Constantin Cranganu · Henri Luchian
Mihaela Elena Breaban *Editors*

# Artificial Intelligent Approaches in Petroleum Geosciences

Springer

# Artificial Intelligent Approaches in Petroleum Geosciences

Constantin Cranganu · Henri Luchian
Mihaela Elena Breaban
Editors

# Artificial Intelligent Approaches in Petroleum Geosciences

Springer

*Editors*
Constantin Cranganu
Brooklyn College
Brooklyn, NY
USA

Mihaela Elena Breaban
University of Iaşi
Iaşi
Romania

Henri Luchian
University of Iaşi
Iaşi
Romania

ISBN 978-3-319-16530-1 ISBN 978-3-319-16531-8 (eBook)
DOI 10.1007/978-3-319-16531-8

Library of Congress Control Number: 2015933823

Springer Cham Heidelberg New York Dordrecht London


Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

# Preface

Integration, handling data of immense size and uncertainty, and dealing with risk management are among crucial issues in petroleum geosciences. The problems one has to solve in this domain are becoming too complex to rely on a single discipline for effective solutions, and the costs associated with poor predictions (e.g., dry holes) increase. Therefore, there is a need to establish new approaches aimed at proper integration of disciplines (such as petroleum engineering, geology, geophysics, and geochemistry), data fusion, risk reduction, and uncertainty management.

This book presents several artificial intelligent approaches[1] for tackling and solving challenging practical problems from the petroleum geosciences and petroleum industry. Written by experienced academics, this book offers state-of-the-art working examples and provides the reader with exposure to the latest developments in the field of artificial intelligent methods applied to oil and gas research, exploration, and production. It also analyzes the strengths and weaknesses of each method presented using benchmarking, while also emphasizing essential parameters such as robustness, accuracy, speed of convergence, computer time, overlearning, or the role of normalization.

The reader of this book will benefit from exposure to the latest developments in the field of modern heuristics applied to oil and gas research, exploration, and production. These approaches can be used for uncertainty analysis, risk assessment, data fusion and mining, data analysis and interpretation, and knowledge discovery, from diverse data such as 3-D seismic, geological data, well logging, and production data. Thus, the book is intended for petroleum scientists, data miners, data scientists and professionals, and postgraduate students involved in the petroleum industry.

Petroleum Geosciences are—like many other fields—a paradigmatic realm of difficult optimization and decision-making real-world problems. As the number,

[1] Artificial Intelligence methods, some of which are grouped together in various ways, under names such as *Computational Intelligence*, *Soft Computing*, *Meta-heuristics*, or *Modern heuristics*.

difficulty, and scale of such specific problems increase steadily, the need for diverse, adjustable problem-solving tools can hardly be satisfied by the necessarily limited number of approaches typically included in a curriculum/syllabus from academic fields other than Computer Science (such as Petroleum Geology). Therefore, the first three chapters of this volume aim at providing working information about modern problem-solving tools, in particular in machine learning and in data mining, and also at inciting the reader to look further into this thriving topic.

Traditionally, solving a given problem in mathematics and in sciences at large implies the construction of an abstract model, the process of proving theoretical results valid in that model, and eventually, based on those theoretical results, the design of a method for solving the problem. This problem-solving paradigm has been and will continue to be immensely successful. Nevertheless, an abstract model is an approximation of the real-world problem; there have been failures triggered by a tiny mismatch between the original problem and the proposed model for it. Furthermore, a problem-solving method developed in this manner is likely to be useful only for the problem at hand. While, ultimately, any problem-solving technique may be—in various degrees—subject to these two observations, some relatively new approaches illustrate alternative lines of attack; it is the editors' hope that the first three chapters of the book illustrate this idea in a way that will prove to be useful to the readers.

In the first chapter, Simovici presents some of the main paradigms of intelligent data analysis provided by machine learning and data mining. After discussing several types of learning (supervised, unsupervised, semi-supervised, active, and reinforcement learning), he examines several classes of learning algorithms (naïve Bayes classifiers, decision trees, support vector machines, and neural networks) and the modalities to evaluate their performance. Examples of specific applications of algorithms are given using System R.

The second and third chapters, by Luchian, Breaban, and Bautu, are dedicated to *meta-heuristics*. After a rather simple introduction to the topic, the second chapter presents, based on working examples, evolutionary computing in general and, in particular, genetic algorithms and differential evolution; particle swarm optimization is also extensively discussed. Topics of particular importance, such as multimodal and multi-objective problems, hybridization, and also applications in petroleum geosciences are discussed based on concrete examples. The third chapter gives a compact presentation of genetic programming, gene expression programming, and also discusses an R package for genetic programming and applications of GP for solving specific problems from the oil and gas industry.

Ashena and Thonhauser discuss the Artificial Neural Networks (ANNs), which has the potential to increase the ability of problem solving in geosciences and in the petroleum industry, particularly in case of limited availability or lack of input data. ANN applications have become widespread because they proved to be able to produce reasonable outputs for inputs they have not learned how to deal with. The following subjects are presented: artificial neural networks basics (neurons, activation function, ANN structure), feed-forward ANN, back-propagation and learning, perceptrons and back-propagation, multilayer ANNs and back-propagation

algorithm, data processing by ANN (training, overfitting, testing, validation), ANN, and statistical parameters. An applied example of ANN, followed by applications of ANN in geosciences and petroleum industry complete the chapter.

Al-Anazi and Gates present the use of support vector regression to accurately estimate two important geomechanical rock properties, Poisson's ratio and Young's modulus. Accurate prediction of rock elastic properties is essential for wellbore stability analysis, hydraulic fracturing design, sand production prediction and management, and other geomechanical applications. The two most common required material properties are Poisson's ratio and Young's modulus. These elastic properties are often reliably determined from laboratory tests by using cores extracted from wells under simulated reservoir conditions. Unfortunately, most wells have limited core data. On the other hand, wells typically have log data. By using suitable regression models, the log data can be used to extend knowledge of core-based elastic properties to the entire field. Artificial neural networks (ANN) have proven to be successful in many reservoir characterization problems. Although nonlinear problems can be well resolved by ANN-based models, extensive numerical experiments (training) must be done to optimize the network structure. In addition, generated regression models from ANNs may not perfectly generalize to unseen input data. Recently, support vector machines (SVMs) have proven successful in several real-world applications for its potential to generalize and converge to a global optimal solution. SVM models are based on the structural risk minimization principle that minimizes the generalization error by striking a balance between empirical training errors and learning machine capacity. This has proven superior in several applications to the empirical risk minimization principle adopted by ANNs that aims to reduce the training error only. Here, support vector regression (SVR) to predict Poisson's ratio and Young's modulus is described. The method uses a fuzzy-based ranking algorithm to select the most significant input variables and filter out dependency. The learning and predictive capabilities of the SVR method is compared to that of a back-propagation neural network (BPNN). The results demonstrate that SVR has similar or superior learning and prediction capabilities to that of the BPNN. Parameter sensitivity analysis was performed to investigate the effect of the SVM regularization parameter, the regression tube radius, and the type of kernel function used. The result shows that the capability of the SVM approximation depends strongly on these parameters.

The next three chapters introduce the active learning method (ALM) and present various applications of it in petroleum geosciences.

First, Cranganu, and Bahrpeyma use ALM to predict a missing log (DT or sonic log) when only two other logs (GR and REID) are present. In their approach, applying ALM involves three steps: (1) supervised training of the model, using available GR, REID, and DT logs; (2) confirmation and validation of the model by blind-testing the results in a well containing both the predictors (GR, REID) and the target (DT) values; and (3) applying the predicted model to wells containing the predictor data and obtaining the synthetic (simulated) DT values. Their results indicate that the performance of the algorithm is satisfactory, while the performance time is significantly low. The quality of the simulation procedure was assessed by

three parameters, namely mean square error (MSE), mean relative error (MRE), and Pearson product momentum correlation coefficient (R). The authors employed both the measured and simulated sonic log DT to predict the presence and estimate the depth intervals where overpressured fluid zone may develop in the Anadarko Basin, Oklahoma. Based on interpretation of the sonic log trends, they inferred that overpressure regions are developing between ~1,250 and 2,500 m depth and the overpressured intervals have thicknesses varying between ~700 and 1,000 m. These results match very well previous published results reported in the Anadarko Basin, using the same wells, but different artificial intelligent approaches.

Second, Bahrpeyma et al. employed ALM to estimate another missing log in hydrocarbon reservoirs, namely the density log. The regression and normalized mean squared error (MSE) for estimating density log using ALM were equal to 0.9 and 0.042, respectively. The results, including errors and regression coefficients, proved that ALM was successful in processing the density estimation. In their chapter, the authors illustrated ALM by an example of a petroleum field in the NW Persian Gulf.

Third, Bahrpeyma et al. tackled the common issue when reservoir engineers should analyze the reservoirs with small sets of measurements (this problem is known as the small sample size problem). Because of small sample size problem, modeling techniques commonly fail to accurately extract the true relationships between the inputs and the outputs used for reservoir properties prediction or modeling. In this chapter, small sample size problem is addressed for modeling carbonate reservoirs by using the active learning method (ALM). Noise injection technique, which is a popular solution to small sample size problem, is employed to recover the impact of separating the validation and test sets from the entire sample set in the process of ALM. The proposed method is used to model hydraulic flow units (HFUs). HFUs are defined as correlatable and mappable zones within a reservoir controlling the fluid flow. This research presents quantitative formulation between flow units and well log data in one of the heterogeneous carbonate reservoirs in Persian Gulf. The results for $R$ and $nMSE$ are 85 % and 0.0042, respectively, which reflect the ability of the proposed method to improve generalization ability of the ALM when facing with sample size problem.

Dobróka and Szabó carried out a well log analysis by global optimization-based interval inversion method. Global optimization procedures, such as genetic algorithms and simulated annealing methods, offer robust and highly accurate solution to several problems in petroleum geosciences. The authors argue that these methods can be used effectively in the solution of well-logging inverse problems. Traditional inversion methods are used to process the borehole geophysical data collected at a given depth point. As having barely more types of probes than unknowns in a given depth, a set of marginally overdetermined inverse problems has to be solved along a borehole. This single inversion scheme represents a relatively noise-sensitive interpretation procedure. To reduce the noise, the degree of overdetermination of the inverse problem must be increased. This condition can be achieved by using a so-called interval inversion method, which inverts all data from a greater depth interval jointly to estimate petrophysical parameters of hydrocarbon reservoirs to

the same interval. The chapter gives a detailed description of the interval inversion problem, which is then solved by a series expansion-based discretization technique. The high degree of overdetermination significantly increases the accuracy of parameter estimation. The quality improvement in the accuracy of estimated model parameters often leads to a more reliable calculation of hydrocarbon reserves. The knowledge of formation boundaries is also required for reserve calculation. Well logs contain information about layer thicknesses, which cannot be extracted by the traditional local inversion approach. The interval inversion method is applicable to derive the layer boundary coordinates and certain zone parameters involved in the interpretation problem automatically. In this chapter, the authors analyzed how to apply a fully automated procedure for the determination of rock interfaces and petrophysical parameters of hydrocarbon formations. Cluster analysis of well-logging data is performed as a preliminary data-processing step before inversion. The analysis of cluster number log allows the separation of formations and gives an initial estimate for layer thicknesses. In the global inversion phase, the model including petrophysical parameters and layer boundary coordinates is progressively refined to achieve an optimal solution. The very fast simulated reannealing method ensures the best fit between the measured data and theoretical data calculated on the model. The inversion methodology is demonstrated by a hydrocarbon field example, with an application for shaly sand reservoirs.

Finally, Mohebbi and Kaydani undertake a detailed review of meta-heuristics dealing with permeability estimation in petroleum reservoirs. They argue that proper permeability distribution in reservoir models is very important for the determination of oil and gas reservoir quality. In fact, it is not possible to have accurate solutions in many petroleum engineering problems without having accurate values for this key parameter of hydrocarbon reservoir. Permeability estimation by individual techniques within the various porous media can vary with the state of in situ environment, fluid distribution, and the scale of the medium under investigation. Recently, attempts have been made to utilize meta-heuristics for the identification of the relationship that may exist between the well log data and core permeability. This chapter overviews the different meta-heuristics in permeability prediction, indicating the advantages of each method. In the end, some suggestions and comments about how to choose the best method are presented.

December 2014

Constantin Cranganu
Henri Luchian
Mihaela Elena Breaban

# Contents

**Intelligent Data Analysis Techniques—Machine Learning and Data Mining** . . . . . 1
Dan Simovici

**On Meta-heuristics in Optimization and Data Analysis. Application to Geosciences** . . . . . 53
Henri Luchian, Mihaela Elena Breaban and Andrei Bautu

**Genetic Programming Techniques with Applications in the Oil and Gas Industry** . . . . . 101
Henri Luchian, Andrei Băutu and Elena Băutu

**Application of Artificial Neural Networks in Geoscience and Petroleum Industry** . . . . . 127
Rahman Ashena and Gerhard Thonhauser

**On Support Vector Regression to Predict Poisson's Ratio and Young's Modulus of Reservoir Rock** . . . . . 167
A.F. Al-Anazi and I.D. Gates

**Use of Active Learning Method to Determine the Presence and Estimate the Magnitude of Abnormally Pressured Fluid Zones: A Case Study from the Anadarko Basin, Oklahoma** . . . . . 191
Constantin Cranganu and Fouad Bahrpeyma

**Active Learning Method for Estimating Missing Logs in Hydrocarbon Reservoirs** . . . . . 209
Fouad Bahrpeyma, Constantin Cranganu and Behrouz Zamani Dadaneh

**Improving the Accuracy of Active Learning Method via Noise Injection for Estimating Hydraulic Flow Units: An Example from a Heterogeneous Carbonate Reservoir** . . . . . 225
Fouad Bahrpeyma, Constantin Cranganu and Bahman Golchin

**Well Log Analysis by Global Optimization-based Interval Inversion Method** . . . . . 245
Mihály Dobróka and Norbert Péter Szabó

**Permeability Estimation in Petroleum Reservoir by Meta-heuristics: An Overview** . . . . . 269
Ali Mohebbi and Hossein Kaydani

**Index** . . . . . 287

# Intelligent Data Analysis Techniques—Machine Learning and Data Mining

**Dan Simovici**

**Abstract** This introductory chapter presents some of the main paradigms of intelligent data analysis provided by machine learning and data mining. After discussing several types of learning (supervised, unsupervised, semi-supervised, active and reinforcement learning) we examine several classes of learning algorithms (naive Bayes classifiers, decision trees, support vector machines, and neural networks) and the modalities to evaluate their performance. Examples of specific applications of algorithms are given using System R.

**Keywords** Supervised learning · Unsupervised learning · Clustering · Generalization · Overfitting · Active learning · Classifiers · A priori probabilities · A posteriori probabilities · Decision trees · Entropy · Impurity · Naive Bayes classifiers · Perceptrons · Neural Networks

## 1 Introduction

Machine learning and its applied counterpart, data mining, deal with problems that present difficulties in formulating algorithms that can be readily translated into programs, due to their complexity. Examples of such problems are finding diagnosis for patients starting with a series of their symptoms, determining credit worthiness of customers based on their demographics and credit history. In each of these problems, the challenge is to compute a label for each analyzed piece of data that depends on the characteristics of data.

The general approach known as *supervised learning* is to begin with a number of labeled examples (where answers are known) in order to generate an algorithm that computes the function that gives the answers starting from these examples.

D. Simovici (✉)
Department of Computer Science, University of Massachusetts Boston, Boston, MA, USA
e-mail: dsim@cs.umb.edu



C. Cranganu et al. (eds.), *Artificial Intelligent Approaches in Petroleum Geosciences*,
DOI 10.1007/978-3-319-16531-8_1

In other approaches in machine learning, the challenge is to identify structure that is hidden in data, e.g., identifying groups of data such that strong similarity exists between objects that belong to the same group and also that objects that belong to different groups are sufficiently distinct. This activity is known as *clustering* and belongs to the category of *unsupervised learning*. The term "unsupervised" refers to the fact that this type of learning does not require operator intervention. Other machine learning activities of this type include outlier identification and density estimation.

An intermediate type of activity, referred as *semi-supervised learning*, requires a limited involvement of the operator. For example, in the case of clustering, this may allow the operator to specify pairs of objects that must belong to the same group and pairs of objects that may not belong to the same group.

The quality of the learning process is assessed through its capability for *generalization*, that is, the capacity of the produced algorithm for computing correct labels for yet unseen examples. It is important to note that the correct behavior of an algorithm relative to the training data is no guarantee, in general, for its generalization prowess. Indeed, it is sometime the case that the pursuit of a perfect fit of the learning algorithm to the training data leads to *overfitting*. This term describes the situation when the algorithm acts correctly on the training data but is unable to predict unseen data. In an extreme case, a *rote learner* will memorize the labels of its training data and nothing else. Such a learner will be perfectly accurate on its training data but lack completely any generalization capability.

A machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns, that is, to apply *active learning*. An active learner may pose queries soliciting a human operator to label a data instance. Since unlabeled data are abundant and, in many cases, easily obtained, there are good reasons to use this learning paradigm.

Reinforcement learning is a machine learning paradigm inspired by psychology which emphasizes learning by an agent from its direct interaction with the data in order to attain certain goals of learning, e.g., accuracy of label prediction. The framework of this type of learning makes use of states and actions of an agent, and the rewards and deals with uncertainty and non-determinism.

Machine learning techniques can be applied to a wide variety of problems and tend to avoid the difficulties of standard problem-solving techniques where a complete understanding of data is required at the beginning of the problem-solving process.

We have selected system R to provide examples of applications of algorithms presented in this chapter. This is one of the most popular, freely available software system for statistics and machine learning, that is continuously expanded by a large community of developers that have created packages that address certain problems. The basic software is available from http://www.r-project.org/. Packages can be obtained from many mirrors of the software that can be easily accessed after the basic system is installed.

Data sets used in R are either part of the basic software or can be downloaded from the University of California Irvine machine learning repository whose URL is http://archive.ics.uci.edu/ml/. The basic R system is capable of reading files in the

csv format, which is one of the most common modalities for uploading data. For example, to create a data frame d by reading the file d.csv, one could use

```
d <- read.csv("d.csv")
```

To learn the basics of R, the reader is invited to consult one of the basic references (Lander 2014; Maindonald and Braun 2004) or seek help on the Web.

## 2 Simple Classifiers

We present now several types of classifiers using two of the most popular data sets, namely Fisher's *iris* data and the *tennis* data.

*Example 2.1* The *iris* data were collected by Anderson (1936), an American botanist who was interested in the study of variations in three species of iris flowers in Gaspè peninsula in northeastern Canada and was made popular in statistics by Fisher (1936).

Fisher's iris data consist of measurements on 150 of iris specimens and include measurements of sepal length, sepal width, petal length, and petal width, as well as the species of the plants. The attributes that are distinct from the class are numerical, so each plant is represented by a point in $\mathbb{R}^4$. The species identified are *iris setosa*, *iris versicolor*, and *iris virginica,* and there are 50 specimens from each of these species, as shown in Table 1.

We will use various types of classifiers as they are implemented in system R, one of the most used pieces of software for data analysis, which is freely available on the Internet.

The *iris* data set is a part of the basic R package and can be loaded using

```
> data(iris)
```

The structure of this data set can be obtained using

```
> str(iris)
```

which returns a summary description:

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..:1 1 ...
```

*Example 2.2* The tennis data set shown in Table 2 is a fictitious small data set that specifies conditions from playing an outdoor game. It contains five attributes: outlook, temperature, humidity, windy, and play.

**Table 1** Fisher's iris data set

| Sepal length | Sepal width | Petal length | Petal width | Species |
|---|---|---|---|---|
| SL | SW | PL | PW | |
| 5.1 | 3.5 | 1.4 | 0.2 | Setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | Setosa |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 5.3 | 3.7 | 1.5 | 0.2 | Setosa |
| 5.0 | 3.3 | 1.4 | 0.2 | Setosa |
| 7.0 | 3.2 | 4.7 | 1.4 | Versicolor |
| 6.4 | 3.2 | 4.5 | 1.5 | Versicolor |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 5.1 | 2.5 | 3.0 | 1.1 | Versicolor |
| 5.7 | 2.8 | 4.1 | 1.3 | Versicolor |
| 6.3 | 3.3 | 6.0 | 2.5 | Virginica |
| 5.8 | 2.7 | 5.1 | 1.9 | virginica |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 6.2 | 3.4 | 5.4 | 2.3 | Virginica |
| 5.9 | 3.0 | 5.1 | 1.8 | Virginica |

**Table 2** Tennis data set

| `outlook` | `temperature` | `humidity` | `windy` | `play` |
|---|---|---|---|---|
| Sunny | Hot | High | No | No |
| Sunny | Hot | High | Yes | No |
| Overcast | Hot | High | No | Yes |
| Rainy | Mild | High | No | Yes |
| Rainy | Cool | Normal | No | Yes |
| Rainy | Cool | Normal | Yes | No |
| Overcast | Cool | Normal | Yes | Yes |
| Sunny | Mild | High | No | No |
| Sunny | Cool | Normal | No | Yes |
| Rainy | Mild | Normal | No | Yes |
| Sunny | Mild | Normal | Yes | Yes |
| Overcast | Mild | High | Yes | Yes |
| Overcast | Hot | Normal | No | Yes |
| Rainy | Mild | High | Yes | No |

The data can be placed in a comma-separated EXCEL file `tennis.csv` and then loaded in R using a statement of the form

```
tennis <- read.csv("tennis.csv")
```

## 2.1 Bayes Classification and Naive Bayesian Classifiers

Suppose that a data set $D$ consists of $n$ non-empty and mutually disjoint classes

$$C_1, \ldots, C_m.$$

Let $P(D|C_i)$ be the probability that a datum $\mathbf{x}$ in $D$ belongs to $C_i$ for $1 \leqslant i \leqslant m$.

Bayes classifiers determine the class of $\mathbf{x} \in D$ as one of the classes $C_1, \ldots, C_n$. by computing the conditional probabilities $P(C_i|x)$ and assigning $x$ to the class $C_k$ where

$$k = \mathsf{argmax}_i P(C_i|\mathbf{x}).$$

The probabilities $P(C_i|\mathbf{x})$ are known as a posteriori probabilities, since they are evaluated after the datum $\mathbf{x}$ is observed and the class $C_k$ is occasionally referred to as the *maximum* a *posteriori class*.

By the Bayes' law, we have

$$P(C_i|\mathbf{x}) = \frac{P(\mathbf{x}|C_i)P(C_i)}{P(\mathbf{x})}$$

for $1 \leqslant i \leqslant n$. Note that $P(\mathbf{x})$ does not influence the selection of $C_k$.

Generally, the probabilities of the classes $P(C_i)$ are referred to as the *prior* or a priori probability of classes, and they may be estimated using one of the following methods:

(i) they may be assumed to be equal, $P(C_i) = \cdots = P(C_n) = \frac{1}{n}$, or
(ii) they can be estimated as the frequencies of the classes $C_i$ in the training population, or
(iii) estimations can be obtained from general domain knowledge.

Another challenge in Bayesian classification is to evaluate probabilities of the form $P(\mathbf{x}|C_i)$. *Naïve Bayes classifiers* add a supplementary independence hypothesis. Namely, if $\mathbf{x} = (x_1, \ldots, x_m)$, we assume that the components $x_1, \ldots, x_m$ are independent of each other, which allows us to write

$$P(\mathbf{x}|C_i) = \prod_{j=1}^{m} P\big(x_j|C_i\big)$$

for $1 \leqslant i \leqslant n$. The probabilities $P(x_j|C_i)$ are usually estimated from the training examples, and the estimation method depends on the nature of each of the attributes $A_1, \ldots, A_m$ that define these components. The classifier will assign $\mathbf{x}$ to the most likely class, that is to the $C_i$ that corresponds to the maximum value of $P(C_i|x)$ and therefore to the class $C_i$ for which $P(\mathbf{x}|C_i) = \prod_{j=1}^{m} P\big(x_j|C_i\big)$ is maximal.

If $A_j$ is a categorical attribute, $P(x_j|C_i)$ can be estimated as $\frac{f_{ji}}{c_i}$, where $c_i$ is the number of training examples in class $C_i$ and $f_{ji}$ is the number of training examples in the class $C_i$ having the value of the $A_j$ component equal to $x_j$.

If $A_j$ is continuous, $P(x_j|C_i)$ can be approximated with the normal distribution. If $\mu_i$ and $\sigma_i$ are the mean and the standard deviation of the examples of the class $C_i$, then we may adopt as an estimate of $P(x_j|C_i)$ the value

$$\frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x_j-\mu_j)^2}{2\sigma_i^2}}.$$

*Example 2.3* In the *tennis* data set, there are two classes determined by the attribute `play`: $C_{\text{yes}}$ and $C_{\text{No}}$, which contain 9 and 5 records, respectively. If the probabilities of these classes are estimated by their frequencies, we will have $P(C_{\text{Yes}}) = \frac{9}{14}$ and $P(C_{\text{No}}) = \frac{5}{14}$. Since all attributes in this example are categorical, the probabilities $P(x_j|C_i)$ are estimated as $\frac{f_{ij}}{c_i}$, where $c_i$ is the number of training examples in class $C_i$ and $f_{ji}$ is the number of training examples in the class $C_i$ having the value of the $A_j$ component equal to $x_j$. In this case, the frequencies are computed in Table 3.

A naive Bayes classifier for this categorical data set is created in R with the package `e1071`. After installing this package, `e1071` is loaded using the directive

```
-> library(e1071)
```

The naive Bayes classifier `nbc` is created by writing:

```
nbc <- naiveBayes(Play ~ .,data = tennis)
```

In the definition of `nbc`, the expression

```
Play ~ .
```

**Table 3** Frequencies in the tennis data set

| Attributes | Values | $P(x\|C_{\text{yes}})$ | $P(x\|C_{\text{no}})$ |
|---|---|---|---|
| `outlook` | Sunny | 2/9 | 3/5 |
| | Overcast | 4/9 | 0/5 |
| | Rainy | 3/9 | 2/5 |
| `temperature` | Hot | 2/9 | 2/5 |
| | Mild | 4/9 | 2/5 |
| | Cool | 3/9 | 1/5 |
| `humidity` | High | 3/9 | 4/5 |
| | Normal | 6/9 | 1/5 |
| `windy` | No | 6/9 | 2/5 |
| | Yes | 3/9 | 3/5 |

is a *model formula* that has the general form

$$\text{class variable} \sim \text{list of explanatory variables}.$$

In this case, `Play` is clearly the class variable; the period "." replaces all other variables. If several variables participate in the list of explanatory variables, they are linked by +.

Displaying the components of `nbc` gives us the prior probabilities and the conditional probabilities $P(x|C)$:

```
A-priori probabilities:
       No       Yes
0.3571429 0.6428571

Conditional probabilities:
     Outlook
      Overcast     Rainy     Sunny
  No  0.0000000 0.4000000 0.6000000
  Yes 0.4444444 0.3333333 0.2222222

     Temp
          Cool       Hot      Mild
  No  0.2000000 0.4000000 0.4000000
  Yes 0.3333333 0.2222222 0.4444444

     Humidity
          High    Normal
  No  0.8000000 0.2000000
  Yes 0.3333333 0.6666667

     Windy
         NO      YES
  No  0.4000000 0.6000000
  Yes 0.6666667 0.3333333
```

We seek to predict the value of the attribute `Play` when the values of the other attributes form a tuple that is absent from the table. This happens when we have the datum **x** given below

| outlook | temperature | humidity | windy |
|---|---|---|---|
| Rainy | Hot | High | YES |

We need to compute the conditional probabilities

$$\begin{aligned} P(\mathbf{x}|C_{\text{yes}}) &= P(\texttt{outlook} = \text{Rainy}|C_{\text{yes}}) \cdot P(\texttt{temperature} = \text{Hot}|C_{\text{yes}}) \\ &\quad \cdot P(\texttt{humidity} = \text{High}\ |C_{\text{yes}}) P(\texttt{windy} = \text{YES}\ |C_{\text{yes}}) \\ &= 3/9 \cdot 2/9 \cdot 3/9 \cdot 3/9 = 54/6561 = 0.0082, \end{aligned}$$

and

$$\begin{aligned} P(\mathbf{x}|C_{\text{no}}) &= P(\texttt{outlook} = \text{Rainy}|C_{\text{no}}) \cdot P(\texttt{temperature} = \text{Hot}|C_{\text{no}}) \\ &\quad \cdot P(\texttt{humidity} = \text{High } |C_{\text{no}}) P(\texttt{windy} = \text{YES } |C_{\text{no}}) \\ &= 2/5 \cdot 2/5 \cdot 4/5 \cdot 3/5 = 48/625 = 0.0768. \end{aligned}$$

A posteriori probabilities are given by

$$\begin{aligned} P(C_{\text{yes}}|\mathbf{x}) &= \frac{P(\mathbf{x}|C_{\text{yes}})P(C_{\text{yes}})}{P(\mathbf{x})} \\ &= \frac{0.0082 \cdot 0.6428571}{P(\mathbf{x})} = \frac{0.00527}{P(\mathbf{x})}, \\ P(C_{\text{no}}|\mathbf{x}) &= \frac{P(x|C_{\text{no}})P(C_{\text{no}})}{P(\mathbf{x})} \\ &= \frac{0.0768 \cdot 0.3571429}{P(\mathbf{x})} = \frac{0.02742}{P(\mathbf{x})}. \end{aligned}$$

Since $P(C_{\text{no}}|\mathbf{x}) > P\big(C_{\text{yes}}|\mathbf{x}\big)$, the classifier will predict "no" for $\mathbf{x}$.

Note that there is no example in the data set where `outlook` = "*Overcast*" and `Play` = "*Yes*". Therefore, $P$ (`outlook` = "*Overcast*"|`Play` = "*Yes*") = 0 and any product of probabilities that includes this factor will be 0. This problem can be fixed by using a technique known as *Laplace correction*. Namely, if the fractions

$$\frac{p_1}{q_1}, \ldots, \frac{p_m}{q_m}$$

are $m$ probabilities such that $\sum_{i=1}^{m} \frac{p_i}{q_i} = 1$, we replace these fractions by

$$\frac{p_1 + \lambda}{q_1 + m\lambda}, \ldots, \frac{p_m + \lambda}{q_m + m\lambda},$$

respectively. None of the newly defined numbers is 0 and we have

$$\frac{p_i}{q_i} \leqslant \frac{p_i + \lambda}{q_i + m\lambda} \leqslant \frac{1}{m}.$$

The parameter $\lambda$ is, in general, a small positive number and is determining how influential the priori values are compared to knowledge extracted from the training set.

To apply a Laplace correction with $\lambda = 1$, we need to write

```
> nbc <- naiveBayes(Play ~ .,data=tennis,laplace=1)
```

Note that the conditional probabilities are modified and there is no null values:

```
A-priori probabilities:

       No       Yes
0.3571429 0.6428571

Conditional probabilities:
     Outlook
      Overcast     Rainy     Sunny
  No  0.1250000 0.3750000 0.5000000
  Yes 0.4166667 0.3333333 0.2500000

     Temp
          Cool       Hot      Mild
  No  0.2500000 0.3750000 0.3750000
  Yes 0.3333333 0.2500000 0.4166667

     Humidity
          High    Normal
  No  0.7142857 0.2857143
  Yes 0.3636364 0.6363636

     Windy
         NO      YES
  No  0.6000000 0.8000000
  Yes 0.7777778 0.4444444
```

*Example 2.4* In this example, we seek to construct a Bayes classifier for a data set that has numerical attributes using the *iris* data set and the package `e1071`.

```
> nbc <- naiveBayes(iris[,1:4],iris[,5])

> table(predict(nbc,iris[,1:4]), iris[,5],
+ dnn=list("predicted","actual"))
```

This will return

```
            actual
 predicted    setosa versicolor virginica
   setosa         50          0         0
   versicolor      0         47         3
   virginica       0          3        47
```

The structure of the classifier returned can be inspected using the statement

```
> str(nbc)
```

which returns

```
List of 4
 $ apriori: 'table' int [1:3(1d)] 50 50 50
  ..- attr(*, "dimnames")=List of 1
  .. ..$ iris[, 5]: chr [1:3] "setosa" "versicolor" "virginica"
 $ tables :List of 4
  ..$ Sepal.Length: num [1:3, 1:2] 5.006 5.936 6.588 0.352 0.516 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ iris[, 5]   : chr [1:3] "setosa" "versicolor" "virginica"
  .. .. ..$ Sepal.Length: NULL
  ..$ Sepal.Width : num [1:3, 1:2] 3.428 2.77 2.974 0.379 0.314 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ iris[, 5]  : chr [1:3] "setosa" "versicolor" "virginica"
  .. .. ..$ Sepal.Width: NULL
  ..$ Petal.Length: num [1:3, 1:2] 1.462 4.26 5.552 0.174 0.47 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ iris[, 5]   : chr [1:3] "setosa" "versicolor" "virginica"
  .. .. ..$ Petal.Length: NULL
  ..$ Petal.Width : num [1:3, 1:2] 0.246 1.326 2.026 0.105 0.198 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ iris[, 5]  : chr [1:3] "setosa" "versicolor" "virginica"
  .. .. ..$ Petal.Width: NULL
 $ levels : chr [1:3] "setosa" "versicolor" "virginica"
 $ call   : language naiveBayes.default(x = iris[, 1:4], y = iris[, 5])
 - attr(*, "class")= chr "naiveBayes"
```

## 2.2 Decision Trees

Decision trees are algorithms that build classification models based on a chain of partitions of the training set. Depending on the nature of data (categorical or numerical), we need to choose a particular type of decision tree.

Decision trees are built through recursive data partitioning, where in each iteration, the training data are split according to the values of a selected attribute. Each node $n$ corresponds to a subset $D(n)$ of the training data set $D$ and to a partition $\pi(n)$ of $D(n)$. If $n_0$ is the root of the decision tree, then $D(n_0) = D$. If $n$ is a node that has the descendants $n_1, \ldots, n_k$, then

$$\pi(n) = \{D(n_1), \ldots, D(n_k)\}.$$

In other words, the blocks of the partition $\pi(n)$ are the data sets that correspond to the descendant nodes $n_1, \ldots, n_k$. Partitioning of a set $D(n)$ is done, in general, on the basis of the values of the attributes of the objects assigned to the node $n$.

Suppose that the training data is labeled by $c_1, \ldots, c_m$. This, in turn, determines a partition $\sigma = \{C_1, \ldots, C_m\}$ of the training set, where the block $C_j$ contains the data records labeled $c_j$ for $i \leqslant j \leqslant m$. If $E$ is a subset of $D$, the *purity* of $E$ equals the entropy of the trace partition $\sigma_E$ (see Sect. 8B). The set $E$ is pure if $\sigma_E$ consists of

exactly one block, that is, $\mathcal{H}(\sigma_E) = 0$; in other words, $E$ is pure if its elements belong to exactly one classes.

The recursive splitting of the nodes stops at nodes that correspond to "pure" or "almost pure" data subsets, that is, when the data of the node consist of instances of the same class, or when a class is strongly predominant at that node. Nodes where splitting stops are the leaves of the decision trees.

There are three issues in constructing a decision tree (Breiman et al. 1998):

(i) choosing a splitting criterion that generates a partition of $D(n)$;
(ii) deciding when a node should not be split further, that is, when a node is *terminal*;
(iii) the assignment of each terminal node to a class.

Splitting the data set $D(n)$ aims to produce nodes with increasing purity. Assume that $n$ is split $k$ ways to generate the descendants $n_1, \ldots, n_k$ that contain the data sets $D(n_1), \ldots, D(n_k)$. The *splitting partition* $\sigma_n$ at $n$ is defined as

$$\sigma = \{D(n_1), \ldots, D(n_k)\}.$$

If $\kappa$ is the partition of the original data set in classes, the $a$-impurity at $n$ is $\mathcal{H}_a(\kappa_{D(n)})$. The aggregate $a$-impurity of the descendants of $n$ is

$$\sum_{j=1}^{k} \mathcal{H}_a\left(\kappa_{D(n_j)}\right)\left(\frac{|D(n_j)|}{|D(n)|}\right)^a = \mathcal{H}_a(\kappa_{D(n)}|\sigma_n)$$

and, therefore, the decrease in impurity afforded by the splitting $\sigma_n$ is

$$\mathcal{H}_a(\kappa_{D(n)}) - \mathcal{H}_a(\kappa_{D(n)}|\sigma_n).$$

This quantity is known as the *information gain* caused by $\sigma_n$, and it is the basis of one of the best known method for constructing decision trees, namely the C5.0 algorithm of Quinlan (1993). Variants of this algorithm are also popular [e.g., the `J48` of the WEKA software package (Witten et al. 2011)].

The construction of a C5.0 tree in the C50 package can be achieved by writing

```
C5.0(trainData,classVector, trials = t, costs = c)
```

where the first parameter specifies the data set on which the classifier is constructed and the second parameter is a factor vector which contains the class for each row of the training data; the remaining parameters are optional and will be discussed in the sequel.

*Example 2.5* To generate a decision tree for the `iris` data set, we split this data into a training data set, `trainIris`, and a test data set, `testIris` by writing

```
> index <- sample(2,nrow(iris),replace=TRUE,prob=c(0.9,0.1))
> trainIris <- iris[index==1,]
> testIris <- iris[index==2,]
```

About 90 % of the entries in this index have value 1 and about 10 % contain the value 2, which correspond to the training set and the test set, respectively.

The classifier `dt` is built using the syntax

```
dt <- C5.0(trainIris[,1:4],trainIris[,5])
```

The classes predicted for the test set are obtained with

```
> pred <- predict(dt,testIris[,1:4],type="class")
> pred
     setosa     setosa     setosa     setosa     setosa
     versicolor versicolor versicolor versicolor versicolor
     versicolor versicolor virginica  virginica  virginica
     virginica  virginica  virginica  virginica
Levels: setosa versicolor virginica
```

A summary of the classifier `summary (dt)` returns the specifics of the decision tree

```
Decision tree:

Petal.Length <= 1.9: setosa (45)
Petal.Length > 1.9:
:...Petal.Width > 1.7: virginica (39/1)
    Petal.Width <= 1.7:
    :...Petal.Length <= 4.9: versicolor (41/1)
        Petal.Length > 4.9: virginica (6/2)
Evaluation on training data (131 cases):

            Decision Tree
          ----------------
          Size      Errors
             4    4( 3.1%)   <<


           (a)   (b)   (c)    <-classified as
          ----  ----  ----
            45                (a): class setosa

                  40     3    (b): class versicolor
                   1    42    (c): class virginica
```

The parameter `trials` refers to a very important technique in machine learning called *boosting*. Boosting refers to a method of producing a very accurate classifier by combining moderately inaccurate classifiers. Using `trials`, we can specify the number of boosting iterations.

Note that the classifier generated in Example 2.5 produced four erroneous predictions. A matrix of costs can be associated with these mistakes such that the costs depend on the nature of the errors. For instance, since we have three classes designated as (a), (b), and (c), we could consider the cost matrix

$$\texttt{costs} = \begin{pmatrix} 0 & 2 & 0 \\ 4 & 0 & 5 \\ 0 & 1 & 0 \end{pmatrix}$$

These entries of this matrix assign a cost to mistakes made during the classification. Rows correspond to predicted values and columns to actual values; the diagonal elements are 0. Thus, the costliest error of the classifier is to predict (b) for an object in the class (c).

## 3 Evaluation of Performance of Classifiers

Consider a simple classification algorithm involving the diagnosis of a condition based on the value of a test result. A disease is predicted when the value of a test $t$ result is greater than 5; patients who satisfy this condition constitute the positive set which contains $P(t)$ elements; the other patients form the set of negative cases which consist of $N(t)$, as we show in Fig. 1. Suppose initially that the distribution of cases is the one shown in Fig. 1a. In this case, the test results are decisive: Patients with test values of at least 5 have a positive diagnosis, while patients with values lower than 5 have a negative diagnosis. Such well-delimited situations are infrequent. More likely, the curves that give the probability densities intersect, as we show in Fig. 1b.

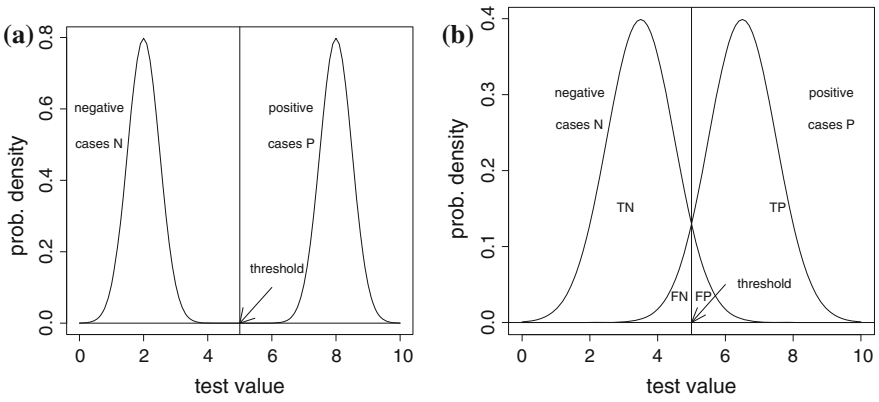


**Fig. 1** Relative positions of distributions of test results. **a** Well-separated results. **b** Positive and negative results overlap

Note that the set of test values of individuals who have the disease overlaps with the set of test values of those who do not have the disease. These sets are represented in Fig. 1b by the areas $\mathsf{P}$ and $\mathsf{N}$ located under each of the two curves.

The diagnosis is determined by the value of a test threshold: Patients whose test values exceed the threshold are deemed to be positive (that is, have the disease); patients whose test values are lower than the threshold are deemed to be negative.

Some patients who have the disease but whose test results are lower than the threshold will be classified by this simple test among the negative cases (there are the *false-negative cases*); others, who do not have the disease but whose test values are larger than the threshold, will be classified among the positive cases (they are the *false-positive cases*). The number of elements of these sets is denoted by $\mathsf{FN}(\mathsf{t})$ and $\mathsf{FN}(t)$, respectively.

The set of patients who have the disease and are correctly identified by the test forms the set of *true-positive cases*; the number of elements of this set is denoted by $\mathsf{TP}(t)$. Also, the set of patients who do not have the disease and are correctly identified forms the set of *true-negative cases*; the number of elements of this set is $\mathsf{TN}(t)$. Clearly, we have

$$\begin{aligned}\mathsf{N} &= \mathsf{TN}(t) + \mathsf{FP}(t),\\ \mathsf{P} &= \mathsf{TP}(t) + \mathsf{FN}(t).\end{aligned}$$

Note that the total number of cases $\mathsf{N}$ and $\mathsf{P}$ does not depend on $t$. The definitions are summarized in Table 4 known as the confusion matrix or confusion table.

Among these cases, the number of incorrectly classified cases is $\mathsf{FP}(t) + \mathsf{FN}(t)$; this motivates the introduction of the *error rate* $\mathsf{error}(t)$ as

$$\mathsf{error}(t) = \frac{\mathsf{FP}(t) + \mathsf{FN}(t)}{\mathsf{N} + \mathsf{P}}.$$

Note that $\mathsf{error}(t) \in [0, 1]$ for every value of $t$. The *accuracy* at $t$ is

$$\mathsf{acc}(t) = 1 - \mathsf{error} = \frac{\mathsf{TP}(t) + \mathsf{TN}(t)}{\mathsf{P} + \mathsf{N}}.$$

The *specificity* at $t$ (also known as the *true-negative rate*) is defined as:

**Table 4** Confusion table

| | | True class | |
|---|---|---|---|
| | | Positive | Negative |
| Classifier result for threshold $t$ | Positive | $\mathsf{TP}(t)$ | $\mathsf{FP}(t)$ |
| | Negative | $\mathsf{FN}(t)$ | $\mathsf{TN}(t)$ |
| Totals | | $\mathsf{P}$ | $\mathsf{N}$ |

$$\mathsf{specificity}(t) = \frac{\mathsf{TN}(t)}{\mathsf{N}}.$$

Specificity can be regarded as a conditional probability, namely,

$$\mathsf{specificity}(t) = P(\mathsf{TN}(t)|\mathsf{N}).$$

Similarly, the *sensitivity* at $t$ (also known as the *true-positive rate*) or the *recall* is given by

$$\mathsf{sensitivity}(t) = \frac{\mathsf{TP}(t)}{\mathsf{P}},$$

and can be expressed as the conditional probability $\mathsf{sensitivity}(t) = P(\mathsf{TP}(t)|\mathsf{P})$.

High values of specificity occur when there are few false positives; low sensitivity indicates the presence of many false negatives.

The *precision* at $t$ is

$$\mathsf{precision}(t) = \frac{\mathsf{TP}(t)}{\mathsf{TP}(t) + \mathsf{FP}(t)}.$$

Note that

$$0 \leqslant \mathsf{specificity}(t),\ \mathsf{sensitivity}(t),\ \mathsf{precision}(t) \leqslant 1$$

for every value of $t$. Also, we have

$$\begin{aligned}
\mathsf{specificity}(t) &= \frac{\mathsf{TN}(t)}{\mathsf{TN}(t) + \mathsf{FP}(t)},\\
\mathsf{sensitivity}(t) &= \frac{\mathsf{TP}(t)}{\mathsf{TP}(t) + \mathsf{FN}(t)},\\
\mathsf{precision}(t) &= \frac{\mathsf{TP}(t)}{\mathsf{TP}(t) + \mathsf{FP}(t)}.
\end{aligned}$$

It is easy to verify that for any four positive numbers $a, b, c, d$, we have the double inequality

$$\min\left\{\frac{a}{b}, \frac{c}{d}\right\} \leqslant \frac{a+c}{b+d} \leqslant \max\left\{\frac{a}{b}, \frac{c}{d}\right\}.$$

This implies

$$\min\left\{\frac{\mathsf{TP}(t)}{\mathsf{P}}, \frac{\mathsf{TN}(t)}{\mathsf{N}}\right\} \leqslant \frac{\mathsf{TP}(t) + \mathsf{TN}(t)}{\mathsf{P} + \mathsf{N}} \leqslant \max\left\{\frac{\mathsf{TP}(t)}{\mathsf{P}}, \frac{\mathsf{TN}(t)}{\mathsf{N}}\right\},$$

which is equivalent to

$$\min\{\mathsf{specificity}(t),\ \mathsf{sensitivity}(t)\} \leqslant \mathsf{acc}(t) \leqslant \mathsf{max}\{\mathsf{specificity}(t),\ \mathsf{sensitivity}(t)\}.$$

In other words, the accuracy of $t$ always lies between the sensitivity and the specificity at $t$.

Note that $1 - \mathsf{specificity}(t) = 1 - \frac{\mathsf{TN}(t)}{\mathsf{N}} = \frac{\mathsf{FP}(t)}{\mathsf{N}}$. This justifies referring to $1 - \mathsf{specificity}(t)$ as the *false-positive rate*.

The $F_1$ score considers both the precision and the sensitivity rates and is defined as their harmonic mean

$$F_1(t) = 2\frac{\mathsf{precision}(t)\ *\ \mathsf{sensitivity}(t)}{\mathsf{precision}(t) + \mathsf{sensitivity}(t)}.$$

A more general measure is $F_\beta$ given by

$$F_\beta(t) = (1+\beta^2)\frac{\mathsf{precision}(t)\ *\ \mathsf{sensitivity}(t)}{\beta^2\mathsf{precision}(t) + \mathsf{sensitivity}(t)}.$$

Note that $F_2$ weighs sensitivity higher than precision, while $F_{0.5}$ weighs precision higher than sensitivity.

## 4 Support Vector Machines

Support vector machines (SVMs) represent a powerful technique in classification, regression, and outlier detection. SVMs were developed by Cortes and Vapnik (1995) for binary classification.

The simplest application of these algorithms is solving the binary classification problem which seeks to separate two classes of vectors in $\mathbb{R}^n$ by determining an optimum separating hyperplane for the classes involved. The two classes of vectors involved are known as the positive examples and the negative examples, and the separating hyperplane must be determined such that the separation between the closest representatives of the two classes is maximized.

Building a separating hyperplane amounts to building a classifier model and the process begins, as it is customary in classification, with a *training set* $T$ that consists of $m$ pairs of the form

$$(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_m, \mathbf{y}_m),$$

where $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^n$ and $\mathbf{y}_i \in \{-1, 1\}$ for $1 \leqslant i \leqslant m$. The sets

$$T_+ = \{\mathbf{x}_i | (\mathbf{x}_i, 1) \in T\},$$
$$T_- = \{\mathbf{x}_i | (\mathbf{x}_i, -1) \in T\}$$

are the set of *positive examples* and the set of *negative examples*, respectively.

$T$ is *linearly separable* if there exists a hyperplane $H_{\mathbf{v},\mathrm{a}} : \mathbf{v}'\mathbf{x} = a$ (called the *separating hyperplane*) such that all positive examples lie in one half-space determined by $H_{\mathbf{v},a}$ and all negative examples lie in the other half-space as shown in Fig. 2. In other words, $\mathbf{v}$ and $a$ can be chosen such that for all positive examples we shall have $\mathbf{v}'\mathbf{x}_i - a > 0$ and for all negative examples we shall have $\mathbf{v}'\mathbf{x}_i - a < 0$. Both conditions can be stated as

$$y_i(\mathbf{v}'\mathbf{x}_i - a) > 0 \tag{1}$$

for $1 \leqslant i \leqslant m$.

The distance between a point $\mathbf{x}_i$ and the hyperplane $H_{\mathbf{v},a}$ is

$$d_i = \frac{|\mathbf{v}'\mathbf{x}_i - a|}{\|\mathbf{v}\|} = \frac{y_i(\mathbf{v}'\mathbf{x}_i - a)}{\|\mathbf{v}\|},$$

and we refer to this distance as the *geometric margin* of $\mathbf{x}_i$.

We need to ensure that the geometric margins have a guaranteed minimum $\mu$, that is,

$$y_i(\mathbf{v}'\mathbf{x}_i - a) \geqslant \mu\|\mathbf{v}\|$$

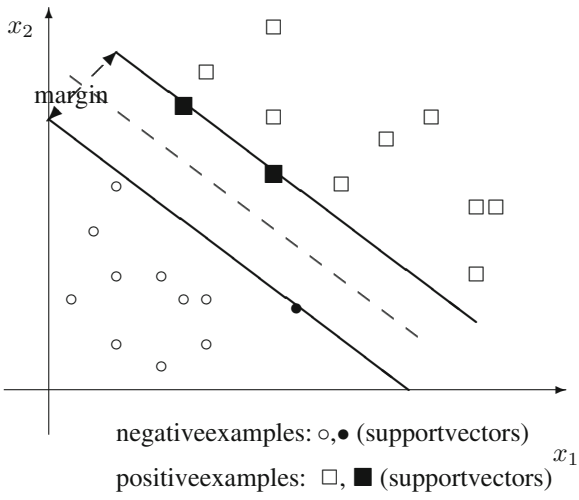


**Fig. 2** Linearly separable data set

for $1 \leqslant i \leqslant m$. This would imply that for the positive examples, we shall have

$$\mathbf{v}'\mathbf{x}_i - a - \mu\|\mathbf{v}\| \geqslant 0,$$

and for the negative examples,

$$\mathbf{v}'\mathbf{x}_i - a + \mu\|\mathbf{v}\| \leqslant 0.$$

These conditions can be written equivalently as

$$\mathbf{w}'\mathbf{x}_i - b - 1 \geqslant 0 \tag{2}$$

for the positive examples, and

$$\mathbf{w}'\mathbf{x}_i - b + 1 \leqslant 0, \tag{3}$$

for the negative examples, where $\mathbf{w} = \frac{1}{\mu}\frac{\mathbf{v}}{\|\mathbf{v}\|}$ and $b = \frac{a}{\mu\|\mathbf{w}\|}$. In a unified form, these restrictions can be now written

$$y_i(\mathbf{w}'\mathbf{x}_i - b) \geqslant 1$$

for $1 \leqslant i \leqslant m$.

The distance between the hyperplanes $\mathbf{w}'\mathbf{x}_i - a = 1$ and $\mathbf{w}'\mathbf{x} - a = -1$ is $\frac{2}{\|\mathbf{w}\|}$, and we seek to maximize this distance in order to obtain a good separation between the classes. Thus, we need to minimize $\|\mathbf{w}\|$ subjected to the restrictions $y_i(\mathbf{w}'\mathbf{x}_i - b) \geqslant 1$ for $1 \leqslant i \leqslant m$. An equivalent formulation brings this problem to a quadratic optimization problem, namely seeking $\mathbf{w}$ that is a solution of the problem:

$$\begin{aligned}\text{minimize } \frac{1}{2}\|\mathbf{w}\|^2, \text{where } \mathbf{w} \in,\\ \text{subject to } 1 - y_i(\mathbf{w}'\mathbf{x}_i - b) \leqslant 0 \quad \text{for } 1 \leqslant i \leqslant m\end{aligned}$$

The separating hyperplane is $H_{\mathbf{v},a}$.

To obtain the dual of this problem (see Sect. 8C), we start from the Lagrangean

$$\begin{aligned}L(\mathbf{w}, a, \mathbf{u}) &= \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{m} u_i(1 - y_i(\mathbf{w}'\mathbf{x}_i))\\ &= \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{m} u_i - \sum_{i=1}^{m} u_i y_i\left(\sum_{k=1}^{n} \mathbf{w}_k \mathbf{x}_{ki} - b\right),\end{aligned}$$

where $u_i \geqslant 0$ are the Lagrange multipliers. The dual objective function is obtained by as $g(\mathbf{u}) = \inf_{\mathbf{w},a} L(\mathbf{w}, \mathbf{u})$. This requires the stationarity conditions

$$\frac{\partial L}{\partial w_i} = 0 \quad \text{for } 1 \leqslant i \leqslant n \quad \text{and} \quad \frac{\partial L}{\partial a} = 0,$$

which amount to

$$\frac{\partial L}{\partial w_j} = w_j - \sum_{i=1}^{m} y_i u_i x_{ji} = 0 \quad \text{for } 1 \leqslant j \leqslant n,$$
$$\frac{\partial L}{\partial a} = \sum_{i=1}^{n} u_i y_i = 0,$$

In a vectorial form, the first $n$ stationary conditions can be written as $\mathbf{w} - \sum_{i=1}^{n} y_i u_i \mathbf{x}_i = \mathbf{0}_n$. Since

$$\mathbf{w} = \sum_{i=1}^{n} y_i u_i \mathbf{x}_i \tag{4}$$

and $\sum_{i=1}^{n} u_i y_i = 0$, the dual objective function is

$$g(\mathbf{u}) = \sum_{i=1}^{m} u_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i=1}^{n} y_i y_i u_i u_i \mathbf{x}'_i \mathbf{x}_j,$$

which is a quadratic function subject to $\mathbf{u} \geqslant \mathbf{0}_m$.

There are several important aspects of the dual formulation of the SVM:

(i) Equality (4) shows that the weight vector $\mathbf{w}$ is located in the hyperplane determined by the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Moreover, one can show that $u_i \neq 0$ if and only if $\mathbf{x}_i$ is a support vector and, therefore, $\mathbf{w}$ is determined by the support vectors.
(ii) The advantage of the dual formulation is that the number $m$ of variables $u_i$ may be a lot smaller that the original number of variables $n$.
(iii) The dual optimization problem needs no access to the original data $\{\mathbf{x}_i | 1 \leqslant i \leqslant n\}$. Instead, only the inner products $\mathbf{x}'_i \mathbf{x}_j$ are necessary in the construction of the dual objective function.

A data point $\mathbf{x} \in \mathbb{R}^n$ is classified by the SVM determined here based on the sign of the expression $\mathbf{w}'\mathbf{x} - a$; in other words, the class $y$ of an yet unseen point is given by $y = \text{sign}(\mathbf{w}'\mathbf{x} - a)$.

If the data are "almost" linearly separable, a separation hyperplane exists such that the majority (but not all) of the positive examples inhabit the positive half-space of the hyperplane and the majority (but not all) of the negative examples inhabit the negative half-space. In this case, we shall seek a "separating hyperplane" that separates the two classes with the smallest error. This is achieved by assigning to each object $\mathbf{x}_i$ in the data set a *slack variable* $\xi_i$, where $\xi_i \geqslant 0$, by relaxing Inequalities 2 and 3 as

$$\mathbf{w}'\mathbf{x}_i - b - 1 \geqslant -\xi_i \tag{5}$$

for the positive examples and

$$\mathbf{w}'\mathbf{x}_i - b + 1 \leqslant \xi_i, \tag{6}$$

for the negative examples, respectively, where $\mathbf{w} = \frac{1}{\mu}\frac{\mathbf{v}}{\|\mathbf{v}\|}$ and $b = \frac{a}{\mu\|\mathbf{w}\|}$. In turn, in a unified form these restrictions can be written as

$$1 - y_i(\mathbf{w}'\mathbf{x}_i - b) \leqslant \xi_i$$

for $1 \leqslant i \leqslant m$. The *soft-margin* SVM primal problem is

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i, \text{ where } \mathbf{w} \in, \\ &\qquad \text{subject to } 1 - y_i(\mathbf{w}'\mathbf{x}_i - a) \leqslant \xi_i \quad \text{for } 1 \leqslant i \leqslant m, \end{aligned}$$

where $C$ and $\xi_i$ are user-defined parameters referred usually as *hyper-parameters*.

The *dual of the soft-margin problem* is similar to the previous dual and it amounts to

$$\begin{aligned} &\text{maximize } g(\mathbf{u}) = \sum_{i=1}^{m} u_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{i=1}^{n} y_i y_j u_i u_j \mathbf{x}'_i \mathbf{x}_j, \\ &\qquad \text{subject to } 0 \leqslant u_i \leqslant C \text{ and } \sum_{i=1}^{m} u_i y_i = 0. \end{aligned}$$

*Example 4.1* The `kernlab` library is described in (Karatzoglou et al. 2004) which provides users with essential access to support vector machine techniques. After installing the package, its loading is achieved using

```
> library(kernlab)
```

We split the data set `iris` into a training set `trainIris` and a test set `testIris` in the same manner used in Example 2.5.

The classifier is created by writing

```
> svm <- ksvm(Species ~ .,data=trainIris,kernel="vanilladot",
          C = 1,prob.model=TRUE)
```

and is used to generate distribution probabilities for each of the 12 entries of the test set by writing

```
> pred_p <- predict(svm,testIris,type = "probabilities")
```

Note the use of the parameter `kernel` = "`vanilladot`". We will explain later the use of kernels.

These distributions can be examined:

```
> pred_p
           setosa   versicolor   virginica
 [1,] 0.948669677 0.0365527398 0.014777583
 [2,] 0.971508823 0.0190805740 0.009410603
 [3,] 0.987012019 0.0080849105 0.004903071
 [4,] 0.950002416 0.0357236471 0.014273937
 [5,] 0.659161885 0.2879288429 0.052909272
 [6,] 0.017947111 0.9594198514 0.022633038
 [7,] 0.012561988 0.9829687166 0.004469296
 [8,] 0.017910234 0.9784817276 0.003608038
 [9,] 0.008436607 0.9467301478 0.044833245
[10,] 0.012126227 0.9815816669 0.006292106
[11,] 0.028265376 0.9660266137 0.005708011
[12,] 0.052250902 0.9359484109 0.011800687
[13,] 0.001837466 0.0003496850 0.997812849
[14,] 0.006546816 0.0065769958 0.986876188
[15,] 0.005543471 0.0006948435 0.993761686
[16,] 0.001242060 0.0002903663 0.998467574
[17,] 0.012187320 0.0324955786 0.955317101
[18,] 0.019265185 0.3263600533 0.654374762
[19,] 0.005646642 0.0255939953 0.968759363
```

Note that in each case, one of the numbers strongly dominates the others, a consequence of the linear separability of this data set. Alternatively, a prediction that returns directly the class of various objects can be generated by

```
pred <- predict(svm,testIris,type="response")
```

and generates

```
> pred
 [1] setosa     setosa     setosa     setosa     setosa
     versicolor versicolor versicolor
 [9] versicolor versicolor versicolor versicolor virginica
     virginica  virginica  virginica
[17] virginica  virginica  virginica

Levels: setosa versicolor virginica.
```

A contingency table can be obtained with

```
table(pred,testIris$Species)

pred         setosa versicolor virginica
  setosa          5          0         0
  versicolor      0          7         0
  virginica       0          0         7
```

In many situations, data are not linearly separable; that is, there is no separating hyperplane between classes. Consider, for example, the set of points is shown in Fig. 3, which are separated into positive and negative examples by a nonlinear surface rather than a hyperplane (in our two-dimensional case, by a curve rather than a line). The solution is to transform data into another space, where the separating surface is transformed into a hyperplane such that the positive and negative examples will inhabit the two half-spaces determined by the hyperplane. The data transformation is defined by a function $\phi : \mathsf{R}^n \to H$, where $H$ is a new linear space that is referred as the *feature space*. The intention is to use a linear classifier in the new space to achieve separation between the representation of the positive and the negative examples in this new space.

We assume that the feature space $H$ is equipped with an inner product $(\cdot, \cdot) : H \to \mathbb{R}_{\geqslant 0}$. In view of Equality (4), if the data are approximately linearly separable in the new space, the classification decision is based on computing

$$\sum_{i=1}^{n} y_i u_i \phi(\mathbf{x}_i)' \phi(\mathbf{x}) - a$$

Let $K : H^2 \to \mathbb{R}$ be the function defined by $K(\mathbf{u}, \mathbf{v}) = (\Phi(\mathbf{u}), \Phi(\mathbf{v}))$; this function is referred to as a *kernel function*, and the decision in the new space is based on the sign of the expression $\sum_{i=1}^{n} (y_i u_i K \ \mathbf{x}_i, \mathbf{x}) - a$. Thus, we need to specify only the kernel function rather than the explicit transformation $\phi$. In Example 4.1, $\phi$ is the identical transformation and the corresponding kernel, $K(\mathbf{u}, \mathbf{v}) = \mathbf{u}'\mathbf{v}$, is known as the *vannila kernel*. Among the most frequently used kernels, we mention the *Gaussian kernel* defined by $K(\mathbf{u}, \mathbf{v}) = e^{-\gamma \|\mathbf{u}-\mathbf{v}\|^2}$, the *exponential kernel* given by $K(\mathbf{u}, \mathbf{v}) = e^{-\gamma \|\mathbf{u}-\mathbf{v}\|}$, and the *polynomial kernel* $K(\mathbf{u}, \mathbf{v}) = (k + \mathbf{u}'\mathbf{v})^p$.

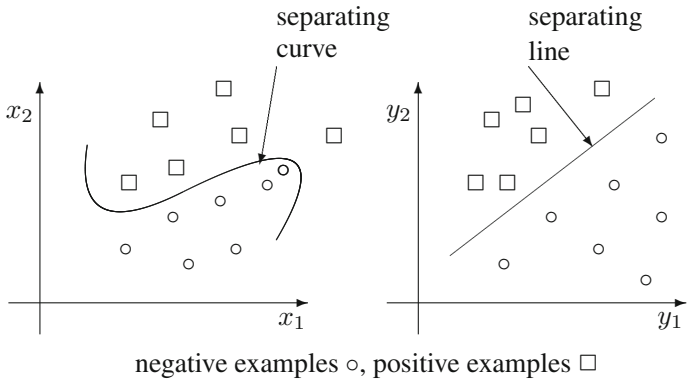


**Fig. 3** Inseparable data set

*Example 4.2* The two-dimensional data set shown in Fig. 4 is clearly not linearly separable because no line can be drawn such that all positive points will be on one side of the line and all negative points on the other.

Again, we use the `kernlab` and the function `ksvm` of this package. We apply a Gaussian kernel, which can be called using the `rbfdot` value:

```
> svmrbf <- ksvm(class ~ x + y, data=points,
+ kernel="rbfdot", C = 1)
```

If the data frame `testdata` contains the vectors $\begin{pmatrix}6\\6\end{pmatrix}$, $\begin{pmatrix}7\\8\end{pmatrix}$, and $\begin{pmatrix}8\\11\end{pmatrix}$, then the predictions of the classifier `svmrbf` obtained with

```
> pred_points <- predict(svmrbf,testdata,type="response")
```

returns

```
> pred_points
             [,1]
[1,] -0.03084342
[2,] -1.03816317
[3,]  1.21256792
```

Note that the first two test data that are close to negative training examples get negative predictions; the remaining test data that are close to positive examples get a positive prediction.

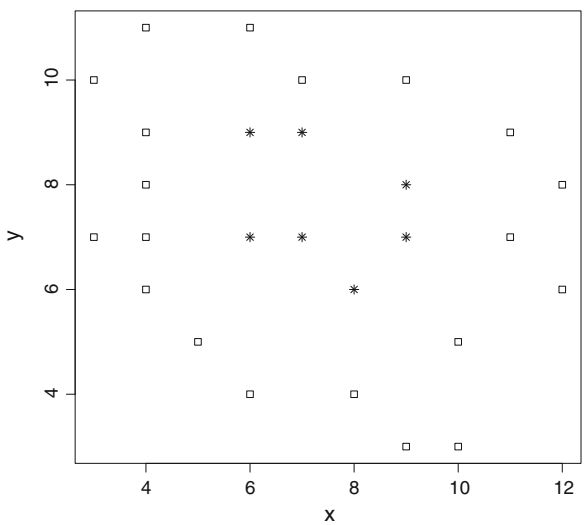


**Fig. 4** Data set that is not linearly separable; positive examples are shown as *white square* and negative examples as *asterisk*

## 5 Regression

Regression seeks functions that model data with minimal errors. It aims to describe the relationships between dependent variables and independent variables and to estimate values of dependent variables starting from values of independent variables.

There are several types of regression: linear regression, logistic regression, nonlinear regression, Cox regression, etc. We present here an introduction to linear regression.

Linear regression considers models that assume that a variable $Y$ is estimated to be a linear function of the independent variables $X_1, \ldots, X_n$:

$$Y = a_0 + a_1 X_1 + \cdots + a_n X_n.$$

$Y$ must be continuous, while $X_1, \ldots, X_n$ may be continuous or discrete (categorial).

*Example 5.1* Consider a data set that records the height and weight of several individuals. We seek a linear dependency of the height on the weight (the regressor), specified by the model formula `weight ~ height`. Data can be placed in R using

```
height <- c(1.6,1.62,1.65,1.72,1.74,1.74,1.76,1.77,1.79,1.8,1.8,1.81,
            1.83,1.84,1.86,1.87,1.9,1.91,1.91,1.92);
weight <- c(55,53,54,57,64,69,73,65,80,72,77,81,73,80,84,86,84,88,
            91,89);
```

and can be displayed using the usual `plot` function, as shown in Fig. 5. To produce the regression line, we call the linear modeling function `lm`:

```
lm.r <- lm(formula = weight ~ height)
```

The coefficients of the regression line are

```
Coefficients:
(Intercept)       height
     -148.0        123.8
```

and `abline(lm.r)` places the regression line on the plot.

In a multivariable regression, we seek a similar linear dependency that involves several regressors.

*Example 5.2* Suppose that we collect a data set that shows the dependency of systolic blood pressure numbers on the body mass index (BMI), sex, and age by writing

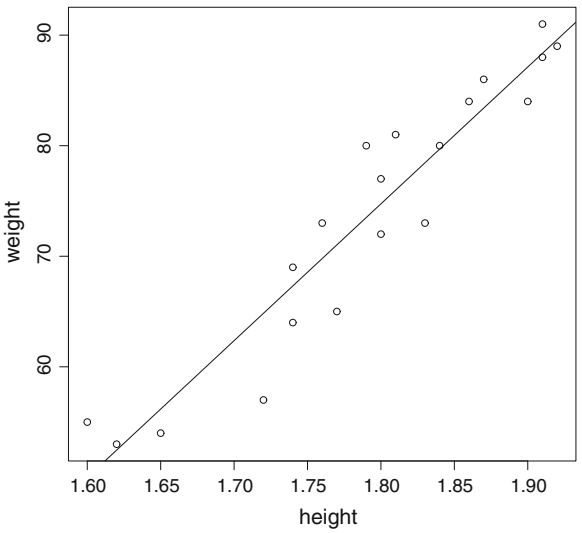


**Fig. 5** Data and regression line

```
BMI <- c(21.48,19.83,19.26,21.13,22.79,23.56,20.74,24.96,22.22,23.76,
         24.72,21.79,23.62,24.28,24.59,23.26,24.12,24.94,24.14);
sex <- c(0,0,1,0,1,1,1,1,1,1,0,1,0,1,0,1,1,1,1);
age <- c(21,31,20,32,41,25,40,38,50,45,41,65,37,60,51,65,40,55,40);
sys <- c(125,120,110,130,141,155,110,120,130,140,130,120,135,167,
         130,150,140,145,120);
```

The linear model is obtained with

```
lm.r = lm(sys ~ BMI + sex + age)
```

This results in the linear function defined by the following coefficients:

```
(Intercept)         BMI         sex         age
    31.5458      4.0081      2.4310      0.1791
```

In other words, the linear model is

$$\texttt{sys} = 31.5458 + 4.0081 * \texttt{BMI} + 2.4310 * \texttt{sex} + 0.1791 * \texttt{age}.$$

The use of support vector machines for regression was proposed in (Drucker et al. 1996). The model produced by support vector classification depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Another SVM version known as least squares support vector machine has been proposed in (Suykens and Vandewalle 1999).

## 6 Active Learning

Learning, as has been discussed up to this point, involves *passive learners*, that is, learning algorithms where the information flows from data to learner.

A machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns, that is, to apply *active learning*. An active learner may pose queries, usually in the form of unlabeled data instances to be labeled by a human operator. The flow of information between data and the learner is bidirectional as shown in Fig. 6.

Since unlabeled data are abundant and, in many cases, easily obtainable, there are good reasons to use this learning paradigm.

The training processes that allow us to construct data mining models often require a large volume of labeled data. For example, to produce a topic-based text classifier through text mining, a large number of documents must be labeled with the pertinent topics. This is an expensive process that requires numerous human readers capable of understanding these topics and attaches appropriate labels to the documents. Similarly, speech recognition requires labeling of a large number of speech fragments by specialized linguists, which is time consuming and prone to errors.

Active learning requires a querying strategy (see Settles 2012). One such strategy is *query by uncertainty* (also known as *uncertainty sampling*), in which a single classifier is learned from labeled data and is subsequently utilized for examining the unlabeled data. Those instances in the unlabeled data set that the classifier is least certain about are subject to classification by a human annotator. Query by uncertainty has been realized using a range of learners, such as logistic regression (Lewis and Gale 1994), support vector machines (Schohn and Cohn 2000), and Markov models (Scheffer et al. 2001). The amount of data that require annotation in order to reach a given performance, compared to passively learning from examples provided in a random order, is significantly reduced using query by uncertainty.

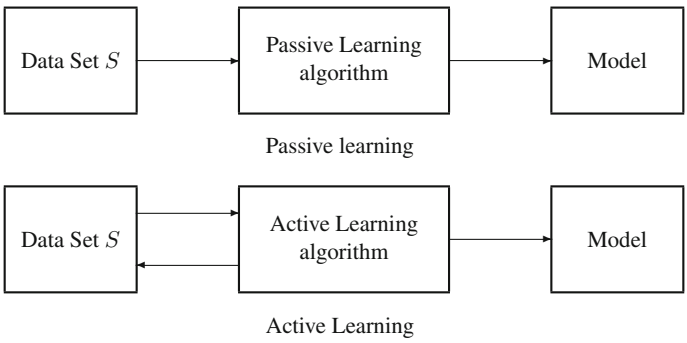


**Fig. 6** Information flow in passive versus active learning

There are several modalities to implement query by uncertainty, and they require to determine the data item $x_{\text{lc}}$ for which the learner is the least confident about its labeling.

The most common approach for selecting $x_{\text{lc}}$ is the use of entropy as a measure of uncertainty. If $Y$ is a random variable that ranges over all possible labels, then we shall seek $x_{\text{lc}}$ as $x_{\text{lc}} = \mathsf{argmax}_x H(Y|x)$.

Another approach requires the learner $C$ to evaluate the degree of confidence in its predictions. Let $x$ be a data item and let $\hat{y}$ be the label with the highest posterior probability according to $C$, that is, $\hat{y} = \mathsf{argmax}_y P_C(\mathrm{y}|x)$. Then, $1 - P(\hat{y}|x)$ is the lack of confidence of $C$ in the label $\hat{y}$ and $x_{\text{lc}} = \mathsf{argmax}_x(1 - P(\hat{y}|x))$ is a data item for which $C$ is the least confident. The intervention of the human annotator will be required for $x_{\text{lc}}$.

Yet another strategy makes use of the *output margin* of a data item $x$ defined as the difference $P(\hat{y}_1|x) - P(\hat{y}_2|x))$ between the probability of the most likely label $\hat{y}_1$ and the second most likely label $\hat{y}_2$ of an item $x$. For items with large margins, there is little uncertainty on the choice of the most likely label; therefore, items with small margin benefit most from an external annotation, and so, an external annotation will be required for $x_m$ defined by

$$x_m = \mathsf{argmin}_x(P(\hat{y}_1|x) - P(\hat{y}_2|x)).$$

Active learning may run into difficulties because, as shown in (Schütze et al. 2006; Velipasaoglu et al. 2007), a mix of learnable and unlearnable classes co-occur in a data set. A class can be regarded as *learnable* if there exists a learning procedure that generates a classifier with a performance (e.g., the $F_1$ measure) that exceeds a certain threshold with a certain level of confidence.

For small classes, it is difficult or impossible to create reliable classifiers. For example, if a class contains 1 % of 1000 records, we have just ten examples for that class and this is often not sufficient for creating a classifier.

In Dasgupta (2011), the following simple but paradigmatic example is used to describe the effect of active learning. Suppose that we have a data set $S = \{(x_i, y_i)|1 \geqslant i \geqslant n\}$, where $x_i \in \mathbb{R}$ and $y_i \in \{-1, 1\}$, and we use a collection $H$ of simple thresholding classifiers of the form $h_t : \mathbb{R} \to \{-1, 1\}$, where

$$\begin{cases} -1 & \text{if } x < t, \\ 1 & \text{if } x \geqslant t, \end{cases}$$

where $t$ is the threshold that defines the classifier $h_t$. The empirical error of the classifier $h_t$ is

$$\text{err}(h_t) = \frac{|\{x_i|h_t(x_i) \neq y_i\}|}{n}.$$

The data are separable if a value $t_0$ exists such that $err(h_{t_0}) = 0$. Note that if $n = 2$, the data are separable.

To determine effectively a threshold classifier that achieves an approximative separation of the data (in case that data are not separable), with an error less than $\epsilon$, we need approximately $\frac{1}{\epsilon}$ randomly drawn examples [see, for example Blumer et al. (1989)].

If $n_\epsilon = \left\lceil \frac{1}{\epsilon} \right\rceil$ unlabeled examples are drawn at random, finding a classifier involves asking for $\log_2 n_\epsilon$ labels by a binary process that begins by asking for the label of the median point, then for the label of the 25 percentile point (or to the label of the 75 percentile point), as so on. This opens the possibility that active learning reduces exponentially the number of labels needed to construct a classifier (Dasgupta 2011).

## 7 Perceptrons and Neural Networks

Artificial neural networks (NN) aim to emulate cognitive processes that take place in the human brain. Research in this direction started in the 1940s with the work of McCulloch and Pitts (1943), Pitts and McCulloch (1947) who developed a computational model of the brain.

The human brain is a highly organized collection of a large number of interconnected and specialized cells called *neurons*. Neurons are engaged in certain computing activities that are carried out using chemical and electrical signals; connections between neurons are referred to as *synapses* and the brain, as a large collection of simple computers has a high degree of parallelism.

The current model of NN consists in a series of layers $L_1, \ldots, L_\ell$ of computing units. Units on the first layer $L_1$ are referred to as *input units*; those on the last layer $L_\ell$ are the *output units,* and the units in each layer beyond the first layer are neurons. Connections exist only between neurons that belong to consecutive layers.

A simple example of a NN is a perceptron that consists of $n$ input units and one neuron. Perceptrons can be trained to perform classification on sets of objects of the form $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, 1\}$, and they achieve this by constructing a separating hyperplane between the set of positive examples and the set of negative examples whenever these sets are linearly separable. In this respect, perceptrons are similar to support vector machines. However, the model building is done in an iterative, specific way proposed by Rosenblatt (1958). Several variants of this algorithm exist (Freund and Shapire 1999; Novikoff 1962).

A perceptron intended to analyze vectors $\mathbf{x} \in \mathbb{R}^n$ is defined by $n + 1$ numbers: the weights $w_1, \ldots, w_n$ of the input units and a bias $b$ as shown in Fig. 7.

In the simplest case (discussed next), the neuron itself is characterized by a transfer function that computes the answer $y = \text{sign}(\text{net}(\mathbf{x}))$, where $\text{net}(\mathbf{x}) = \mathbf{w}'\mathbf{x} + b$.

The hyperplane defined by this perceptron is $\mathbf{w}'\mathbf{x} + b = 0$.

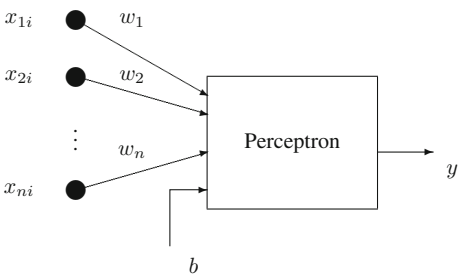


**Fig. 7** Perceptron acting on $n$-dimensional inputs

*Example 7.1* Let

$$\mathbf{x}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \mathbf{x}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \mathbf{x}_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \mathbf{x}_4 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

The sequence

$$S_1 = (\mathbf{x}_1, 1), (\mathbf{x}_2, 1), (\mathbf{x}_3, -1), (\mathbf{x}_4, 1))$$

is linearly separable, as shown in Fig. 8a. On the other hand, the sequence

$$S_1 = (\mathbf{x}_1, 1), (\mathbf{x}_2, -1), (\mathbf{x}_3, 1), (\mathbf{x}_4, -1))$$

shown in Fig. 8b is not linearly separable.

Let $R$ be the minimum radius of a closed ball centered in $\mathbf{0}$, that is, $R = \max\{\| \mathbf{x}_i \| | 1 \leqslant i \leqslant m\}$.

If $(\mathbf{x}_i, y_i)$ is a member of the sequence $S$ and $H$ is the target hyperplane $\mathbf{w}'\mathbf{x} + b = 0$, where $\|\mathbf{w}\| = 1$, define the *functional margin* of $(\mathbf{x}_i, y_i)$ as $\gamma_i = y_i(\mathbf{w}'\mathbf{x}_i + b)$. As before, if $y_i$ and $\mathbf{w}'\mathbf{x}_i + b$ have the same sign, then $(\mathbf{x}_i, y_i)$ is classified correctly; otherwise, it is incorrectly classified and we say that a mistake occurred.

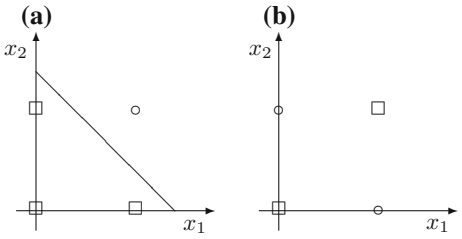


**Fig. 8** A linearly separable sequence and a sequence that is not linearly separable; positive examples are designated by *square*, while *circle* symbols correspond to negative examples

A perceptron is constructed starting from the sequence $S$ and from a parameter $\eta \in (0, 1)$ known as a *learning rate*.

**Input:** labelled training sequence $S$ and learning rate $\eta$
**Output:** weight vector $\mathbf{w}$ and parameter $b$ defining classifier
initialize $\mathbf{w} = \mathbf{0}$, $b_0 = 0$, $k = 0$
define $R = \max\{\| \mathbf{x}_i \| \mid 1 \leqslant i \leqslant m\}$
**repeat until** (no mistakes are made in the **for** loop)
**for** $i = 1$ **to** $m$ **do**
**if** $(y_i(\mathbf{w}_k'\mathbf{x}_i + b_k) \leqslant 0)$
$\mathbf{w}_{k+1} = \mathbf{w}_k + \eta y_i \mathbf{x}_i$;
$b_{k+1} = b_k + \eta y_i R^2$;
$k = k + 1$;
**end if**
**end for**
**end repeat**
return $k, (\mathbf{w}_k, b_k)$ where $k$ is the number of mistakes;

Suppose there exists an optimal weight vector $\mathbf{w}_{\text{opt}}$ and an optimal bias $b_{\text{opt}}$ such that

$$\left\|\mathbf{w}_{\text{opt}}\right\| = 1 \text{ and } y_i(\mathbf{w}'_{\text{opt}}\mathbf{x}_i + b_{\text{opt}}) \geqslant \gamma,$$

for $1 \leqslant i \leqslant m$. Then, we claim that the number of mistakes made by the algorithm is at most

$$\left(\frac{2R}{\gamma}\right)^2$$

Indeed, let $t$ be the update counter,

$$\hat{\mathbf{w}} = \begin{pmatrix} \mathbf{w} \\ \frac{b}{R} \end{pmatrix} \text{ and } \hat{\mathbf{x}}_i = \begin{pmatrix} \mathbf{x}_i \\ R \end{pmatrix}$$

for $1 \leqslant i \leqslant m$.

The algorithm begins with an augmented vector $\hat{\mathbf{w}}_0 = \mathbf{0}$ and updates it at each mistake.

Let $\hat{\mathbf{w}}_{t-1}$ be the augmented weight vector prior to the $t$th mistake. The $t$th update is performed when

$$y_i\hat{\mathbf{w}}'_{t-1}\hat{\mathbf{x}}_i = y_i(\hat{\mathbf{w}}'_{t-1}\mathbf{x}_i + b_{t-1}) \leqslant 0,$$

where $(\mathbf{x}_i, y_i)$ is the example incorrectly classified by

$$\hat{\mathbf{w}}_{t-1} = \begin{pmatrix} w_{t-1} \\ \frac{b_{t-1}}{R} \end{pmatrix}.$$

The update is

$$\begin{aligned} \hat{\mathbf{w}}_t &= \begin{pmatrix} \mathbf{w}_t \\ \frac{b_t}{R} \end{pmatrix} = \begin{pmatrix} \mathbf{w}_{t-1} + \eta y_i \mathbf{x}_i \\ \frac{b_{t-1} + \eta y_i R^2}{R} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{w}_{t-1} + \eta \mathbf{y}_i \mathbf{x}_i \\ \frac{b_{t-1}}{R} + \eta y_i R \end{pmatrix} = \begin{pmatrix} \mathbf{w}_{t-1} \\ \frac{b_{t-1}}{R} \end{pmatrix} + \begin{pmatrix} \eta y_i \mathbf{x}_i \\ \eta y_i R \end{pmatrix} \\ &= \hat{\mathbf{w}}_{t-1} + \eta y_i \hat{\mathbf{x}}_i, \end{aligned}$$

where we used the fact that $b_t = b_{t-1} + \eta y_i R^2$.

Since

$$y_i \hat{\mathbf{w}}'_{\text{opt}} \hat{\mathbf{x}}_i = y_i \left( \hat{\mathbf{w}}'_{\text{opt}} \frac{b}{R} \right) \begin{pmatrix} \mathbf{x}_i \\ R \end{pmatrix} = y_i \left( \hat{\mathbf{w}}'_{\text{opt}} \mathbf{x}_i + b \right) \geqslant \gamma,$$

we have

$$\hat{\mathbf{w}}'_{\text{opt}} \hat{\mathbf{w}}_t = \hat{\mathbf{w}}'_{\text{opt}} \hat{\mathbf{w}}'_{t-1} + \eta y_i \hat{\mathbf{w}}'_{\text{opt}} \hat{\mathbf{x}}_i \geqslant \hat{\mathbf{w}}'_{\text{opt}} \hat{\mathbf{w}}_{t-1} + \eta\gamma.$$

By repeated application of the inequality $\hat{\mathbf{w}}'_{\text{opt}} \hat{\mathbf{w}}_t \geqslant \eta\gamma$, we obtain

$$\hat{\mathbf{w}}'_{\text{opt}} \hat{\mathbf{w}}_t \geqslant t\eta\gamma.$$

Since $\hat{\mathbf{w}}_t = \hat{\mathbf{w}}_{t-1} + \eta y_i \hat{\mathbf{x}}_i$, we have

$$\begin{aligned} \|\hat{\mathbf{w}}_t\|^2 = \hat{\mathbf{w}}'_t \hat{\mathbf{w}}_t &= (\hat{\mathbf{w}}'_{t-1} + |\eta y_i \hat{\mathbf{x}}_{i'})(\hat{\mathbf{w}}_{t-1} + \eta y_i \hat{\mathbf{x}}_i) \\ &= \|\hat{\mathbf{w}}_{t-1}\|^2 + 2\eta y_i \hat{\mathbf{w}}'_{t-1} \hat{\mathbf{x}}_i + \eta^2 \|\hat{\mathbf{x}}_i\|^2 \\ &\quad \text{(because } y_i \hat{\mathbf{w}}'_{t-1} \hat{\mathbf{x}}_i \leqslant 0 \text{ when an update occurs)} \\ &\leqslant \|\hat{\mathbf{w}}_{t-1}\|^2 + \eta^2 \|\hat{\mathbf{x}}_i\|^2 \\ &\leqslant \|\hat{\mathbf{w}}_{t-1}\|^2 + \eta^2 (\|\hat{\mathbf{x}}_i\|^2 + R^2) \\ &\leqslant \|\hat{\mathbf{w}}_{t-1}\|^2 + 2\eta^2 R^2, \end{aligned}$$

which implies $\|\hat{\mathbf{w}}_t\|^2 \leqslant 2t\eta^2 R^2$. By combining the inequalities

$$\hat{\mathbf{w}}'_{\text{opt}} \mathbf{w}_t \geqslant t\eta\gamma \text{ and } \|\hat{\mathbf{w}}_i\|^2 \leqslant 2t\eta^2 R^2$$

we have

$$\left\|\hat{\mathbf{w}}_{\text{opt}}\right\|\sqrt{2t}\eta R \geqslant \left\|\hat{\mathbf{w}}_{\text{opt}}\right\|\|\hat{\mathbf{w}}_t\| \geqslant \hat{\mathbf{w}}'_{\text{opt}}\hat{\mathbf{w}}_t \geqslant t\eta\gamma,$$

which imply

$$t \leqslant 2\left(\frac{R}{\gamma}\right)^2 \left\|\hat{\mathbf{w}}_{\text{opt}}\right\|^2 \leqslant \left(\frac{2R}{\gamma}\right)^2$$

because $b_{\text{opt}} \leqslant R$ for a non-trivial separation of data and hence

$$\left\|\hat{\mathbf{w}}_{\text{opt}}\right\|^2 \leqslant \left\|\hat{\mathbf{w}}_{\text{opt}}\right\|^2 + 1 = 2.$$

In the case of the perceptron considered above, the transfer function is the signum function

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geqslant 0, \\ -1 & \text{if } x > 0 \end{cases}$$

for $x \in \mathbb{R}$. We mention a few other choices that exist for the transfer function:

- the sigmoid or the logistic function $h(x) = \frac{1}{1+e^{-x}}$,
- the hyperbolic tangent $h(x) = \tanh(x)$,
- the Gaussian function $h(x) = ae^{-\frac{x^2}{2}}$.

for $x \in \mathbb{R}$. The advantage of these last three choices is their differentiability that enables us to apply optimization techniques to more complex NNs. Note, in particular, that if $h$ is a sigmoid transfer function, then

$$h'(x) = \frac{1}{(1-e^{-x})2}e^{-x} = h(x)\,(1-h(x)), \tag{7}$$

which turns out to be a very useful property. To emphasize the choices that we have for the transfer function, it is useful to think that a neuron has the structure shown in Fig. 9.

A multilayer NN is a much more capable classifier compared to the perceptron. It has, however, a degree of complexity because of topology of the neuron network which entails multiple connection weights, the multiple outputs, and the more complex neurons.

The specification of the architecture of a NN encompasses the following elements (see Fig. 10):

(i) the choice of $\ell$ is the number of levels; the first level $L_1$ contains the input units, the last level $L_\ell$ contains the output units, and the intermediate levels $L_2, \ldots, L_{\ell-1}$ contain the *hidden units*;

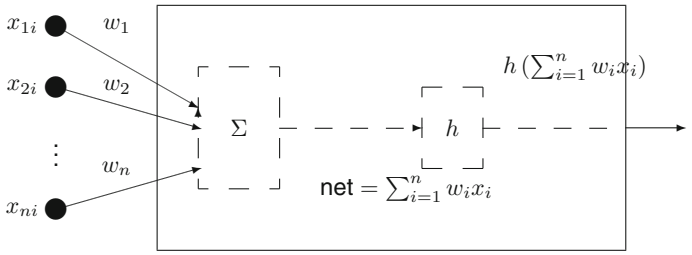


**Fig. 9** Summation and activation components of a neuron

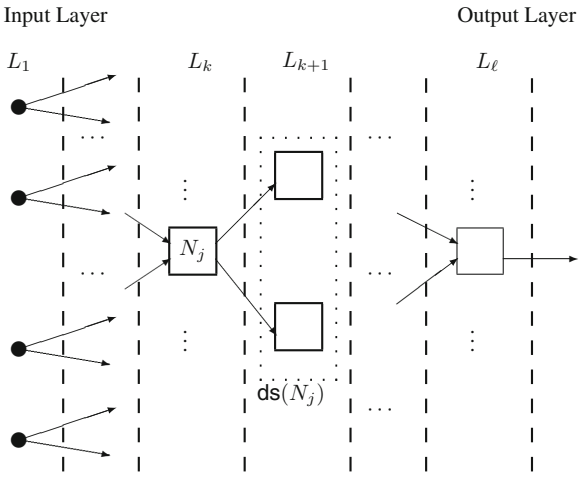


**Fig. 10** Structure of a neural network

(ii) the connection from unit $N_i$ on level $L_k$ to unit $N_j$ on level $k+1$ has the weight $w_{ji}$; the set of units on level $L_{k+2}$ that are connected to unit $N_j$ is the *downstream set* of $N_j$ denoted by $(N_j)$;
(iii) the type of neurons used in the network as defined by their transfer functions.

Let $X$ be the set of examples that are used in training the network. For $\mathbf{x} \in X$, we have a vector of *target outputs* $\mathbf{t}(\mathbf{x})$ and a vector of *actual outputs* $\mathbf{o}(\mathbf{x})$, both in $\mathbf{x}^p$, where $p$ is the number of output units. The outputs that correspond to a unit $N_j$ are denoted by $\mathbf{o}_{\mathbf{x},j}$. For a weight vector $\mathbf{w}$ of the network, the *total error* is

$$E(\mathbf{w}) = \frac{1}{2} \sum_{\mathbf{x} \in X} \|\mathbf{t}(\mathbf{x}) - \mathbf{o}(\mathbf{x})\|^2.$$

The information is propagated from the input to the output layer. This justifies referring to the architecture of this network as a feed-forward network.

We discuss here the *backpropagation* training algorithm for a feed-forward NN. The training process consists in readjusting the weights of the connection taking into account the error rate.

Note that $E(\mathbf{w})$ depends on a large number of parameters of the form $w_{ji}$, which poses a significant challenge as an optimization problem. Finding a local minimum of $E(\mathbf{w})$ can be achieved by applying a gradient descent algorithm. It is known that the fastest decrease of a function is in the opposite direction of its gradient, and the components of the gradient of $E(\mathbf{w})$ are given by $\frac{\partial E(\mathbf{w})}{\partial w_{kj}}$. The learning algorithm will modify the weights of the network $w_{ji}$ in the opposite direction of the gradient of the error $E(\mathbf{w})$. Consequently, the change in $w_{ji}$ will be given by

$$\Delta w_{ji} = -\eta \frac{\partial E(\mathbf{w})}{\partial w_{ji}}$$

where the learning rate $\eta$ is a small positive number. Initially, the weights of the edges are randomly set as numbers having small absolute values (e.g., between $-0.05$ and $0.05$) (cf. Mitchell 1997). These rates are successively modified as we show next.

To evaluate the partial derivatives of the form $\frac{\partial E(\mathbf{w})}{\partial w_{ji}}$, we need to take into account that $E(\mathbf{w})$ depends on $w_{ji}$ through $net_j$ and therefore,

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \frac{\partial E(\mathbf{w})}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ji}} = \frac{\partial E(\mathbf{w})}{\partial \text{net}_j} x_{ji}.$$

The position of the neuron $N_j$ in the network must be considered in computing $\frac{\partial E(\mathbf{w})}{\partial \text{net}_j}$, and we have two cases.

(i) If $N_j$ is an output neuron, then $E(w)$ depends on $net_j$ through the output $o_j$ of the unit $N_j$, where $o_j = h(\text{net}_j)$. Thus,

$$\frac{\partial E(\mathbf{w})}{\partial \text{net}_j} = \frac{\partial E(\mathbf{w})}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} = \frac{\partial E(\mathbf{w})}{\partial o_j} h'(\text{net}_j).$$

Since $N_j$ is an output neuron, we have $\frac{\partial E(\mathbf{w})}{\partial o_j} = -(t_j - o_j)$ for $1 \leqslant j \leqslant p$. If we assume that $N_j$ has a sigmoidal transfer function, then [by Equality (7)] we have

$$\frac{\partial E(\mathbf{w})}{\partial \text{net}_j} = -\big(t_j - o_j\big) h\big(\text{net}_j\big)\big(1 - h\big(\text{net}_j\big)\big).$$

(ii) When $N_j$ is a hidden unit $E(\mathbf{w})$ depends on $\text{net}_j$ via the functions $\text{net}_k$ for all neurons $N_k$ situated downstream from $N_j$. In turn, each $\text{net}_k$ depends on $o_j$, which depends on $\text{net}_j$. This allows us to write:

$$\frac{\partial E(\mathbf{w})}{\partial \text{net}_j} = \sum_{N_k \in \mathsf{ds}(N_j)} \frac{\partial E(\mathbf{w})}{\partial \text{net}_k} \frac{\partial \text{net}_k}{\partial \text{net}_j}$$
$$= \sum_{N_k \in \mathsf{ds}(N_j)} \frac{\partial E(\mathbf{w})}{\partial \text{net}_j} \frac{\partial \text{net}_k}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j}.$$

Observe that

$$\frac{\partial o_j}{\partial \text{net}_j} = h'(\text{net}_j) = h(\text{net}_j)(1 - h(\text{net}_j))$$

because $h$ is a sigmoid function, and $\frac{\partial \text{net}_k}{\partial o_j} = w_{kj}$ because $N_k \in \mathsf{ds}(N_j)$. This yields

$$\frac{\partial E(\mathbf{w})}{\partial \text{net}_j} = o_j(1 - o_j) \sum_{N_k \in \mathsf{ds}(N_j)} \frac{\partial E(\mathbf{w})}{\partial \text{net}_k} w_{kj}.$$

If $\delta_i = -\frac{\partial E(\mathbf{w})}{\partial \text{net}_i}$ for every neuron $N_i$, then

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \begin{cases} -\big(t_j - o_j\big)h\big(\text{net}_j\big)\big(1 - h\big(\text{net}_j\big)\big) & \text{if } N_j \text{ is an output neuron} \\ -o_j\big(1 - o_j\big)\sum_{N_k \in \mathsf{ds}(N_j)} \delta_k w_{kj} & \text{if } N_j \text{ is a hidden neuron.} \end{cases}$$

The changes in the weights can now be written as

$$\Delta w_{ji} = \begin{cases} \eta(t_j - o_j)h(\text{net}_j)(1 - h(\text{net}_j)) & \text{if } N_j \text{ is an output neuron} \\ \eta o_j(1 - o_j)\sum_{N_k \in \mathsf{ds}(N_j)} \delta_k w k_j & \text{if } N_j \text{ is a hidden neuron.} \end{cases}$$

The *backpropagation algorithm* consists of the following steps:

**for** each training example $(\mathbf{x}, \mathbf{t})$, where $\mathbf{x} \in X$ do
  input $\mathbf{x}$ in the network and obtain $\mathbf{o}_{\mathbf{x},j}$ for each unit $N_j$;
  for each output unit $N_j$ compute $\Delta w_{ji} = \eta(t_j - o_j)h(net_j)(1 - h(net_j))$
    and update $w_{ji}$;
  for each hidden unit $N_j$ compute $\Delta w_{ji} = \eta o_j(1 - o_j)\sum_{N_k \in \mathsf{ds}(N_j)} \delta_k w_{kj}$
    and update $w_{ji}$;
**end for**

Observe that the weight updates proceed from the output layer toward the inner layers, which justifies the name of the algorithm.

Next, we present an example for NN construction using the package `neuralnet` developed in (Günther and Fritsch 2010). The package computes NN with

one hidden layer with a prescribed number of neurons. The computation of the NN model is achieved by calling

```
nnmodel <- neuralnet(target ~ predictors, data = inputdata,
+ hidden = h)
```

where `target ~ predictors` is the formula that specifies the model, and `hidden` gives the number of neurons in the hidden layer.

*Example 7.2* We use the data set `Concrete_Compressive_Strength` (CCS) that is available from the data mining repository at UCI. Ingredients included in concrete include cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate. The data set records 1030 observations and has nine numerical attributes. Data are presented in a raw form (it is not scaled), and various attributes have distinct ranges (see Table 5).

The first seven attributes are expressed in $kgm^3$. Data (originally in the `xls` format) are read in R using the `csv` format as

```
> CCS <- read.csv("CCS.csv")
> head(CCS)
    cem blast ash water plast coarse  fine age strength
1 540.0   0.0   0   162   2.5 1040.0 676.0  28    79.99
2 540.0   0.0   0   162   2.5 1055.0 676.0  28    61.89
3 332.5 142.5   0   228   0.0  932.0 594.0 270    40.27
4 332.5 142.5   0   228   0.0  932.0 594.0 365    41.05
5 198.6 132.4   0   192   0.0  978.4 825.5 360    44.30
6 266.0 114.0   0   228   0.0  932.0 670.0  90    47.03
```

Since the scale of the attributes is quite distinct, the data are normalized using the function `normalize` defined in (Lantz 2013) as

```
 normalize <- function(x) {
+ return((x - min(x)) / (max(x) - min(x)))
+ }
```

**Table 5** Attributes of CCS data set

| Attribute in original set | Attribute in our data set |
|---|---|
| Cement | Cem |
| Blast furnace slag | Bla |
| Fly ash | Ash |
| Water | Water |
| Superplasticizer | Plast |
| Coarse aggregate | Coarse |
| Fine aggregate | Fine |
| Age | Age in days |
| Concrete compressive strength | Strength |

and the data set is normalized using

```
CCSN <- as.data.frame(lapply(CCS,normalize))
```

This results in normalized data; its first few records (truncated to two decimals) are:

```
> head(CCSN)
  cem  blast  ash  water  plast coarse fine   age   strength
1 1.00 0.00   0    0.32   0.07  0.69   0.20   0.07  0.96
2 1.00 0.00   0    0.32   0.07  0.73   0.20   0.07  0.74
3 0.52 0.39   0    0.84   0.00  0.38   0.00   0.73  0.47
4 0.52 0.39   0    0.84   0.00  0.38   0.00   1.00  0.48
5 0.22 0.36   0    0.56   0.00  0.51   0.58   0.98  0.52
6 0.37 0.31   0    0.84   0.00  0.38   0.19   0.24  0.55
```

A neural net model with four hidden neurons is computed by

```
nnet4 <- neuralnet(strength ~ cem + blast + ash + water +
+ plast + coarse + fine + age,
+ data = CCSN, hidden = 4)
```

The resulting neural net can be seen using `plot(nnet4)` and is shown in Fig. 11.

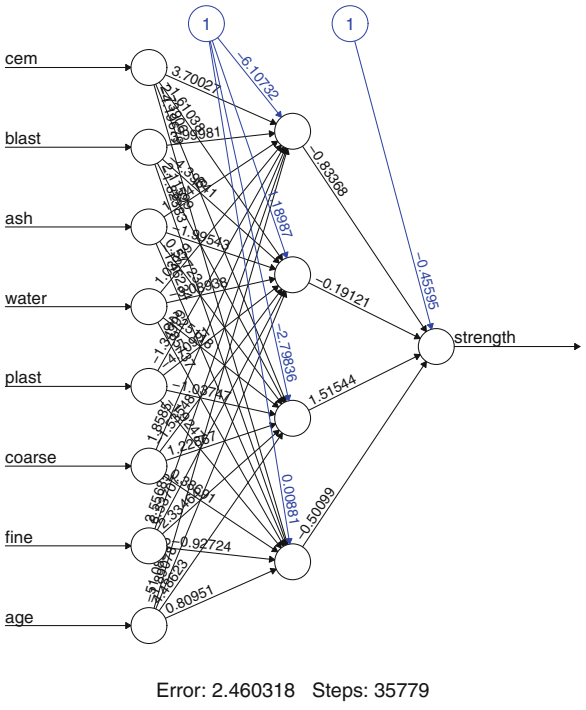


**Fig. 11** Neural nets with four hidden neurons

Once a neural net is created, the `compute` function of the `neuralnet` can be used to calculate and summarize the output of each neuron; it can be used to predict outputs formed by new combinations of values of attributes.

*Example 7.3* Consider some new combinations of values for the eight predictive attributes of `CCSN` defined by

```
newconc <- matrix(c(1.00, 0.2, 0.1, 0.1, 0.1, 0.8, 0.8, 0.9,
                    0.9, 0.5, 0.1, 0.4, 0.1, 0.5, 0.5, 0.2),
                    byrow = TRUE, ncol = 8)
```

Using

```
new.output <- compute(nnet4,newconc)
new.output$net.result
```

yields the following predictions for strength

```
                    [,1]
[1,] 0.8891098520
[2,] 0.9530817536
```

## 8 Bibliographic Guide

Data mining and machine learning have generated a vast collection of references. Among more advanced texts, we recommend (Abu-Mostafa et al. 2012; Bishop 2007; Murphy 2012; Shalev-Shwartz and Ben-David 2014; Zaki and Meira 2014; Mohri et al. 2012).

A large number of books exist that deal with the R system and its applications to machine learning and data mining. We mention (Lander 2014; Maindonald and Braun 2004; Matloff 2011; Wickham 2009) as general references on R; books specialized in machine learning applications are (Lantz 2013; Zhao 2013; Shao and Cen 2014).

A very lucid and helpful survey of active learning is (Settles 2012).

The current literature dedicated to support vector machines includes book written at various levels of mathematical sophistication ranging from accessible titles (Cristianini and Shawe-Taylor 2000; Kung 2014; Statnikov et al. 2011; Suykens et al. 2005) to more advanced (Shawe-Taylor and Cristianini 2005; 2008). A comprehensive discussion related to the implementation of SVM in the `kernlab` package of R is presented in (Karatzoglou et al. 2004; Karatzoglu et al. 2006).

## A Subspaces and Hyperplanes

We assume that the reader is familiar with the notion of linear space, as presented, for example in (Simovici and Djeraba 2014). If $L$ is a real linear space, a subspace of $L$ is a subset $M$ of $L$ such that $\mathbf{x}, \mathbf{y} \in M$ implies $\mathbf{x} + \mathbf{y} \in M$ and $a\mathbf{x} \in M$ for every $a \in \mathbb{R}$.

Note that $L$ is a subspace of $L$ and that the smallest subspace of $L$ is $\{\mathbf{0}\}$. Any intersection of subspaces of $L$ is a subspace of $L$. Therefore, if $X$ is a subset of $L$, then the intersection of all subspaces that contain $X$ is a subspace; we refer to this subspace as the subspace generated by $X$, and we denote it by $\mathsf{span}(X)$.

Let $\mathbf{v} \in \mathbb{R}^n - \{\mathbf{0}\}$ and let $a \in \mathbb{R}$. The *hyperplane* determined by $\mathbf{v}$ and $a$ is the set $H_{\mathbf{v},a} = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{v}'\mathbf{x} = a\}$.

If $\mathbf{x}_0 \in H_{\mathbf{v},a}$, then $\mathbf{v}'\mathbf{x}_0 = a$, so $H_{\mathbf{v},a}$ is also described by the equality

$$H_{\mathbf{v},a} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{v}'(\mathbf{x} - \mathbf{x}_0) = 0\},$$

where $\mathbf{x}_0 \in H_{\mathbf{v},a}$.

Any hyperplane $H_{\mathbf{v},a}$ partitions $\mathbb{R}^n$ into three sets:

$$\begin{aligned} H^{>}_{\mathbf{v},a} &= \{\mathbf{x} \in \mathbb{R}^n | \mathbf{v}'\mathbf{x} > a\}, \\ H^{0}_{\mathbf{v},a} &= H_{\mathbf{v},a}, \\ H^{<}_{\mathbf{v},a} &= \{\mathbf{x} \in \mathbb{R}^n | \mathbf{v}'\mathbf{x} < a\}. \end{aligned}$$

The sets $H^{>}_{\mathbf{v},a}$ and $H^{<}_{\mathbf{v},a}$ are the *positive* and *negative open* half-spaces determined by $H_{\mathbf{v},a}$, respectively. The sets

$$\begin{aligned} H^{\geqslant}_{\mathbf{v},a} &= \{\mathbf{x} \in \mathbb{R}^n | \mathbf{v}'\mathbf{x} \geqslant a\}, \\ H^{\leqslant}_{\mathbf{v},a} &= \{\mathbf{x} \in \mathbb{R}^n | \mathbf{v}'\mathbf{x} \leqslant a\}. \end{aligned}$$

are the *positive* and *negative closed* half-spaces determined by $H_{\mathbf{v},a}$, respectively.

If $\mathbf{x}_1, \mathbf{x}_2 \in H_{\mathbf{v},a}$, then $\mathbf{v} \perp \mathbf{x}_1 - \mathbf{x}_2$. This justifies referring to $\mathbf{v}$ as the *normal to the hyperplane* $H_{\mathbf{v},a}$. Observe that a hyperplane is fully determined by a vector $\mathbf{x}_0 \in H_{\mathbf{v},a}$ and by $\mathbf{v}$.

Let $\mathbf{x}_0 \in \mathbb{R}^n$ and let $H_{\mathbf{x},a}$ be a hyperplane. We seek $\mathbf{x} \in H_{\mathbf{x},a}$ such that $\|\mathbf{x} - \mathbf{x}_0\|_2$ is minimal. Finding $\mathbf{x}$ amounts to minimizing the function $f(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_0\|_2^2 = \sum_{i=1}^{n} (x_i - x_{0i})^2$ subjected to the constraint $\mathbf{v}_1\mathbf{x}_1 + \cdots + w_n\mathbf{x}_n - a = 0$. Using the Lagrangean $L(\mathbf{x}) = f(\mathbf{x}) + \lambda(\mathbf{v}'\mathbf{x} - a)$ and the multiplier $\lambda$, we impose the conditions

$$\frac{\partial L}{\partial x_i} = 0 \quad \text{for} \quad 1 \leqslant i \leqslant n$$

which amount to

$$\frac{\partial f}{\partial x_i} + \lambda w_i = 0$$

for $1 \leqslant i \leqslant n$. These equalities yield $2(x_i - x_{0i}) + \lambda \mathbf{v}_i = 0$, so we have $x_i = x_{0i} - \frac{1}{2}\lambda \mathbf{v}_i$. Consequently, we have $\mathbf{x} = \mathbf{x}_0 - \frac{1}{2}\lambda \mathbf{v}$. Since $\mathbf{x} \in H_{\mathbf{v},a}$, this implies

$$\mathbf{v}'\mathbf{x} = \mathbf{v}'\mathbf{x}_0 - \frac{1}{2}\lambda \mathbf{v}'\mathbf{v} = a.$$

Thus,

$$\lambda = 2\frac{\mathbf{v}'\mathbf{x}_0 - |a}{\mathbf{v}'\mathbf{v}} = 2\frac{\mathbf{v}'\mathbf{x}_0 - a}{\|\mathbf{v}\|_2^2}.$$

We conclude that the closest point in $H_{\mathbf{v},a}$ to $\mathbf{x}_0$ is

$$\mathbf{x} = \mathbf{x}_0 - \frac{\mathbf{v}'\mathbf{x}_0 - a}{\|\mathbf{v}\|_2^2}\mathbf{v}.$$

The smallest distance between $\mathbf{x}_0$ and a point in the hyperplane $H_{\mathbf{v},a}$ is given by

$$\|\mathbf{x}_0 - \mathbf{x}\| = \left\| \frac{|\mathbf{v}'\mathbf{x}_0 - a|}{\|\mathbf{v}\|_2^2}\mathbf{v} \right\| = \frac{|\mathbf{v}'\mathbf{x}_0 - a|}{\|\mathbf{v}\|}$$

If we define the distance $d(H_{\mathbf{v},a}, \mathbf{x}_0)$ between $\mathbf{x}_0$ and $H_{\mathbf{v},a}$ as this smallest distance, we have:

$$d\big(H_{\mathbf{v},a}, \mathbf{x}_0\big) = \frac{|\mathbf{v}'\mathbf{x}_0 - a|}{\|\mathbf{v}\|_2} \tag{8}$$

## B Convexity, Partitions, and Entropy

Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. The *closed segment* determined by $\mathbf{x}$ and $\mathbf{y}$ is the set

$$[\mathbf{x}, \mathbf{y}] = \{a\mathbf{x} + (1-a)\mathbf{y} \mid 0 \leqslant a \leqslant 1\}.$$

A set $C$, $C \subseteq \mathbb{R}^n$ is convex if $\mathbf{x}, \mathbf{y} \in C$ implies $[\mathbf{x}, \mathbf{y}] \subseteq C$.

Let $S$ be a non-empty convex subset of $\mathbb{R}^n$. A function $f : S \to \mathbb{R}$ is *convex* if $f(t\mathbf{x} + (1-t)\mathbf{y}) \leqslant tf(\mathbf{x}) + (1-t)f(\mathbf{y})$ for every $\mathbf{x}, \mathbf{y} \in S$ and $t \in [0, 1]$.

**Theorem B.1** (Jensen's Theorem) *Let $f$ be a function that is convex on an interval $I$. If $t_1, \ldots, t_n \in [0, 1]$ are $n$ numbers such that $\sum_{i=1}^n t_i = 1$, then*

$$f\left(\sum_{i=1}^{n} t_i x_i\right) \leqslant \sum_{i=1}^{n} t_i f(x_i)$$

for every $x_1, \ldots, x_n \in I$.

*Proof* The argument is by induction on $n$, where $n \geqslant 2$. The basis step, $n = 2$, follows immediately from the definition of convex functions.

Suppose that the statement holds for $n$, and let $t_1, \ldots, t_n, t_{n+1}$ be $n + 1$ numbers such that $\sum_{i=1}^{n+1} t_i = 1$. We have

$$\begin{aligned} &f(t_1 x_1 + \cdots + t_{n-1} x_{n-1} + t_n x_n + t_{n+1} x_{n+1}) \\ &= f\left(t_1 x_1 + \cdots + t_{n-1} x_{n-1} + (t_n + t_{n+1}) \frac{t_n x_n + t_{n+1} x_{n+1}}{t_n + t_{n+1}}\right). \end{aligned}$$

By the inductive hypothesis, we can write

$$\begin{aligned} &f(t_1 x_1 + \cdots + t_{n-1} x_{n-1} + t_n x_n + t_{n+1} x_{n+1}) \\ &\leqslant t_1 f(x_1) + \cdots + t_{n-1} f(x_{n-1}) + (t_n + t_{n+1}) f\left(\frac{t_n x_n + t_{n+1} x_{n+1}}{t_n + t_{n+1}}\right). \end{aligned}$$

Next, by the convexity of $f$, we have

$$f\left(\frac{t_n x_n + t_{n+1} x_{n+1}}{t_n + t_{n+1}}\right) \leqslant \frac{t_n}{t_n + t_{n+1}} f(x_n) + \frac{t_{n+1}}{t_n + t_{n+1}} f(x_{n+1}).$$

Combining this inequality with the previous inequality gives the desired conclusion. □

*Example B.2* It is easy to verify that the function $f(x) = x^a$ is convex if $a \geqslant 1$ because $f''(x) = \frac{1}{x} > 0$ for $x \in \mathbb{R}_{>0}$. Therefore, if $t_1, \ldots, t_n \in [0, 1]$ and $\sum_{i=1}^{n} t_i = 1$, by applying Jensen's inequality to $f$, we obtain the inequality:

$$\left(\sum_{i=1}^{n} t_i x_i\right)^a \leq \sum_{i=1}^{n} t_i x_i^a.$$

In particular, if $t_1 = \cdots = t_n = \frac{1}{n}$, it follows that

$$\left(\sum_{i=1}^{n} x_i\right)^a \leqslant n^{a-1} \sum_{i=1}^{n} x_i^a,$$

so

$$\sum_{i=1}^{n} x_i^a \geqslant n^{1-a}\left(\sum_{i=1}^{n} x_i\right)^a.$$

When $\sum_{i=1}^{n} x_i = 1$, the previous inequality implies $\sum_{i=1}^{n} x_i^a \geqslant n^{1-a}$.

A *partition* of a finite and non-empty set $S$ is a collection of non-empty subsets $B_1, \ldots, B_n$ such that

(i) if $1 \leqslant i, j \leqslant n$ and $i \neq j$, $B_i \cap B_j = \emptyset$;
(ii) $\bigcup_{i=1}^{n} B_i = S$. The sets $B_1, \ldots, B_n$ are known as the *blocks* of $\pi$.

Let $\mathsf{part}(S)$ be the set of partitions of the set $S$. A partial order "$\leq$" can be defined on $\mathsf{part}(S)$ as $\pi \leqslant \pi'$ if each block $B'$ of $\pi$ is a union of blocks of the partition $\pi$.

*Example B.3* For $S = \{x_i | 1 \leqslant i \leqslant 6\}$ consider the partitions

$$\pi = \{\{x_1, x_2\}, \{x_6\}, \{x_3, x_5\}, \{x_4\}\},$$
$$\pi' = \{\{x_1, x_2, x_6\}, \{x_3, x_4, x_5\}\}.$$

We have $\pi \leqslant \pi'$ because each of the blocks of $\pi'$ is a union of blocks of $\pi$.

The partition $\iota_S$ whose blocks are singletons $\{x\}$, where $x \in S$, is the least partition defined on $S$. The partition $\theta_S$ that consists of a single block equal to $S$ is the largest partition on $S$.

Let $\pi, \sigma$ be two partitions of a set $S$, where $\pi = \{B_1, \ldots, B_n\}$ and $\sigma = \{C_1, \ldots, C_m\}$. The partition $\pi \wedge \sigma$ of $S$ consists of all non-empty intersections of the form $B_i \cap C_j$, where $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$. Clearly, we have $\pi \wedge \sigma \leqslant \pi$ and $\pi \wedge \sigma \leqslant \sigma$. Moreover, if $\tau$ is a partition of $S$ such that $\tau \leqslant \pi$ and $\tau \leqslant \sigma$, then $\tau \leqslant \pi \wedge \sigma$.

If $T \subset S$ is a non-empty subset of $S$, then any partition $\pi = \{B_1, \ldots, B_n\}$ of $S$ determines a partition $\pi_T$ on $T$ defined by

$$\pi_T = \{T \cap B_i | B_i \in \pi \quad \text{and} \quad T \cap B_i \neq \emptyset\}.$$

For example, if $\pi = \{\{x_1, x_2\}, \{x_6\}, \{x_3, x_5\}, \{x_4\}\}$, the trace on $\pi$ on the set $\{x_1, x_2, x_5, x_6\}$ is the partition $\pi_T = \{\{x_1, x_2\}, \{x_6\}, \{x_5\}\}$.

A subset $T$ of $S$ is $\pi$-*pure*, if $T$ is included in a block of $\pi$, or, equivalently, if $\pi_T = \omega_T$.

Let $\pi = \{B_1, \ldots, B_n\}$ be a partition of a finite set $S$ and let $x_i = \frac{|B_i|}{|S|}$ for $1 \geqslant i \leqslant n$. Since $\sum_{i=1} n x_i = 1$, we have the inequality

$$1 - \sum_{i=1}^{n} \left(\frac{|B_i|}{|S|}\right)^a \leqslant 1 - n^{1-a}.$$

Note that if $|B_1| = \cdots = |B_n| = \frac{|S|}{n}$, the left member of the above equality equals $1 - n^{1-a}$.

**Definition B.4** The *a-entropy* $\mathcal{H}_a(\pi)$ of the partition $\pi = \{B_1, \ldots, B_n\}$ of the set $S$ is given by

$$\mathcal{H}_a(\pi) = \frac{1}{1 - 2^{1-a}} \left( 1 - \sum_{i=1}^{m} \left( \frac{|B_i|}{|S|} \right)^a \right).$$

By the previous considerations, the maximum value of the expression $\mathcal{H}_a(\pi)$ is obtained when the blocks of the partition $\pi$ have equal size and are equal to $\frac{1-n^{1-a}}{1-2^{1-a}}$.

When $a = 2$, we obtain the Gini index of $\pi$, $\text{gini}(\pi) = 2\left( 1 - \sum_{i=1}^{n} \left( \frac{|B_i|}{|S|} \right)^2 \right)$.

*Example B.5* Starting with the convex function $g(x) = x \ln x$ (whose second derivative $g''(x) = \frac{1}{x}$ is positive), the Jensen equality implies:

$$\left( \sum_{i=1}^{n} t_i x_i \right) \ln \left( \sum_{i=1}^{n} t_i x_i \right) \leqslant \sum_{i=1}^{n} t_i x_i \ln x_i$$

for every $x_1, \ldots, x_n \in I$. As before, for $t_1 = \cdots = t_n = \frac{1}{n}$, we have

$$(x_1 + \cdots + x_n) \ln \frac{x_1 + \cdots + x_n}{n} \leqslant \sum_{i=1}^{n} x_i \ln x_i.$$

Applying this inequalities to $x_i = \frac{|B_i|}{|S|}$, where $\pi$ is a partition of $S$ given by $\pi = \{B_1, \ldots, B_n\}$, we have

$$\ln n \geqslant - \sum_{i=1}^{n} \frac{|B_i|}{|S|} \ln \frac{|B_i|}{|S|}.$$

The quantity $- \sum_{i=1}^{n} \frac{|B_i|}{|S|} \ln \frac{|B_i|}{|S|}$ is the *Shannon entropy of* $\pi$. Its maximum value $\ln n$ is obtained when the blocks of $\pi$ have equal size.

Note that $\lim_{a \to \infty} \mathcal{H}_a(\pi) = \mathcal{H}(\pi)$. In other words, Shannon's entropy is a limit case of the $\mathcal{H}_a$-entropy.

Let $\pi, \sigma$ be two partitions of a set $S$, where $\pi = \{B_1, \ldots, B_n\}$ and $\sigma = \{C_1, \ldots, C_m\}$. The *conditional entropy* $\mathcal{H}_a(\pi|\sigma)$ is defined by

$$\mathcal{H}_a(\pi|\sigma) = \sum_{j=1}^{m} \mathcal{H}_a(\pi_{C_j}) \left( \frac{|C_j|}{|S|} \right)^a.$$

Since $\mathcal{H}_a(\pi_{C_j}) = \frac{1}{1-2^{1-a}}\left(1 - \sum_{i=1}^{m}\left(\frac{|B_i \cap C_j|}{|C_j|}\right)^a\right)$, it follows that

$$\mathcal{H}_a(\pi \wedge \sigma) = \mathcal{H}_a(\pi|\sigma) + \mathcal{H}_a(\sigma).$$

Various types of entropies are used to evaluate the impurity of a set relative to a partition. Namely, for a partition $\kappa$ of $S$, $\mathcal{H}_a(\kappa)$ ranges from 0 (when the partition $\kappa$ consists of one block and, therefore, is pure) to $\frac{1-n^{1-a}}{1-2^{1-a}}$ when the partition consists of $n$-singletons, and, therefore, it has the highest degree of impurity.

## C Optimization with Constraints

An *optimization problem consists* in finding a local minimum or a local maximum of a function $f : \mathbb{R}^n \to \mathbb{R}$, when such a minimum exists. The function $f$ is referred to as the *objective function*. Note that finding a local minimum of a function $f$ is equivalent to finding a local maximum for the function $-f$.

In *constrained optimization,* additional conditions are imposed on the argument of the objective function. A typical formulation of a constrained optimization problem is

$$\begin{aligned}\text{minimize} & f(\mathbf{x}), \text{where } \mathbf{x} \in \mathbb{R}^n \\ & \text{subject to } c_i(\mathbf{x}) = 0, \text{ where } 1 \leqslant i \leqslant p, \\ & \text{and } c_j(\mathbf{x}) \geqslant 0, \text{ where } 1 \leqslant j \leqslant q.\end{aligned}$$

Here, $c_i$ are functions that specify *equality constraints* placed on $\mathbf{x}$, while $c_j$ define *inequality constraints*. The *feasible region* of the constrained optimization problem is the set

$$R = \left\{\mathbf{x} \in \mathbb{R}^n | c_i(\mathbf{x}) = 0 \quad \text{for} \quad 1 \leqslant i \leqslant p \text{ and } c_j(\mathbf{x}) \geqslant 0 \quad \text{for} \quad 1 \leqslant j \leqslant q\right\}.$$

If the feasible region $R$ is non-empty and bounded, then, under certain conditions, a solution exists.

If $R = \emptyset$, we say that the constraints are *inconsistent*.

Note that equality constraints can be replaced in a constrained optimization problem by inequality constraints. Indeed, a constraint of the form $c(\mathbf{x}) = 0$ can be replaced by a pair of constrains $c(\mathbf{x}) \geqslant 0$ and $-c(\mathbf{x}) \geqslant 0$.

Let $\mathbf{x} \in R$ be a feasible solution and let $c(\mathbf{x}) \geqslant 0$ be an inequality constraint used to define $R$. If $\mathbf{x} \in \mathbb{R}$ and $c(\mathbf{x}) = 0$, we say that $c$ is an *active constraint*.

Consider the following optimization problem for an object function $f : \mathbb{R}^n \to \mathbb{R}$, the compact set $S \subseteq \mathbb{R}^n$, and the constraint functions $\mathbf{c} : \mathbb{R}^n \to \mathbb{R}^m$ and $\mathbf{d} : \mathbb{R}^n \to \mathbb{R}^p$:

$$\begin{aligned}\text{minimize} & f(\mathbf{x}), \text{where } \mathbf{x} \in S, \\ & \text{subject } u \in \mathbb{R}^m \text{ to } \mathbf{c}(\mathbf{x}) \leqslant \mathbf{0}_m \\ & \text{and } \mathbf{d}(\mathbf{x}) = \mathbf{0}_p\end{aligned}$$

Both the object function $f$ and the constraint functions $\mathbf{c}, \mathbf{d}$ are assumed to be continuously differentiable. We shall refer to this optimization problem as the *primal problem*.

**Definition C.1** The *Lagrangean* associated with this optimization problem is the function $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ given by

$$L(\mathbf{x}, \mathbf{u}, \mathbf{v}) = f(\mathbf{x}) + \mathbf{u}'\mathbf{c}(\mathbf{x}) + \mathbf{v}'\mathbf{d}(\mathbf{x})$$

for $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}^m$, and $\mathbf{v} \in \mathbb{R}^p$. The component $u_i$ of $u$ is the *Lagrangean multiplier* corresponding to the constraint $c_i(\mathbf{x}) \leqslant 0$; the component $v_j$ of $\mathbf{v}$ is the *Lagrangean multiplier* corresponding to the constraint $h_j(\mathbf{x}) = 0$.

The *dual optimization problem* starts with the *Lagrange dual function* $g : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ defined by

$$g(\mathbf{u}, \mathbf{v}) = \inf_{x \in S} L(\mathbf{x}, \mathbf{u}, \mathbf{v}) \tag{9}$$

and consists of

$$\begin{aligned} &\text{maximize } g(\mathbf{u},\ \mathbf{v}), \text{where } \mathbf{u} \in \mathbb{R}^m \text{and } \mathbf{v} \in \mathbb{R}^p, \\ &\qquad\qquad \text{subject to } \mathbf{u} \geqslant \mathbf{0}_m. \end{aligned}$$

**Theorem C.2** *The function $g : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ defined by Equality (9) is concave over $\mathbb{R}^m \times \mathbb{R}^p$.*

*Proof* For $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{R}^m$ and $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^p$, we have:

$$\begin{aligned} &g(t\mathbf{u}_1 + (1-t)\mathbf{u}_2, t\mathbf{v}_1 + (1-t)v_2) \\ &\quad = \inf\{f(\mathbf{x}) + (t\mathbf{u}_1' + (1-t)\mathbf{u}_2')\mathbf{c}(\mathbf{x}) + (t\mathbf{v}_1' + (1-t)\mathbf{v}_2')\mathbf{d}(\mathbf{x}) \,|\, \mathbf{x} \in S\} \\ &\quad = \inf\{t(f(\mathbf{x}) + \mathbf{u}_1'\mathbf{c} + \mathbf{v}_1'\mathbf{d}) + (1-t)(f(\mathbf{x}) + \mathbf{u}_2'\mathbf{c}(\mathbf{x}) + \mathbf{v}_2'\mathbf{d}(\mathbf{x})) \,|\, \mathbf{x} \in S\} \\ &\quad \geqslant t \inf\{f(\mathbf{x}) + \mathbf{u}'_1\mathbf{c} + \mathbf{v}'_1\mathbf{d} | \mathbf{x} \in S\} \\ &\qquad + (1-t)\inf\{f(\mathbf{x}) + \mathbf{u}_2'\mathbf{c}(\mathbf{x}) + \mathbf{v}_2'\mathbf{d}(\mathbf{x}) \,|\, \mathbf{x} \in S\} \\ &\quad = tg(\mathbf{u}_1, \mathbf{v}_1) + (1-t)g(\mathbf{u}_2, \mathbf{v}_2), \end{aligned}$$

which shows that $g$ is concave.

Theorem C.2 is significant because a local optimum of $g$ is a global optimum regardless of convexity properties of $f, \mathbf{c}$ or $\mathbf{d}$. Although the dual function $g$ is not given explicitly, the restrictions of the dual have a simpler form and this may be an advantage in specific cases. □

*Example C.3* Let $f : \mathbb{R}^n \to \mathbb{R}$ be the linear function $f(\mathbf{x}) = \mathbf{a}'\mathbf{x}$, $A \in \mathbb{R}^{p \times n}$, and $\mathbf{b} \in \mathbb{R}^p$. Consider the primal problem:

$$\begin{aligned} &\text{maximize } \mathbf{a}', \mathbf{x}, \text{where } \mathbf{x} \in \mathbb{R}^n, \\ &\qquad\qquad \text{subject to } \mathbf{x} \geqslant \mathbf{0}_n \text{ and} \\ &\qquad\qquad A\mathbf{x} - \mathbf{b} = \mathbf{0}_p. \end{aligned}$$

The constraint functions are $\mathbf{c}(\mathbf{x}) = -\mathbf{x}$ and $\mathbf{d}(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$, and the Lagrangean $L$ is

$$\begin{aligned} L(\mathbf{x}, \mathbf{u}, \mathbf{v}) &= \mathbf{a}'\mathbf{x} - \mathbf{u}'\mathbf{x} + \mathbf{v}'(A\mathbf{x} - \mathbf{b}) \\ &= -\mathbf{v}'\mathbf{b} + (\mathbf{a}' - \mathbf{u}' + \mathbf{v}'A)\mathbf{x}. \end{aligned}$$

This yields the dual function

$$g(\mathbf{u}, \mathbf{v}) = -\mathbf{v}'\mathbf{b} + \inf_{x \in \mathbb{R}^n} (\mathbf{a}' - \mathbf{u}' + \mathbf{v}'A)\mathbf{x}.$$

Unless $\mathbf{a}' - \mathbf{u}' + \mathbf{v}'A = \mathbf{0}_{n'}$, we have $g(\mathbf{u}, \mathbf{v}) = -\infty$. Therefore, we have

$$g(\mathbf{u}, \mathbf{v}) = \begin{cases} -\mathbf{v}'\mathbf{b} & \text{if } \mathbf{a} - \mathbf{u} + A'\mathbf{v} = \mathbf{0}_n, \\ -\infty & \text{otherwise.} \end{cases}$$

Thus, the dual problem is

maximize $g(\mathbf{u}, \mathbf{v})$ subject to $\mathbf{u} \geqslant \mathbf{0}_m$.

An equivalent of the dual problem is

maximize $-\mathbf{v}'\mathbf{b}$ subject to $\mathbf{a} - \mathbf{u} + A'\mathbf{v} = \mathbf{0}_n$ and $\mathbf{u} \geqslant \mathbf{0}_m$.

In turn, this problem is equivalent to:

maximize $-\mathbf{v}'\mathbf{b}$ subject to $\mathbf{a} + A'\mathbf{v} \geqslant \mathbf{0}_n$.

*Example C.4* Let us consider a variant of the primal problem discussed in Example C.3. The objective function is again $f(\mathbf{x}) = \mathbf{a}'\mathbf{x}$. However, now we have only the inequality constraints $\mathbf{c}(\mathbf{x}) \leqslant \mathbf{0}_m$, where $\mathbf{c}(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$, $A \in \mathbb{R}^{m \times n}$, and $\mathbf{b} \in \mathbb{R}^m$. Thus, the primal problem can be stated as

$$\begin{aligned} &\text{maximize } \mathbf{a}'\mathbf{x}, \text{where } \mathbf{x} \in \mathbb{R}^n, \\ &\qquad\qquad \text{subject to } A\mathbf{x} \geqslant \mathbf{b}. \end{aligned}$$

The Lagrangean $L$ is

$$L(\mathbf{x}, \mathbf{u}) = \mathbf{a}'\mathbf{x} + \mathbf{u}'(A\mathbf{x} - \mathbf{b}) = -\mathbf{u}'\mathbf{b} + (\mathbf{a}' + \mathbf{u}'A),$$

which yields the dual function:

$$g(\mathbf{u}) = \begin{cases} -\mathbf{u}'\mathbf{b} & \text{if } \mathbf{a}' + \mathbf{u}'A = \mathbf{0}_m, \\ -\infty & \text{otherwise .} \end{cases}$$

and the dual problem is

$$\begin{gathered} \text{maximize} - \mathbf{b}'\mathbf{u} \,\text{subject to}\, \mathbf{a}' + \mathbf{u}'A = \mathbf{0}_m \\ \text{and}\, \mathbf{u} \geqslant \mathbf{0} \end{gathered}$$

*Example C.5* The following optimization problem

$$\begin{gathered} \text{minimize} \frac{1}{2}\mathbf{x}'Q\mathbf{x} - \mathbf{r}'\mathbf{x}, \text{where}\, \mathbf{x} \in \mathbb{R}^n, \\ \text{subject to}\, A\mathbf{x} \geqslant \mathbf{b}, \end{gathered}$$

where $Q \in \mathbb{R}^{n \times n}$ is a positive definite matrix, and $\mathbf{r} \in \mathbb{R}^n$, $A \in \mathbb{R}^{p \times n}$, and $\mathbf{b} \in \mathbb{R}^p$ are known as a *quadratic optimization problem*.

The Lagrangean $L$ is

$$L(\mathbf{x}, \mathbf{u}) = \frac{1}{2}\mathbf{x}'Q\mathbf{x} - \mathbf{r}'\mathbf{x} + \mathbf{u}'(A\mathbf{x} - \mathbf{b}) = \frac{1}{2}\mathbf{x}'Q\mathbf{x} + (\mathbf{u}'A - \mathbf{r}')\mathbf{x} - \mathbf{u}'\mathbf{b}$$

and the dual function is $g(\mathbf{u}) = \inf_{\mathbf{x} \in \mathbb{R}^n} L(\mathbf{x}, \mathbf{u})$ subject to $\mathbf{u} \geqslant \mathbf{0}_m$. Since $\mathbf{x}$ is unconstrained in the definition of $g$, the minimum is attained when we have the equalities

$$\frac{\partial \frac{1}{2}\mathbf{x}'Q\mathbf{x} + (\mathbf{u}'A - r')\mathbf{x} - \mathbf{u}'b}{\partial x_i} = 0$$

for $1 \leqslant i \leqslant n$, which amount to $\mathbf{x} = Q^{-1}(\mathbf{r} - A\mathbf{u})$. Thus, the dual optimization function is: $g(\mathbf{u}) = -\frac{1}{2}\mathbf{u}'P\mathbf{u} - \mathbf{u}'\mathbf{d} - \frac{1}{2}\mathbf{r}'Q\mathbf{r}$ subject to $\mathbf{u} \geqslant \mathbf{0}_p$, where $P = AQ^{-1}A'$, $\mathbf{d} = \mathbf{b} - AQ^{-1}\mathbf{r}$. This shows that the dual problem of this quadratic optimization problem is itself a quadratic optimization problem.

**Theorem C.6** (The Weak Duality Theorem) *Let* $\boldsymbol{x}_0$ *be a solution of the primal problem and let* $(\mathbf{u}, \mathbf{v})$ *be a solution of the dual problem. We have* $g(\boldsymbol{u}, \boldsymbol{v}) \leqslant f(\boldsymbol{x}_0)$.

*Proof* We have

$$\begin{aligned} g(\mathbf{u}, \mathbf{v}) &= \inf\{f(\mathbf{x}) + \mathbf{u}'\mathbf{c}(\mathbf{x}) + \mathbf{v}'\mathbf{d}(\mathbf{x}) | \mathbf{x} \in S\} \\ &\leqslant f(\mathbf{x}_0) + \mathbf{u}'\mathbf{c}(\mathbf{x}_0) + \mathbf{v}'\mathbf{d}(\mathbf{x}_0) \leqslant f(\mathbf{x}_0), \end{aligned}$$

because $\mathbf{u} \geqslant 0$, $\mathbf{c}(\mathbf{x}_0) \leqslant \mathbf{0}_m$, and $\mathbf{d}(\mathbf{x}_0) = \mathbf{0}_p$ which yields the desired inequality. □

**Corollary C.7** *For the function involved in the primal and dual problems, we have*

$$\sup\{g(\boldsymbol{u},\boldsymbol{v})|\boldsymbol{u}\geqslant \boldsymbol{0}_n\}\leqslant \inf\{f(\boldsymbol{x})|\boldsymbol{x}\in S,\boldsymbol{c}(\boldsymbol{x})\leqslant \boldsymbol{0}_m\}.$$

*Proof* This inequality follows immediately from the proof of Theorem C.6. □

**Corollary C.8** *If $f(\boldsymbol{x}_*)\leqslant g(\boldsymbol{u},\boldsymbol{v})$, where $\boldsymbol{u}\geqslant \boldsymbol{0}_m$ and $\boldsymbol{c}(\boldsymbol{x}_*)\leqslant \boldsymbol{0}_m$, then $\boldsymbol{x}_*$ is a solution of the primal problem and $\boldsymbol{u}$ is a solution of the dual problem.*

*Furthermore, if $\sup\{g(\boldsymbol{u},\boldsymbol{v})|\boldsymbol{u}\geqslant \boldsymbol{0}_m\}=\infty$, then there is no solution of the primal problem.*

*Proof* These statements are an immediate consequence of Corollary C.7. □

*Example C.9* Consider the primal problem

$$\begin{aligned}\text{minimize}\, x_1^2+x_2^2, \text{where}\, x_1\, x_2\in\mathbb{R}\\ \text{subject to}\, x_1-1\leqslant 0.\end{aligned}$$

It is clear that the minimum if $f(\mathbf{x})$ is obtained for $x_1=1$ and $x_2=0$ and this minimum is 1. The Lagrangean is

$$L(\mathbf{u})=x_1^2+x_2^2+u_1(x_1-1)$$

and the dual function is

$$g(\mathbf{u})=\inf\{x_1^2+x_2^2+u_1(x_1-1)|\mathbf{x}\in\mathbb{R}^2\}=-\frac{u_1^2}{4}.$$

Then, $\sup\{g(u_1)|u_1\geqslant 0\}=0$, and a gap exists between the minimal value of the primal function and the maximal value of the dual function.

The possible gap that exists between $\inf\{f(\mathbf{x})|\mathbf{x}\in S,\mathbf{c}(\mathbf{x})\leqslant \mathbf{0}_m\}$ and $\sup\{g(\mathbf{u},\mathbf{v})|\geqslant \mathbf{0}_n\}$ is known as the *duality gap*.

A stronger result holds if certain conditions involving the restrictions are satisfied:

**Theorem C.10** (Strong Duality Theorem) *Let $C$ be a non-empty convex subset of $\mathbb{R}^n$, $f:\mathbb{R}^n\to\mathbb{R}$ and $c:\mathbb{R}^n\to\mathbb{R}^m$ be convex functions, and let $\mathbf{d}:\mathbb{R}^n\to\mathbb{R}^p$ be given by $d(\mathbf{x})=A\mathbf{x}-\mathbf{b}$, where $A\in\mathbb{R}^{p\times n}$ and $b\in\mathbb{R}^p$.*

*Consider the primal problem*

$$\begin{aligned}\textit{minimize}\, f(\boldsymbol{x}), \textit{where}\, \boldsymbol{x}\in S,\\ \textit{subject to}\, \boldsymbol{c}(\boldsymbol{x})\leqslant \boldsymbol{0}_m\\ \textit{and}\, \boldsymbol{d}(\boldsymbol{x})=\boldsymbol{0}_p,\end{aligned}$$

*and its dual*

$$\begin{aligned} &\textit{maxmize } g(\boldsymbol{u}, \boldsymbol{v}), \textit{where } \boldsymbol{u} \in \mathbb{R}^m \textit{ and } \boldsymbol{v} \in \mathbb{R}^p, \\ &\textit{subject to } \boldsymbol{u} \geqslant \boldsymbol{0}_m \end{aligned}$$

*Suppose that there exists* $z \in C$ such that $\mathbf{c}(\mathbf{z}) < \mathbf{0}_m$ *and* $\mathbf{d}(\mathbf{z}) = \mathbf{0}_p$*; additionally,* $\mathbf{0}_p \in \mathbf{I}(\mathbf{d}(C))$. *We have:*

$$\sup\{g(\mathbf{u}, \mathbf{v}) | \mathbf{u} \geqslant \mathbf{0}_m\} = \inf\{f(\mathbf{x}) | \mathbf{x} \in C, \mathbf{c}(\mathbf{x}) \leqslant \mathbf{0}_m, \mathbf{d}(\mathbf{x}) = \mathbf{0}_p\}. \tag{10}$$

*Moreover, if* $\inf\{f(\mathbf{x}) | \mathbf{x} \in C, \mathbf{c}(\mathbf{x}) \leqslant \mathbf{0}_m, \mathbf{d}(\mathbf{x}) = \mathbf{0}_p\}$ *is finite, then there exists* $\mathbf{u}_1, \mathbf{v}_1$ *with* $\mathbf{u}_1 \geqslant \mathbf{0}_m$ *such that* $g(\mathbf{u}_1, \mathbf{v}_1) = \sup\{g(\mathbf{u}_1, \mathbf{v}_1) | \mathbf{u}_1 \geqslant \mathbf{0}_m\}$*; if* $f(\bar{\mathbf{x}}) = \inf\{f(\mathbf{x}) | \mathbf{x} \in C, \mathbf{c}(\mathbf{x}) \leqslant \mathbf{0}_m, \mathbf{d}(\mathbf{x}) = \mathbf{0}_p\}$ *(which means that the infimum is achieved at* $\bar{\mathbf{x}}$*), then* $\mathbf{u}'_1 c(\bar{\mathbf{x}}) = 0$.

If $L$ is the Lagrangean of the primary optimization problem

$$\begin{aligned} &\text{minimize } f(\mathbf{x}), \text{where } \mathbf{x} \in S, \\ &\text{subject to } \mathbf{c}(\mathbf{x}) \leqslant \mathbf{0}_m \\ &\text{and } \mathbf{d}(\mathbf{x}) = \mathbf{0}_{p,} \end{aligned}$$

then a *saddle point* is a triplet $\mathbf{x}_*, \mathbf{u}_*, \mathbf{v}_*$ with $\mathbf{x}_* \in S$ and $\mathbf{u}_* \leqslant \mathbf{0}$ such that

$$L(\mathbf{x}_*, \mathbf{u}, \mathbf{v}) \leqslant L(\mathbf{x}_*, \mathbf{u}_*, \mathbf{v}_*) \leqslant L(\mathbf{x}, \mathbf{u}_*, \mathbf{v}_*)$$

for $\mathbf{x} \in S$ and $\mathbf{u} \geqslant \mathbf{0}$.

The duality gap disappears, and then, a *saddle point* occurs for the primal problem, as stated by the next theorem.

**Theorem C.11** *The triplet* $(\mathbf{x}_*, \mathbf{u}_*, \mathbf{v}_*)$ *is a saddle point of the Lagrangean of the primal problem if and only if its components* $\mathbf{x}_*$ *and* $\mathbf{u}_*, \mathbf{v}_*$ *are solutions of the primal and dual problems, respectively, and there is no duality gap, that is,* $f(\mathbf{x}_*) = g(\mathbf{u}_*, \mathbf{v}_*)$.

## References


Abu-Mostafa YS, Magdon-Ismail M, Lin HT (2012) Learning from data. AML Book, AMLbook.com

Anderson E (1936) The species problem in iris. Ann Mo Bot Gard 23:457–509

Bishop CM (2007) Pattern recognition and machine learning. Springer, New York

Blumer A, Ehrenfeucht A, Haussler D, Warmuth MK (1989) Learnability and the vapnik-chervonenkis dimension. J ACM 36(4):929–965

Breiman L, Friedman JH, Olshen RO, Stone CS (1998) Classification and regression trees. Chapman and Hall, Boca Raton (reprint edition)

Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297

Cristianini N, Shawe-Taylor J (2000) Support vector machines and other kernel-based learning methods. Cambridge University Press, Cambridge
Dasgupta S (2011) Two faces of active learning. Theoret Comput Sci 412:1767–1781
Drucker H, Burges CJC, Kaufman L, Smola AJ, Vapnik V (1996) Support vector regression machines. In: Advances in neural information processing systems 9, NIPS, Denver, CO, USA, 2–5 Dec 1996, pp 155–161
Fisher RA (1936) The use of multiple measurements in taxonomic problems. Ann Eugenics 7:179–188
Freund Y, Shapire RE (1999) Large margin classification using the perceptron algorithm. Mach Learn 37:277–296
Günther F, Fritsch S (2010) Neuralnet: training of neural networks. R J 2(1):30–38
Karatzoglou A, Smola A, Hornik K, Zeileis A (2004) kernlab—an s4 package for kernel methods in R. J Stat Softw 11:1–20
Karatzoglu A, Meyer DM, Hornik K (2006) Support vector machines in R. J Stat Softw 15:1–28
Kung SY (2014) Kernel methods and machine learning. Cambridge University Press, Cambridge
Lander J (2014) R for everyone. Addison-Wesley, Upper Saddle River
Lantz B (2013) Machine learning with R. PACKT Publishing, Birmingham
Lewis DD, Gale WA (1994) A sequential algorithm for training text classifiers. In: Proceedings of the 17th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR'94, pp 3–12. Springer-Verlag New York, Inc, New York
Maindonald J, Braun J (2004) Data analysis and graphics using R—an example-based approach. Cambridge University Press, Cambridge
Matloff N (2011) The art of R programming—a tour of statistical software design. No starch press, San Francisco
McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophy 5:115–133
Mitchell TM (1997) Machine learning. McGraw-Hill, Boston
Mohri M, Rostamizadeh A, Talwalkar A (2012) Foundations of machine learning. MIT Press, Cambridge
Murphy KP (2012) Machine learning: a probabilistic perspective. MIT Press, Cambridge
Novikoff ABJ (1962) On convergence proofs on perceptrons. In: Proceedings of the symposium on mathematical theory of automata 12:615–622
Pitts W, McCulloch WS (1947) How we know universals—the perception of auditory and visual forms. Bull Math Biophys 9:127–147
Quinlan JR (1993) C 4.5 programs for machine learning. Morgan Kaufmann Publ., San Mateo
Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65:386–407
Scheffer T, Decomain C, Wrobel S (2001) Active hidden Markov models for information extraction. In: Advances in intelligent data analysis, 4th international conference, IDA 2001. Cascais, Portugal, Sept 13–15, 2001. Proceedings, pp 309–318
Schohn G, Cohn D (2000) Less is more: active learning with support vector machines. In: Proceedings of the seventeenth international conference on machine learning (ICML 2000), Stanford University, Stanford, CA, June 29–July 2, 2000, pp 839–846
Schütze H, Velipasaoglu E, Pedersen JO (2006) Performance thresholding in practical text classification. In: Proceedings of the 2006 ACM CIKM international conference on information and knowledge management, Arlington, 6–11 Nov 2006, pp 662–671
Settles B (2012) Active learning. Morgan and Claypool
Shalev-Shwartz S, Ben-David S (2014) Understanding machine learning. Cambridge University Press, Cambridge
Shao Y, Cen Y (2014) Data mining applications with R. Academic Press, San Diego
Shawe-Taylor J, Cristianini N (2005) Kernel methods for pattern analysis. Cambridge University Press, Cambridge
Simovici DA, Djeraba C (2014) Mathematical tools for data mining, 2nd edn. Springer, London

Statnikov A, Aliferis CF, Hardin DP, Guyon I (2011) A gentle introduction to support vector machines in biomedicine. World Scientific, Singapore
Steinwart I, Christman A (2008) Support vector machines. Springer, Berlin
Suykens JAK, van Gestel T, De Brabanter J, De Moor B, Vandewalle J (2005) Least squares support vector machines. World Scientific, New Jersey
Suykens JAK, Vandewalle J (1999) Least squares support vector machine classifiers. Neural Process Lett 9(3):293–300
Velipasaoglu E, Schütze H, Pedersen JO (2007) Improving active learning recall via disjunctive boolean constraints. In SIGIR 2007: proceedings of the 30th annual international ACM SIGIR conference on research and development in information retrieval, Amsterdam, The Netherlands, July 23–27, 2007, pp 893–894
Wickham H (2009) ggplot2—Elegant graphics for data analysis. Springer, Dordrecht
Witten IH, Frank E, Hall MA (2011) Data mining—practical machine learning tools and techniques, 3rd edn. Elsevier (Morgan Kaufmann), Amsterdam
Zaki MJ, Meira WM (2014) Data mining and analysis. Cambrige University Press, Cambrige
Zhao Y (2013) R and data mining—example and case studies. Academic Press, San Diego

# On Meta-heuristics in Optimization and Data Analysis. Application to Geosciences

**Henri Luchian, Mihaela Elena Breaban and Andrei Bautu**

**Abstract** This chapter presents popular meta-heuristics inspired from nature focusing on evolutionary computation (EC). The first section, as an elevator pitch, briefly walks through problem solving, touching upon notions such as *optimization problems*, *meta-heuristics*, *constraint handling*, *hybridization*, and the *No Free Lunch Theorem for optimization*, and also giving very short introductions into several most popular meta-heuristics. The next two sections are dedicated to evolutionary algorithms and swarm intelligence (SI), two of the main areas of EC. Three particular optimization methods illustrating these two areas are presented in more detail: genetic algorithms (GAs), differential evolution (DE), and particle swarm optimization (PSO). For a better understanding of these algorithms, references to R packages implementing the algorithms and code samples to solve numerical and combinatorial problems are given. The fourth section is dedicated to the use of EC techniques in data analysis. Optimization of the hyper-parameters of conventional machine learning techniques is illustrated by a case study. The last section reviews applications of meta-heuristics in geosciences.

**Keywords** Meta-heuristics · Numerical and combinatorial optimization · Genetic algorithms · Differential evolution · Particle swarm optimization · Hyper-parameters optimization · Problems in geosciences

H. Luchian · M.E. Breaban (✉)
Faculty of Computer Science, Alexandru Ioan Cuza University of Iasi, Iasi, Romania
e-mail: pmihaela@infoiasi.ro

A. Bautu
Faculty of Navigation and Naval Management, Romanian Naval Academy, Constanta, Romania



C. Cranganu et al. (eds.), *Artificial Intelligent Approaches in Petroleum Geosciences*,
DOI 10.1007/978-3-319-16531-8_2

# 1 A Painless Introduction

A particular characteristic of problem solving becomes evident if computers are used for searching solutions to problems. Namely, when asked to solve a given problem, one is simultaneously, if implicitly, asked to solve the meta-problem of finding the best method to solve the problem. Best may refer to saving resources most often, time in the process of finding a solution; it may also point to the required accuracy/precision of the solution or to the set of instances of the problem which must be solved, or to a threshold for positive/negative errors, etc. In many cases, simply finding a method which can successfully look for a solution to the given problem is not sufficient; the method should comply with requirements such as those enumerated above, and moreover, it should do this in the best possible way. Therefore, irrespective of what best means, in order to deal with the companion meta-problem, one needs to be acquainted with a comprehensive set of methods for solving problems: the larger the set of methods one chooses from, the better the proposed method should be.

This may be the reason why, along with the ever increasing use of computers for solving problems, a wealth of new approaches to problem solving has been proposed.

## *1.1 Briefly, on Problems and Methods to Solve Them*

**How many problem-solving methods does one need to master?** Indeed, many new methods for solving problems were invented (some may say discovered) lately. As opposed to exact deterministic algorithms, many of these new methods are weak methods; a weak method is not rigidly related to one specific problem, but rather it can be applied for solving various problems. At times, one or another such problem-solving technique appears to be most fashionable. To an outsider, genetic algorithms (GAs), artificial neural networks, particle swarm optimization, and support vector machines to name just a few seemed to successively take by storm the proscenium over the last decades. Is each new method better than the previous ones and, consequently, is the choice of the method to solve ones specific problem a matter of keeping pace with fashion? Is there one particular method that solves best, among all existing methods and all problems? A positive answer to either question would mean that we actually have a free lunch when trying to solve a given problem: we could spare the time needed to identify the best method for finding solutions to the problem. However, a theorem proven in 1995 by Wolpert and McReady (1997), called the *No Free Lunch Theorem for optimization*, shows that the answer to both questions above is negative. Informally (and leaving aside details and nuances of the theorem), the *NFLTO* states that, averaging overall problems, all solving methods have the same performance, no matter what indicator of performance is used. Obviously, the common average is obtained from various

sets of values of the performance indicators for each method and various levels of each method's performance when applied to each specific problem. This means that in general, two different methods perform at their respective best on different problems, and consequently, each of them has a poorer performance on remaining problems. It follows that there is no problem-solving method which is the "best" method to solve all problems (indeed, if a method M would have equally good performances on all problems, then this would be Ms average performance; then, any method with scattered values of the performance indicator would outperform M on some problems). Therefore, for each problem-solving method, there is a subset of all problems for which it is the best solving method in some cases, and the subset may consist of only one problem or even zero problems. Conversely, given a problem to be solved, one has to find a particular method that works best for that problem which proves that the *meta-problem* mentioned above is non-trivial. Actually, it may be a very difficult problem; similar to the way some problem-solving methods are widely used even if they are not guaranteed to provide the exact solution, an approximate but acceptably good solution to the meta-problem may be useful.

**Optimization problems** There is an informal conjecture stating that anything we are doing, we optimize something; or, as Clerc put it in (2006), iterative optimization is as old as life itself. While each of these two statements may be the subject of subtle philosophical debates, it is true that many problems can be stated as optimization problems. Finding the average of $n$ real numbers is an optimization problem (*find the number a which minimizes the sum of its distances absolute values of the differences to each of the given numbers*); the same goes for decision-making problems, for machine learning ones, and many others.

An optimization problem asks to find—if it exists—an extreme value (either minimum or maximum) of a given function. Finding the required solution is, in fact, a search process performed in the space of all candidate solutions; this is why the terms *optimization method* and *search method* are sometimes loosely used as synonyms, although the term *optimization* refers to the values of the function, while *search* (through the set of candidate solutions) usually points to values of the variables of the respective function. Several simple taxonomies of optimization problems are useful when studying meta-heuristics: optimization of functions of continuous variable/discrete variable; optimization with/without constraints; optimization with a fixed/moving optimum; single objective/multiple objective optimization. Here are some examples:

- constraint optimization raises the critical problem of handling constraints;
- continuous/discrete variables point to specific meta-heuristics that originally specialize in one the two types of optimization (e.g., GAs for discrete variables; differential evolution (DE) for continuous variables);
- self-adapting meta-heuristics are recommended for solving problems with a moving optimum;

- particular variants of existing meta-heuristics have been defined for multi-objective optimization (e.g., in DE).

*Meta-heuristics*, described below, are seen as optimization methods (i.e., methods for solving optimization problems). While meta-heuristics can also be used for solving, for example, complex-system-design problems or machine learning problems, such problems can also be stated as optimization problems.

**Meta-heuristics** Any problem-solving method belongs to one of three categories: exact deterministic methods, approximate deterministic methods, and non-deterministic methods. This chapter is concerned with the second and third categories, which flourished over the last few decades.

A *heuristic* is a problem-solving method which is able to find approximate solutions to the given problem either in a (significantly) shorter time than an exact algorithm or simply when no exact algorithm can find a solution. Approximate solutions may be acceptable in various situations; Simon (1969) argues that humans tend to *satisfice* (use an acceptable approximate solution obtained reasonably quickly) when it comes to complex situations/domains.

*Meta-heuristic* is a relatively recent term, introduced by Glover in 1986. Various definitions and taxonomies of *meta-heuristics* were subsequently proposed; the works mentioned below discuss these in detail. It is generally accepted that meta-heuristics are problem-independent high-level strategies which guide the process of finding (approximate) solutions to given problems. However, problem-independent methods (also called *weak methods*) may well be fine-tuned by incorporating in the search procedure some problem-specific knowledge; an early paper on this is (Grefenstette 1987).

Among several existing taxonomies of meta-heuristics, the most interesting one for our discussion is the classification concerned with the number of current solutions. A *trajectory* or *single-point meta-heuristic* works with only one current solution; the current solution is iteratively subject to conditional change. Local search meta-heuristics, such as Tabu Search, Iterated Local Search, and Variable Neighborhood Search (Blum and Roli 2003), fall into this category. A *population-based meta-heuristic* iteratively change a set of candidate solutions collectively called *population*; genetic algorithm (GA) or particle swarm Optimization, among others, belong in this category.

This section briefly discusses two trajectory-based methods: iterated hill climbing and simulated annealing.

**Hill climbing** Hill climbing is a *weak* optimization heuristic: In order to be applied for solving a given problem, the only properties that are required are that the function to be optimized takes on values which can always be compared against each other (a totally ordered set of values such as the real numbers or the natural numbers) and that it allows for a step-by-step improvement of candidate solutions (i.e., the problem is not akin to finding the needle in the haystack). Hill climbing does not use any other properties of the function to be optimized and does not

organize the search for the optimum following a tree structure—or any other structure. Therefore, it requires little computer memory. Hill climbing starts with an initial candidate solution and iteratively aims at improving the current candidate solution by replacing it with any (or the best) neighbor solution which is better than the current one; when there are no more possible improvements, the search stops. The neighborhood can be considered either in the set over which the function is defined (a neighbor can be obtained through a slight modification of a number which is a component of the candidate solution) or in the set of computer representations of candidate solutions (a neighbor there is reached by flipping one bit).

While the procedure sketched above is very effective for any mono-modal function (informally, a function whose graph has only one hilltop), it may get stuck in local optima if the function is multi-modal. In the latter case, the graph of the function will also have a second-highest hill, a third highest one, etc.; one run of the hill-climbing procedure having the initial solution at the shoulder of the second-highest hill will find the second-highest hilltop (a local optimum), but then, it will get stuck there, since no improvement is possible anymore in the neighborhood. This is why for multi-modal functions iterated hill climbing is used instead of one-iteration hill climbing: The method is applied several times in a row, with different initial candidate solutions, thus increasing the chance that one run of the method will start at the foot of the hill which contains the global optimum.

**Simulated Annealing** The problem described above—optimization methods getting stuck in local optima—was actually impairing potential advances in optimization methods. A breakthrough has been the Metropolis algorithm (Metropolis et al. 1953). The new idea was to occasionally allow for candidate solutions which are worse than the current one to replace the current solution. This is compatible with the hill-climbing metaphor: Indeed, when one wanders through a hilly landscape aiming at reaching the top of the highest hill, he/she may have to occasionally climb down a hill in order to reach a higher one.

The idea of expanding the exploration capabilities of the optimization method at the expense of the quality of the current solution proved to be very productive. Nevertheless, a better idea is to also keep under some kind of control the ratio between the number of steps when the current solution is actually improved and the number of steps when the current solution is worsened. This is where simulated annealing comes into scene. Beings of nature have not been the only inspiration for problem-solving researchers; non-living-world processes are also a rich source for metaphors and simulations in problem solving. One celebrating example is annealing: Cooled gradually, a metal can gain most desirable physical properties (e.g., ductility and flexibility), while sudden cooling of a metal hardens it.

Kirckpatrick et al. (1983) proposed a simulation of annealing which uses a parameter (the temperature) for controlling the improvement/worsening ratio mentioned above: The lower the temperature, the fewer steps which worsen the current solution are allowed. Analogously to what happens in the physical–chemical process

of annealing, the temperature starts at a (relatively) high value and decreases at each iteration of the current-solution-changing process. Simulated annealing has been successfully applied to solve many discrete and continuous optimization problems, including optimal design.

The rest of this chapter and Chapter "Genetic Programming Techniques with Applications in the Oil and Gas Industry" present several population-based meta-heuristics: GAs and genetic programing, DE, and particle swarm optimization. We briefly introduce each of them in the following paragraphs. Four particular topics of interest, in particular for the meta-heuristics under discussion, are then briefly touched upon.

Many more meta-heuristics have been proposed and new ones continue to appear. Monographs and surveys on meta-heuristics such as Glover (1986); Talbi (2009); Voß (2001) give comprehensive insights into the topic. The *International Journal of Meta-heuristics* publishes both theoretical and application papers on methods including: neighborhood search algorithms, evolutionary algorithms, ant systems, particle swarms, variable neighborhood search, artificial neural networks, and artificial immune systems. Those interested in approaches to solving the *meta-problem* above may wish to read about *hyper-heuristics*—a term coined by Burke; a survey is provided in Burke et al. (2013).

## 1.2 What Will the Rest of This Chapter and the Next One Elaborate On?

We introduce briefly the main topics of the two chapters.

**Genetic Algorithms** Ingo Rechenberg, a professor with the Technical University of Berlin and a parent of evolution strategies, made a statement which supports the use of evolutionary techniques for problem solving: "Natural evolution is, or comprises, a very efficient optimization process, which, by simulation, can conduct to solving difficult optimization processes" Rechenberg (1973). The statement is empirically supported by many successful applications of evolutionary techniques for solving various optimization problems. The field of evolutionary computing now includes various techniques; the pioneering ones have been the GAs (Holland 1975), the evolution programs Fogel et al. (1966), and the evolution strategies Rechenberg (1973; Schwefel 1993). Excellent textbooks on GAs are widely used: Michalewicz (1992; Mitchell 1996), or a more general one, on evolutionary computing (Jong 2006).

As the title of the groundbreaking book by Holland suggests, *adaptation* has been the core idea that led to GAs; *self-adapting* techniques became ever since more and more popular. Trying to reach the optimum starting from initial guesses as candidate solutions, such techniques self-adapt their search using properties of the search space of (the instance of) the problem.

GAs simulate a few basic factors of natural evolution: mutation, crossover, and selection. The implementation of each of these simulated factors involves generating random numbers: like all evolutionary methods, GAs are non-deterministic. Adaptation, which is instrumental in natural evolution, is simulated by calculating values of a function (the environment) and, on this basis, making candidate solutions compete for survival for the next generation. The evolution of the population of solutions can be seen as a learning process where candidate solutions learn collectively.

More sophisticated variants of GAs simulate further factors of natural evolution, such as the integrated evolution of two species [coevolution (Hillis 1990) the host–parasite model].

One particular feature of GAs is that the whole computation process takes place in two dual spaces: the space of candidate solutions to the given problem (where the evaluation and the subsequent selection for survival take place the *phenotype*) and the space of the representations of such solutions (where genetic operators such as mutation and crossover are applied the *genotype*). This characteristic is also borrowed from natural evolution, where the genetic code and the actual being evolved from that code are instantiations of the two-space paradigm: In natural evolution, the genetic code is altered through mutations and through crossover between parents; subsequently, the being evolved from the genetic code is evaluated with respect to its adaptation to the environment.

The genetic code in GAs is actually the way candidate solutions are represented in the computer. The standard GAs (Michalewicz 1992) works with chromosomes (representations of candidate solutions) which are strings of bits. When applied to solve real-world problems, GAs evolved toward sophisticated representations of candidate solutions, including varying-length chromosomes and multi-dimensional chromosomes. One particular representation of candidate solutions has been groundbreaking: trees from graph theory.

**Genetic Programing** emerged as a distinct area of GA. In his seminal book (Koza 1992), Koza uses a particular definition for the solution to a problem: A solution is a computer program which solves the problem. Adding to this the idea that such computer programs can be developed automatically, in particular through genetic programing, a flourishing field of research and applications emerged. As Poli et. al. put it, genetic programing automatically solves problems without requiring the user to know or specify the form or structure of the solution in advance (Poli et al. 2008).

A tree can be seen as representing a calculation, in particular, a computer program. In genetic programing, computer programs evolve in an automated manner through self-adaptation of a population of trees each tree representing a candidate program. Evaluation of candidate solutions is carried out using a set of instances of the problem to be solved for which the actual solutions are known beforehand. Specific operators have been introduced to cope with peculiarities of the evolution of trees as abstract representations.

Spectacular results have been obtained using genetic programing, including patentable inventions.

**Differential Evolution** Since 1996, when it was publicly proposed by Price and Storn (1997), DE became a popular optimization technique. It is a population-based method designed for minimizing multi-dimensional real-valued functions through vector processing; the function needs not be continuous (let alone differentiable) and, even if it is differentiable, no information on the gradient is used.

DE follows the general steps of an evolutionary scheme: initialisation, applying specific operators (see the one described below), evaluation, and selection; this sequence being iterated from the second step until a halting condition is met. The basic operation in DE is to add the weighted difference of two vectors in the population to a third one. Thus, the candidate solutions learn from each other; the computation is a self-adapting process.

From its early days, DE proved to be a powerful optimization technique: It won the general-purpose algorithms competition in the First International Contest on Evolutionary Optimization, 1996 (at the IEEE International Conference on Evolutionary Computation). As was the case with other evolutionary techniques, DE evolved to incorporate new elements such as elitism or coevolution. Pareto-based approaches have been proposed for tackling multiple objective optimization problems using DE (Madavan 2002).

**Particle Swarm Optimization** *Collective intelligence* (Nguyen and Kowalczyk 2012) is a rich source of inspiration for designing meta-heuristics through simulation. Particularly, successful among such meta-heuristics are *Ant Colony Optimization* (Dorigo and Stützle 2004) and *Particle Swarm Optimization*.

The seminal paper for the latter meta-heuristic is (Kennedy and Eberhart 1995); a textbook dedicated to PSO is (Clerc 2006). Bird flocking or fish schooling can be considered as being the inspiring metaphors from nature. The core idea is that at each iteration, each *particle* (candidate solution) moves through the search space according to a (linear) combination of the particles current move, of the best personal previous position, and of the best previous position of the neighbors (what neighbors means, is a parameter of the procedure). This powerful combination of the backtracking flavor (keeping track somehow of the previous personal best) and collective learning (partly aiming at the regional/global previous best) makes PSO well suited for optimization problems with a moving optimum.

## *1.3 Short Comments on Four Transversal Issues*

**Parameter Control** A key element for the successful design of any meta-heuristic is a proper adjustment of its parameters. Suffices it to think of the number of candidate GAs one has to select from when designing a GA for a given problem:

Mutation rates and crossover rates can, at least theoretically, take on any value between 0 and 1; there are tens of choices for the population size; the selection procedure can be any of at least ten popular ones (new ones can be invented), etc. This makes a search space for properly designing a GAs for a given problem in the range of at least hundreds of thousands candidate GAs; of these, only a few will probably have a good performance and finding these among all possible GAs for that problem is a non-trivial task.

In the design phase of a meta-heuristic, parameters can be set by hand or automatically. For example, for GAs, a supervisor GAs have been proposed (Grefenstette 1986) which can be used for off-line improvement of the parameters of a given GAs such as the population size, the mutation, and crossover rates.

If one chooses to have dynamic parameter values during the run of the algorithm, this can be done automatically, for example, upon automatedly checking whether or not any change of the best-so-far solution happened during a given number of iterations.

**Constraint Handling** When the problem to be solved belongs to the *constraint optimization* class, a major concern along the iterative solution-improving process is that of preserving the *feasability* of candidate solutions, i.e., keeping only solutions which satisfy all the constraints. The way a feasible solution is obtained in the first place is beyond the scope of this paragraph—this may happen, for example, by applying a heuristic which ends up with a feasible but, very likely, non-optimal solution. Subsequently, the iterative solution-improvement process successively changes the current solution; every such change may turn a current solution which is feasible into one which is not. When unfeasible solutions (candidate solutions which do not satisfy the problem constraints) are obtained, the optimization method should address this.

There are three main ways of tackling unfeasible solutions. A first approach is to penalize unfeasible solutions and otherwise let them continue to be part of the search process. In this way, an unfeasible solution becomes even less competitive than it actually is with respect to the search-for-the-optimum process (see *fitness function* in the GAs section of this chapter). A second approach is to *repair* the new solution in case it is unfeasible (repairing means changing the solution in such a way that it becomes feasible); the fact that repairing may have the same complexity as the original given problem makes this approach least recommendable. The best approach seems to be that of including the constraints (or at least some of them) into the representation of solutions. This idea is convincingly illustrated for numerical problems in Michalewicz (1992) where bit string representations are used: Any bit string is decoded into a feasible solution. This approach has the decisive advantage that there is no need to check whether or not candidate solutions obtained from existing ones are feasible. When including the problem constraints into the codification of candidate solutions, one actually uses *hybridisation with the problem*, which is mentioned in the next paragraph.

**Hybridisation** According to one of the definitions in Blum and Roli (2003), a basic idea of meta-heuristics in general is to combine two or more heuristics in one problem-tailored procedure and use it as a specific meta-heuristic. *Hybridisation* has even more subtle aspects. Hybridisation happens when inserting an element from one meta-heuristic into another meta-heuristic (e.g., using crossover, a defining operator for GA, in an evolution strategy which, in its standard form, uses only mutations). Another form of hybridisation could be called hybridisation with the problem: Problem-specific properties can be used for defining particular operators in a meta-heuristic. An example can be found in Michalewicz (1992): For the transportation problem, a feasible solution remains feasible after applying on it a certain transformation; this transformation is then used to define the mutation operator. An example of hybridisation is illustrated in this book, in the chapter on genetic programing.

Hybridisation is recommended, in general, for improving the problem-solving method. This could be called intended hybridisation, and it has proven its beneficial effects in countless successful applications.

There also exists an unintended hybridisation which one should be aware of. For example, when trying to optimize a Royal Road function (Mitchell et al. 1992), the search in a large plateau (while a substring of 8 bits does not yet contain only 1s) is akin to a blind search, even though we run a GA for solving the problem. Indeed, the probability field constructed for the selection has, for the whole plateau, equal probabilities, and consequently, the selection is not biased toward solutions closer to the optimum—it is rather a random selection. This way, the GA designed to solve the Royal Road problem is (unwillingly) hybridised with random search which takes over temporarily while walking the plateau.

**Experiments** Non-deterministic methods are used in a way which differs from that of deterministic ones. The latter will always provide the same output for a given input, while the former may give different results when run repeatedly with the same input. This behavior leads to the need of assessing the quality of a non-deterministic algorithm by repeatedly running it with the same input. Various statistics can be used —usually, the average of the respective best solutions and their standard deviation, over a number of runs. Therefore, the proper use of non-deterministic methods requires at least basic knowledge of probabilities and statistics, in particular *Experiment design*. Testing statistical hypothesis gives substance to the study of the performance of (non-deterministic) meta-heuristics.

## *1.4 Going into Practice: Two Running Examples*

In order to illustrate the optimization process conducted within the methods described in this chapter, two optimization problems are formulated here. Sample code in R (including the output) invoking the algorithms under consideration is listed in the next sections in an attempt to familiarize the reader with some available, easy-to-use software.

The first optimization problem, known as Six Hump Camel Back, is commonly used as a benchmark function to assess the performance of optimization algorithms to which its multi-modal complex landscape imposes serious difficulties. It is formulated as a minimization problem over two continuous variables. The problem is defined as follows:

$$\begin{array}{ll} \text{Minimize} & f(x_1, x_2) = (4 - 2.1x_1^2 + x_1^4/3)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2 \\ \text{where} & -3 \leq x_1 \leq 3, \\ & -2 \leq x_2 \leq 2. \end{array} \tag{1}$$

The landscape of the function is illustrated in Fig. 1 with the aid of perspective and contour plots in R.

Visible on the plots above, the function has six local minima and two global minima. The two global minima lie at locations $(x_1, x_2) = (-0.0898, 0.7126)$ and $(x_1, x_2) = (0.0898, -0.7126)$; the value returned at these locations corresponds to $f(x_1, x_2) = -1.0316$.

The R code defining the Six Hump Camel Back function is shown below.

```
> SixHump <- function (x1, x2)
 {
   (4-2.1*x1^2+x1^4/3)*x1^2+x1*x2+(-4+4*x2^2)*x2^2
 }
```

An equivalent function can be implemented in R using as argument a vector. This formulation is more appropriate for our goals because, this general form which does not impose restrictions on the size of the input, can be further called by other R routines implementing the meta-heuristics presented in this chapter.

```
> SixHumpV <- function (x)
 {
   (4-2.1*x[1]^2+x[1]^4/3)*x[1]^2+x[1]*x[2]+(-4+4*x[2]^2)*x[2]^2
 }
```

We also illustrate the use of meta-heuristics on a constrained optimization problem with discrete variables, frequently arising in the oil and gas industry: portfolio selection. While this problem may be found under various formulations, we tackle here the variant presented in Shakhsi-Niaei et al. (2013). Given a firm with a budget $b$, $n$ projects, with the net value of the $i$th project denoted by $f_i$ and the cost of the $i$th project denoted by $c_i$, one must find the combination of projects that maximizes the total utility for the firm, as computed in Eq. 2:

$$\text{Maximize } z = \sum_{i=1}^{n} f_i x_i, \tag{2}$$

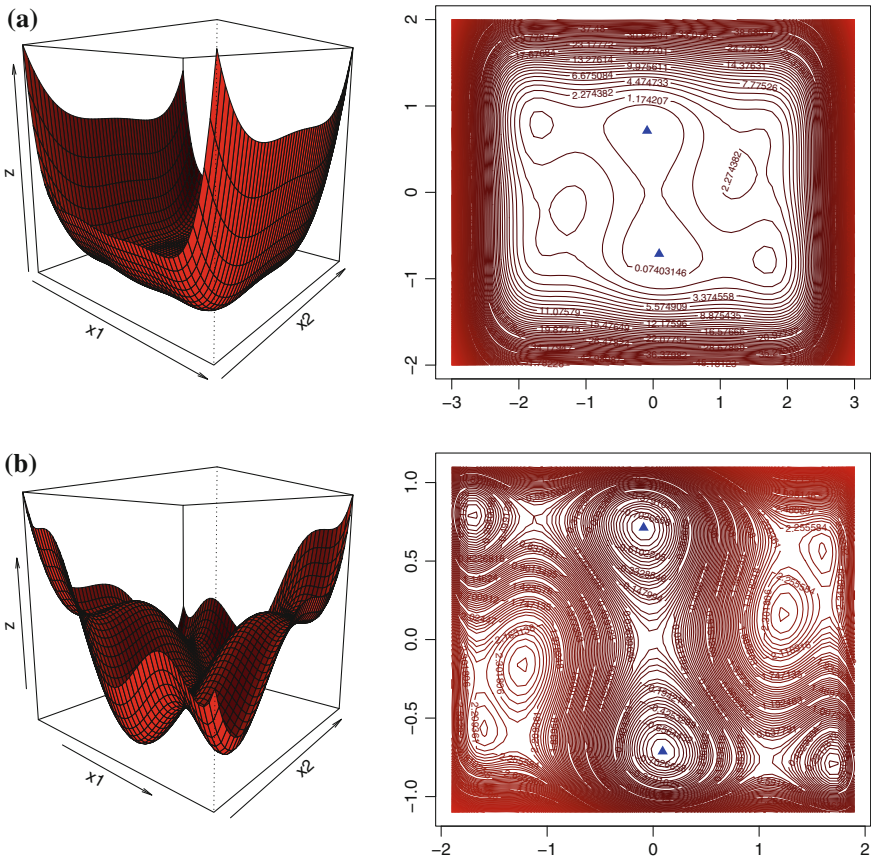


**Fig. 1** Perspective and contour plots for Six Hump Camel Back: **a** for the entire domain of definition: $x_1 \in [-3, 3]$, $x_2 \in [-2, 2]$, **b** restricted to $x_1 \in [-1.9, 1.9]$, $x_2 \in [-1.1, 1.1]$. The two global optima are illustrated as *blue triangles* at locations $(x_1, x_2) = (-0.0898, 0.7126)$ and $(x_1, x_2) = (0.0898, -0.7126)$

$$\text{Subject to: } \sum_{i=1}^{n} c_i x_i \leq b, \tag{3}$$

$$x_i \in 0, 1, \quad i = \overline{1, n}. \tag{4}$$

$$x_1 + x_2 \leq 1, \tag{5}$$

$$x_5 + x_3 \leq 1. \tag{6}$$

$$x_5 + x_3 + x_4 \leq 2. \tag{7}$$

The variables $x_i$ represent the decision to select project $i$ ($x_i = 0$ means the project is not selected, whereas $x_i = 1$ means the project gets selected for implementation)—

constraint expressed by Eq. 4. The total budget of the firm must not be exceeded by the total costs of the projects selected (Eq. 3). Other constraints may be imposed on the problem (especially in a real-world context), such as Eq. 5 expresses the condition that either project 1 or project 2 gets implemented; Eq. 6 expresses the condition that either project 3 or project 5 gets implemented; Eq. 7 expresses the condition that at most 2 out of the 3 projects (3, 4, and 5) may get implemented.

## 2 Evolutionary Algorithms

Evolutionary algorithms (EAs) are simplified computational models of the evolutionary processes that occur in nature. They are search methods implementing principles of natural selection and genetics. Parts of this section follow closely the text in (Breaban 2011).

### *2.1 Terminology*

Evolutionary algorithms use a vocabulary borrowed from genetics. They simulate the evolution across a sequence of *generations* (iterations within an iterative process) of a *population* (set) of candidate solutions. A candidate solution is internally represented as a string of *genes* and is called *chromosome* or *individual*. The position of a gene in a chromosome is called *locus,* and all the possible values for the gene form the set of *alleles* of the respective gene. The internal representation (encoding) of a candidate solution in an evolutionary algorithm form the *genotype*; this information is processed by the evolutionary algorithm. Each chromosome corresponds to a candidate solution in the search space of the problem which represents its *phenotype*. A decoding function is necessary to translate the genotype into phenotype. If the search space is finite, it is desirable that this function should satisfy the bijection property in order to avoid redundancy in chromosomes encoding (which would slow down the convergence) and to ensure the coverage of the entire search space.

The population maintained by an evolutionary algorithm evolves with the aid of genetic operators that simulate the fundamental elements in genetics: *Mutation* consists in a random perturbation of a gene, while *crossover* aims at exchanging genetic information among several chromosomes. The chromosome subjected to a genetic operator is called *parent* and the resulted chromosome is called *offspring*.

A process called *selection* involving some degree of randomness selects the individuals to breed and create offsprings, mainly based on individual merit. The individual merit is measured using a **fitness function** which quantifies how fitted the candidate solution encoded by the chromosome is for the problem being solved. The fitness function is formulated based on the mathematical function to be optimized.

The solution returned by an evolutionary algorithm is usually the most fitted chromosome in the last generation.

## 2.2 Directions in Evolutionary Algorithms

First efforts to develop computational models of evolutionary systems date back to 1950s (Bremermann 1958; Fraser 1957). Several distinct interpretations, which are widely used nowadays, were independently developed later. The main differences between these classes of evolutionary algorithms consist in solution encoding, operators implementation, and selection schemes.

*Evolutionary programing* crystallized in 1963 in the USA at San Diego University, when Fogel (1966) generated simple programs as simple finite-state machines; this technique was developed further by his son David Fogel. A random mutation operator was applied on state transition diagrams, and the best chromosome was selected for survival.

*Evolutionary strategies* (ES) were introduced in 1960s when Hans-Paul Schwefel and Ingo Rechenberg, working on a problem from mechanics involving shape optimization, designed a new optimization technique because existing mathematical methods were unable to provide a solution. The first ES algorithm was initially proposed by Schwefel in 1965 and developed further by Rechenberg (1973). Their method was designed to solve optimization problems with continuous variables; it used one candidate solution and applied random mutations followed by the selection of the fittest. ES were later strongly promoted by Back (1996) who incorporated the idea of population of solutions.

*GAs* were developed by John Henry Holland in 1973 after years of study of the idea of simulating the natural evolution. These algorithms model the genetic inheritance and the Darwinian competition for survival. GAs are described in more detail in Sect. 2.3.

*Genetic programing* is a specialized form of a GA. The specialization consists in manipulating a very specific type of encoding and, consequently, in using modified versions of the genetic operators. GP was introduced by Koza in 1992 in an attempt to perform automatic programing. GP manipulates directly phenotypes, which are computer programs (hierarchical structures) expressed as trees. It is currently intensively used to solve symbolic regression problems. Genetic programing and one important variation—gene expression programing—are described in Chapter "Genetic Programming Techniques with Applications in the Oil and Gas Industry" of this book.

*DE* (Storn and Price 1997) is a more recent class of evolutionary algorithms whose operators are specifically designed for numerical optimization. DE is described in detail in Sect. 2.4.

An in-depth analysis under a unified view of these distinct directions in evolutionary algorithms is presented in De Jong (2006).

**Fig. 2** A generic genetic algorithm

$t := 0$
Initialize $P_0$
Evaluate $P_0$
**while** *halting condition not met* **do**
    $t := t + 1$
    select $P_t$ from $P_{t-1}$
    apply crossover and mutation in $P_t$
    evaluate $P_t$
**end while**

## 2.3 Genetic Algorithms

GAs (Holland 1998) are the most well known and the most intensively used class of evolutionary algorithms.

A GA performs a multi-dimensional search by means of a population of candidate solutions which exchange information and evolve during an iterative process. The process is illustrated by the pseudo-code in Fig. 2.

In order to solve a problem with a GA, one must define the following elements:

- an encoding for candidate solutions (the genotype);
- an initialization procedure to generate the initial population of candidate solutions;
- a fitness function which defines the environment and measures the quality of the candidate solutions;
- a selection scheme;
- genetic operators (mutation and crossover);
- numerical parameters.

The encoding is considered to be the main factor determining the success or failure of a GA.

The standard **encoding** in GAs consists in binary strings of fixed length. The main advantage of this encoding is offered by the existence of a theoretical model (the Schema theorem) explaining the search process until convergence. Another advantage shown by Holland is the high implicit parallelism in the GA. A widely used extension to the binary encoding is gray coding.

Unfortunately, for many problems, this encoding is not a natural one and it is difficult to be adapted. However, GAs themselves evolved and the encoding extended to strings of integer and real numbers, permutations, trees, and multi-dimensional structures. Decoding the chromosome onto a candidate solution to the problem sometimes necessitates problem-specific heuristics.

Important factors that need to be analyzed with regard to the encoding are the size of the search space induced by a representation and the coverage of the phenotype space: Whether the phenotype space is entirely covered and/or reachable, whether the mapping from genotype to phenotype is injective, or "degenerate," and

whether particular (groups of) phenotypes are over-represented (Radcliffe et al. 1995). Also, the "heritability" and "locality" of the representation under crossover and mutation need to be studied (Raidl and Gottlieb 2005).

The **initialization** of the population is usually performed randomly. There exist approaches which make use of greedy strategies to construct some initial good solutions or other specific methods depending on the problem.

The **fitness function** is constructed based on the mathematical function to be optimized. For more complex problems, the fitness function may involve very complex computations and increase the intrinsic polynomial complexity of the GA.

Several probabilistic procedures based on the fitness distribution in population can be used to select the individuals to survive in the next generations and produce offsprings; this phase of the algorithm is known as **selection for variation**. All these procedures encourage to some degree the survival of the fittest individuals, allowing at the same time that the worst adapted individual survive and contribute with local information (short-length substrings) to the structure of the optimal solution. The most essential feature which differentiates them is the selection pressure: the degree to which the better individuals are favored; the higher the selection pressure, the more the better individuals are favored. The selection pressure has a great impact on the diversity in population and consequently on the convergence of GAs. If the selection pressure is too high, the algorithm will suffer from insufficient exploration of the search space and premature convergence occurs, resulting in sub-optimal solutions. On the contrary, if the selection pressure is too low, the algorithm will unnecessarily take longer time to reach the optimal solution. Various selection schemes were proposed and studied from this perspective. They can be grouped into two classes: proportionate-based selection and ordinal-based selection. Proportionate-based selection takes into account the absolute values of the fitness. The most known procedures in this class are as follows: roulette wheel (Holland 1975) and stochastic universal sampling (Baker 1987).

Because of its wide use and popularity among all GAs flavors, we describe, in the following, roulette wheel selection. For this procedure, each individual is assigned a probability of being selected proportional with its fitness value. The sum of these probability values over the set of all the individuals in a generation is 1. Let $f_i$ be the fitness of the $i$th individual of the current population, then $p_i$ is the probability of the individual for being selected:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j},$$

where $N$ is the number of individuals in the population (see, for a simple example, Fig. 3 which assumes a population of 5 individuals). On each application of the selection scheme, a random number is generated $r \in [0, 1)$ and the individual $i$ with the highest cumulative frequency smaller than this random $r$ is selected to survive to the next generation:

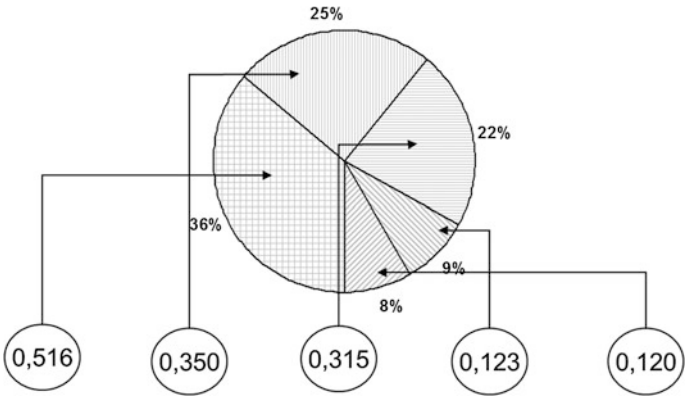


**Fig. 3** Fitness values in a population of 5 individuals. The *bottom row* contains the fitness values of the individuals. Their associated probabilities are the labels of the circular sectors

$$i = k = \overset{\min}{1}\ldots n\{k| \sum_{j=1}^{k} \geq r\}.$$

Ordinal-based selection takes into account only the relative order of individuals according to their fitness values. The most used procedures of this kind are the linear ranking selection (Baker 1985) and the tournament selection (Goldberg 1989).

New individuals are created in population with the aid of two genetic operators: crossover and mutation. The classical **crossover** operator aims at exchanging genetic material between two chromosomes in two steps: A locus is chosen randomly to play the role of a cut point and splits each of the two chromosomes in two segments; then, two new chromosomes are generated by merging the first segment from the first chromosome with the second segment from the second chromosome and vice versa. This operator is called in literature one-point crossover and is illustrated in Fig. 4. Generalizations exist to three or more cut points. Uniform crossover builds sequentially the offspring by copying at each locus the allele randomly chosen from one of the two parents.

Various constraints imposed by real-world problems led to various encodings for candidate solutions; these problem-specific encodings subsequently necessitate the redefinition of crossover. Thus, algebraic operators are implied for the case of numerical optimization with real encoding; an impressive number of papers focused on permutation-based encodings proposing various operators and performing comparative studies. It is now a common procedure to wrap a problem-specific heuristic within the crossover operator in Ionita et al. (2006), the authors propose new operators for constraint satisfaction; (Luchian et al. 1994) presents new operators in the context of clustering]. Crossover in GAs stands at the moment for any procedure which combines the information encoded within two or several chromosomes to create new and hopefully better individuals.

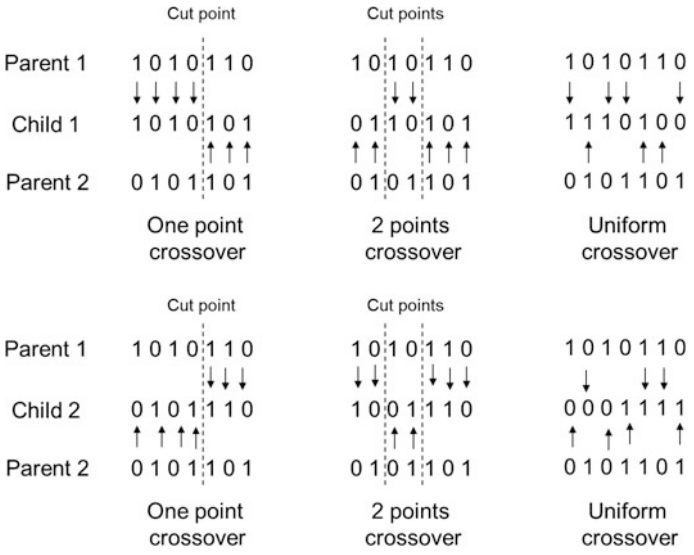


**Fig. 4** Crossover operators in bit string GA

**Mutation** is a unary operator designed to introduce variability in population. In the case of binary GAs, the mutation operator modifies each gene (from 0 to 1 or from 1 to 0) with a given probability. As in the case of crossover, mutation takes various forms depending on the problem and the encoding used (see Fig. 5 for examples of how mutation works for different chromosome representations).

When designing a GA, decisions have to be made with regard to several **parameters**: population size, crossover and mutation rate, and a halting criterion. Except some general considerations (i.e., high mutation rate in first iterations, decreasing during the run, combined with a complementary evolution for crossover), finding the optimum parameter values comes more to empiricism than to abstract studies.

In the following, we illustrate the search process conducted by a GA using the package called "GA" (Scrucca 2013) in R to minimize the Six Hump Camel Back function, previously defined in Sect. 1.4.

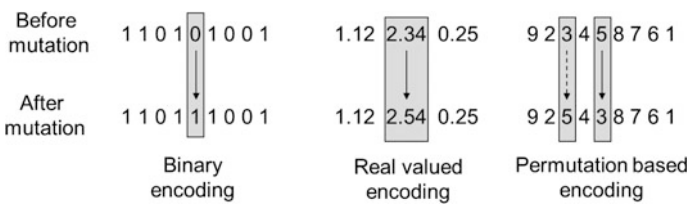


**Fig. 5** The behavior of the mutation operator for different encodings

Because this is a problem with a continuous bi-dimensional search space, a real encoding and arithmetical operators are a natural choice. Moreover, empirical studies have reported that these settings obtain better performance compared to natural encoding and standard operators in the case of numerical optimization problems. The initialization scheme consists in randomly generating points (candidate solutions, chromosomes) in the bi-dimensional search space defined by the problem. We have to define further the fitness function that should be used to measure the quality of the chromosomes. Naturally, this is based on the objective function of our problem, but requires some minimal modifications: the GAs necessitate that the fitness function is designed for maximization: The higher the fitness value is, the better the candidate solution for our problem is. Because the problem we tackle is defined for minimization, low values of our objective function (previously defined in R as *SixHumpV*) correspond to better solutions, while high values to worse ones. Therefore, we need to build a new function playing as fitness in the GA, simply by multiplying our objective function with (−1):

```
SixHumpMax <- function(x)
+ {
+ -SixHumpV(x)
+ }
```

The lines of code below call the **ga** function to execute a GA which maximizes our newly defined function with a population of 20 chromosomes using real encoding and arithmetic operators for 50 iterations:

```
> library("GA")
> GA.sols <- ga(type = "real-valued", fitness = SixHumpMax,
+  min = c(-3, -2), max = c(3, 2), maxiter=50, popSize=20)
Iter = 1  | Mean = -20.10513  | Best = 0.3900806
Iter = 2  | Mean = -8.679598  | Best = 0.3900806
Iter = 3  | Mean = -1.909435  | Best = 0.3900806
Iter = 4  | Mean = -0.7739577  | Best = 0.521566
Iter = 5  | Mean = -0.4207289  | Best = 0.521566
...
Iter = 50  | Mean = 0.9275536  | Best = 1.020383
```

During its execution, the **ga** function prints at each iteration the mean of the fitness in population and the best fitness value. To show the final results, we call the **summary** function:

```
> summary(GA.sols)
+-----------------------------------+
|         Genetic Algorithm         |
+-----------------------------------+

GA settings:
Type                  =  real-valued
Population size       =  20
Number of generations =  50
Elitism               =  1
Crossover probability =  0.8
Mutation probability  =  0.1
Search domain
    x1 x2
Min -3 -2
Max  3  2

GA results:

Iterations             = 50
Fitness function value = 1.020383
Solution               =
             x1        x2
[1,] -0.1262185 0.6870156
```

The best solution obtained over 50 iterations corresponds to Six Hump (−0.1262185, 0.6870156) = −1.020383. The evolution of the best value of the objective function in population during the run is illustrated in Fig. 6.

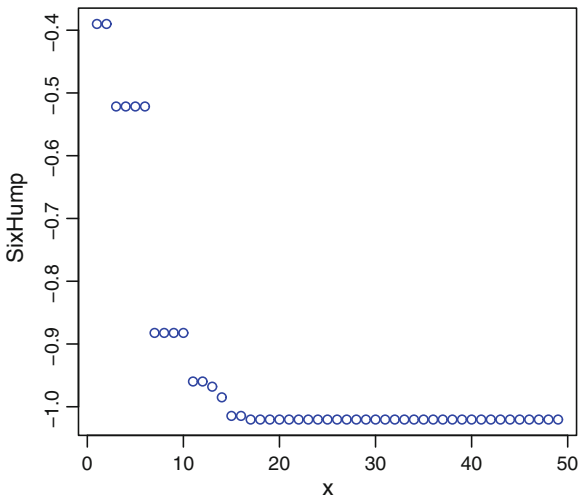


**Fig. 6** The evolution of the best objective value in one run of the GA

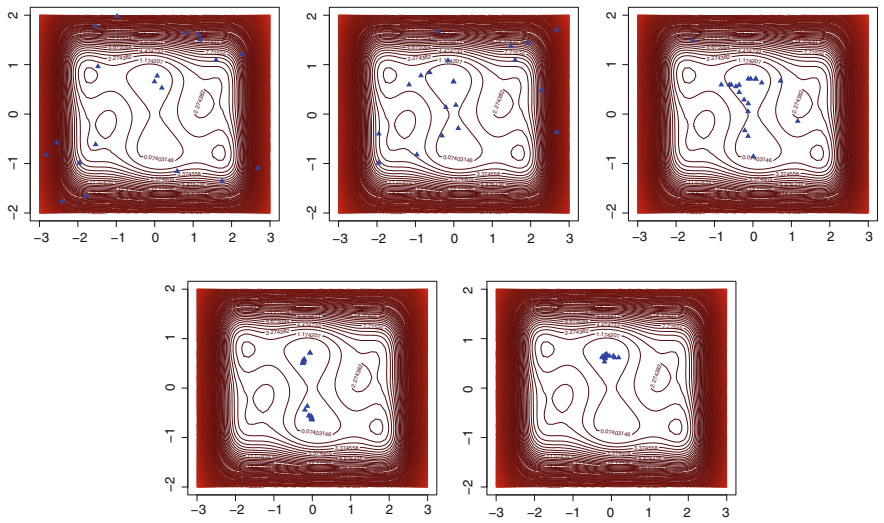

**Fig. 7** The evolution of the population in GA during one run of the algorithm: the distribution of the candidate solutions at iterations 1, 2, 5, 10, and 15

Figure 7 illustrates the distribution of the individuals in population during one run of the GA, at iterations 1, 2, 5, 10, and 50. The GA shows a very quick convergence toward the regions containing the global minima. The evolution of the fitness for the run illustrated here shows that the GA is able to locate in only a few number of iterations the promising area in the search space due to its good exploration abilities. However, by comparing the final solution to the minimum of the objective function (−1.020383 vs. −1.0316), we may conclude that in this run, the GA is deficient at exploitation: Even if very close to the global optima, starting at iteration number 17, the algorithm stopped improving the best solution achieved so far.

By illustrating only one run of the GA, a general conclusion on its convergence cannot be drawn on this basis due to the stochastic nature of the algorithm. To study its performance, 30 runs are performed with the same settings and for each run, the objective value corresponding to the solution returned is collected. In this manner, we obtain a sample of 30 values with mean −1.030361—which is closer to the optimum than the particular run reported previously, and standard deviation 0.0037—which indicates that the algorithm is stable, returning each time solutions very close to the optimum. The confidence interval for the mean supports these conclusions: The mean of the objective values returned by the GA is less than −1.028960 (we are interested in the minimum) with probability 0.95. In the code below, "fitness" is a vector with 30 values corresponding to the objective values returned in 30 runs:

```
 > t.test(fitness)

        One Sample t-test

data:  fitness
t = -1503.688, df = 29, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -1.031762 -1.028960
sample estimates:
mean of x
-1.030361
```

Although the reported results are satisfactory, the GAs are usually enhanced in practice by hybridizing them with local search algorithms.

With a standard binary encoding, GAs are the most appropriate candidates when attempting to solve the portfolio selection problem by means of meta-heuristics.

In order to illustrate such an approach, we consider the problem defined in Sect. 1.1 with the following instantiation: the number of projects $n = 6$, the budget of the firm $b = 1000$, and the costs and the utilities of the projects as in Table 1. An optimal solution to this problem involves the selection of projects 1, 4, 5, and 6; it has total cost 850 and utility 1700.

One way to deal, within a GA, with the constraints imposed by the problem, is to encourage the search in the feasible region of the search space by penalizing the unfeasible candidate solutions. Under this approach, any solution that violates a constraint gets a lower fitness. Identifying the most appropriate scheme to penalize solutions is, by itself, an optimization problem. The code below implements one possible fitness function for our problem:

```
> portfolio <- function(x){
+  cost <- c(250,350,100,200,300,100)
+  utility <- c(500,400,150,300,600,300)
+  totalUtility <- sum (utility*x)
+  totalCost <- sum (cost*x)
+  penalty <- 0
+  if (totalCost > 1000)
+    penalty <- totalCost #penalty for exceeding the budget
+  p=sum(cost)
+  if (x[1]+x[2] > 1) penalty <- penalty+p #violating constraint 5)
+  if (x[3]+x[5] > 1) penalty <- penalty+p #violating constraint 6)
+  if (x[3]+x[4]+x[5] > 2) penalty <- penalty+p #violating constraint 7)
+  totalUtility - penalty
+ }
```

**Table 1** Cost and utility of projects

| Project | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Cost | 250 | 350 | 100 | 200 | 300 | 100 |
| Utility | 500 | 400 | 150 | 300 | 600 | 300 |

A GA with binary encoding is called to solve this problem instance:

```
> GA <- ga(type = "binary", fitness = portfolio, nBits = 6,
+  maxiter = 50, popSize = 10)
Iter = 1  | Mean = 270  | Best = 1300
Iter = 2  | Mean = 850  | Best = 1400
Iter = 3  | Mean = 1225  | Best = 1700
Iter = 4  | Mean = 1160  | Best = 1700
I...
Iter = 49  | Mean = 832.5  | Best = 1700

> summary(GA)
+-----------------------------------+
|         Genetic Algorithm         |
+-----------------------------------+

GA settings:
Type                  =  binary
Population size       =  20
Number of generations =  50
Elitism               =  1
Crossover probability =  0.8
Mutation probability  =  0.1

GA results:
Iterations             = 50
Fitness function value = 1700
Solution               =
     x1 x2 x3 x4 x5 x6
[1,]  1  0  0  1  1  1
```

## 2.4 *Differential Evolution*

Adhering by design to the area of evolutionary algorithms, but targeting in particular the field of numerical optimization, a method called DE was developed by Ken Price and Rainer Storn during 1994–1996 (Storn and Price 1997). The results

**Fig. 8** The differential evolution algorithm

1. $t := 0$
2. Initialize population $P_0 = \{x_1^{(0)}, x_2^{(0)}, ..., x_m^{(0)}\}$ of size $m$
3. Evaluate $P_0$
4. **while** *halting condition not met* **do**
5. $t := t + 1$
6. **for** i=1 **to** m **do**
7. $y_i$ = generateMutant($P_{t-1}$)
8. $z_i = \text{crossover}(x_i^{(t-1)}, y_i)$
9. Evaluate $z_i$
10. **if** $z_i$ is better than $x_i^{(t-1)}$ **then**
11. $x_i^{(t)} = z_i$
12. **else**
13. $x_i^{(t)} = x_i^{(t-1)}$
14. **end if**
15. **end for**
16. **end while**

in their seminal paper show that DE outperforms GAs in numerical optimization and this hypothesis was subsequently confirmed in competitions dedicated to real-valued function minimization.

DE makes use of the same terminology as GAs: A population of candidate solutions evolves by means of selection, mutation, and crossover. The differences occur at several levels: the encoding of the candidate solutions, the definition of the genetic operators, and the selection scheme.

Designed for numerical optimization, the internal **encoding** of the candidate solution (the genotype) is identical to the phenotype: A string of real values that correspond to the decision variables defined by the problem.

The **selection for variation** is replaced in DE by a simple pass through the entire population: Each chromosome is participating in the variation phase to create a new offspring by means of genetic operators. However, DE implements selection at replacement: The offspring is introduced in the new population only if it is better than its parent with regard to the fitness function. The pseudo-code of the DE algorithm is illustrated in Fig. 8.

There are several versions of the **mutation** operator (line 7 of the algorithm). However, they all share a mechanism that is a distinctive feature of DE within the EA framework: The perturbation term is obtained as the difference between some randomly selected chromosomes. This perturbation mechanism, particular to DE, suggestively gives the name of this method. The general formula creating one mutant $y_i$ at time $t$ is given below:

$$y_i = \lambda x_*^{(t-1)} + (1-\lambda) x_{I_i}^{(t-1)} + \sum_{l=1}^{L} F_l (x_{J_{il}}^{(t-1)} - x_{K_{il}}^{(t-1)}) \tag{8}$$

where $\lambda$ is a numerical value in range [0,1] controlling the influence of the best element in the current population, which is $x_*^{(t-1)}$. $x_{I_i}^{(t-1)}$ is a chromosome from the

current population, chosen at random ($I_i \in \{1, 2, \ldots m\}$). $L >= 1$ is an integer value specifying the number of pairs of chromosomes of the form $(x_{J_{il}}^{(t-1)}, x_{K_{il}}^{(t-1)})$ randomly chosen from the current population ($J_{il}, K_{il} \in \{1, 2, \ldots m\}$, $J_{il} \neq K_{il}$) and which are used in the perturbation mechanism. $F_l > 0, l = \overline{1 \ldots m}$ are scaling factors decisive for the influence of each difference.

Different settings of the numerical parameters $\lambda$ and $L$ lead to distinct DE algorithms. In order to specify, in a concise manner, the DE variant, a simple notation, was introduced based on three variables: DE/$a/L/c$ where a depends on the value of $\lambda$, $L$ is the number of vector differences used, and $c$ is the type of crossover. The most popular versions of the DE algorithm are DE/best/$1/*$ and DE/rand/$1/*$. Both versions correspond to the case when only one difference is used to compute the mutant. The first case corresponds to $\lambda = 1$, respectively, to

$$y_i = x_*^{(t-1)} + F(x_{J_i}^{(t-1)} - x_{K_i}^{(t-1)}) \tag{9}$$

while the second case corresponds to $\lambda = 0$, respectively, to

$$y_i = x_{I_i}^{(t-1)} + F(x_{J_i}^{(t-1)} - x_{K_i}^{(t-1)}) \tag{10}$$

It must be noted that the mutation mechanism described above does not alter the current/selected chromosome $x_i$. It is the role of crossover to build an offspring of the current chromosome, by combining its genetic material with the one encoded by the mutant chromosome. From this perspective, DE is not entirely compliant with the general specifications of the two genetic operators.

Two versions of **crossover** are proposed in DE. A first one, called binomial crossover, is similar to the uniform crossover in GAs: It is a binary operator that mixes the components of the two chromosomes based on a given probability CR:

$$z_{i,d} = \begin{cases} y_{i,d} & \text{if } r_d < \text{CR or } d = d_0 \\ x_{i,d} & \text{otherwise} \end{cases} \quad d = \overline{1 \ldots D} \tag{11}$$

where $r_d$ is a random number uniformly distributed in [0,1] and $d_0 \in [1, D]$ is a random position in the chromosome guaranteeing that the offspring contains at least one element from the mutant. $D$ denotes the dimensionality of the problem, i.e., the length of the string representing a chromosome.

The second variant of the crossover operator is called exponential crossover and can be expressed by the following formulation:

$$z_{i,d} = \begin{cases} y_{i,d} & \text{for } d \in \Theta \\ x_{i,d} & \text{otherwise} \end{cases} \tag{12}$$

where $\Theta$ is a series of size at most $D$ of consecutive circular numbers in range 1,2, … $D$, starting with a value $d_0$ and continuing with $(d_0 + 1) \div D$, $(d_0 + 2) \div D, \ldots,$ $(d_0 + k) \div D$ where $a \div b$ expresses the modulus operator returning the remainder

of the division of $a$ to $b$; $k$ is the first trial that satisfies that a random uniformly generated number in [0,1] is higher than CR, thus following a truncated geometric distribution. For example, considering $d_0 = 6$ and $D = 10$, $\Theta$ could be the series 6, 7, 8 or 6, 7, 8, 9, 10, 1, 2, depending on the parameter CR; these two examples clearly illustrate the similarity of the exponential crossover in DE with the 2-point crossover in GAs.

In both versions of the crossover operator, CR is a parameter deciding the influence of the mutant on the structure of the offspring. A theoretical analysis of the two crossover variants and their influence on the sensitivity of DE to different values of CR are presented in Zaharie (2007).

An **elitist replacement strategy** guarantees survival of the fittest chromosome among the parent and the offspring.

To simulate a run of the DE algorithm on our minimization problem, we use the R package called DEoptim (Mullen et al. 2011).[1] The following code calls the DEoptim function which executes the DE/rand/1/bin algorithm (the variant implementing mutation based on a random candidate and one difference, and binary crossover) to minimize the SixHump function with a population consisting of 20 candidate solutions over 50 iterations; with the trace parameter set on TRUE, the best candidate solution (its value for the objective function and its components) in each iteration is shown during the run:

```
> library("DEoptim")
> DE.sols <- DEoptim(SixHumpV, lower = c(-3, -2), upper = c(3, 2),
+  control = list(strategy = 1, NP=20, itermax=50, storepopfrom = 1,
+  trace = TRUE))
Iteration: 1 bestvalit: -0.343676 bestmemit:    0.424858   -0.515384
Iteration: 2 bestvalit: -0.343676 bestmemit:    0.424858   -0.515384
Iteration: 3 bestvalit: -0.343676 bestmemit:    0.424858   -0.515384
Iteration: 4 bestvalit: -0.722848 bestmemit:   -0.090842    0.885970
Iteration: 5 bestvalit: -0.811161 bestmemit:    0.138414    0.742059
...
```

The performance of DE is highly dependent on the values of the numerical parameters. The authors of DE recommend setting CR to 0.9 and selecting $F$ from the interval [0.5, 1.0]. The run illustrated here uses the default values in DEoptim: CR = 0.9 and $F = 0.8$.

The following lines of code list the best solution in the last iteration and output two plots: One representing the evolution of the best value of the objective function (the minimum) in the population and one representing the distribution of the candidate solutions during the run. The resulting plots are illustrated in Fig. 9.

[1]The package can be freely downloaded from http://cran.r-project.org/web/packages/DEoptim/index.html.

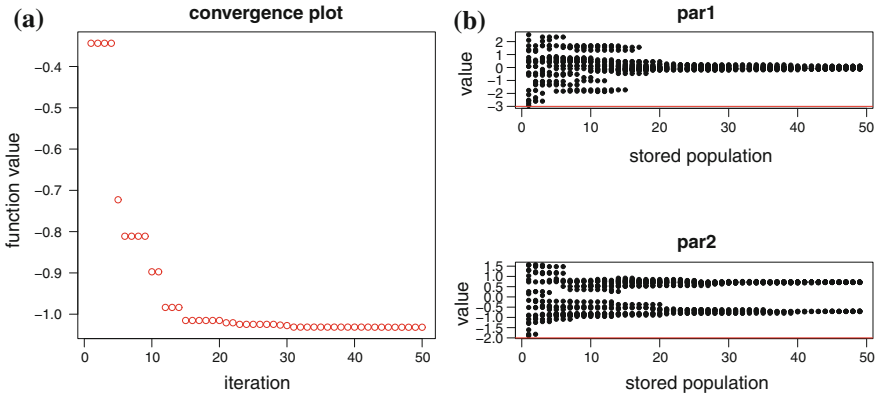


**Fig. 9** The evolution of the population in DE during one run of the algorithm: **a** the evolution of the best fitness value in population and **b** the distribution of the candidate solutions (the genotype)

```
> DE.sols$optim
$bestmem
       par1        par2
 0.08984226 -0.71265649

$bestval
[1] -1.031628
...
> plot(DE.sols, plot.type = "bestvalit", col="red", pch=1)
> plot(DE.sols, plot.type = "storepop")
```

Figure 9 clearly illustrates the convergence toward the optimal solution in DE. In our run, the optimum is found after 31 iterations, as indicated by Fig. 9a. The diversity in population decreases significantly during the run according to Fig. 9b which presents in two distinct plots the distribution of the values in each iteration for each parameter of the objective function. This plot indicates an interesting behavior: convergence toward two distinct regions in the search space.

In order to get more insight into the dynamics of the population within DE, Fig. 10 illustrates the candidate solutions in the population at distinct moments during the run distributed over the contour plot illustrating the landscape of the objective function. The series (a) of plots show the distribution of the candidate solutions at iterations 1, 5, 10, and 15. The series (b) offers a zoomed-in perspective of the landscape (restricted to $x_1 \in [-1.9, 1.9]$ and $x_2 \in [-1.1, 1.1]$) showing the distribution of the candidate solutions at iterations 15, 20, 30, and 50. In the first iteration of the algorithm, the population is spread at random in the search space. At iteration number 10 (Fig. 10a)-3rd plot), groups of individuals were formed around local and global optima. Toward the end of our run, all the candidate solutions migrate in the regions corresponding to the two global optima.

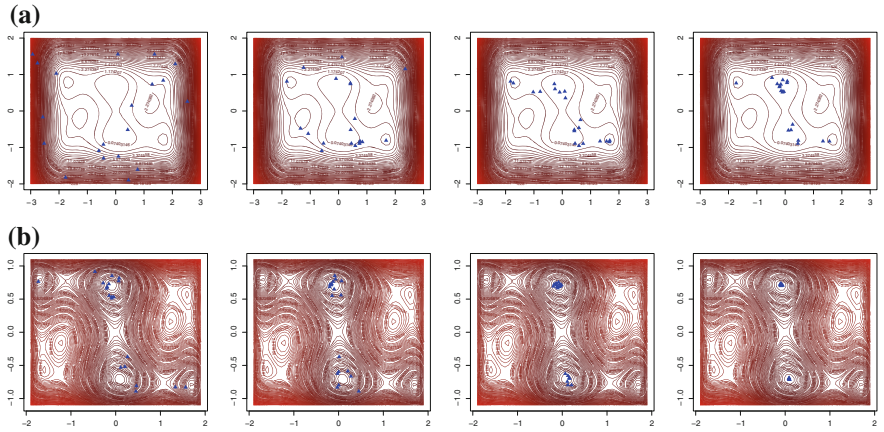


**Fig. 10** The evolution of the population in DE during one run of the algorithm: **a** the distribution of the candidate solutions at iterations 1, 5, 10, and 15 and **b** a zoomed-in landscape showing the distribution of the candidate solutions at iterations 15, 20, 30, and 50

The mean of the objective values after 30 runs is −1.031615, with a standard deviation of 3.74e−05.

## 2.5 *Extensions of EAs for Multi-modal and Multi-objective Problems*

Variations were brought to the classical EAs not only at the encoding and operators level. In order to face the challenges imposed by real-world problems, modifications are also recorded in the general scheme of the algorithms.

EAs are generally preferred to trajectory-based meta-heuristics (i.e., hill climbing, simulated annealing, Tabu Search) in **multi-modal environments**, mostly due to their increased exploration capabilities. However, a standard EA still can be trapped in a local optimum due to premature attraction of the entire population into its basin of attraction. Therefore, the main concern of EAs for multi-modal optimization is to maintain diversity for a longer time in order to detect multiple (local) optima. To discover the global optima, the EA must be able to intensify the search in several promising regions and eventually encourage simultaneous convergence toward several local optima. This strategy is called *niching*: The algorithm forces the population to preserve subpopulations, each subpopulation corresponding to a niche in the search space, and different niches represent different (local) optimal regions.

Several strategies exist in the literature to introduce niching capabilities into evolutionary algorithms. Deb and Goldberg (1989) propose *fitness sharing*: The

fitness of each individual is modified by taking into account the number and fitness of its closely ranged individuals. This strategy determine the number of individuals in the attraction basin of an optimum to be dependent on the height of that peak.

Another widely used strategy is to arrange the candidate solutions into groups of individuals that can only interact between themselves. The *island model* evolves independently several populations of candidate solutions; after a number of generations, individuals in neighboring populations migrates between the islands (Whitley et al. 1998).

There are techniques, which divide the population, based on the distances between individuals (the so-called radii-based multi-modal search GAs). Genetic chromodynamics (Dumitrescu 2000) introduces a set of restrictions with regard to the way selection is applied or the way recombination takes place. A merging operator is introduced which merges very similar individuals after perturbation takes place. In Stoean et al. (2010), best successive local individuals are conserved, while sub-populations are topological separated.

De Jong introduced a new scheme of inserting the descendants into the population, called the *crowding* method (Kenneth 1975). To preserve diversity, the offspring replace only similar individuals in the population.

A field of intensive research within the evolutionary computation (EC) community is **multi-objective optimization**. Most real-world problems necessitate the optimization of several, often conflicting objectives. Population-based optimization methods offer an elegant and very efficient approach to this kind of problems: With small modifications of the basic algorithmic scheme, they are able to offer an approximation of the Pareto optimal solution set. While moving from one Pareto solution to another, there is always a certain amount of sacrifice in one objective(s) to achieve a certain amount of gain in the other(s). Pareto optimal solution sets are often preferred to single solutions in practice, because the trade-off between objectives can be analyzed and optimal decisions can be made on the specific problem instance.

Zitzler et al. (2000) formulate three goals to be achieved by multi-objective search algorithms:

- the Pareto solution set should be as close as possible to the true Pareto front,
- the Pareto solution set should be uniformly distributed and diverse over of the Pareto front in order to provide the decision maker a true picture of trade-offs,
- the set of solutions should capture the whole spectrum of the Pareto front. This requires investigating solutions at the extreme ends of the objective function space.

GAs have been the most popular heuristic approach to multi-objective design and optimization problems mostly because of their ability to simultaneously search different regions of a solution space and find a diverse set of solutions. The

crossover operator may exploit structures of good solutions with respect to different objectives to create new non-dominated solutions in unexplored parts of the Pareto front. In addition, most multi-objective GAs do not require the user to prioritize, scale, or weigh objectives. There are many variations of multi-objective GAs in the literature and several comparative studies. As in multi-modal environments, the main concern in multi-objective GAs optimization is to maintain diversity throughout the search in order to cover the whole Pareto front. Konak et al. (2006) provide a survey on the most known multi-objective GAs, describing common techniques used in multi-objective GA to attain the three above-mentioned goals.

## 3 Swarm Intelligence

Swarm intelligence (SI) is a computational paradigm inspired from the collective behavior in auto-organized decentralized systems. It stipulates that problem solving can emerge at the level of a collection of agents which are not aware of the problem itself, but collective interactions lead to the solution. SI systems are typically made up of a population of simple autonomous agents interacting locally with one another and with their environment. Although there is no centralized control, the local interactions between agents lead to the emergence of global behavior. Examples of systems like this can be found in nature, including ant colonies, bird flocking, animal herding, bacteria molding, and fish schooling.

The most successful SI techniques are ant colony optimization (ACO) and particle swarm optimization (PSO). In ACO (Dorigo and Stützle 2004), artificial ants build solutions walking in the graph of the problem and (simulating real ants) leaving artificial pheromone so that other ants will be able to build better solutions. ACO was successfully applied to an impressive number of optimization problems. PSO is an optimization method initially designed for continuous optimization; however, it was further adapted to solve various combinatorial problems. PSO is presented in more detail in the next section.

### *3.1 Particle Swarm Optimization*

The PSO model was introduced in 1995 by Kennedy and Eberhart (1995), being discovered through simulation of a simplified social model such as fish schooling or bird flocking. It was originally conceived as a method for optimization of continuous nonlinear functions. Latter studies showed that PSO can be successfully adapted to solve combinatorial problems.

The evolutionary cultural model proposed by (Boyd and Richerson 1985) stands as the basic principle of PSO. According to this model, individuals of a society have two learning sources: individual learning and cultural transmission. Individual learning is efficient only in homogenous environments: The patterns acquired

through local interactions with the environment are generally applicable. For heterogenous environments, social learning—the essential feature of cultural transmission—is necessary.

In line with the evolutionary cultural model, the PSO algorithm uses a set of simple agents which collaborate in order to find solutions of a given optimization problem.

In the PSO paradigm, the environment corresponds to the search space of the optimization problem to be solved. A swarm of particles is placed in this environment. The location of each particle corresponds therefore to a candidate solution to the problem. A fitness function is formulated in accordance with the optimization criterion to measure the quality of each location. The particles move in their environment collecting information on the quality of the solutions they visit and share this information to the neighboring particles in the swarm. Each particle is endowed with memory to store the information gathered by individual interactions with the environment, simulating thus *individual learning*. The information acquired from neighboring particles corresponds to the *social learning* component. Eventually, the swarm is likely to move toward "more" optimum locations of the search space, similar to a flock of birds that collectively forage for food.

Unlike GAs, in PSO, there exist no evolution operators and no competition for survival; all particles survive and share information for the welfare of the swarm. The driving force is the emergent SI and attained by the sharing of local information between particles in order to produce global knowledge. It is important to note that problem solving is a population-wide phenomenon, because a particle by itself is probably incapable of solving even simple problems (Poli et al. 2007).

Usually, the swarm is composed of particles that share the same structural and behavioral features. Each particle is characterized by its current position in the search space, its velocity, and one or more of its best positions in the past (usually, only one position). Each particle uses the objective (fitness) function so that it can find out how good its current status is. The particles use a communication channel in order to exchange information with (some) of its peers. The topology of the swarm's social network is defined by the structure of the communication channel, where cliques of interconnected particles form neighborhoods.

In the classical PSO algorithm, the position of a particle in the search space is updated in each iteration depending on the position and velocity of the particle in the previous iteration. The formulas used to update the particles and the procedures are inspired from and conceived for continuous spaces. Therefore, each particle is represented by a vector $x$ of length $n$ indicating the position in the $n$-dimensional search space and has a velocity vector $v$ used to update the current position. The velocity vector is computed following the rules:

- every particle tends to keep its current direction (an inertia term);
- every particle is attracted to the best position $p$ it has achieved so far (implements the individual learning component);
- every particle is attracted to the best particle $g$ in the neighborhood (implements the social learning component).

The velocity vector is computed as a weighted sum of the three terms above. Two random multipliers $r_1, r_2$ are used to gain stochastic exploration capability, while $w, c_1, c_2$ are weights usually empirically determined. The formulae used to update each of the individuals in the population at iteration t + 1 are as follows:

$$v_i^t = w \cdot v_i^{t-1} + c_1 \cdot r_1 \cdot (p_i^{t-1} - x_i^{t-1}) + c_2 \cdot r_2 \cdot (g \cdot t^{t-1} - x_i^{t-1}) \tag{13a}$$

$$x_i^t = x_i^{t-1} + v_i^t \tag{13b}$$

As a side effect of these changes, the velocity of the particle could enter a divergence process, throwing the particle further, and further away form $p$. To prevent this behavior, Kennedy and Eberhart clamped the amplitude of the velocity to a maximum value, denoted by $v_{\max}$:

$$v_i^t = \min(v_{\max}, \max(-v_{\max}, v_i^t)). \tag{14}$$

Equation 13b generates a new position in the search space (corresponding to a candidate solution). It can be associated to some extent to the mutation operator in evolutionary programing. However, in PSO, this mutation is guided by the past experience of both the particle and other members of the swarm. In other words, "PSO performs mutation with a conscience" (Jong 2006). Considering the best visited solutions stored in the personal memory of each individual as additional members of the population, PSO implements a weak form of selection (Angeline 1998).

The shape of the search space is unknown; hence, there exists no known optimum combination of the two learning sources (i.e., individual learning and cultural transmission). The classical PSO algorithm compensates this lack of information with random values for learning factors $c_1 \cdot r_1$ and $c_2 \cdot r_2$, which change in each iteration in order to weigh differently the learning sources. The velocity change produced by each term depends on the distance between the compared positions (i.e., the particle will move faster if values are larger) and the random learning factors. This allows PSO to simulate, during a single run, various search strategies. The solution that the algorithm outputs at the end of the run is obtained from the information stored in the memory of each particle after the last iteration is completed.

The search for the optimal solution in PSO is described by the iterative procedure in Fig. 11. The fitness function is denoted by $f$ and is formulated for maximization.

Particle $p_i$ is chosen in the basic version of the algorithm to be the best position in the problem space visited by particle $i$. However, the best position is not always dependent only on the fitness function. Constraints can be applied in order to adapt PSO to various problems, without slowing down the convergence of the algorithm. In constrained nonlinear optimization, the particles store only feasible solutions and ignore the infeasible ones (Hu and Eberhart 2002). In multi-objective optimization, only the Pareto-dominant solutions are stored (Coello and Lechunga 2002; Hu and

**Fig. 11** Basic PSO

1. $t := 0$
2. Initialize $x_i^t, i = \overline{1..n}$
3. Initialize $v_i^t, i = \overline{1..n}$
4. Store personal best $p_i^t = x_i^t,\ i = \overline{1..n}$
5. Find neighborhood best $g_i^t = argmax_{y \in Nx_i^t}(f(y)),\ i = \overline{1..n}$
6. **while** *halting condition not met* **do**
7. $\quad t := t + 1$
8. $\quad$ Update $v_i^t,\ i = \overline{1..n}$ using equation 13a
9. $\quad$ Update $x_i^t,\ i = \overline{1..n}$ using equation 13b
10. $\quad$ Update personal best $p_i^t = argmax(f(p_i^{t-1}), f(x_i^t))$
11. $\quad$ Find neighborhood best $g_i^t = argmax_{y \in Nx_i^t}(f(y))$
12. **end while**

Eberhart 2002). In dynamic environments, particle $p$ is reset to the current position if a change in the environment is detected (Hu and Eberhart 2001).

The selection of particle $g_i$ is performed in two steps: neighborhood selection followed by particle selection. The size of the neighborhood has a great impact on the convergence of the algorithm. It is generally accepted that a large neighborhood speeds-up the convergence, while small neighborhoods prevent the algorithm from premature convergence. Various neighborhood topologies were investigated with regard to their impact on the performance of the algorithm (Kennedy 2002; Kennedy and Mendes 2003); however, as expected, there is no free lunch: Different topologies are appropriate to different problems.

A major problem investigated in the PSO literature is the **premature convergence** of the algorithm in multi-modal optimization. This problem has been addressed in several papers and solutions include addition of a queen particle (Clerc 1999), alternation of the neighborhood topology (Kennedy 1999), introduction of subpopulations (Lïvbjerg et al. 2001), giving the particles a physical extension (Krink et al. 2002), alternation between phases of attraction and repulsion (Riget and Vesterstrøm 2002), giving different temporary search goals to groups of particles (Al-kazemi and Mohan 2002), giving particles quantum behavior (Sun et al. 2004), and the use of specific swarm-inspired operators (Breaban and Luchian 2005).

Another crucial problem is **parameter control**. The values and choices for some of these parameters may have significant impact on the efficiency and reliability of the PSO. There are several papers that address this problem; in most of them, the values for parameters are established through repeated experiments but there also exist attempts to adjust them dynamically, using EC algorithms.

The role played by the inertia weight was compared to that of the temperature parameter in simulated annealing (Shi and Eberhart 1998). A large inertia weight facilitates a global search, while a small inertia weight facilitates a local search. The parameters $c_1$ and $c_2$ are called generically learning factors; because of their distinct roles, $c_1$ was named the cognitive parameter (it gives the magnitude of the information gathered by each individual) and $c_2$ the social parameter (it weights the cooperation between particles). Another parameter used in PSO is the maximum

velocity which determines the maximum change each particle can take during one iteration. This parameter is usually proportional with the search domain.

One run of the PSO algorithm can be illustrated using package pso built for R which is consistent with standard PSO, as described in Bratton and Kennedy (2007):

```
> library(pso)
> PSO.sols <- psoptim(rep(NA,2),SixHumpV,lower=c(-3,-2),upper=c(3,2),
   control=list( maxit=50, s=20, trace=1, REPORT=1))
S=20, K=3, p=0.1426, w0=0.7213, w1=0.7213, c.p=1.193, c.g=1.193
v.max=NA, d=7.211, vectorize=FALSE, hybrid=off
It 1: fitness=-0.3635
It 2: fitness=-0.8261
It 3: fitness=-0.8261
It 4: fitness=-0.8623
It 5: fitness=-0.9337
...
```

The final solution obtained in 50 iterations with a population of 20 individuals reaches the global optima:

```
> show(PSO.sols)
$par
[1]  0.09041749 -0.71296641

$value
[1] -1.031627
```

The algorithm reaches quickly the global optima, as shown in Fig. 12.

Figure 13 illustrates the distribution of the individuals in population during one run, at iterations 1, 2, 5, 10, 20, and 50.

## *3.2 PSO on Binary Domains*

Although PSO was conceived for continuous optimization, an effort was done to adapt the algorithm in order to be used for solving a wide range of combinatorial and binary optimization problems. A short discussion of the binary version of PSO is presented in this section, following the presentation from (Bautu 2010).

Kennedy and Eberhart (1997) introduced a first variant of binary PSO, combining the evolutionary cultural model with the reasoned action model. According to the latter, the action performed by an individual is the stochastic result of the intention to do that action. The strength of the intention results from the interaction of the personal attitude and the social attitude on the matter (Hale et al. 2002).

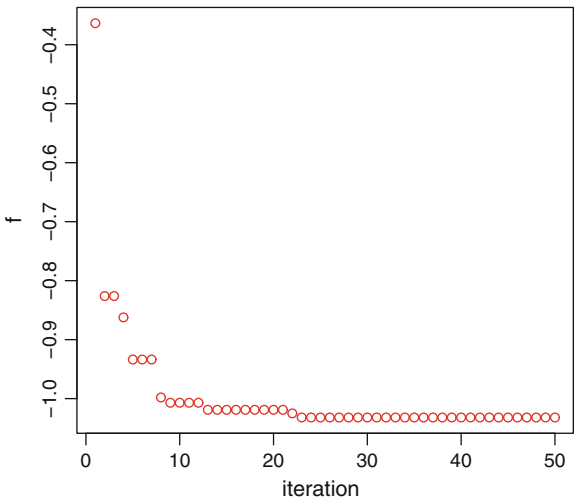


**Fig. 12** The evolution of the best value of the objective function for one run of PSO

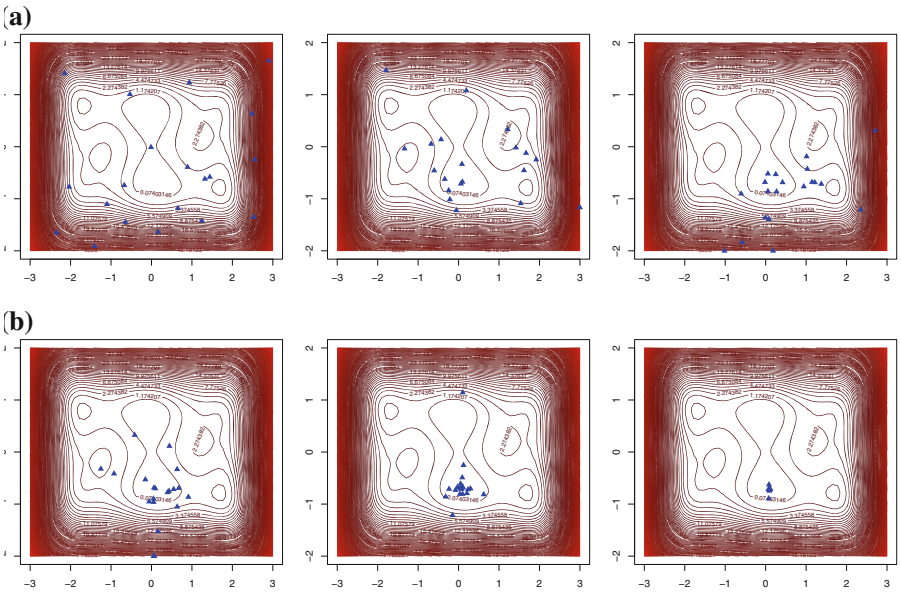


**Fig. 13** The evolution of the population in PSO during one run of the algorithm: the distribution of the candidate solutions at iterations 1, 2, 5, 10, 20, and 15

The PSO algorithm for real-valued optimization updates the positions of particles based on a function that depends (indirectly) of various personal and social factors. In the binary domain, the intention of particles to move between the two allowed positions: 0 and 1 is modeled in a similar manner. The *probability* that the particle will move to position 1 is computed by:

$$P(p^t = 1) = f'(p^{t-1}, v^{t-1}, p_i^{t-1}, g_g^{t-1}). \tag{15}$$

The individual learning factor and the social learning factor act as personal and social attitudes that help to select one of the two binary options.

In particular, with respect to classical PSO, in binary PSO:

- the domain of particle positions in the context of binary optimization problems is $P = \{0, 1\}^n$;
- the cost function that describes the optimization problem is hence defined $c : \{0, 1\}^n \rightarrow \mathbb{R}$;
- the position of a particle consists in the responses of the particle to the $n$ binary queries of the problem. The position in the search space is updated during each iteration depending on its velocity.

Let $p^t \in P$ and $v^t \in \mathbb{R}$ denote the position and the velocity of a particle at iteration $t$. The update equation for the particle's position in binary PSO is as follows:

$$p = \begin{cases} 1, & \text{if } \phi_3 < (1 + \exp(-v))^{-1} \\ 0, & \text{otherwise} \end{cases}, \tag{16}$$

where $\phi_3$ is a random uniformly distributed variable in $[0, 1)$. It results that higher velocity induces higher probabilities for the particle to choose 1. The equation for the particle ensures that the particle stays within the search space domain; hence, no relocation procedure is required.

The velocity of the particle is updated using the same equation as in classical PSO. The semantics of each term in (13a) for binary PSO are special cases of their original meaning. For example, if the best position of the particle ($p_{\mathrm{i}}^t$) is 1 and the current position ($p^t$) is 0, then $p_{\mathrm{i}}^t - p^t = 1$. In this case, the second term in (13a) will increase the value of $v^t$; hence, the probability that the particle with choose 1 will also increase. Similarly, the velocity will decrease if $p_{\mathrm{i}}^t = 0$ and $p^t = 1$. If the two positions are the same, the individual learning term will not change the velocity in order to try to maintain the current choice. The same is true for the velocity updates produced by the social learning term. The position of the particle may change due to the stochastic nature of (16), even if the velocity does not change between iterations.

The complete PSO algorithm for binary optimization problems is presented in vector form in (Fig. 14).

**Require:** $c$ - the objective function
**Ensure:** $S$ - the position that encodes the best solution
1. $t = 0$
2. Initialize particle positions ($p^t$)
3. Initialize particle velocities ($v^t$)
4. Store particle best solutions ($g_{\mathrm{i}}^t = p^t$)
5. **while** searching allowed **do**
6. $t = t + 1$
7. Update positions using equation (16)
8. Find neighborhood best solutions with neighborhood operator $N$ ($g_{\mathrm{g}}^t = \mathrm{argmin}_{x \in \{b_{\mathrm{i}}^t | N\}} c(x)$)
9. Update velocity using equation (13a)
10. Limit velocity using equation (14)
11. **end while**
12. Retrieve the solution.
13. **return** $S$

**Fig. 14** The particle swarm optimization algorithm for binary optimization

Other PSO variants can also be successfully used on binary spaces. In Wang et al. (2008), the authors propose the outcome of the binary queries to be established randomly based on the position of the particle within a continuous space. Khanesar et al. (2009) present a variation of the binary PSO in which the particle toggles its binary position with probability depending its velocity.

## 4 Integrating Meta-heuristics with Conventional Methods in Data Analysis: A Practical Example

Meta-heuristics stand as basis for the design of efficient algorithms for various data analysis tasks. Such approaches are extensions of conventional techniques, obtained as hybridizations with meta-heuristics, or evolved as new self-contained data analysis methods.

There is a large variety of approaches for data clustering based on GAs (Breaban et al. 2012; Hruschka et al. 2009; Luchian et al. 1994), DE (Zaharie 2005), PSO (Breaban and Luchian 2011; Rana et al. 2011), and ACO (Shelokar et al. 2004). Learning Classifier Systems (Lanzi et al. 2000) are one of the major families of techniques that apply EC to machine learning; these systems evolve a set of condition–action rules able to solve classification problems. Decision trees (Turney 1995) and support vector machines (Stoean et al. 2009, 2011) are also evolved with GAs. The representative application example of EAs in regression analysis is the use of genetic programing for symbolic regression, topic covered in detail in Chapter "Genetic Programming Techniques with Applications in the Oil and Gas Industry" of this book. Many algorithms based on meta-heuristics tackle feature selection and feature extraction.

We restrict the discussion in this section to one particular application: Optimization of the parameters of machine learning algorithms used in data analysis. The performance of several machine learning algorithms depends heavily on some parameters involved in their design; such parameters are often called meta-parameters or hyper-parameters. The problem of choosing the best settings for these parameters is also known as model selection.

Examples may vary from simple algorithms such as k-nearest neighbors where $k$ is such a hyper-parameter, to more complex algorithms. In the case of artificial neural networks, the structure of the network (the number of hidden layers, the number of neurons in each layer, the activation function) has a high impact on the accuracy of the results in classification or regression analysis; the degree of complexity of the network is a critical factor in the trade-off between overfitting the model to the training data and underfitting, and the right balance can be achieved only with extensive experiments. In the definition of support vector machines (SVMs), two numerical parameters play important roles: a constant $C$ called regularization parameter and a constant $\varepsilon$ corresponding to the width of the $\epsilon$-insensitive zone, influence the number of support vectors used in the model, controlling the trade-off between two goals, fitting the training set well, and avoiding overfitting; parameters characterizing various kernel functions are also involved.

We illustrate here a simple model selection scheme by means of EAs for regression analysis. A small dataset called "rock," included in R, is used with this purpose. It consists of 48 rock samples from a petroleum reservoir characterized by the area of pores, total perimeter of pores, shape, and permeability.

```
> show(rock)
    area     peri     shape   perm
1   4990 2791.900 0.0903296    6.3
2   7002 3892.600 0.1486220    6.3
3   7558 3930.660 0.1833120    6.3
4   7352 3869.320 0.1170630    6.3
...
48  9718 1485.580 0.2004470  580.0
```

We illustrate regression analysis by training a support vector machine to learn a model able to predict permeability. The quality of the regression model is usually measured by the mean squared error, as defined below.

```
> MSE <- function(x,y)
+ {
+  mean((x-y)^2)
+ }
```

Support vector regression is implemented in R under package "e1071." The results obtained using radial kernel are shown below:

```
> library(e1071)
> svr <- svm(perm ~ area+peri+shape, data=rock,
+ type="eps-regression", kernel = "radial")
> predicted <-predict(svr,newdata=rock,type="response")
> MSE(predicted, rock$perm)
[1] 35316.21
> cor(predicted, rock$perm)
[1] 0.9040716
> plot(predicted, rock$perm)
```

The default settings of the three hyper-parameters used in the run above can be inspected next: **Cost** is the regularization parameter, **gamma** is a parameter of the kernel function, and epsilon is the size of the insensitive tube.

```
> summary(svr)
Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.3333333
    epsilon:  0.1
```

These numerical parameters can be optimized in order to minimize the prediction error measured by MSE. We formulate this task as a numerical optimization problem defined over three numerical parameters (cost, gamma, and epsilon), aiming to minimize the MSE of the predictions obtained with support vector regression under the given settings:

```
> trainingError <- function(params)
+ {
+     svr <- svm(perm ~ area+peri+shape, data=rock, type="eps-regression",
+       kernel = "radial", gamma=params[1], cost = params[2], epsilon = params[3])
+     predicted <-predict(svr,newdata=rock,type="response")
+     MSE(predicted, rock$perm)
+ }
```

Any of the meta-heuristics presented in this chapter can be used to tackle this minimization problem. We illustrate here the use of DE:

```
> DEparams <- DEoptim(trainingError, lower = c(0, 0, 0), upper = c(4, 4, 1),
+ control = list(strategy = 1,NP=20, itermax=20, trace = TRUE))
Iteration: 1 bestvalit: 5937.692186 bestmemit:    1.929174    2.872409    0.012022
Iteration: 2 bestvalit: 5630.575260 bestmemit:    3.110530    3.717773    0.166768
Iteration: 3 bestvalit: 3623.210268 bestmemit:    2.818071    3.682759    0.077892
...
Iteration: 20 bestvalit: 1473.135923 bestmemit:    3.983884    3.812688    0.046011
```

The solution obtained by DE is stored next in the vector **params** and is used to train a new SVM.

```
> params <- DEparams$optim$bestmem
> svr <- svm(perm ~ area+peri+shape, data=rock, scale = TRUE, type="eps-regression",
+ kernel = "radial", gamma=params[1], cost = params[2], epsilon = params[3])
> predicted <-predict(svr,newdata=rock,type="response")
> MSE(predicted, rock$perm)
[1] 1473.136
> cor(predicted, rock$perm)
[1] 0.9968882
```

Figure 15 illustrates the predicted values compared to real values for the case of SVR with default settings (a) and for the case of SVR with optimized hyper-parameters (b).

Nevertheless, the optimized model gives much better results with regard to the error of predictions, but is prone to overfitting: A single dataset was used both for training and testing; in this situation, the model is highly adapted to the dataset and may suffer from poor generalization power. We can avoid overfitting by using distinct sets for training and testing. The new function to be optimized should be formulated as shown below. Very similar with the previous version regarding its definition, this function is significantly different in behavior: It invokes a "training" dataset in the learning phase but computes the prediction error on a "testing" dataset:

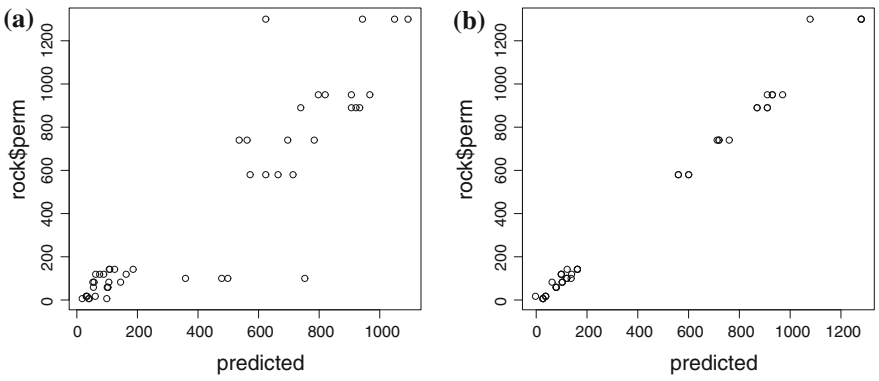


**Fig. 15** Predicted over expected values in regression analysis with SVR using: **a** default hyper-parameters settings and **b** optimized settings

```
> testingError <- function(params)
+ {
+     svr <- svm(perm ~ area+peri+shape, data=training, type="eps-regression",
+       kernel = "radial", gamma=params[1], cost = params[2], epsilon = params[3])
+     predicted <-predict(svr,newdata=testing,type="response")
+     MSE(predicted, rock$perm)
+ }
```

The validation of the regression model obtained with the optimized hyper-parameters requires in this case a third dataset called validation set. This phase closes the analysis which, as recommended in the case of any supervised learning task, is composed of three phases: training, testing, and validation. If the accuracy/error obtained in the validation phase is satisfactory, the model can be used in production.

## 5 Applications of Meta-Heuristics in Geosciences

Evolutionary algorithms have been used in solving geophysics optimization problems in two main directions: either by performing the optimization, or by optimizing parameters of other methods (e.g., neural networks) used in specific problems.

Evolutionary methods are compared to PSO in a study on optimization of reservoir models to match past petroleum production data in Yasin Hajizadeh et al. (2011). ACO, DE, PSO, and the neighborhood algorithm are integrated in a Bayesian framework in order to measure the uncertainty of the predictions obtained by each algorithm, in a case study involving two petroleum reservoirs. Ahmadi et al. (2013) perform the task of predicting reservoir permeability using a soft sensor implemented on the basis of a feed-forward artificial neural network, which was then optimized using a hybrid GA and PSO method. History matching is also the research topic in Park et al. (2014). A multi-objective evolutionary algorithm identifies optimal solutions and outperforms a traditional weighted-sum approach.

GAs are acknowledged as important tools for successful neural network data-driven models with applications in the oil and gas industry (Mohaghegh 2005; Shahab et al. 2005). Intelligent software tools used in the industry integrate hard (statistical) and soft (intelligent) computing techniques, such as fuzzy cluster analysis, genetic optimization, or neural computing (Shahab et al. 2005).

Direct use of a GA helps to evaluate hydrocarbon resource in a field dataset from North Cambay basin, India (Thander et al. 2014). Several parameters are required for resource estimation (e.g., areal extent, net pay thickness, oil saturation, etc.), yet a limited set is recorded in the exploration phase. Also, recordings are done with uncertainty, due to reservoir heterogeneity. GA copes well with the uncertainty in data and delivers estimations of the oil reserve using a real dataset.

An oil production planning problem that appears in the context of oil wells with insufficient oil pressure and which consists in identifying the amount of gas that should be injected in a well in order to maximize the amount of oil extracted from that well is solved by an evolutionary algorithm in Singh et al. (2013). The problem is more difficult since it is constrained by the total amount of gas available daily. The authors propose a multi-objective approach to the problem and also formulate a single objective version, focused on the maximization of profit, instead of the oil quantity. The problem of gas allocation among oil wells is also tackled in Ghaedi et al. (2013), by means of a hybrid GA, and in Abdel Rasoul et al. (2014). The problem of gas allocation among oil wells is also tackled in Ghaedi et al. (2013), by means of a hybrid GA, and in Abdel Rasoul et al. (2014).

The optimal well type and location are determined with PSO in (Onwunalu and Durlofsky 2010), in a study involving vertical, deviated, and dual-lateral wells. Comparisons with a GA over multiple runs of both algorithms show that PSO outperforms, on average, the GA, yet the advantages of using PSO over GA are varied among the cases surveyed. Driven by the goal of maximizing the total hydrocarbon recovery, an well placement problem is tackled in Nwankwor et al. (2013) with a hybrid PSO-DE algorithm is proposed for the problem. The hybrid is compared to basic variants of PSO and DE on three problem cases concerning the placement of vertical wells in 2D and 3D reservoir models. Optimal well placement under uncertainty is tackled in a two-stage approach in Lyons and Nasrabadi (2013). First, an ensemble Kalman filter is used to perform history matching on the reservoir data. Then, well placement is solved by a GA combined with pseudohistory matching.

Carbon dioxide ($CO_2$) sequestration is of great interest for oil engineers. In recent years, the idea of storing $CO_2$ in deep geological formations, such as depleted oil and gas reservoirs (with impermeable rocks), gained a lot of focus from the community as a solution for greenhouse gas mitigation by avoiding $CO_2$ from emission into the atmosphere. The $CO_2$ sequestration also helps by enhancing methods for oil or gas recovery (Zangeneh et al. 2013). Evolutionary algorithms are used in order to identify carbon dioxide seepage areas in Cortis et al. (2008). In Zangeneh et al. (2013), the parameters of a $CO_2$ storage model are optimized using a GA. A multi-objective GA (NSGA) is implemented for optimizing gas storage alongside oil recovery in Safarzadeh and Motahhari (2014). Based on the results from the GA, the authors are able to propose some production scenarios.

In (Fichter et al. 2000), a portfolio optimization problem for the oil and gas industry is tackled by means of a GA. GAs are chosen for this task both due to their scalability to extremely large portfolios and because they allow the analysis of portfolios from the point of view of value and risk measures.

GA and PSO are used to find the optimal parameters of a linear and an exponential model for the demand of oil in Iran in Assareh et al. (2010). The models use as input variables the population, the gross domestic product, import, and export data; they are used to forecast demand of oil up to 2030.

PSO emerged as a powerful algorithm for geophysical inverse problems when compared to GAs and simulated annealing in Martnez et al. (2010), Shaw, and

Srivastava (2007). Other applications include inversion of seismic refraction data Poormirzaee et al. (2014), crosshole traveltime tomography Tronicke et al. (2012), or reservoir characterization Fernández Martìnez et al. (2012).

A large number of meta-heuristics are compared with respect to training an artificial neural network for the task of forecasting the water temperature of a natural river in Piotrowski et al. (2014). The study involves a comparison of several versions of PSO, DE, direct search to the levenberg–Marquardt (LM) algorithm for ANN training. The study concludes that only the DE algorithm obtains results competitive to the LM algorithm. A similar optimization idea is described in Ahmadi and Ebadi (2014), where a hybrid combination between an artificial neural network and PSO, extended with dew point pressure data, leads to a better understanding of reservoir fluid behavior.

## References

Abdel Rasoul RR, Daoud A, El Tayeb ESA (2014) Production allocation in multi-layers gas producing wells using temperature measurements with the application of a genetic algorithm. Pet Sci Technol 32(3):363–370

Ahmadi MA, Ebadi M (2014) Robust intelligent tool for estimation dew point pressure in retrograded condensate gas reservoirs: application of particle swarm optimization. J Pet Sci Eng 123:7–19

Ahmadi MA, Zendehboudi S, Lohi A, Elkamel A, Chatzis I (2013) Reservoir permeability prediction by neural networks combined with hybrid genetic algorithm and particle swarm optimization. Geophys Prospect 61(3):582–598

Al-kazemi B, Mohan CK (2002) Multi-phase generalization of the particle swarm optimization algorithm. In: Proceedings of the IEEE congress on evolutionary computation. IEEE Press

Angeline PJ (1998) Using selection to improve particle swarm optimization. In: Proceedings of the IEEE international conference on evolutionary computation. IEEE Press, pp 84–89. ISBN 0-7803-4869-9

Assareh E, Behrang MA, Assari MR, Ghanbarzadeh A (2010) Application of PSO (particle swarm optimization) and GA (genetic algorithm) techniques on demand estimation of oil in iran. Energy 35(12):5223–5229

Back T (1996) Evolutionary algorithms in theory and practice. Oxford University Press, New York

Baker JD (1985) Adaptive selection methods for genetic algorithms. In: Proceedings of an International Conference on Genetic Algorithms and their applications. Hillsdale, New Jersey, pp 101–111

Baker JD (1987) Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the second international conference on genetic algorithms. pp 14–21

Bautu A (2010) Generalizations of Particle Swarm Optimization: applications of particle swarm algorithms to statistical physics and bioinformatics problems. PhD Thesis, Department of Computer Science, Al. I. Cuza University, Lambert Academic Publishing. ISBN 978-3848417315

Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv 35(3):268–308. ISSN 0360-0300. doi:http://doi.acm.org/10.1145/937503.937505

Boyd R, Richerson PJ (1985) Culture and the evolutionary process. The University of Chicago Press, Chicago

Bratton D, Kennedy J (2007) Defining a standard for particle swarm optimization. In: Swarm intelligence symposium, 2007. SIS 2007, IEEE, pp 120–127

Breaban M (2011) Clustering: evolutionary approaches. PhD Thesis, Department of Computer Science, Al. I. Cuza University

Breaban M, Luchian H (2005) PSO under an adaptive scheme. In: Proceedings of the IEEE congress on evolutionary computation. IEEE Press, pp 1212–1217

Breaban ME, Luchian H (2011) PSO aided k-means clustering: introducing connectivity in k-means. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation. ACM, pp 1227–1234

Breaban ME, Luchian H, Simovici D (2012) A genetic clustering algorithm by monomial projection pursuit. In Symbolic and numeric algorithms for scientific computing (SYNASC), 14th international symposium on 2012. IEEE, pp 214–219

Bremermann HJ (1958) The evolution of intelligence: the nervous system as a model of its environment. Technical Report No. 1, Department of Mathematics, University of Washington, Seattle

Burke EK, Gendreau M, Hyde M, Kendall G, Ochoa G, Özcan E, Qu R (2013) Hyper-heuristics: a survey of the state of the art. J Oper Res Soc 64(12):1695–1724

Clerc M (1999) The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In: Proceedings of the IEEE congress on evolutionary computation, vol 3, pp 1951–1957. doi:10.1109/CEC.1999.785513

Clerc M (2006) Particle swarm optimization. Hermes Sci, London. ISBN 1905209045

Coello CAC, Lechunga MS (2002) Mopso: a proposal for multiple objective particle swarm optimization. In Proceedings of the IEEE congress on evolutionary computation. IEEE Press, pp 1051–1056

Cortis A, Oldenburg CM, Benson SM (2008) The role of optimality in characterizing $CO_2$ seepage from geologic carbon sequestration sites. Int J Greenh Gas Control 2(4):640–652

De Jong KA (2006) Evolutionary computation. A unified approach. MIT Press, Cambridge

Deb K, Goldberg DE (1989) An investigation of niche and species formation in genetic function optimization. In: Proceedings of the 3rd international conference on genetic algorithms, San Francisco. Morgan Kaufmann Publishers Inc., pp 42–50, ISBN 1-55860-066-3. http://portal.acm.org/citation.cfm?id=645512.657099

Dorigo M, Stützle T (2004) Ant colony optimization. Bradford Company, Scituate. ISBN 0262042193

Dumitrescu D (2000) Genetic chromodynamics. Studia Universitatis Babes-Bolyai Cluj-Napoca, Ser. Informatica 45:39–50

Fernández Martnez JL, Mukerji T, Garca Gonzalo E, Suman A (2012) Reservoir characterization and inversion uncertainty via a family of particle swarm optimizers. Geophysics 77(1): M1–M16

Fichter DP et al (2000) Application of genetic algorithms in portfolio optimization for the oil and gas industry. In: SPE annual technical conference and exhibition. Society of Petroleum Engineers

Fogel LJ, Owens AJ, Walsh MJ (1966) Artifficial intelligence through simulated evolution. Wiley, New York

Fraser AS (1957) Simulations of genetic systems by automatic digital computers. Aust J Biol Sci 10:492–499

Ghaedi M, Ghotbi C, Aminshahidy B (2013) Optimization of gas allocation to a group of wells in gas lift in one of the iranian oil fields using an efficient hybrid genetic algorithm (HGA). Pet Sci Technol 31(9):949–959

Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13(5):533–549. ISSN 0305-0548. doi:10.1016/0305-0548(86)90048-1

Grefenstette JJ (1986) Optimization of control parameters for genetic algorithms. IEEE Trans Syst Man Cybern 16(1): 122–128

Grefenstette JJ (1987) Incorporating problem specific knowledge into genetic algorithms. Genet Algorithms Simul Annealing 4:42–60

Hajizadeh Y, Demyanov V, Mohamed L, Christie M (2011) Comparison of evolutionary and swarm intelligence methods for history matching and uncertainty quantification in petroleum

reservoir models. In: Intelligent computational optimization in engineering. Springer, Berlin, pp 209–240
Hale JL, Householder BJ, Greene KL (2002) The theory of reasoned action. Sage Publications, Thousand Oaks, pp 259–286
Hillis WD (1990) Co-evolving parasites improve simulated evolution as an optimization procedure. Phys D Nonlinear Phenom 42(1):228–234
Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
Holland JH (1998) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence. MIT Press, Cambridge. ISBN 0-262-58111
Hruschka ER, Campello RJGB., Freitas AA, De Carvalho APLF (2009) A survey of evolutionary algorithms for clustering. IEEE Trans Syst Man Cybern Part C Appl Rev 39(2):133–155
Hu X, Eberhart RC (2001) Tracking dynamic systems with PSO: where's the cheese? In Proceedings of the workshop on particle swarm optimization, pp 80–83
Hu X, Eberhart RC (2002) Multiobjective optimization using dynamic neighborhood particle swarm optimization. In: Proceedings of the IEEE congress on evolutionary computation. IEEE Press, pp 1677–1681
Hu X, Eberhart RC (2002) Solving constrained nonlinear optimization problems with particle swarm optimization. In: Proceedings of the sixth world multiconference on systemics, cybernetics and informatics
Ionita M, Croitoru C, Breaban M (2006) Incorporating inference into evolutionary algorithms for max-csp. In: 3rd international workshop on hybrid metaheuristics, LNCS 4030. Springer, Berlin, pp 139–149
Jong KD (2006) Evolutionary computation: a unified approach. MIT Press. ISBN 0-262-04194
Kennedy J (1999) Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In: Proceedings of the IEEE congress of evolutionary computation, vol 3. IEEE Press, pp 931–1938. doi:10.1109/CEC.1999.785513
Kennedy J (2002) Population structure and particle swarm performance. In: Proceedings of the congress on evolutionary computation (CEC 2002). IEEE Press, pp 1671–1676
Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceedings of the 1995 IEEE international conference on neural networks, vol 4. IEEE Press, pp 1942–1948
Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: Proceedings of IEEE international conference on neural networks, pp 1942–1948
Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In: Proceedings of the world multiconference on systemics, cybernetics and informatics, vol 5, Piscataway. IEEE Press, pp 4104–4109
Kennedy J, Mendes R (2003) Neighborhood topologies in fully-informed and best-of neighborhood particle swarms. In: Proceedings of the 2003 IEEE SMC workshop on soft computing in industrial applications (SMCia03). IEEE Computer Society, pp 45–50
Kenneth ADJ (1975) An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan, Dissertation Abstracts International, vol 36, no 10, Ann Arbor, AAI7609381
Khanesar MA, Tavakoli H, Teshnehlab M, Shoorehdeli MA (2009) Novel binary particle swarm optimization. In: Tech Education and Publishing, pp 1–10. ISBN 978-953-7619-48-0
Kirkpatrick S, Gelatt CD, Vecchi MP et al (1983) Optimization by simmulated annealing. Science 220(4598):671–680
Konak A, Coit DW, Smith AE (2006) Multi-objective optimization using genetic algorithms: a tutorial. Reliab Eng Syst Safety 910(9):992–1007. http://www.sciencedirect.com/science/article/B6V4T-4J0NY2F-2/2/97db869c46fc43f457f3d509adaa15b5
Koza J (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
Krink T, Vesterstrom JS, Riget J (2002) Particle swarm optimisation with spatial particle extension. In: Proceedings of the evolutionary computation on 2002. CEC'02. Proceedings of

the 2002 Congress—vol 02, CEC'02. IEEE Computer Society, Washington, pp 1474–1479. ISBN 0-7803-7282-4. http://portal.acm.org/citation.cfm?id=1251972.1252447
Lanzi PL, Stolzmann W, Wilson SW (2000) Learning classifier systems: from foundations to applications (No. 1813). Springer, Berlin
Lïvbjerg M, Rasmussen TK, Krink T (2001) Hybrid particle swarm optimiser with breeding and subpopulations. In: Proceedings of the genetic and evolutionary computation conference (GECCO-2001). Morgan Kaufmann, pp 469–476
Luchian S, Luchian H, Petriuc M (1994) Evolutionary automated classification. In: Proceedings of 1st congress on evolutionary computation, pp 585–588
Lyons J, Nasrabadi H (2013) Well placement optimization under time-dependent uncertainty using an ensemble kalman filter and a genetic algorithm. J Petrol Sci Eng 109:70–79
Martnez JLF, Gonzalo EG, Álvarez JPF, Kuzma HA, Pérez COM (2010) PSO: A powerful algorithm to solve geophysical inverse problems: Application to a 1D-DC resistivity case. J Appl Geophys 710(1):13–25
Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953) Equation of state calculations by fast computing machines. J Chem Phys 21(6):1087–1092
Michalewicz Z (1992) Genetic algorithms + data structures = evolution programs (3rd edn). Springer, Berlin. ISBN 3-540-60676-9
Mitchell M (1996) An introduction to genetic algorithms. MIT Press, Cambridge. ISBN 0-262-13316-4
Mitchell M, Forrest S, Holland JH (1992) The royal road for genetic algorithms: fitness landscapes and ga performance. In: Proceedings of the first European conference on artificial life, pp 245–254. The MIT Press, Cambridge
Mohaghegh SD (2005) A new methodology for the identification of best practices in the oil and gas industry, using intelligent systems. J Pet Sci Eng 49(3):239–260
Mohaghegh SD et al (2005) Recent developments in application of artificial intelligence in petroleum engineering. J Pet Technol 57(4):86–91
Mullen KM, Ardia D, Gil DL, Windover D, Cline J (2011) DEoptim: an R package for global optimization by differential evolution. J Stat Softw 40(6):1–26
Nateri K Madavan (2002) Multiobjective optimization using a pareto differential evolution approach. In: Proceedings of the world on congress on computational intelligence, vol 2. IEEE, pp 1145–1150
Nguyen NT, Kowalczyk R (2012) Transactions on computational collective intelligence III. Springer, Berlin
Nwankwor E, Nagar AK, Reid DC (2013) Hybrid differential evolution and particle swarm optimization for optimal well placement. Comput Geosci 17(2):249–268
Onwunalu JE, Durlofsky LJ (2010) Application of a particle swarm optimization algorithm for determining optimum well location and type. Comput Geosci 14(1):183–198
Park H-Y, Datta-Gupta A, King MJ (2014) Handling conflicting multiple objectives using pareto-based evolutionary algorithm during history matching of reservoir performance. J Pet Sci Eng
Piotrowski AP, Osuch M, Napiorkowski MJ, Rowinski PM, Napiorkowski JJ (2014) Comparing large number of metaheuristics for artificial neural networks training to predict water temperature in a natural river. Comput Geosci 64:136–151
Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization. Swarm Intell 1(1):33–57
Poli R, Langdon WB, McPhee NF (2008) A field guide to genetic programming. http://www.gp-field-guide.org.uk. (With contributions by JR Koza)
Poormirzaee R, Moghadam RH, Zarean A (2014) Inversion seismic refraction data using particle swarm optimization: a case study of Tabriz, Iran. Arab J Geosci 1–9
Radcliffe NJ, Surry PD, Jz E (1995) Fitness variance of formae and performance prediction. In: Foundations of genetic algorithms, pp 51–72
Raidl GR, Gottlieb J (2005) Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: a case study for the multidimensional knapsack problem. Evol Comput 13(4):441–475

Rana S, Jasola S, Kumar R (2011) A review on particle swarm optimization algorithms and their applications to data clustering. Artif Intell Rev 35(3):211–222
Rechenberg I (1973) Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution. In: Frommann-Holzboog
Rechenberg I (1973) Evolutionstrategie: optimierung Technisher Systeme nach Prinzipien der Biologischen Evolution. Frommann-Holzboog Verlag, Stuttgart
Riget J, Vesterstrøm JS (2002) A diversity-guided particle swarm optimizer-the ARPSO. Department of Computer Science, University of Aarhus, Aarhus, Denmark, Technical Report, vol 2. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.2929
Safarzadeh MA, Motahhari SM (2014) Co-optimization of carbon dioxide storage and enhanced oil recovery in oil reservoirs using a multi-objective genetic algorithm (NSGA-II). Pet Sci 11 (3):460–468
Schwefel H-PP (1993) Evolution and optimum seeking. Wiley, Hoboken
Scrucca L (2013) GA: a package for genetic algorithms in R. J Stat Softw 53(4):1–37. http://www.jstatsoft.org/v53/i04/
Shakhsi-Niaei M, Iranmanesh SH, Torabi SA (2013) A review of mathematical optimization applications in oil-and-gas upstream & midstream management. Int J Energy Stat 1 (02):143–154
Shaw R, Srivastava S (2007) Particle swarm optimization: a new tool to invert geophysical data. Geophysics 72(2):F75–F83
Shelokar PS, Jayaraman VK, Kulkarni BD (2004) An ant colony approach for clustering. Analytica Chimica Acta 509(2):187–195
Shi Y, Eberhart RC (1998) Parameter selection in particle swarm optimization. In: EP'98: proceedings of the 7th international conference on evolutionary programming VII. Springer, London, pp 591–600. ISBN 3540648917
Simon HA (1969) The sciences of the artificial, vol 136. MIT Press, Cambridge
Singh HK, Ray T, Sarker R (2013) Optimum oil production planning using infeasibility driven evolutionary algorithm. Evolut Comput 21(1):65–82
Stoean R, Preuss M, Stoean C, El-Darzi E, Dumitrescu D (2009) Support vector machine learning with an evolutionary engine. J Oper Res Soc 60(8):1116–1122
Stoean C, Preuss M, Stoean R, Dumitrescu D (2010) Multimodal optimization by means of a topological species conservation algorithm. IEEE Trans Evolut Comput 14(6):842–864
Stoean R, Stoean C, Lupsor M, Stefanescu H, Badea R (2011) Evolutionary-driven support vector machines for determining the degree of liver fibrosis in chronic hepatitis C. Artif Intell Med 51:53–65. ISSN 0933-3657
Storn R, Price K (1997) Differential evolution: a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11(4):341–359. ISSN 09255001. doi:10.1023/A:1008202821328
Sun J, Feng B, Xu W (2004) Particle swarm optimization with particles having quantum behavior. In Proceedings of the IEEE congress on evolutionary computation. IEEE Press, pp 325–331
Talbi E-G (2009) Metaheuristics: from design to implementation, vol 74. Wiley, Hoboken
Thander B, Sircar A, Karmakar GP (2014) Hydrocarbon resource estimation: a stochastic approach. J Pet Explor Prod Technol 1–8
Tronicke J, Paasche H, Böniger U (2012) Crosshole traveltime tomography using particle swarm optimization: a near-surface field example. Geophysics 77(1):R19–R32
Turney P (1995) Cost-sensitive classification: empirical evaluation of a hybrid genetic decision tree induction algorithm. J Artif Intell Res 2:369–409
Voß S (2001) Meta-heuristics: the state of the art. In: Local search for planning and scheduling. Springer, Berlin, pp 1–23
Wang L, Wang X, Fu J, Zhen L (2008) A novel probability binary particle swarm optimization algorithm and its application. J Softw 3(9):28–35
Whitley Darrell, Rana Soraya, Heckendorn Robert B (1998) The island model genetic algorithm: on separability, population size and convergence. J Comput Inf Technol 7:33–47

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Comput 1(1):67–82

Zaharie D (2005) Density based clustering with crowding differential evolution. In: International symposium on symbolic and numeric algorithms for scientific computing, pp 343–350

Zaharie D (2007) A comparative analysis of crossover variants in differential evolution. In: Proceedings of IMCSIT 2007, pp 171–181

Zangeneh H, Jamshidi S, Soltanieh M (2013) Coupled optimization of enhanced gas recovery and carbon dioxide sequestration in natural gas reservoirs: case study in a real gas field in the south of Iran. Int J Greenhouse Gas Control 17:515–522

Zitzler E, Deb K, Thiele L (2000) Comparison of multiobjective evolutionary algorithms: empirical results. Evol Comput 8:173–195

# Genetic Programming Techniques with Applications in the Oil and Gas Industry

**Henri Luchian, Andrei Băutu and Elena Băutu**

**Abstract** The chapter, entitled "Genetic Programming Techniques with Applications in the Oil and Gas Industry", consists of four parts. The first part presents theoretical features of the genetic programming algorithm, describing its main components, such as individual representation, initialization of the population, evaluation of the individuals, genetic operators, and selection scheme. The second part is concerned with a hybrid evolutionary algorithm—Gene Expression Programming, which combines features from genetic algorithms and genetic programming. In the third part, references towards software frameworks that implement GP are provided. This part then focuses on the use of the R package for genetic programming—RGP and provides a guide for the package, using two model problems to exemplify its usage. The last part reviews applications of genetic programming for petroleum engineering problems.

**Keywords** Genetic programming · Regression · Gene expression · Programming · RGP · Petroleum engineering problems

This chapter presents the theoretical background behind the evolutionary algorithm variant known as genetic programming (GP). Details on the features that make GP a remarkable algorithm for data analysis are provided. Gene Expression Programming (GEP) is a GP variant proposed by Ferreira (2001), which has since gained a lot of interest from researchers for applications in various fields of science. We chose to present it in this chapter since it is a good example of a hybrid evolutionary algorithm that combines advantages from both GAs and GP, and it is among the most used flavors of GP in applications. Insight into the inner workings

H. Luchian
Faculty of Computer Science, Alexandru Ioan Cuza University, Iasi, Romania

A. Băutu
Faculty of Navigation and Naval Management, Romanian Naval Academy, Constanta, Romania

E. Băutu (✉)
Faculty of Mathematics and Computer Science, Ovidius University, Constanta, Romania
e-mail: ebautu@gmail.com



C. Cranganu et al. (eds.), *Artificial Intelligent Approaches in Petroleum Geosciences*,
DOI 10.1007/978-3-319-16531-8_3

of GP is gained by means of two practical examples: a synthetic regression problem and a real problem from the field of petroleum engineering, both modeled by GP.

We provide references to existing software packages that offer implementations of the GP paradigm and focus on the R package (RGP) for genetic programming. A step by step guide to using RGP for the aforementioned problems is provided. Further, we review applications of GP to petroleum engineering related problems, such as well log analysis, reservoir characterization, or pressure analysis.

Following the directions set by John Holland for the presentation of adaptive systems (Holland 1992), we explain the basics of GP and GEP by describing the following features (Bautu 2010):

- the encoding method used by the individuals (**representation**) and the decoding procedure;
- the procedure to **generate individuals**, used especially during the initialization of the population, but also in the context of some genetic operators;
- the **evaluation procedure** (fitness function);
- procedures for **genetic operators** (e.g., mutation, crossover);
- the **selection scheme**.

## 1 Genetic Programming

Nicheal Cramer's work from 1985 stands at the root of the genetic programming paradigm; he proposed a type of genetic algorithm with individuals represented by computer programs (Cramer 1985). Cramer used the proposed algorithm to automatically evolve simple mathematical expressions. His work was followed by Schmidhuber's idea of using Prolog and Lisp as support for evolutionary algorithms, which led to a meta-learning algorithm based on GP (Dickmanns et al. 1987; Schmidhuber 1987). The inventor of modern GP is considered to be John Koza, a former professor at Stanford University, who layed the foundation of what is currently known as GP in his first book on the topic (Koza 1992). He envisioned a genetic algorithm that evolved Lisp S-expressions, that automatically solves problems. Recent accounts on the topic of GP are provided in (Poli and Koza 2014; Poli 2008); insights into the theoretical foundations of GP are provided in Langdon and Poli (2002). We will briefly describe in the following the main traits of GP that differentiate it from GAs, following the description provided in (Bautu 2010; Bautu and Bautu 2009).

### *1.1 Representation of Individuals*

Traditional GP appeared from the need to automatically solve problems, based on a high-level statement of the problem, without any prior knowledge of the form or structure of the solution. The structures (individuals) that evolve are at the base of

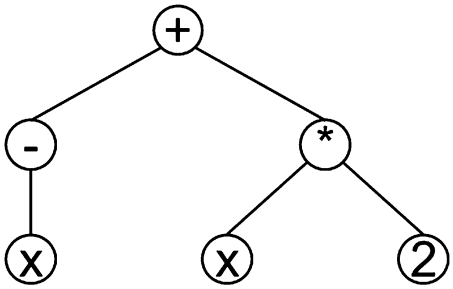


**Fig. 1** GP syntax tree representing the individual $(-x) + x * 2$

any adaptive (or learning based) system. GP individuals are computer programs, encoded as syntax trees (e.g,. Fig. 1). The nodes in the tree are labeled with symbols. The leaves of the tree are labeled with terminal symbols (the variables and the constants in the program—in our example, $x$, 2), while the internal nodes are labeled with functional symbols (e.g., algebraic operators, trigonometric functions, or other common mathematical functions, etc.). During evolution, the sizes and shapes of the trees are changing in order to adapt to the environment provided by the problem. The search space for the GP algorithm is graphically depicted in Fig. 2.

It is important for the symbol set of the algorithm, comprised of all the functions and terminals, to be carefully selected prior to running the GP algorithm, in order to provide the prerequisites to model the proposed problem (Koza 1992). We refer, in the following, to two features that must be met by the symbol set: **closure** and **completeness**.

The **closure** property refers to each function of the set of functions being well defined and closely relative to any combination of parameters it may receive during evolution. This is usually achieved by the special treatment of a relatively small

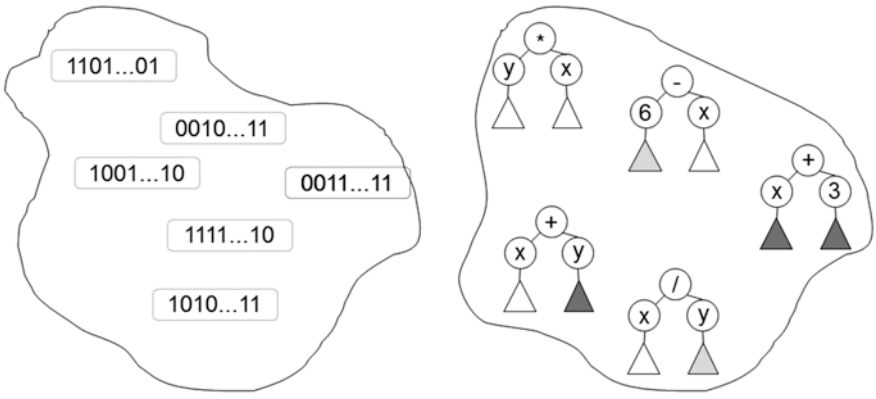


**Fig. 2** Graphical representation of the search space for GA (*left*) and GP (*right*)

number of situations. For example, for divide operations, which are not allowed to receive zero as the second parameter, it is clear that the closure property is not satisfied; likewise, the logarithm function should not receive negative parameters.

Examples of closed symbols sets (i.e., it is guaranteed that all syntactically valid expressions formed with these symbols are also semantically valid):

- $C = \{AND,\ OR,\ NOT,\ x,\ y,\ TRUE,\ FALSE\}$, where $x$ and $y$ are Boolean variables, and *TRUE* and *FALSE* are Boolean constants;
- $C = \{+,\ -,\ *,\ x,\ y,\ 0,\ 1\}$, where $x$ and $y$ are integers variables.

Examples of functions sets that are not closed are:

- $C = \{+, -, *, /, x, y, 0, 1\}$, where $x$ and $y$ are real variables—the set is not closed because it is possible to generate expressions which are semantically invalid due to division by 0:

$$f(x, y) = (x - x)/(y - y) \text{ or } f(x, y) = (x - y)/0,$$

- $C = \{+, -, \log, x\}$, where $x$ is a real variable—the set is not closed; in case the log function receives negative or null parameters, the resulting expression is not semantically valid

$$f(x) = x + \log(x - x) \text{ or } f(x) = \log(x)/\log(-x) \tag{1}$$

A possible solution for achieving closure of the symbol set is by means of the definition of *protected* functions. Protected functions return a special value of the terminal set whenever an exceptional situation is detected. For example, in case of the division operator, a protected function can return 0 if the second parameter is 0:

$$/_{\text{prot}}(x, y) = \begin{cases} x/y, & \text{if } y \neq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

In this way, the *protected divide* operation has a well-defined result for any values of its parameters. The advantage of this approach is its simplicity, from the implementation point of view.

In order to meet the **completeness** property, one must make sure that the symbol set for the algorithm is sufficient in order to express a solution to the problem; in general, expert knowledge is needed to implement this part. This property is guaranteed only for some problem cases where there exist theoretical arguments or empirical evidence favoring a particular choice of symbols.

The selection of the input variables necessary for a given problem can be straightforward, or it may be solved by a feature extraction algorithm (Veeramachaneni et al. 2010). Similarly, the function set that is sufficient to express a problem solution is very dependent on the problem to be solved.

For example, the functions set {*AND*, *OR*, *NOT*} is sufficient to express any Boolean function. By removing the *AND* function, the remaining set still meets the sufficient condition because the *AND* Boolean function can be simulated with:

$$AND(x, y) = NOT(OR(NOT(x), NOT(y))).$$

In case of removing the *NOT* function, the remaining set no longer meets the sufficient condition, because its effect can not be simulated with the functions left in the set. Thus, functions such as *XOR* can not be expressed. As with the terminals, the responsibility to establish the set of functions appropriate for the problem remains to the user.

GP builds approximations of the real solution, in case the symbols included in the symbol set are not sufficient to express a solution to the problem. For this reason, the general set of symbols used in GP to express a solution to a given problem does not coincide with the minimal set of symbols required to express the solution; it usually contains additional symbols. The effect that these additional symbols may have on the quality of solutions identified by the algorithm is difficult to assess a priori. For example, the presence of additional variables in the set of terminals may lead to a decrease in the algorithm performance in finding solutions (Fig. 3); in this case, the GP algorithm also performs a feature selection task, identifying automatically the variables that are significant for the model.

For example, suppose GP is used to infer a formula for the exponential function $e^x$. This function cannot be expressed exactly by a finite algebraic expression. If GP uses the set of symbols,

$$C = \{+, -, *, /, x, y, 0, 1, 2\},$$

it will, most likely, provide finite approximations for this function, such as 1, $1, 1 + x,\ 1 + x + \frac{1}{x^2},\ 1 + x + \frac{1}{x^2} + \frac{1}{x^3}$.

## 1.2 Generating Individuals

The generation of GP individuals is used for the initialization of the population in the first generation, as well as for implementing certain genetic operators, like

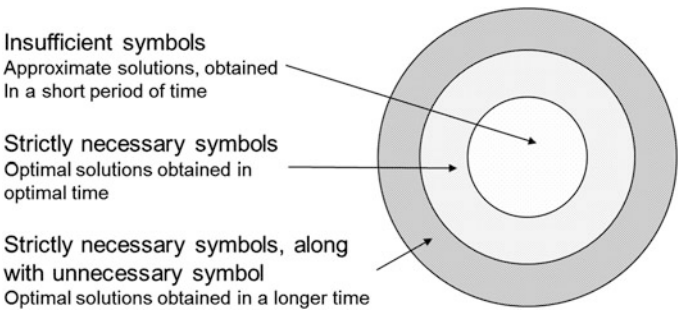


**Fig. 3** Completeness of the symbols set and its effect on the solution

subtree mutation. GP individuals are generated in a random manner, usually recursively, node by node. In the beginning, a symbol chosen randomly from the symbol set of the algorithm is assigned as the root of the tree. If the symbol is a terminal (e.g., variable, constant, or a function without parameters), then the generating process stops. The individual obtained is a (degenerate) tree consisting of a single node, labeled with a terminal. If the symbol chosen is a function *f* with arity *a* (*f*), then the recursive process builds up *a*(*f*) descendants as parameters of this symbol. If a descendant is labeled with a terminal, then the generation process is considered completed for that node. If a descendant is labeled with a function, then the generation process continues recursively until all leaf nodes of the tree are labeled with terminals. The tree depth is the longest direct path from the root to any leaf node. Using pseudocode, this process is described by the algorithm (as shown in Fig. 4).

In practice, the algorithm (as shown in Fig. 4) should be enhanced with a provided mechanism for limiting the size of the tree produced; such a mechanism can be implemented in different ways. Koza proposed three generating methods that provide control over the sizes and complexity of the trees (Bautu 2010; Koza 1992):

- the *full* method creates full trees (i.e., the length of the direct path from the root to any leaf node is equal to the depth of the tree);
- the *grow* method creates trees with different shapes and sizes;
- the *ramped half-and-half* method combines the previous methods to produce a larger variety of full and irregular trees (Fig. 5).

In the initial population, it is essential to have a wide variety of individuals, such that they ensure a good coverage of the search space, and a good diversity. Diversity is key for the evolutionary process. The ramped half-and-half method is very suitable to create a wide variety of trees in the initial population. For example, with depths between 2 and 5, 12.5 % of the trees are full trees of depth 2, 12.5 % are irregular trees of depth 2, 12.5 % are full trees of depth 3, and so on until the maximum depth. Other methods, more sophisticated, are discussed in the literature and are usually already implemented in dedicated GP packages (Luke 2000a, b).

**Require:** $\mathcal{S}$ – the symbols set
**Ensure:** $T$ – result tree
1. $c = RandomSymbol(\mathcal{S})$
2. $root(T) = c$
3. **for** $i = 1 \ldots z(c)$ **do**
4. $\quad Child(T, i) = RandomTree(\mathcal{S})$
5. **end for**
6. **return** $T$

**Fig. 4** Random generation of an individual in GP

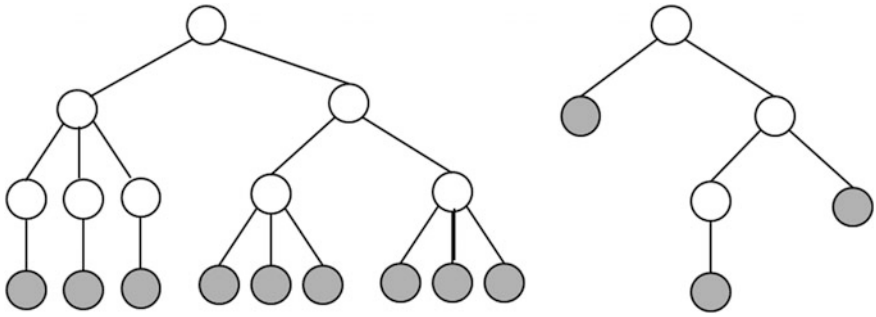

**Fig. 5** Trees with depth 4, generated by the full method (*left*) and the grow method (*right*). *Gray nodes* are terminal nodes

## *1.3 Evaluation of Individuals*

The central idea to all EC techniques is *adaptation to the environment*. In nature, the number of offspring the individual has is usually used as a measure of the individual's adaptation to its environment. In EC, a reverse approach is employed: The specific adaptation of each individual controls the number of offspring. An explicit measure for the adaptation of individuals is the *fitness* value, evaluated using a procedure specific to the problem addressed.

In the case of GP, each individual fitness is evaluated against a given set of input data—particular cases of the problem search space. The selection of the input data should be representative for the problem, because it is the foundation based on which the algorithm generalizes the results obtained to the whole problem space. All the individuals in a generation should train using the same input data, such that they can be compared against each other.

A formula that is very frequently used to evaluate individuals is the *mean squared error* of the individual, with respect to the input data:

$$\text{fitness}(i) = \sqrt{\frac{\sum_{j=1}^{N} (S(i,j) - C(j))^2}{n}}, \tag{3}$$

where $N$ is the total number of cases for assessing individuals, $S(i, J)$ is the value obtained by assessing the individual $i$ of the population for variables in the case $j$ of input data, and $C(j)$ is the correct (expected) value for the case $j$. For the sake of comparing individuals across different generations and algorithm runs, John Koza introduced several types of fitness which offer different abstraction degrees of the individual performances, all of them based on the distance between the input data and the estimations made by the GP individual (Koza 1992).

## 1.4 Genetic Operators

The main GP operators are selection and crossover. Mutation is considered a secondary type operator. Also, specific GP operators exist, such as permutation, editing, encapsulation, and decimation—they are also considered secondary type operators. Other operators may be defined in order to target particular aspects of the problems addressed with GP.

Crossover is deemed the most important GP operator. The basic idea is common to that of the crossover operator from GAs: Parent individuals are selected from the population, and offspring are produced such that they inherit parts from each parent. There exist several traditional versions of the crossover operator, and others may be defined, depending on the specificity of the problem.

The standard crossover operator uses two cut points, one in each parent. The routine for this operator chooses, with uniform probability, one point in each of the two parent chromosomes. Then, it swaps the subtree rooted in the corresponding cut point with the subtree from the other parent (see Fig. 6). This process is illustrated in the algorithm (as shown in Fig. 7) and represented in Fig. 6.

The offspring produced are always valid structures, due to the closure property of the symbol set. It can be noted that the operator produces diversity in the population; in the GA, when 2 identical individuals were subject to crossover, the offspring were identical to the parents. It is not the case of the standard crossover operator in GP. Hence, premature convergence is not an issue for GP when standard crossover is used.

**Mutation** The mutation operator is a secondary type operator, mainly responsible for producing diversity in the GP population. The standard implementation of the operator proceeds by randomly selecting a node from the individual. The subtree rooted in that node is then replaced by a randomly generated subtree. The generation of the new subtree makes use of one of the algorithms we discussed earlier. Similar to crossover, a maximum depth limit can be used to restrict the size

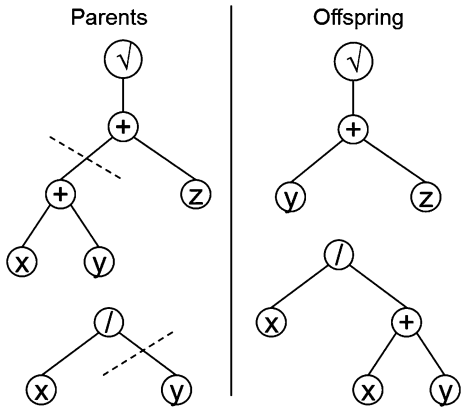


**Fig. 6** Two-points crossover operator exchange subtrees rooted in the cut points (marked with a *dashed line*)

**Require:** $C1, C2$ – parent chromosomes
**Ensure:** $O1, O2$ – offspring chromosomes
1. $O1 = C1$ ▷ Clone parent chromosomes
2. $O2 = C2$
3. $P1 = RandomNodeSelect(C1)$ ▷ Select the cut points
4. $P2 = RandomNodeSelect(C2)$
5. $nod(O1, P1) = Subtree(C2, P2)$ ▷ Swap subtrees
6. $nod(O2, P2) = Subtree(C1, P1)$
7. **return** $O1, O2$

**Fig. 7** The standard (two points) crossover operator in GP

of the offspring. This process is illustrated in the algorithm (as shown in Fig. 8) and exemplified in Fig. 9.

When the cut point for subtree mutation is close to the root of the syntax tree, the operator has a highly destructive effect; similarly, a mutation point near to the leaves of the tree has small chances to alter completely the expression encoded by the individual. A practical solution to this problem is to assign variable mutation probabilities to nodes on different levels of the tree, e.g., mutation probability that increases from the root to the frontier of the tree.

**Permutation** This operator randomly selects an internal node of the syntax tree. Assume this node is labeled with a function of arity $k$. The permutation operator

**Require:** $C$ – the chromosome undergoing mutation
**Require:** $\mathcal{S}$ – the symbols set
**Ensure:** $O$ – cromozomul obinut pentru mutaie
1. $O = C$ ▷ clone the parent
2. $P = RandomNodeSelect(O)$ ▷ select the mutation point
3. $T = RandomTree(\mathcal{S})$
4. $nod(O, P) = T$ ▷ replace the subtree
5. **return** $O$

**Fig. 8** Subtree mutation operator

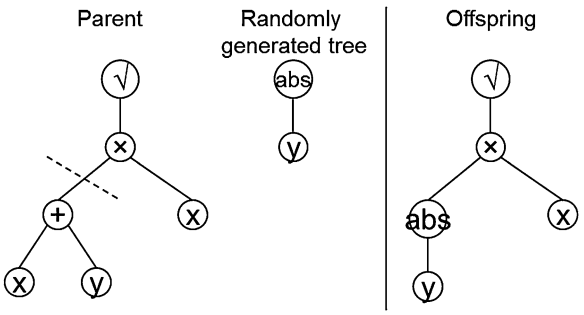


**Fig. 9** The subtree mutation operator replaces a subtree with a newly generated tree

generates a random permutation of the $k$ children and swaps the children nodes according to this permutation. In case the label of the target node is a commutative function, the effect of this operator on the phenotype encoded by the tree is actually null.

**Editing** The editing operator provides a way to reduce the complexity of individuals chromosomes dynamically, at runtime. For example, the editing operator might evaluate functions that are context-free and have only constants as parameters and then replace these functions with the result of the evaluation. Complex editing rules might require large computing resources. The use of this operator is justified by the necessity of limiting code bloat (Luke 2000a, b), or if individuals need to be made more readable (for example, one might process the solution of the algorithm in order to obtain a more user-friendly solution).

**Encapsulation** Reusability of code may be implemented in GP by means of the encapsulation operator. This operator works by assigning names to subtrees of chosen individuals, in order for them to be referred later in GP chromosomes as symbols. Encapsulation operates on a single individual by extracting parts of its chromosome and mapping them to a new symbol name. The encapsulation operator works by randomly selecting an internal node of the tree encoded in the individual, saves the subtree with root at that point by a new symbol name, and replaces it with the new symbol name. The new symbol points to the original subtree and it is included in the terminal set because it is a complete subtree and does not require any parameters to be evaluated. The main benefit of this operator is that it protects the subtree used to define the new symbol from the destructing effects of genetic operators. This operator stands at the base of the automatically defined functions idea in GP (Poli 2008).

## *1.5 Selection Scheme*

In GP, selection is viewed as an operator that acts on a population of individuals and results in a single individual. The selection operator works in two stages: first, an individual from the population is chosen according to a selection scheme, and then, this individual is copied into the population in the next generation of the algorithm. The selection schemes available for genetic algorithms are used in the case of GP, too. Among them, roulette wheel selection stands out as being very highly used—whether used directly with the fitness values of the individuals, or with their ranks (assigned based on fitness values, too).

In the context of GP, tournament selection is also of wide use. For this scheme, a number of individuals (e.g., 2 or 4) are chosen randomly from the population. The one with the best fitness is selected to survive in the next generation. The parents remain in the population, so they may participate in future tournaments.

Usual selection schemes are coupled with elitist survival of a number or of a percent of the individuals in the population. This scheme ensures the survival of the best individuals from one generation to the next.

## 2 Gene Expression Programming

Different representations used in the GP algorithm led to different flavors of GP, oftentimes with their own names. Driven by the idea that every terminal has a type and every function has a specification of types for its parameters, Montana introduced strongly typed genetic programming (David 1995). This variant is useful for implementing type constraints, such as those encountered in physics equations. The existence of multiple objectives for practical problems led to the proposal of Pareto GP (Vladislavleva et al. 2009).

Gene Expression Programming is a GP-based algorithm, proposed in (Ferreira 2001), very popular in applications in many domains (Zhou et al. 2003). This variant combines the advantages of the classical GA representation (linear strings of fixed size, which leads to easy implementation of genetic operators), with those exhibited by the individuals in GP (hierarchical structures with different sizes and shapes, which leads to the possibility of encoding highly complex programs). We will describe GEP in the following, using the same structure as for the description of the GP algorithm.

### *2.1 Representation of GEP Individuals*

The *phenotype* of a GEP individual is a complex mathematical expression that may be viewed as a hierarchical structure with variable sizes and shapes. In GEP jargon, it is called an *expression tree*. The *genotype* of a GEP individual is a fixed size string of symbols.

The expression tree from Fig. 10 represents the mathematical expression

$$\sqrt{(x-y)\times(y+x)},$$

which is the phenotype of the genotype:

$$\sqrt{}\times - + x\ y\ y\ x, \tag{4}$$

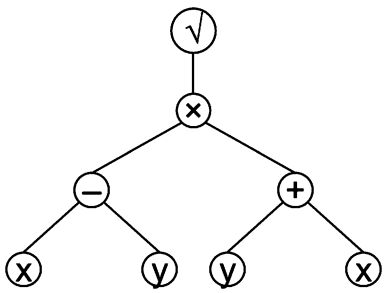


**Fig. 10** An expression parse tree

where $\sqrt{}$ denotes the square root function. This encoding is obtained by the *breadth* traversal of the expression tree in Fig. 10. The expression is different from the prefixed notation, as well as from the postfix notation obtained by depth traversing, which are used by some vector-based or stack-based variants of GP (Keith and Martin 1994).

Decoding the genotype into the equivalent phenotype follows the same rules. For example, the genotype $\sqrt{}/-x\,y\,x$ is equivalent to the following expression tree: The start symbol ($\sqrt{}$) is of arity 1; hence, it is linked with the following symbol (/); /has arity 2, and it is linked with the following two symbols—and $x$. The process continues until each symbol is linked with a number of symbols equal to its arity. The symbols with arity 0 are leaf nodes in the phenotype's expression tree. The translation process builds the expression tree corresponding to $\sqrt{\frac{(y-x)}{x}}$.

GEP genes are divided into two structural units: head and tail. The head may contain functions and terminals, and the tail is constrained to contain only terminals. The tail size depends on the head size and on the set of symbols used in the gene,

$$t = h(n-1)+1,$$

where $t$ is the required minimum size of the tail, $h$ is the size of the head, and $n$ is the maximum arity of the symbols that may appear inside the gene. In this organization, GEP genes are padded at the end with symbols that may not be used in the decodification (they are inactive). This structural organization of GEP genes ensures syntactic validity of all obtained programs. Also, GEP genetic operators always produce syntactically correct expressions.

GEP individuals are multi-genic chromosomes, where each gene encodes a valid expression tree which interacts with the other genes to create a complex entity. The interaction is given by a *linking function*.

For example, consider the set of symbols $\{\times, /, +, -, \sqrt{}\} \cup \{x, y, z, t\}$. The maximum arity of symbols is $n = 2$. Hence, if the head size is $h = 4$ symbols, then the tail size is $t = 5$ symbols. If chromosomes contain three genes, then the total size of each chromosome is $c = 3 \times (4+5) = 27$ symbols. In this context, the chromosome:

$$\sqrt{}\times + x\,z\,y\,z\,x\,x \quad -y\,z\,x\,y\,z\,t\,x\,z \quad /\sqrt{}\,x\,z\,z\,t\,x\,t\,t \tag{5}$$

contains the code for the following mathematical expressions:

$$E_1 = \sqrt{(z+y)\times x}$$
$$E_2 = y - z$$
$$E_3 = \sqrt{z}/x$$

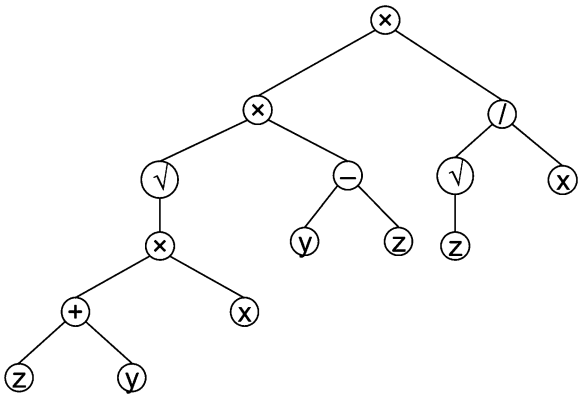


**Fig. 11** The parse tree of the solution encoded by the trigenic chromosome from eq. (5), using multiplication as the linking function

Finally, these three subexpressions are connected via the linking function (e.g., addition, multiplication), which determines the solution encoded by chromosome. If multiplication is the linking function, the parse tree in Fig. 11 is obtained. It corresponds to the mathematical expression:

$$(\sqrt{(z+y)\times x})\times(y-z)\times(\sqrt{z}/x).$$

The solution can be encoded with a single expression; hence, it is possible to express the same thing as a single-gene chromosome:

$$\times\times/\sqrt{}-\sqrt{}x\times y\,z\,z+x\,z\,y. \tag{6}$$

However, using a multi-genic chromosome presents some advantages for complex problems, including the construction of modular and hierarchical solutions. In this way, each gene forms a small building block, isolated from the others and evolving independently.

The interpretation of a chromosome requires the interpretation of each of its genes and deciding the type of interaction between them. The interaction method is usually fixed depending on the problem addressed and some general operation (e.g., addition, multiplication, or the *AND* Boolean function). The linking function may be updated at runtime or even be evolved during the algorithm.

## *2.2 Generating Individuals*

GEP uses a random initialization process, similar to that of the traditional genetic algorithms. The process takes into account the fact that the randomly generated

symbols must respect the structural organization of the GEP chromosome. In the tail, only terminal symbols are allowed, whereas in the head both terminals, and functions, may appear. An alternative to random generation of the initial population is to use a heuristic to generate it. A possible problem that may arise is choosing the appropriate heuristic and then selecting the appropriate encoding of the output solutions provided by the heuristic such that they may be expressed into chromosomes of the GEP algorithm.

## 2.3 Evaluation of Individuals

In GEP, depending on the problem, evaluation of the individual is similar to the evaluation methods used in GP. An important application of GEP is symbolic regression or the discovery of models. In this context, the objective of the algorithm is to find a symbolic expression that gives adequate results for the test cases provided as input data. In this respect, a proper solution is sought by minimizing relative or absolute errors.

The fitness function proposed by Ferreira may be written as in (8) for relative errors and in (7) for absolute errors:

$$f_i = \sum_{j=1}^{t} \left(M - |S_{ij} - C_j|\right), \tag{7}$$

$$f_i = \sum_{j=1}^{t} \left(M - \left|\frac{S_{ij} - C_j}{C_j} \cdot 100\right|\right), \tag{8}$$

where $S_{(ij)}$ is the result of individual $i$ on the test case $j$, $C_j$ is the correct value expected in the test case $j$, and $M$ is the range of selection. The algorithm starts with a tolerance between 20 and 100 % for relative errors, which allows it to explore a wider range of the solutions' space before focusing on certain areas.

## 2.4 Selection Scheme

Roulette wheel is widely used in GEP. All selection schemes that have been introduced in the context of genetic algorithms or GP may be used in GEP. Nevertheless, the power of the algorithm is enhanced by the use of a diverse range of genetic operators.

## 2.5 Genetic Operators

Due to the structural similarities between GA and GEP representations, genetic operators in GEP resemble GA operators closer than they resemble GP operators. The GEP operators have to respect the restriction imposed on the tail of the genes (i.e., it can contain only terminal symbols). After the application of GEP operators, it is possible for symbols that were inactive in the parents to become active (i.e., to participate in the actual decoding expression), as it can be observed in Fig. 12. The structure of GEP chromosomes allows rapid adaptation of many operators used in standard GAs, such as mutation (Figs. 12 and 13) and one-point crossover, two-points crossover, or uniform crossover.

The features of GEP allow new genetic operators to be defined, such as gene crossover and transposition operators. The gene crossover operator is a particular case of one-point crossover that selects the cutting point only between genes. Transposition operators randomly select a transposable element and copy it to another location in the chromosome. A transposable element in GEP is a fragment of genome. The structural organization of the chromosome must also be respected —this is the only restriction.

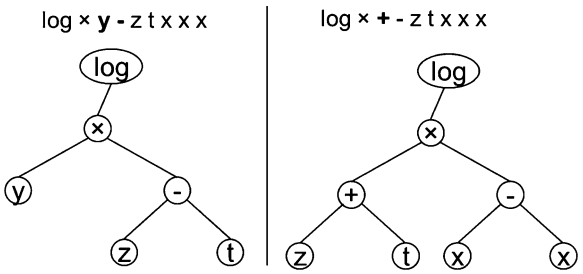


**Fig. 12** The mutation operator in GEP. The mutated symbol is $y$, and it is changed into functional symbol +. The initial encoded expression is $\log(y \times (z - t))$. After mutation, the encoded expression is $\log((z - t) \times (x - x))$. The two *x symbols* in the tail become active after the mutation

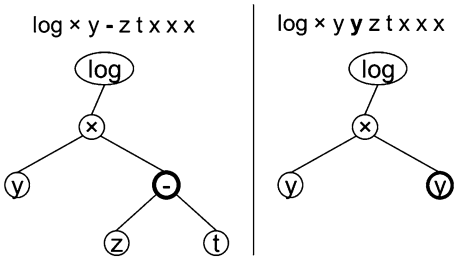


**Fig. 13** The mutation operator in GEP. The mutated symbol is −, and it is changed into terminal $y$. The initial encoded expression is $\log(y \times (z - t))$. After mutation, the encoded expression is $\log(y^2)$

## 3 RGP—R Package for Genetic Programming

Due to its popularity and successful application in a diverse range of real-world problems from different fields of science, many software frameworks are available today for practical and direct application of GP. Among the most well known, we mention Discipulus (Foster 2001), ECJ (Luke et al. 2006), GeneXPro (Ferreira 2010), and GPLab (Silva and Almeida 2003). RGP is a genetic programming package for the R environment that implements various types of genetic programming (classical untyped tree-based GP (Koza 1992), strongly typed GP (Flasch et al. 2010), and Pareto GP (Flasch et al. 2010)) in a way that makes them easy to use, yet highly customizable. It relies on the R environment; hence, it comes together with direct access to the wide set of tools for statistical computing provided by RGP (2014).

RGP is a tree-based GP that stores the individuals as R expressions. This allows most of the existing R functions, variables, and constants to be used in the GP terminal and function set. The R expressions can be directly evaluated by the R interpreter. Besides this flexibility, using individuals as R expressions, many features and packages build on R can be used to further process these individuals. One example is the R rules package which uses a customizable set of a rules to transform such expressions. The default rule set can be used to simplify arithmetic expressions from within GP individuals to reduce the code bloat resulted from the evolution process and to make them easier to grasp by humans.

Out of the box, RGP implements a series of standard GP operators for initialization (grow, full and ramped half-and-half strategy), evolution (subtree crossover, mutation), and selection (various single and multi-objectives methods, such as tournament selection).

The standard tree-based representation and operators are available to allow researchers to get started really fast. However, the chromosome representation and most of the control parameters can be completely changed by the algorithm designer in order to include problem-specific knowledge or to test new settings. The same goes for the evolution pipeline, which can be freely configured with different operators, working in multiple stages, either in parallel or sequentially.

During and after the algorithm runs, researchers can use the vast array of statistical tools available in R and also some special built-in RGP tools to analyze and visualize the structure of GP individuals and populations. In the default implementation, as R expressions, the GP individuals can be presented as the resulting mathematical formulas, as plots of their input/output behavior, as trees (with various detail levels), or as points in a Pareto plot. GP populations can also be presented as forests of schematic trees, as Pareto plots, or as variable presence charts. We will present, in the following, two examples of RGP usage.

## 3.1 Examples

The first time you attempt to use RGP on your system, you have to install it from the comprehensive R archive network (CRAN).[1] Because RGP is published on CRAN, the installation of the package and its dependencies is very easy, requiring only the following command to be run:

```
> install.packages("rgp")
```

After RGP is installed in your local R environment, every time you want to use it you have to load its functionality into the current R session via the library command:

```
> library("rgp")
```

RGP offers three major ways in which its capabilities can be used by researchers.

First, one can use the low-level functions offered by the package. These functions (such as `breed`, `makePopulation`, `functionSet`, and many others) allow many in-depth customizations of how the GP algorithm works. In general, each of these functions targets a very focused step of the algorithm pipeline and they are mostly useful to researchers in the evolutionary computation fields, to be used as the building blocks for prototyping new GP algorithms. We will not discuss this method, but we reference you to the RGP documentation which provides in-depth explanations and some examples of using these functions (RGP 2014).

The high-level functions (such as `geneticProgramming`, `symbolicRegression`, `typedGeneticProgramming`, `dataDrivenGeneticProgramming`) are, in fact, façade for the entire RGP system that makes it easier to set up and run a GP algorithm for a particular task (RGP 2014). This is useful for most scientists with basic knowledge of programming in R that want to use in their work either of the standard versions of GP that RGP offers ready-made, or small variations of them.

For a standard symbolic regression problem, we have available a set of measurement data, divided into input (independent) and output (dependent) variables. The task is to discover a mathematical expression that explains best the functional relationship between the variables. In the following simple example, we will consider a synthetic dataset, sampled from the (simple) function $\sin(x) + 1$. Let us consider the following R data frame:

[1] RGP can be freely downloaded from http://cran.r-project.org/web/packages/rgp/index.html.

| > data | | |
|---|---|---|
| | x | y |
| 1 | 1 | 1.84147098 |
| 2 | 2 | 1.90929743 |
| 3 | 3 | 1.14112001 |
| 4 | 4 | 0.24319750 |
| 5 | 5 | 0.04107573 |
| 6 | 6 | 0.72058450 |
| 7 | 7 | 1.65698660 |
| 8 | 8 | 1.98935825 |
| 9 | 9 | 1.41211849 |
| 10 | 10 | 0.45597889 |

Let us assume we want to use RGP to find a formula that models the dependent variable $Y$ with respect to independent variable $X$. We can use the symbolicRegression `SymbolicRegression` high-level function to set up and run a GP instance. In this case, we provide the model formula we are seeking that computes $Y$ using $X$ ($y \sim x$). We also provide the training data that the algorithm will use as a selection environment for its individuals. Finally, we provide the stopping criterion to use: The algorithm will stop after running for 30 s. The chart that describes the solution obtained in 30 generations of the GP run is depicted in Fig. 14.

Other options would be to stop the algorithm once it reaches a certain fitness (`makeFitnessStopCondition`), after a certain amount of evaluations (`makeEvaluationsStopCondition`), or a certain number of steps (`makeStepsStopCondition`). For advanced scenarios, these simple criteria or custom ones can be combined in more complex expressions using the `orStopCondition`, `andStopCondition`, and `notStopCondition` linking functions.

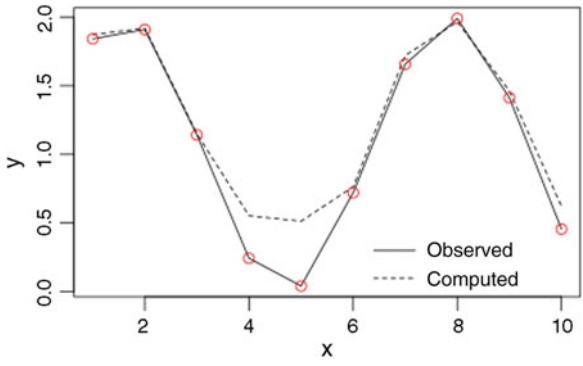


**Fig. 14** R plot of the solution to the simple symbolic regression problem

```
> result <- symbolicRegression(formula = y ~ x,
data = data, + stopCondition = makeTimeStopCondition(30))
> best <- result$population[[which.min(result
$fitnessValues)]]
```

Apart from the parameters presented above, `symbolicRegression` has many other parameters that allow various parts of the algorithm to be configured. For example, the `populationSize` parameter controls the size of the population, `individualSizeLimit` controls the maximum size of individuals, and `functionSet` and `constantSet` control the symbols that GP can use in its expressions.

The third way of using RGP is via its graphical user interface. This is by far the fastest way to get started with RGP, while still making use of its advanced features. If you want to use the graphical user interface of RGP, you will also have to install and load the rgpui package, using the similar install and library commands from page 18, but with the `rgpui` parameter instead of `rgp`. After the library is loaded, the `symbolicRegressionUi()` command will start the RGP user interface.

The rgpui window is divided into four major sections. The first section, called Data, is used for importing existing data files into RGP. In the left column, click on *Choose file* to import a text file that contains the data. The file formatting features are presented below the button. After the import is completed, the data can be visualized in the *Table* area on the right. If any error is detected in the file, the import will stop and the errors will be displayed in the *Table* area. An important part of this section is the *Data-partitioning* controls. Using these settings, the user can select if and how to split imported data into training and validation sets.

In the following, the problem we use as test bed for GP concerns the estimation of reservoir rocks permeability from two-dimensional pore images, an interesting problem for the community of petroleum engineers (Jurgawczynski 2007). While complex tools are used in the industry to address this problem (Jurgawczynski 2007), we employ a symbolic regression approach by means of GP, in order to obtain a model for the estimation of the rock permeability based on measurements of the areas and perimeters of the individual pores.

The R package `datasests` provides a sample of data representative for this problem (Katz 1995), which we use in our experiments—the "rock" dataset. The dataset contains measurements from 12 core samples from petroleum reservoirs, sampled each by 4 cross sections. For each core, a measurement of permeability is available. For each cross section, the total area of pores, the perimeter of pores, and the shape are recorded. Our purpose, in the following, is to exemplify the use of RGP to derive a mathematical formula that explains best the permeability based on the area, perimeter, and shape of the pores, not present a fully described solution for the problem.

The dataset consists of only 48 samples. Each sample registers the variables area, perimeter (for short, peri), and shape—that will pertain to the terminal set of the GP algorithm. The dependent variable is permeability (for short, perm), for which a mathematical formula must be inferred. For our tests, we randomly split the data in

training and validation sets (80 % of the data for training and 20 % for validation), as in Fig. 15. Prior to running the GP algorithm, the data is normalized, by subtracting the mean and by dividing the standard deviation.

After the data are imported into the system, it is time to define our problem based on it (Fig. 16). In the *Objective* section, you can define which of the data columns hold the dependent variable and which of the columns contain the variables that will be used as inputs of our formula. Below, the *Building Blocks* field defines the functional symbols that our formula can contain. Independent variables and random constant symbols are always included in the terminal set, in addition to these

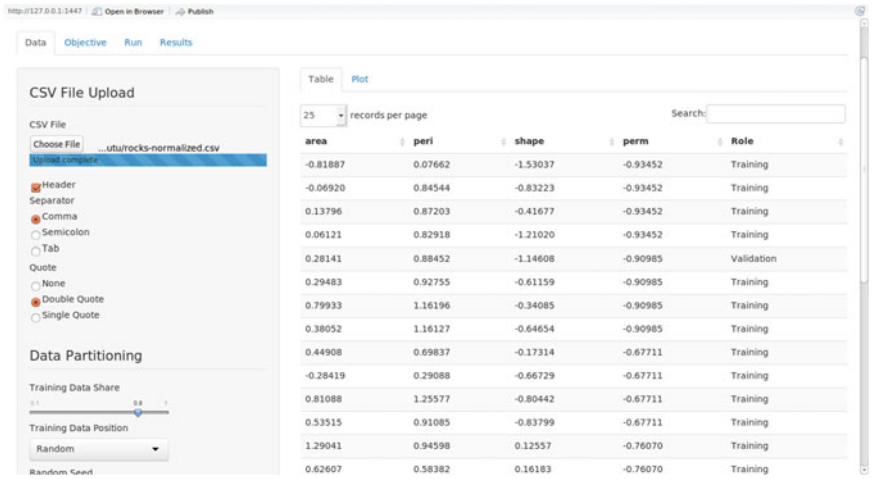


**Fig. 15** Data section

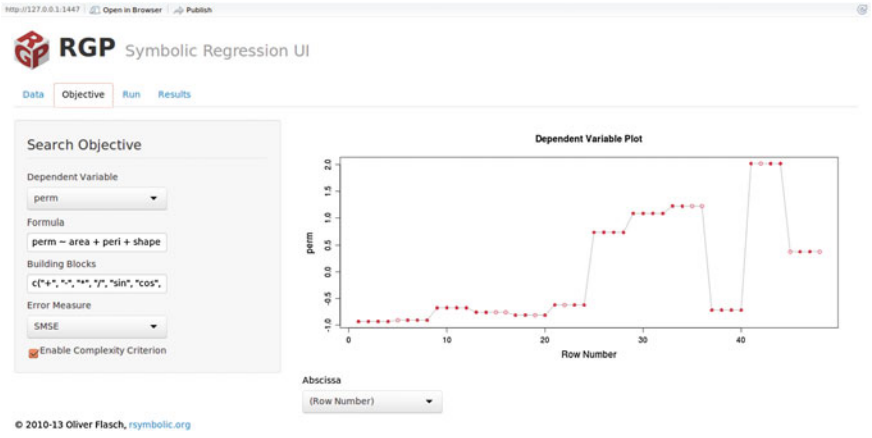


**Fig. 16** Objective section

functional symbols, so they do not need to be explicitly declared here. The *Error Measure* field defines the metric that will be used to measure the quality of each individual. For our experiments, we only set the dependent variable and used the default value for the rest of settings.

After the problem formulation is ready, we can define the algorithm that we will use to tackle it. We do this in section *Run* (Fig. 17). The left side of this section controls the most frequently tuned parameters that RGP offers. Among these, you will find parameters for the size and structure of the population, operators' probabilities, selection scheme, and other parameters. After the parameters have been set to the desired values, the *Start Run* button builds the GP algorithm according to the settings, feeds it with the problem definition, and starts the evolution process. An overview of the entire search process is presented in the right part of the *Run* section.

The algorithm can be paused, resumed, or restarted at any time during the run. The *best solution* area provides information about the best solution found so far (Fig. 18). In our case, even though we used real-world data, within 1 min of running, the RGP evolved complex and quite accurate models for the problem data.

When the algorithm is stopped, the last section (*Results*) presents information about the various solutions that were identified, in addition to the best one (Fig. 19). Although these might have higher errors than the best solution, their visualization can still provide additional insight about the addressed problem.

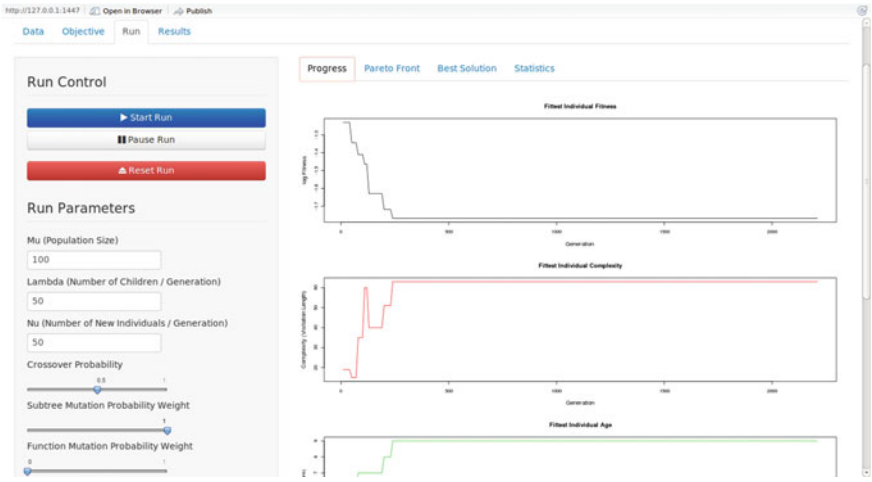


**Fig. 17** Run section. The evolution of the fitness of the best individual is depicted on the *right side*

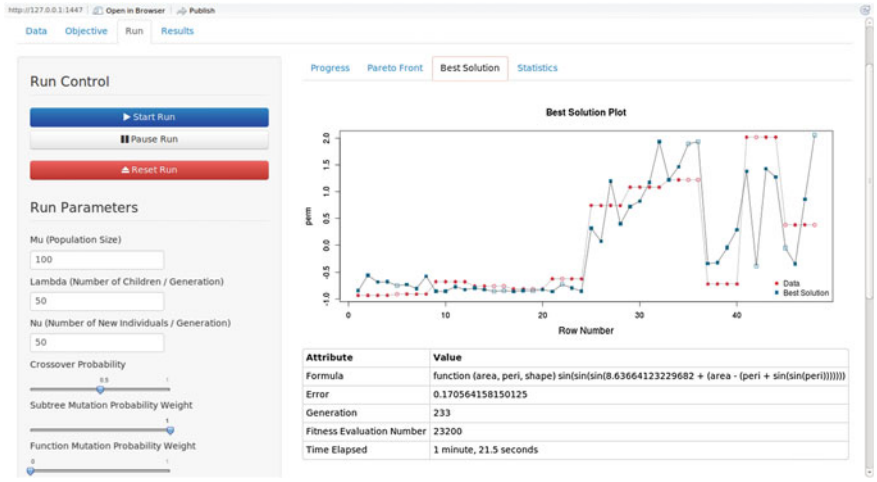


**Fig. 18** Best solution statistics

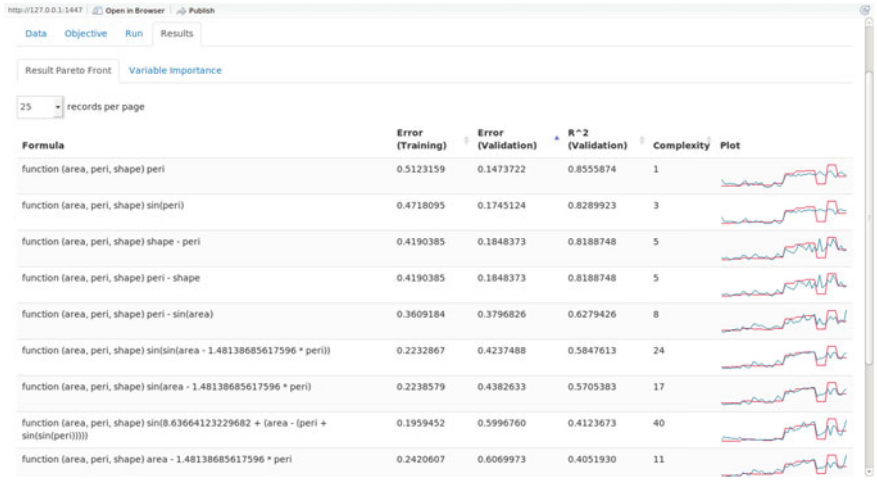


**Fig. 19** Results section

## 4 Applications of GP in the Oil and Gas Industry

Applications of GP techniques in the context of petroleum and natural gas engineering problems are focused on problems of data estimation and forecasting, where classical mathematical modeling methods fail to provide satisfactory results, due to various causes. Among these causes, we mention the complicated nature of

the interrelationships among the parameters, the fact that the parameters of the model may be insufficiently known, or that the simple application of traditional methods is very time consuming or impossible.

Reservoir characterization is a topic of intense research (Cranganu and Bautu 2010; Irani and Nasimi 2011; Kaydani et al. 2014; Saemi et al. 2007; Yu et al. 2007; Yu et al. 2008). GP was used to construct a proxy for a reservoir simulator in (Wilkinson et al. 2010; Yu et al. 2007; Yu et al. 2008). The GP proxy is highly efficient, and it evaluates large numbers of reservoir models that are further used to forecast production. Permeability in a heterogeneous oil reservoir in Iran is predicted using a multi-gene genetic programming algorithm in Kaydani et al. (2014). Oil recovery from a deepwater reservoir is forecasted by means of reservoir characterization and prediction of its producibility. A novel framework for deepwater reservoir characterization is proposed in Yu et al. (2011). It is based on a coevolutionary GP system and fuzzy logic and automates the stratigraphic interpretation (in particular, the gamma ray log is the subject of the interpretation). Dew point pressure measurements are oftentimes not available, yet they are important for forecasting the performance of condensate reservoirs (Eissa and Shokir 2008). A GP-based approach is used to derive a linear model for the dew point pressure, using as inputs the reservoir fluid composition, the reservoir temperature, and the molecular weight of the heptanes-plus fraction. An extensive study involving 245 gas condensate systems for training and 135 systems for validation of the obtained model proved the efficiency of the proposed method in the case of unavailable measurement data.

The soil–water characteristic curve is used in geophysics to describe unsaturated soil behavior. It is estimated successfully by the use of GP in Johari et al. (2006), where it is compared with empirical methods and several indirect methods that prove costly and time consuming. In Garg et al. (2014a), the authors apply a multi-gene genetic programming algorithm to estimate pore water pressure based on soil depth and the soil–water characteristic curve input variables. In Garg et al. (2014b), the previously mentioned GP approach is compared to support vector regression and artificial neural network. The obtained model explains the relationship between water content, stress, and suction. The GP model is satisfactory enough such that it is used to predict water content values by specialists. GP and GEP are used along with evolutionary polynomial regression in order to obtain a model to be used for forecasting settlements of shallow foundation on cohesionless soils (Shahnazari et al. 2014). The problem involves uncertainties around factors such as the compressibility of the soil or the heterogeneous nature of soils, among others.

GEP is used to predict the amount of gas produced by coalfaces in Li et al. (2004). In Qiong et al. (2007), the same problem is approached by GEP, but its application is preceded by feature extraction by means of principal component analysis. GEP outperformed ANN in the task of forecasting performance and emission parameters of an experimental engine in Roy et al. (2014). Prediction models of peak ground acceleration, involved in assessing the damage of structures following an earthquake, are obtained with GEP in Güllü (2012). The models are further evaluated using a likelihood estimation measure, and the results proved the

GEP models to be competitive to regular regression models used previously for this task. GEP is used to infer a model for the estimation of the compressional acoustic (sonic) log (DT) in a given well based on measurements of the natural gamma ray log (GR) and the deep resistivity (REID) in the same well in Cranganu and Bautu (2010). The obtained models allowed the prediction of overpressure zones within the wells used in the experiments.

## References

Bautu E (2010) Intelligent techniques for data modeling problems. Ph.D Thesis, Lambert Academic Publishing, Department of Computer Science, Al. I. Cuza University

Bautu E, Bautu A (2009) Programare genetica Teorie si aplicatii. Al. I Cuza University Publishing House, Romania (In romanian)

Cramer NL (1985) A representation for the adaptive generation of simple sequential programs. In: Grefenstette JJ (ed) Proceedings of an international conference on genetic algorithms and the applications, Carnegie-Mellon University, USA, pp. 183–187

Cranganu C, Bautu E (2010) Using gene expression programming to estimate sonic log distributions based on the natural gamma ray and deep resistivity logs: a case study from the Anadarko basin, oklahoma. J Petrol Sci Eng 70(3):243–255

David J (1995) Montana strongly typed genetic programming. Evol Comput 3(2):199–230

Dickmanns D, Schmidhuber J, Winklhofer A (1987) Der genetische algorithmus: Eine implementierung in Prolog. Technical report, Institut fur Informatik, Lehrstuhl Prof. Radig, Technische Universität München

Eissa M, Shokir El-M (2008) Dewpoint pressure model for gas condensate reservoirs based on genetic programming. Energy Fuels 22(5):3194–3200

Ferreira C (2001) Gene expression programming: a new adaptive algorithm for solving problems. Complex Syst 13(2):87–129

Ferreira C (2010) What is gep? from genexprotools tutorials-a gepsoft web resource. http://www.gepsoft.com/. Accessed 14 Dec 2014

Flasch O, Mersmann O, Bartz-Beielstein T (2010) Rgp: an open source genetic programming system for the r environment. In: Proceedings of the 12th annual conference companion on genetic and evolutionary computation, ACM, 2010, pp 2071–2072

Foster JA (2001) Review: discipulus: a commercial genetic programming system. Genet Program Evolvable Mach 2(2):201–203

Garg A, Garg A, Tai K, Sreedeep S (2014a) Estimation of pore water pressure of soil using genetic programming. Geotech Geol Eng 34:765–772

Garg A, Garg A, Tai K (2014b) A multi-gene genetic programming model for estimating stress-dependent soil water retention curves. Comput Geosci 18(1):45–56

Güllü H (2012) Prediction of peak ground acceleration by genetic expression programming and regression: a comparison using likelihood-based measure. Eng Geol 141:92–113

Holland JH (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. The MIT Press, Massachusetts

Irani R, Nasimi R (2011) Evolving neural network using real coded genetic algorithm for permeability estimation of the reservoir. Expert Syst Appl 38(8):9862–9866

Johari A, Habibagahi G, Ghahramani A (2006) Prediction of soil–water characteristic curve using genetic programming. J Geotech Geoenviron Eng 132(5):661–665

Jurgawczynski M (2007) Predicting absolute and relative permeabilities of carbonate rocks using image analysis and effective medium theory. PhD thesis, Imperial College, Cambridge

Katz R (1995) Measurements on petroleum rock samples. https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/rock.html. Data from BP research, image analysis by Katz R. Department of Statistics, University of Oxford. Accessed 14 Dec 2014

Kaydani H, Mohebbi A Eftekhari M (2014) Permeability estimation in heterogeneous oil reservoirs by multi-gene genetic programming algorithm. J Petrol Sci Eng 123:201

Keith MJ, Martin MC (1994) Genetic programming in C++: implementation issues. In: Kinnear KE (ed) Advances in genetic programming (Chap. 13), MIT Press, Cambridge, pp 285–310

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge

Langdon WB, Poli R (2002) Foundations of genetic programming. Springer, Berlin

LI Q, Zhihua CAI, Zhu L, Zhao Y (2004) Application of gene expression programming in predicting the amount of gas emitted from coal face. J Basic Sci Eng 1:006

Luke S (2000a) Issues in scaling genetic programming: breeding strategies, tree generation, and code bloat. Ph.D. thesis, Department of Computer Science, University of Maryland, College Park, Maryland

Luke S (2000b) Two fast tree-creation algorithms for genetic programming. IEEE Trans Evol Comput 4(3):274–283

Luke S, Panait L, Balan G, Paus S, Skolicki Z, Bassett J, Hubley R, A Chircop (2006) ECJ: a java-based evolutionary computation research system. Downloadable versions and documentation can be found at the following url: http://cs.gmu.edu/eclab/projects/ecj

Poli R, Koza J (2014) Genetic programming. In: Burke EK, Kendall G (eds) Search methodologies, Springer US, pp 143–185

Poli R, Langdon WB, McPhee NF (2008) A field guide to genetic programming. http://www.gp-field-guide.org.uk (With contributions by J.R. Koza)

Qiong G, Zhi-hua CAI, Li Z, Bo H, Du J (2007) A novel gep algorithm based on pca and its application in predicting the amount of gas emitted from coalface. J Basic Sci Eng 4:018

RGP documentation. http://cran.r-project.org/web/packages/rgp/rgp.pdf. Accessed 14 Dec 2014

RGP introduction. http://cran.r-project.org/web/packages/rgp/vignettes/rgp_introduction.pdf. Accessed: 2014-12-14

Roy S, Ghosh A, Dos AK, Banerjee R (2014) A comparative study of GEP and an ANN strategy to model engine performance and emission characteristics of a CRDI assisted single cylinder diesel engine under CNG dual-fuel operation. J Nat Gas Sci Eng 21:814–828

Saemi M, Ahmadi M, Varjani AY (2007) Design of neural networks using genetic algorithm for the permeability estimation of the reservoir. J Petrol Sci Eng 59(1):97–105

Schmidhuber J (1987) Evolutionary principles in self-referential learning. (on learning how to learn: the meta-meta-… hook.). Technical report, Institut f. Informatik, Technische Universität München

Shahnazari H, Shahin MA, Tutunchian MA (2014) Evolutionary-based approaches for settlement prediction of shallow foundations on cohesionless soils. Geotech Eng 12(1):55–64

Silva S, Almeida J (2003) Gplab-a genetic programming toolbox for matlab. In: Proceedings of the Nordic MATLAB conference, Citeseer, pp 273–278

Veeramachaneni K, Vladislavleva K, O'Reilly U-M (2010) Feature extraction from optimization data via datamodeler's ensemble symbolic regression. In: Learning and intelligent optimization, Springer, pp 251–265

Vladislavleva EJ, Smits GF, Hertog DD (2009) Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. Evol Comput IEEE Trans 13(2):333–349

Wilkinson DA, Yu T, Castellini A (2010) Method for forecasting the production of a petroleum reservoir utilizing genetic programming, 2 Feb 2010. US Patent 7,657,494

Yu T, Wilkinson D, Castellini A (2007) Applying genetic programming to reservoir history matching problem. In: Genetic programming theory and practice IV, Springer, Berlin, pp 187–201

Yu T, Wilkinson D, Castellini A (2008) Constructing reservoir flow simulator proxies using genetic programming for history matching and production forecast uncertainty analysis. J Artif Evol Appl 2008:2

Yu T, Wilkinson D, Clark J, Sullivan M (2011) Computational intelligence for deepwater reservoir depositional environments interpretation. J Nat Gas Sci Eng 3(6):716–728

Zhou C, Xiao W, Tirpak TM, Nelson PC (2003) Evolving accurate and compact classification rules with gene expression programming. IEEE Trans Evol Comput 7:519–531

# Application of Artificial Neural Networks in Geoscience and Petroleum Industry

**Rahman Ashena and Gerhard Thonhauser**

**Abstract** It has been shown that artificial neural networks (ANNs), as a method of artificial intelligence, have the potential to increase the ability of problem solving to geoscience and petroleum industry problems particularly in case of limited availability or lack of input data. ANN application has become widespread in engineering including geoscience and petroleum engineering because it has shown to be able to produce reasonable outputs for inputs it has not learned how to deal with. In this chapter, the following subjects are covered: artificial neural networks basics (neurons, activation function, ANN structure), feed-forward ANN, backpropagation and learning (perceptrons and backpropagation, multilayer ANNs and backpropagation algorithm), data processing by ANN (training, over-fitting, testing, validation), ANN and statistical parameters, an applied example of ANN, and applications of ANN in geoscience and petroleum Engineering.

## Nomenclature

| | |
|---|---|
| $\alpha$ | Learning rate |
| ANN | Artificial neural network |
| AAPE | Average absolute percent error |
| APE | Average percent relative error |
| ARMSE | Average root-mean-square error |
| BB | Backpropagation |
| $f$ | Activation or transfer function |
| $\text{Input}_i$ | The input value corresponding to neuron $i$ |
| Logsig | Logistic sigmoid activation/transfer function |
| $m$ | Number of output neurons or nodes |
| MSE | Mean square error |
| $O_{\text{ANN}}$ | Predicted output value by the artificial neural network |
| $R$ | Pearson correlation coefficient |
| $R^2$ | Squared pearson correlation coefficient |

R. Ashena (✉) · G. Thonhauser
Chair of Drilling and Completion Engineering, Petroleum Engineering Department, University of Leoben, Leoben, Austria
e-mail: rahman.ashena@unileoben.ac.at



C. Cranganu et al. (eds.), *Artificial Intelligent Approaches in Petroleum Geosciences*,
DOI 10.1007/978-3-319-16531-8_4

| | |
|---|---|
| SD | Standard deviation |
| $T$ | Number of training samples from known data point given for training the network |
| $V$ | Variance |
| $V_{\text{expected}}$ | Expected real value (known or measured value of output) |
| $W_i$ | The weight corresponding to link or connection $i$ |

## 1 Introduction

Artificial neural networks (ANNs) and their application in geoscience and petroleum industry are considered in this chapter. Application of ANNs has shown to be an effective tool to solve nonlinear complex engineering problems particularly when there are no straightforward analytical or even numerical solutions.

Frequently, when there is no analytical solution or approach to a complex or non-straightforward problem wherein the involved parameters and their exact relationship are not known clearly, ANN is applied. However, one can use ANN even in linearly behaving problems having an analytical solution so that its accuracy and functionality can be determined.

Indeed, a so-called flow of information in the ANN model takes place utilizing basic processing units which are artificial neurons connected to each other. In this way, processing of the data takes place through a network of neurons.

In petroleum industry applications to date, the commonly utilized ANN structure has been Feed-forward artificial neural network (FF-ANN) due to its simplicity compared to other neural structures. FF-ANNs are network structures in which the information or data will propagate only in one direction. This network has a learning ability to recognize the relationship between the inputs and outputs provided that adequate training data are supplied. The FF-ANNs typically consist of three layers including input layer, hidden layer, and output layer. There are other structures of ANNs which are not discussed in this chapter because of their limited usage in petroleum industry. In terms of number of hidden layers, perceptrons and multilayer networks will be discussed.

Although it is possible to have more than one hidden layer in the network, normally a single layer is preferred in many applications. The number of neurons in the input and output layers is normally determined by the problem. However, the optimal number of neurons in the hidden layer (and even the number of hidden layers) must be determined by trial and error in order to obtain a proper network size with highest possible performance.

The network learning or training is attained by adjustment of the weights corresponding to connections or links between the neurons to produce outputs with acceptable errors. After training, the network performance is tested in two stages (test and validation). If the network performance was successful in these three

stages, the network would be recognized as a capable tool for simulation using new inputs and obtaining new results. Recognition and prevention of over-fitting will be discussed.

In most of geoscience and petroleum engineering applications, the FNN will employ backpropagation as its training or learning algorithm. The algorithm is called backpropagation because the output error is propagated backward to the links between neurons in the previous layers during training. In this way, backpropagation helps to modify the weights of links in order to achieve a desired output. The work mechanism of backpropagation algorithm is adjustment of each weight of the network individually based on selection of the path of the steepest gradient descent to minimize the error function (which is usually mean squared error or MSE). As each ANN performance is evaluated by the corresponding errors, knowledge about the relevant statistical model error parameters such as MSE is of great importance.

In this chapter, an attempt is made to give a comprehensive applied discussion of neural networks from basics to application. Thus, an applied example of ANN application in petroleum industry is given after presenting the basics. Then, a short overview of applications of neural network approach would be given.

## 2 Artificial Neural Networks (ANNs) Basics

### *2.1 ANN Structure in General*

Artificial neural networks' structure and function are indeed an extremely simplified version or simulation of the biological human brain. ANNs have been developed as simplification of the mathematical models of biological neural networks (Fig. 1) by having some assumptions which include processing of information takes place in some processing elements called neurons (first assumption). In this simplification,

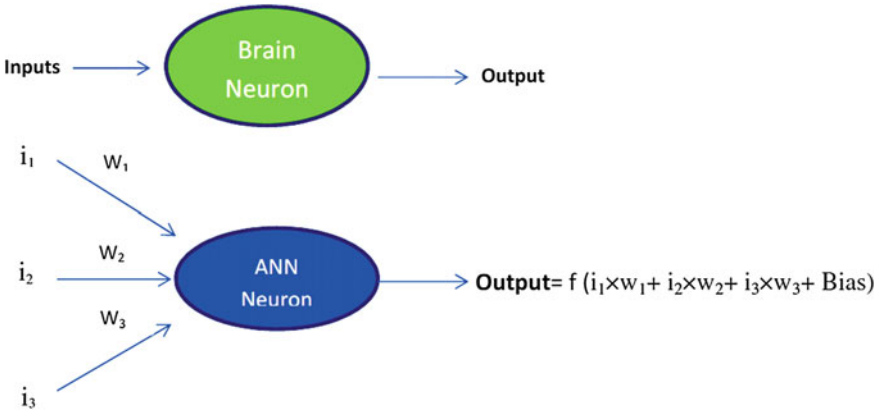


**Fig. 1** Biological and artificial neuron similarity

the information is transferred between neurons using connection links (2nd assumption), and a specified weight is allocated to each link to be multiplied by the information or signal which is passing each link (3rd assumption). Then, each neuron allocates a desired bias or threshold value to be added to the sum to yield a net value (4th assumption), the net value is given as input to an activation or transfer function (which is normally nonlinear function) and in this way the output of the neuron would yield (5th assumption). Simply, the function of the whole ANN structure is just the calculation of the output of all the neurons existing in the network.

## 2.2 *Artificial Neurons*

A typical neuron structure is shown in Fig. 2. As can be seen in this Figure the inputs are multiplied by the corresponding weights, and then, their summation is found. In addition, a bias is added to the summation as an error correction. This value is called the net value. The net value (as the input) is passed through a function called the activation function (Fig. 3). The output of this function is indeed the output of the neuron which would be used as the input to the other neurons in the next layer. This is what actually happens inside each individual neuron.

Please note that the weights and inputs could be assumed as vectors and thus their multiplication is in reality considered as their inner product.

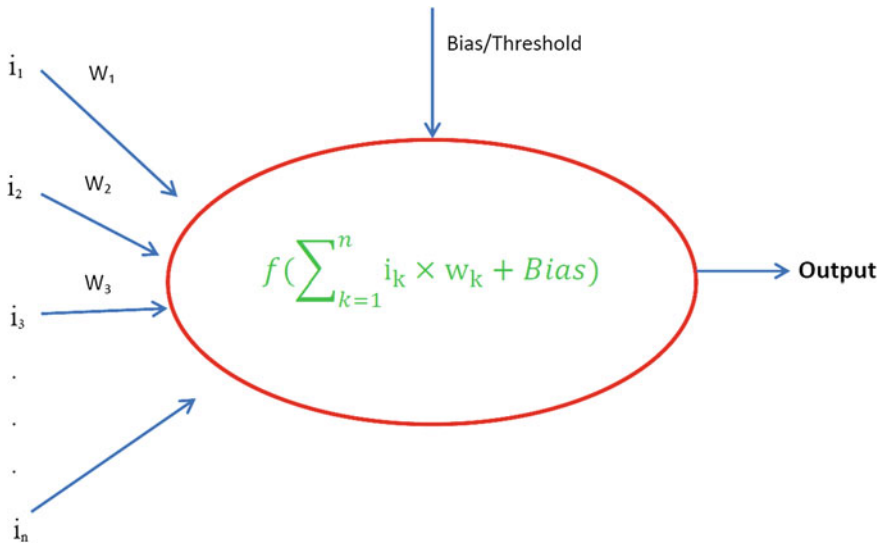


**Fig. 2** A typical neuron structure and the neuron output by applying the activation function on net value

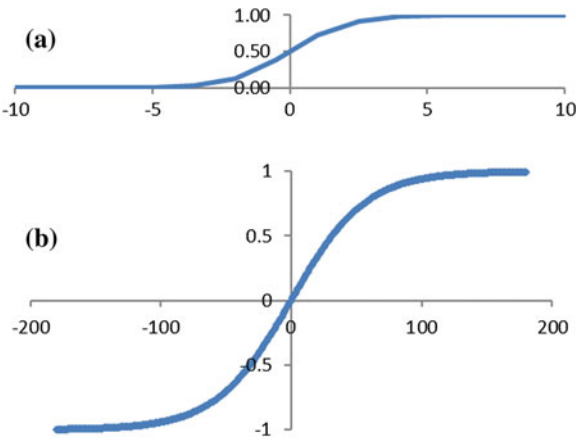


**Fig. 3** Two popular sigmoid activation/transfer functions. **a** logistic function $f(x) = \frac{1}{1+e^{-ax}}$ and **b** hyperbolic tangent function $f(x) = \tanh(x)$

The weight allocated to each connection link, to be multiplied by each input of the neuron, is indeed representative of the importance of the input in the problem, and in this way, this input importance or strength is transferred to the next layer through the corresponding link. For example, in bottom hole pressure (BHP) prediction downhole, there are several input parameters which are of importance and must be taken into consideration. A good trained ANN would allocate higher weights to be multiplied by the more important or stronger input parameter values.

## 2.3 Activation Function

It should be noted that the activation function (Fig. 3) takes the net as its input and its output is considered as the output of the neuron. Typical activation functions are as follows:

- Threshold Function $\{ f = 0$ when $x < 0$, 1 when $x \geq 0\}$,
- Piecewise Linear Function $\{f = 0$ for $x \leq -0.5, f = 0$ for $-0.5 \leq x \leq 0.5$ and $f = 1$ for $x > 0.5\}$,
- Logistic Sigmoid Function $\left\{f(x) = \frac{1}{1+e^{-x}}\right\}$ which has values between 0 and 1,
- Sigmoid Hyperbolic Tangent Function $\{f(x) = \tanh(x)\}$ with values between $-1$ and 1.

Among the activation functions, the sigmoid functions are very common. In Fig. 3, the common sigmoid functions have been illustrated, wherein $\boldsymbol{a}$ is a constant. The sigmoid functions are more frequently used because they are continuous and of course have positive smooth and derivative. The sigmoid functions, unlike some other functions, do have valid derivatives in all points. The derivative of the *logistic sigmoid* function is indeed smooth as shown below:

$$f'(x) = \frac{-e^{-\alpha x}}{(1+e^{-\alpha x})^2} = \alpha f(x) \times \{1 - f(x)\} \quad (1)$$

The derivative of the *sigmoid hyperbolic tangent function* is also smooth as shown below:

$$f'(x) = 1 - \tanh^2(x) = \{1 - f^2(x)\} \quad (2)$$

It is just to note that the derivative of the activation or transfer function in ANN is of great importance in order to give more ability to the network. This is because the neuron activation function must be differentiable and continuous at each point. Consequently, depending on the purpose, the logistic sigmoid and sigmoid hyperbolic tangent functions are normally performing better than the rest.

It is also to be noted that in ANN with no hidden layers, no activation functions are applied and the output of each neuron is the net value (summation plus bias). These ANNs are usually used for simple problems.

## 3 ANN Structure and Feed-forward Artificial Neural Networks (FF-ANNs)

As said above, ANNs structurally consist of some neurons which are linked to each other and cooperate to transform inputs into outputs in the best possible manner. However, on a larger scale, ANN structure is in turn made up of an input layer, one or several hidden layers, and also an output one. In each layer, there are one or several neurons.

There exists only one input and only one output layer all the time. However, there can exist a different number of hidden layers. They can be none, one, two, or even more. As a matter of fact, the number of hidden layers depends on the complexity of the problem.

The FF-ANN has been the first and simplest ANN yet introduced. In this type of ANN, as seen in Fig. 4, the flow or movement of information takes place from the input neurons, through the hidden neurons (if any) to the output neurons (only in the forward direction, without any cycles or loops). Simplicity of this network has helped it to be in common use in most of the petroleum engineering applications to date. FF-ANN is normally capable of learning the implicit governing relationship between the inputs and outputs.

The numbers of neurons in the input layer and output layer are normally determined by the problem. However, the number of neurons in the hidden layer has to be specified by the user. Based on the experience of the user, the optimal number of neurons must be determined so that an efficient neural network is obtained. Yet, the only way to determine the optimal number of neurons in the hidden layer is performed by trial and error (based on the user's experience).

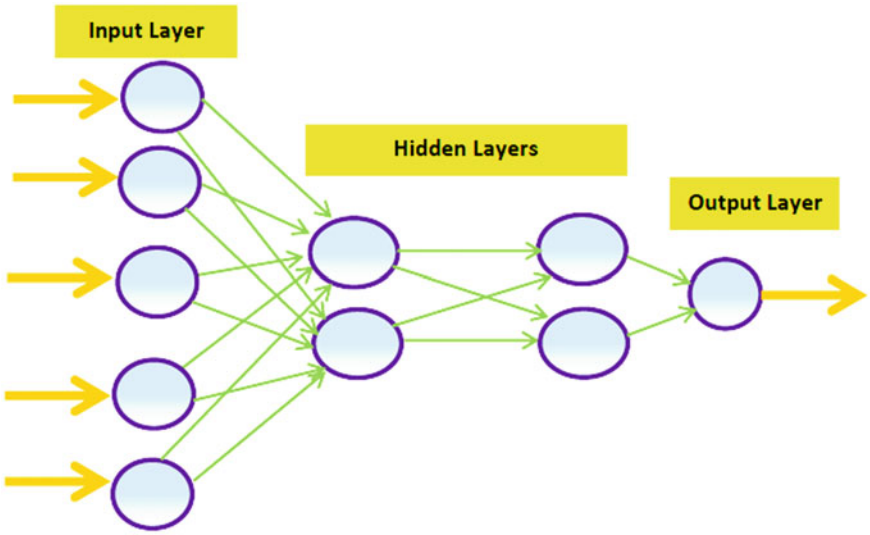


**Fig. 4** Typical feed-forward artificial neural networks (FF-ANNs)

## 4 Backpropagation and Learning

In petroleum engineering application, FF-ANN employs backpropagation of errors in the training phase as their training algorithm (with a supervised learning method). Indeed, backpropagation gets its name from the fact that, during training, the output error is propagated backward to the links or connections between neurons in the previous layers. During this backpropagation of errors, the weights of the links between neurons are adjusted. This process is continued in an iterative manner. In this way, the weights corresponding to links are modified in order to obtain a desired output (with less error), or in other words better learning of the algorithm.

To obtain a more real understanding of the learning mechanism of backpropagation, the expected real output and the predicted output by ANN are compared and an error function is evaluated. Before the training begins, some initial values are allocated to the links between neurons as the weights. Afterwards, by start of training process, a number of data points are fed to the network to be trained using these real examples. For simplicity, a perceptron[1] ANN (with no hidden layers) consisting of 2 input neurons and one output neuron is considered. The training data points have the frame of ($\text{Input}_1$, $\text{Input}_2$, $V_{ex}$). Commonly, the error function utilized to measure the deviation of the expected real output ($V_{ex}$) and the output by ANN is the mean squared error (MSE). The MSE is found by:

[1]A neural network without hidden layer(s).

$$\mathrm{MSE} = (V_{\mathrm{ex}} - O_{\mathrm{ANN}})^2$$

where

$V_{\mathrm{ex}}$ Real expected output value corresponding to a specified input data point (a fixed value, already known)

$O_{\mathrm{ANN}}$ Evaluated value by whole ANN using specified input data points (the output value of the output neuron of the network)

In the mentioned perceptron above (Fig. 5), for instance, suppose a data point of (2, 1, 1) is taken into account for training the network. Please note that the values 2 and 1 are input independent parameter values, and 1 is the dependent value. In this data point, the value of 1 is the expected real output value. If the mean squared error (MSE) values are plotted on the *y*-axis versus the possible output values computed by ANN ($O_{\mathrm{ANN}}$) on the *x*-axis, a parabolic shape is resulted as shown in Fig. 6. The minimum of the parabola is corresponding to the global minimum of the error or MSE (the most favorable point with error equal to zero). The nearer the predicted output value by ANN ($O_{\mathrm{ANN}}$) is to the expected real value ($V_{\mathrm{ex}}$), the less the MSE would be.

Considering $O_{\mathrm{ANN}} = \mathrm{Input}_1 w_1 + \mathrm{Input}_2 w_2$ (ignoring the bias for simplicity), the 3-D map of the error surface (MSE) could be drawn considering the known data point of (2, 1, 1) as an example (Fig. 7).

The work mechanism of backpropagation algorithm is adjustment of each weight of the network individually based on selection of the path of the steepest gradient descent to minimize the error function (which is usually MSE). In more details, backpropagation calculates the gradient descent or derivative of the error of the ANN prediction with respect to all the weights existing in the network. For further simplicity of understanding, the learning mechanism by which backpropagation reduces MSE of the ANN (highest gradient descent) is analogous to the way a mountain climber can descend a hill just by selecting the steepest path down the hill at each point. The hill steepness and path that the climber has to select and go

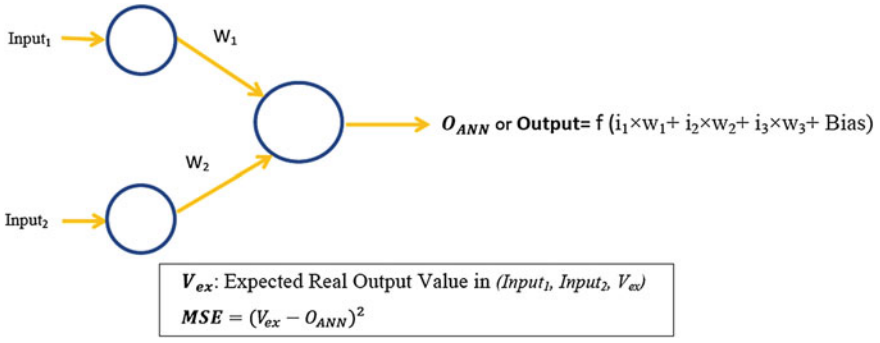


**Fig. 5** Example perceptron with 2 input neurons and 1 output neuron for MSE consideration

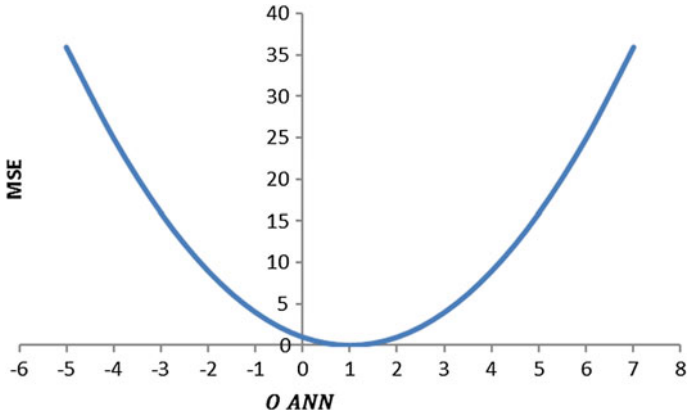


**Fig. 6** Ideal 2-D graph of (MSE) error versus predicted output by ANN ($O_{\mathrm{ANN}}$) considering a known data point of (2, 1, 1) as an example. The value of 1 is the expected real output. Thus, $\mathrm{MSE} = (1 - \boldsymbol{O}_{\mathrm{ANN}})^2$

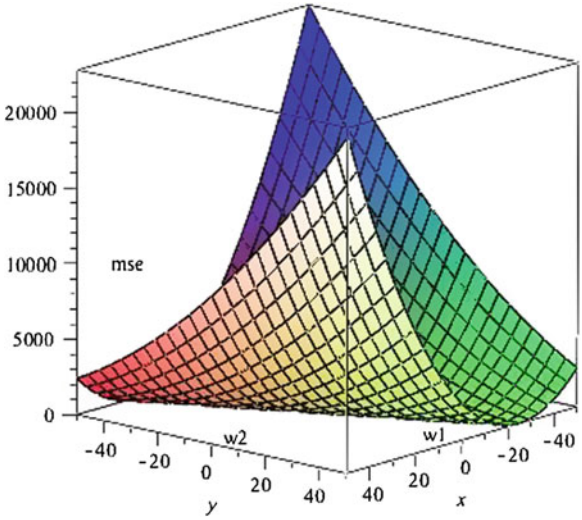


**Fig. 7** Ideal 3-D map of error surface (MSE) versus $x$ ($w_1$) and $y$ ($w_2$) considering a data point of (2, 1, 1) for training a perceptron with 2 input neurons

through at each point could be, respectively, considered as representative of the slope and gradient of the error surface at that point.

Ideally, it is assumed that only one global minimum exists in the error surface. But this is not necessarily the case (there may be a lot of local minima and also maxima in the error surface as shown in Fig. 8). In reality, backpropagation learning algorithm (with gradient descent) would finally converge to an error minimum which is actually a local minimum of error. Undoubtedly, this local minimum of error may not be necessarily global at all. All optimizing algorithms such as genetic

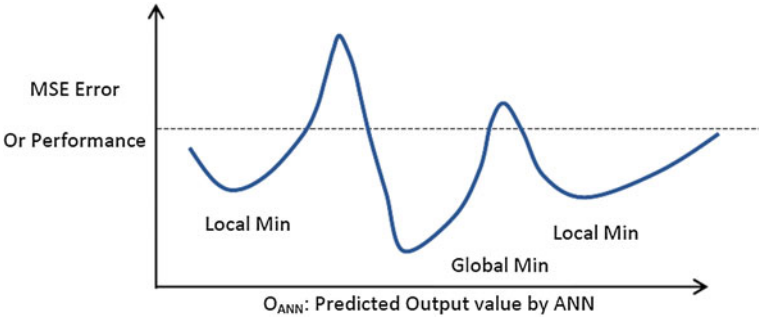


**Fig. 8** Backpropagation (with gradient descent error mitigation mechanism) can only find the local minimum of error, which is not necessarily the global minimum of error

algorithm, ant colony, particle swarm optimization have been designed to give the ANN backpropagation more capability to escape local error minima and reach global minimum of error.

If the initial point of the gradient descent process in backpropagation is somewhere between a local minimum and a local maximum, the training of the network would finally lead to the local minimum, which is not desirable to the user. This dependence of the ANNs which learn by backpropagation algorithm on the initial starting point is an important limitation to this algorithm (Fig. 8). Giving several different random initial values prior to each training is required to prevent trapping into local minima.

It is just to note that weights corresponding to the links of the network are the only variables that can be modified by the network to minimize the error. Still, to the authors' information, modification of ANN structure (number of neurons and hidden layers) with the objective of error reduction is not possible by the network itself, except by self-user's trial and error or automatic procedures.

As backpropagation is based on calculation of the MSE gradient with respect to all weights existing in the network, one of the requirements of the backpropagation is that differentiable activation functions should be used in neurons. This is the reason why sigmoid functions, as differentiable functions, are so popular in petroleum engineering and geoscience applications.

As said before, the MSE is found by:

$$\mathrm{MSE} = \frac{1}{2}(V_{\mathrm{ex}} - O_{\mathrm{ANN}})^2 \tag{3}$$

Please note that the ½ factor has been added so that the derivative has no coefficient: $\mathrm{MSE}' = (V_{\mathrm{ex}} - O_{\mathrm{ANN}})$.

For perceptrons (ANNs with no hidden layers), the activation function is linear or $O_{\mathrm{ANN}}$ is simply the weighted sum of the inputs as follows:

$$O_{\text{ANN}} = \sum_{i=1}^{n} W_i \times \text{Input}_i + \text{Bias} \tag{4}$$

where

$\text{Input}_i$ Input value to output neuron from neuron $i$

$W_i$ Weight corresponding to link $i$ (between input neuron $i$ and the output neuron)

For multilayer ANNs, $O_{\text{ANN}}$ is found after application of a nonlinear activation function as follows:

$$O_{\text{ANN}} = f(\text{Net}) = f\left(\sum_{i=1}^{n} W_i \times \text{Input}_i + \text{Bias}\right) \tag{5}$$

where

$f$ Activation function (usually a sigmoid function)

Since backpropagation applies gradient descent method for error reduction (as said before), the derivate of MSE gradient with respect to weights in the network is calculated utilizing the chain rule of partial derivatives as follows:

$$\frac{\partial \text{MSE}}{\partial W_i} = \frac{\partial \text{MSE}}{\partial O_{\text{ANN}}} \times \frac{\partial O_{\text{ANN}}}{\partial \text{Net}} \times \frac{\partial \text{Net}}{\partial W_i} \tag{6}$$

where

| | | |
|---|---|---|
| $\frac{\partial \text{Net}}{\partial W_i}$ | Rate of change of net value with respect to weight $i$<br>*Note* Net = $\sum_{i=1}^{n} W_i \times \text{Input}_i$ | $= \text{Input}_i$ |
| $\frac{\partial O_{ANN}}{\partial Net}$ | Rate of change of the output value from output neuron with respect to net value.<br>*Note* Activation functions ($f$) is normally considered as sigmoid functions (logistic or tangent hyperbolic sigmoid function):<br>$O_{\text{ANN}} = f(\text{Net}) = \frac{1}{1+e^{-\text{Net}}}$,<br>$f(\text{Net}) = \tanh(\text{Net})$ | $= \frac{\partial f}{\partial \text{Net}} = O_{\text{ANN}} \times (1 - O_{\text{ANN}})$<br>(for logistic $f$)<br>$= \frac{\partial f}{\partial \text{Net}} = (1 - O_{\text{ANN}})^2$ for tanh |
| $\frac{\partial \text{MSE}}{\partial O_{\text{ANN}}}$ | Rate of change of MSE with respect to the output value from output neuron.<br>*Note* $\text{MSE} = \frac{1}{2}(V_{\text{ex}} - O_{\text{ANN}})^2$ | $= V_{\text{ex}} - O_{\text{ANN}}$ |
| $\frac{\partial \text{MSE}}{\partial W_i}$ | Rate of change of MSE with respect to weight $i$<br>(weight corresponding to link from neuron $i$)<br>*Note* $\frac{\partial \text{MSE}}{\partial W_i} = \frac{\partial \text{MSE}}{\partial O_{\text{ANN}}} \times \frac{\partial O_{\text{ANN}}}{\partial \text{Net}} \times \frac{\partial \text{Net}}{\partial W_i}$ | $=(V_{\text{ex}} - O_{\text{ANN}}) \times O_{\text{ANN}}(1 - O_{\text{ANN}}) \times \text{Input}_i$<br>(for logistic f)<br>$=(V_{\text{ex}} - O_{\text{ANN}}) \times (1 - O_{\text{ANN}})^2 \times \text{Input}_i$<br>(for tanh) |

To summarize the above calculations, the gradient of MSE with respect to $W_i$ is equal to:

$$\frac{\partial \text{MSE}}{\partial W_i} = (V_{\text{ex}} - O_{\text{ANN}}) \times f^{'} \times \text{Input}_i \tag{7}$$

If the logistic sigmoid function is used for the activation function or $f$, we have:

$$\frac{\partial \text{MSE}}{\partial W_i} = (V_{\text{ex}} - O_{\text{ANN}}) \times O_{\text{ANN}}(1 - O_{\text{ANN}}) \times \text{Input}_i \tag{8}$$

If the *sigmoid hyperbolic tangent function* is used for the activation function ($f$), we have:

$$\frac{\partial \text{MSE}}{\partial W_i} = (V_{\text{ex}} - O_{\text{ANN}}) \times (1 - O_{\text{ANN}}^2) \times \text{Input}_i \tag{9}$$

$\Delta W_i$ is found by multiplying $\frac{\partial \text{MSE}}{\partial W_i}$ by the learning rate ($\alpha$). Thus, for a multilayer ANN, the value of weight change is equal to:

$$\Delta W_i = \alpha(V_{\text{ex}} - O_{\text{ANN}}) \times f^{'} \times \text{Input}_i \tag{10}$$

For the logistic sigmoid activation function, we have:

$$\Delta W_i = \alpha(V_{\text{ex}} - O_{\text{ANN}}) \times O_{\text{ANN}}(1 - O_{\text{ANN}}) \times \text{Input}_i \tag{11}$$

For the tangent hyperbolic sigmoid activation function, we have:

$$\Delta W_i = \alpha(V_{\text{ex}} - O_{\text{ANN}}) \times (1 - O_{\text{ANN}}^2) \times \text{Input}_i \tag{12}$$

For a perceptron ANN, $f$ is linear (or $f^{'}$ is equal to 1). Thus, the value of weight change is equal to:

$$\Delta W_i = \alpha(V_{\text{ex}} - O_{\text{ANN}}) \times \text{Input}_i \tag{13}$$

Please note that in the above calculations, for simplicity, the value of weight change (stated above) has been evaluated using only one data point. It is also reminded, in the MSE function yet considered, it was assumed that only one neuron exists in the output layer. Generally, mean square error (MSE) can be evaluated using:

$$\text{MSE} = \frac{1}{2}\sum_{k=1}^{T}\sum_{j=1}^{m}\left\{(V_{\text{ex},j}(K) - O_{\text{ANN},j}(K)\right\}^2 \tag{14}$$

where

$T$ Number of training samples from known data point given for training the network

$$(1)\ (\text{Input}_1, \text{Input}_2, \ldots, V_{\text{ex},1}, V_{\text{ex},2}, \ldots$$
$$(2)\ (\text{Input}_1, \text{Input}_2, \ldots, V_{\text{ex},1}, V_{\text{ex},2}, \ldots$$
$$\ldots$$
$$(T)\ (\text{Input}_1, \text{Input}_2, \ldots, V_{\text{ex},1}, V_{\text{ex},2}, \ldots$$

$m$ Number of output neurons or nodes

$V_{\text{ex},j}(K)$ Expected real value of output no. $k$ ($\text{Input}_1$, $\text{Input}_{1,\ \ldots,}$ $V_{\text{ex},1}(K)$

$O_{\text{ANN},j}(K)$ Predicted or estimated value by ANN

## 4.1 Perceptrons and Backpropagation Algorithm

The ANN structure, in which there are no hidden layers, is called perceptron. A percepron is indeed just like a simple neuron. Perceptrons are only applicable in linear simple problems. In perceptrons, only the simple bias/threshold activation function is utilized, namely by adding a bias/threshold value to the summation. A typical perceptron neural network (with no hidden layer) is shown in Fig. 9.

It is noted that training is usually performed in an iterative manner. An *epoch* is indeed the process of providing the network with the entire training data points, calculating the network output and error function, and modifying the network's weights for the next epoch. An epoch is composed to several iterations (providing or presenting one data point to the network can be considered as an iteration). Based on complexity of the problem and the ANN, sometimes a large number of epochs are required for training the network.

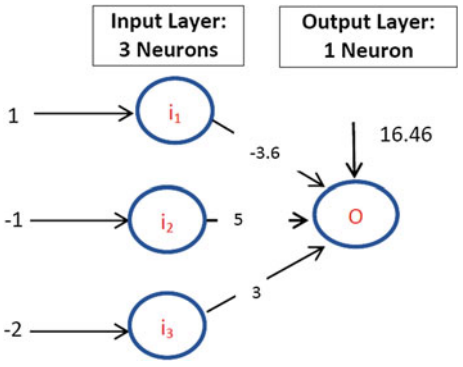


**Fig. 9** A typical perceptron as an ANN example

Indeed, when the input data are presented to the network, the flow of information is forward (FF-ANN). However, as said previously, backpropagation is the backward error propagation of weight adjustments.

The calculations corresponding to the composing neurons of the perceptron in Fig. 9 are shown below.

In each one epoch, the net value (summation plus the bias) and final output value of the neuron (O) are calculated by the perceptron by the ANN as follows:

$$\text{Output} = 1 \times (-3.6) + (-1) \times 5 + (-2) \times 3 + 16.46 = 1.86$$

Now, suppose that the real expected output value is equal to 1. If so, in each next epoch for each data point, the weight values are changed such that the output value gets nearer to the real expected value. This process continues until nearest possible output values to the real expected values are obtained. In this simple ANN, the modified weights are evaluated as follows:

$$\Delta W_i = \alpha(V_{\text{expected}} - O_{\text{ANN}}) \times \text{Input}_i$$

$$W_{i,\text{modified}} = W_{i,\text{previous}} + \boldsymbol{\alpha}(V_{\text{expected}} - O_{\text{ANN}}) \times \text{Input}_i \tag{15}$$

where

| | |
|---|---|
| $\Delta W_i$ | Magnitude of change of the weight corresponding to link $i$ (between neuron $i$ and output neuron) |
| $\boldsymbol{\alpha}$ | Learning rate (a constant) |
| $V_{\text{expected}}$ | Real expected output value corresponding to a specified input data point (already known) |
| $O_{\text{ANN}}$ | Evaluated value by whole ANN using specified input data points (the output value of the output neuron) |
| $\text{Input}_i$ | Input value to output neuron from neuron $i$ |
| $W_{i,\text{previous}}$ | Value of the weight in the previous epoch |

Thus, assuming $\boldsymbol{\alpha}$ (called learning rate) to be equal to 1, the modified weights are as follows:

$$W_{1,\text{mod.}} = -3.6 + 1 \times (1 - 1.86) \times 1 = -4.46$$
$$W_{2,\text{mod.}} = 5 + 1 \times (1 - 1.86) \times (-1) = 5.86$$
$$W_{3,\text{mod.}} = 3 + 1 \times (1 - 1.86) \times (-2) = 4.72$$

Thus, after training the network during one epoch using the data point {(1, −1, −2) as input and 1 as output}, the above modified weight values would be used in the next epoch.

## 4.2 Multilayer ANNs and Backpropagation Algorithm

For more complex problems, normally the ANN should have hidden layers. These neural networks are called multilayer ANNs. Multi-layer ANNs could be considered as a developed or extended perceptron. The structure of a multilayer ANN with one hidden layer has been shown in Fig. 10. Application of multilayer ANN is not only restricted to very simple linear problems, but can also utilized for complex or non-straightforward problems.

Unlike perceptrons, in multilayer ANN, the activation function is not just a bias/threshold, but activation functions (usually sigmoid functions) are usually applied. As could be seen in Fig. 10, the ANN structure is composed of one input layer (consisting of 3 neurons), one output layer (consisting of one neuron), and also one hidden layer (consisting of 2 neurons) as a typical example. For most engineering purposes, only one or 2 hidden layers at most are normally adequate to be used in the structure. The number of neurons in the hidden layers is crucial in the performance and functionality of the network. However, depending on the complexity of the problem, an optimization should be made not to use too many neurons in the hidden layers as this causes over-fitting or over-training.

For the multilayer ANN shown in Fig. 10, in each one epoch, the net value and the output of each neuron in the hidden and output layers are calculated by the ANN as follows:

$\mathbf{H_1}$: the net value and output of this neuron are found as follows:

$$\text{Net} = 1 \times (-3.6) + (-1) \times 5 + (-2) \times 3 + 16.46 = 1.86$$

$$\text{Output} = \frac{1}{1 + e^{-1.86}} = 0.86$$

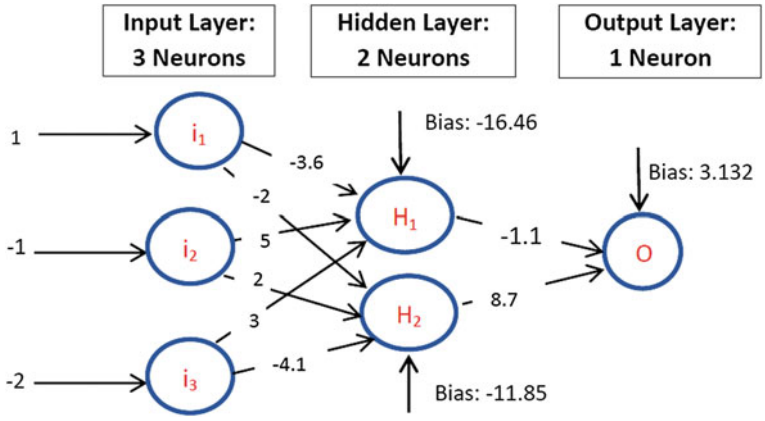


**Fig. 10** A multilayer ANN with one hidden layer (a logistic sigmoid function has been considered as the activation/transfer function)

**$H_2$**:

$$\text{Net} = 1 \times (-2) + (-1) \times 2 + (-2) \times (-4.1) - 11.85 = -7.65$$

$$\text{Output} = \frac{1}{1 + e^{-(-7.65)}} = 4.75 \times 10^{-4}$$

**O**: the net value and output of this neuron (which is the output of the ANN) are found as follows:

$$\text{Net} = 0.86 \times (-1.1) + 4.75 \times 10^{-4} \times 8.7 + 3.132 = 2.187$$

$$\text{Output} = \frac{1}{1 + \text{e}^{-2,187}} = 0.9$$

Now suppose that the real expected value is equal to 1. If so, in each next epoch for each data points, the weight values are changed or modified such that the output value gets nearer to the real expected value (here equal to 1). This process continues until nearest possible output values to real expected values are obtained. In this multilayer ANN, the modified weights are evaluated as follows:

$$\Delta W_i = f'_{\text{output}} \times (V_{\text{expected}} - O_{\text{ANN}}) \times \text{Input}_i$$

Replacing the *logistic sigmoid activation function* derivative for the output neuron yields:

$$\Delta W_i = \alpha O_{\text{ANN}}(1 - O_{\text{ANN}}) \times (V_{\text{expected}} - O_{\text{ANN}}) \times \text{Input}_i$$

The modified weight for this sigmoid function is as follows:

$$W_{\text{modified}} = W_{\text{previous}} + \alpha O_{\text{ANN}}(1 - O_{\text{ANN}}) \times (V_{\text{expected}} - O_{\text{ANN}}) \times \text{Input}_i \quad (16)$$

Replacing the *hyperbolic tangent sigmoid activation function* derivative for the output neuron yields:

$$\Delta W_i = \alpha(1 - O_{\text{ANN}})^2 \times (V_{\text{expected}} - O_{\text{ANN}}) \times \text{Input}_i$$

The modified weight for the *hyperbolic tangent sigmoid function* is as follows:

$$W_{\text{modified}} = W_{\text{previous}} + \alpha(1 - O_{\text{ANN}})^2 \times (V_{\text{expected}} - O_{\text{ANN}}) \times \text{Input}_i \quad (17)$$

where

$O_{\text{ANN}}$ Evaluated value by whole ANN using specified input data points (the output value of the output neuron). This is fixed value for each epoch

| | |
|---|---|
| $f'$ | The derivative of the activation function corresponding to the output neuron |
| $\text{Input}_i$ | Input value of the neuron $i$ |
| $W_{i,\text{previous}}$ | Value of the weight corresponding to link $i$ in the previous epoch |
| $W_{i,\text{modified}}$ | Value of the weight corresponding to link $i$ to be used for the next epoch |

*Input* value is the value enters to each neuron. If the neuron is in the hidden layer, the *input* value is the **net** value (summation plus bias) to the neuron. In perceptrons (without hidden layer ANN) which are applicable to simple linear problems, the activation function is indeed $f(x) = \alpha x$. Therefore, applying the above relations to perceptrons would yield $f'$ to be equal to 1 or $\alpha$ (15).

Thus, assuming $\alpha$ (called learning rate) to be equal to 1, the modified weights are evaluated as follows:

$$W_{\text{modified}} = W_{\text{previous}} + \alpha O_{\text{ANN}}(1 - O_{\text{ANN}}) \times (V_{\text{expected}} - O_{\text{ANN}}) \times \text{Input}_i$$

$$W_{I1-H1} = -3.6 + 1 \times 0.9(1 - 0.9) \times (1 - 0.9) \times 1 = -3.591$$

$$W_{I2-H1} = 5 + 1 \times 0.9(1 - 0.9) \times (1 - 0.9) \times (-1) = 4.991$$

$$W_{I3-H1} = 3 + 1 \times 0.9(1 - 0.9) \times (1 - 0.9) \times (-2) = 2.982$$

$$W_{I1-H2} = -2 + 1 \times 0.9(1 - 0.9) \times (1 - 0.9) \times 1 = -1.991$$

$$W_{I2-H2} = 2 + 1 \times 0.9(1 - 0.9) \times (1 - 0.9) \times (-1) = 1.991$$

$$W_{I3-H2} = -4.1 + 1 \times 0.9(1 - 0.9) \times (1 - 0.9) \times (-2) = -4.118$$

$$W_{H1-O} = -1.1 + 1 \times 0.9(1 - 0.9) \times (1 - 0.9) \times 1.86 = -1.083$$

$$W_{H2-O} = 8.7 + 1 \times 0.9(1 - 0.9) \times (1 - 0.9) \times (-7.65) = 8.631$$

Thus, after training the network during one epoch using the data point {(1, −1, −2) as input and 1 as output}, the above modified weight values would be used in the next epoch. This process takes place for all the data points given to the network in the training stage.

It is noted that if the learning rate is too low, the learning process would be too slow. If the learning rate is too high, the weights and objective function would diverge and no good learning would occur. In linear problems, proper learning rates could be computed using the Hessian matrix (Bertsekas and Tsitsiklis 1996). It is also possible to adjust the learning rate while training. A number of proposals exist in the neural network literature to adjust the learning rate while training. However, most of them are not effective. Among these works, Darken and Moody (1992) could be named.

Please note that the ANN structures discussed above are feed-forward ANN (FF-ANN) with backpropagation algorithm for weight modification. This has been very popular in petroleum engineering practices. Thus, these types of neural networks are further discussed as follows.

## 5 Data Processing by ANN

Typically, processing of ANN data is performed in three sections including training, testing/calibration, and validation/verification phases. For this purpose, the number of the known data points comprising input and output values is divided into two categories. In most of the petroleum engineering practices performed by the authors, typically the number of available data points (inputs and output values available) is divided into two independent parts: (1) 60 % of the data points for utilization in the training phase and (2) 40 % of the data points for utilization in testing. Some users test the trained neural network in two stages: (a) 20 % of the data points for first testing of the trained network and (b) 20 % of the data for second testing purposes. Depending on the number of data points available and experience of the users (which is very important), different users adopt different division of the data. Some may take 70 % of the available data points for training and the rest for testing. The number of the data points plays an important role in successful training. Fake data points could cause trouble for successful training particularly if their number is many.

Then, if testing gives promising results, the user can go for validation or verification phase. In this phase, some new input data is applied to the trained ANN to get the network outputs. The network outputs are thus considered to be very near to the real ones and could be used for instance in petroleum engineering.

Indeed, during the training phase, the desired network is developed. Then, it is tested. When the training and testing process has been finished successfully (after training and simultaneously testing the network by test data points), the network is applied to the validation data points in order to predict outputs using new input data.

### *5.1 Training*

As one of the main similarities between artificial neural networks (ANNs) and biological neural networks, both have the ability to learn or to be trained. As said earlier, the output of a neuron is a function of the net value (which is the weighted sum of the inputs plus a bias). Indeed, after successful training, an ANN can have the capability of creating reasonable outputs using new inputs (during testing and validation phases). The more reasonable the neural networks respond to the new data, the neural networks has got higher functionality in terms of generalized prediction. Good training is thus of great importance to enhance the functionality of ANN.

In the beginning, the ANN allocates a random weight to each input value. Then, during the training process, when the inputs are introduced to the network, the weights (corresponding to the links between all the neurons of the network) would be adjusted or modified such that finally the ANN output(s) is/are very near to the expected real output value(s). Depending on how close the output created by ANN is to the expected real output, the weights between the neurons are modified such that if the same inputs are introduced to the network, the network would provide an output pattern closer to the expected real output.

Surely, if the difference between the created data (by ANN) and real data is considerable, more modification of the weights is performed during training. In this way, at any time in training, a kind of memory is attached to the neurons which store the weights in the past computations. Finally, if convergence is attained after training, we expect the ANN to give outputs which are in proximity to expected real outputs. In Fig. 9 (perceptron) and the corresponding calculations, it was previously shown how the weights were modified ($W_{1,\text{mod}}$, $W_{2,\text{mod}}$, $W_{3,\text{mod}}$) during training.

In more details, it could be stated that the training or the learning stage of the network is accomplished by summing up the errors at the output and creating a cost/risk function in terms of network inputs and weights and minimizing the cost function with respect to the network inputs. The cost function is basically based on mean squared error (MSE) of the outputs. The process of MSE error mitigation during training takes place in an iterative process during many iteration times. Each iteration, which is indeed the process of providing the network with inputs and modifying the network's weights, is called an epoch. Normally, a lot of epochs are required to train a typical ANN.

Training continues as said above until the created output value or pattern complied with the quality criteria based on statistical error values. In other words, the stopping criteria is based on the minimization of MSE (Cacciola et al. 2009). This type of training wherein both inputs and actual outputs are supplied to the network (called Supervised Training) is usually more common in practice. In unsupervised training, only input values are supplied and ANN adjusts its own weights such that similar outputs are given out of the network while inserting similar inputs. Most of the neural network applications in the oil and gas industry are based on supervised training algorithms (Mohaghegh 2000).

## *5.2 Over-fitting*

In iterative backpropagation ANNs which are commonly used in petroleum industry, there is a serious problem which arises after too much training of the network. It is important to know when to stop training the network and go for testing and validation phases. As a matter of fact, too much training would cause over-fitting or over-training. Over-fitting is also called memorization as well because in this case, the network would indeed memorize the data points used in training and give a very accurate match with them, but it would lose its

generalization capability which is required in testing and validation phases. Please note that over-fitting does not really apply to ANNs which are trained using non-iterative processes. However, it can be said that ANNs used in petroleum are normally iterative.

Indeed, during the initial phase of training, both the training and validation set errors show a decreasing trend with time or number of epochs. It is noted that training and validation set errors are, respectively, errors corresponding to training and validation data points. Nevertheless, when the over-fitting of the ANN starts, the validation error suddenly starts to increase, while the training error still continues its decreasing trend (Fig. 11). The blue dots in Fig. 12 illustrate the train data points ($X$, $Y$). The blue and red lines in Fig. 12, respectively, show the true functional relationship and the ANN learned function. As can be seen, due to too much training, there is a large difference between the true functional curve and the ANN learned function in points other than train data points. This is representative of over-fitting. The input values have been shown in the $x$-axis for simplicity, and the output values have been denoted by the values in the $Y$-axis. The saved ANN weights and biases/thresholds are corresponding to the minimum of the validation set error.

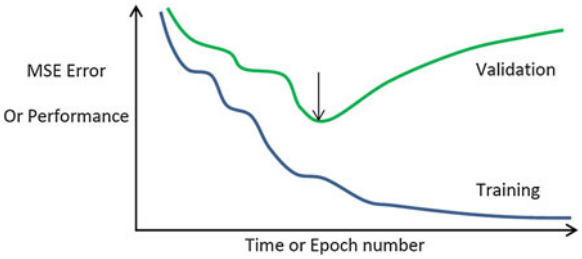


**Fig. 11** The sudden increase of validation set error (*black arrow*) shows over-fitting in the graph of MSE versus time/epoch number

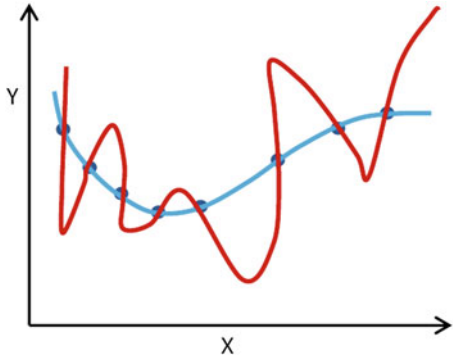


**Fig. 12** Over-fitting. Too much complexity of the ANN due to too much learning (*red curve*) has caused the ANN estimated values by ANN learned function at inputs other than train values to be very different from true functional relationship or curve (*blue curve*). $x$-axis and $y$-axis are, respectively, representative of inputs and ANN outputs

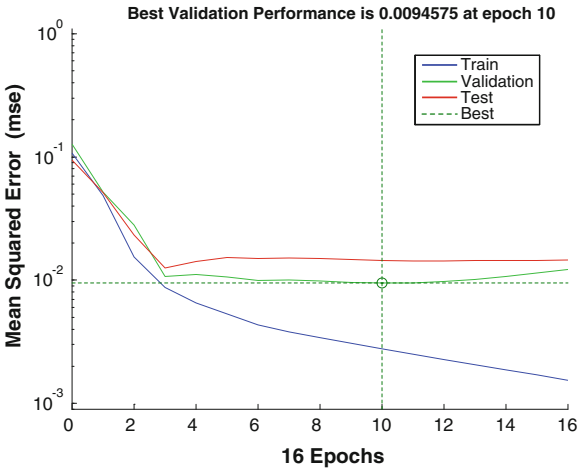


**Fig. 13** At epoch no. 10, the error of second test or validation error (MSE as its set error function) and also first test set error increased, while training set error is still decreasing. But, as the increase of the validation set error is not too much sharp, it is no sign of over-fitting

Figure 12 illustrates the problem of over-fitting in machine learning. The blue dots represent training set data. The blue line represents the true functional relationship, while the red line shows the learned or trained function, which is shown to be under the effect of over-fitting.

It is noted that if the test set error had shown a considerable increase before the validation set error, it could be guessed over-fitting could have happened. Thus, in Fig. 13, for instance, there are no worries about over-fitting until stopping point at epoch number 6. But surely after epoch 6, again over-fitting would happen. The minimum validation error is at epoch no. 6, but the training has continued 6 more epochs until epoch number 12 (the user has instructed the network do so). Thus, it is possible to rely on the ANN weights and biases adjusted based on stopping point at epoch 6. Just to remind that as times passes by (epoch increase), the ANN is getting more and more complex.

In order to prevent over-fitting, first it is required that the network is not too much complex that models even the noises. Second, before the termination of the training process, the process is commonly stopped from time to time so that the network generalization capability is checked using the test data points. If the network performed well in the test phase (if the MSE in the test phase is in the magnitude of 0.001 or a little higher), no further training is required. Since the outputs of the test data points are not utilized in the training phase, the network prediction or generalization capabilities could be analyzed by comparing the network outputs (using the input values in testing phase) with the real test output values.

## *5.3 Testing*

After the training stage, we reach to the testing or calibration stage wherein the weights between the neurons are tested. Testing takes place using input–output

pairs that have not been utilized in the training stage. The difference between the desired and the actual output can show if enough training has happened. It is noted that it is usually a common fault among users with not enough experience to test/calibrate the ANN using the same data points used for training.

### 5.3.1 Validation

Validation is the last stage in the ANN application or neural data processing by ANN. To summarize the data processing by ANN, during the training phase, a trained network is developed. Then, it is tested. If it shows good performance during testing, it is a recognized as a proper network to be used with new data. During validation, the user can apply the trained network to obtain results. In this stage, new input data (with unknown outputs) is supplied to the trained ANN in order to obtain the network outputs. As the trained network has performed well in the testing stage, the outputs obtained in the validation stage are considered to be trustworthy and are taken for granted as nearest to real.

It must be noted that the validation stage is usually considered as the second test. If so, the validation and test error curves versus the number of epochs are plotted along with the training error curve. The validation and test error curves should also have the declining trend just like the training error curve before convergence. After convergence, the validation and test error curve would rise, while training error curve would continue declining showing too much training (over-fitting). Therefore, if the training curve is declining while the validation and test ones are rising and do not show a decline before convergence, it indicates a problem with the network. To help remove the problem with the network, it is suggested to follow the steps below:

1. Re-check the data: Among the data points, there might be some wrong or faulty ones. Try to detect them and delete them from the available data points.
2. Reconsider the effective parameters: Try to consider one or more new input parameter whose effect on the output parameter might have been ignored. This requires a review of the problem and identifying the effective parameters to find its corresponding values.

According to author's experience, the error was reduced dramatically using the above two methods.

Sometimes, the output is not dependent on some of the data or in other words its corresponding sensitivity is very small. This case can be found by sensitivity test analysis.

Please note that normalizing the data can also help to obtain better results from the network performance. Normalizing means arranging all the available data values in the range of 0–1. By knowing the maximum and minimum value among of the data, normalizing is possible. Thus, the normalized value is found by the following relation:

$$\delta = \frac{d - d_{\min}}{d_{\max} - d_{\min}} \tag{18}$$

It is noted that some users consider the MSE error in the validation phase as the model quality. Some others may consider the error in the test phase.

## 6 ANN and Validation Error Statistical Parameters

As mentioned in the text, MSE is the main error function to compare the neural network models and performance and in more detail the discrepancy between the expected real values and the output by ANN. Additionally, several other statistical parameters are also utilized to report this discrepancy. Collectively, these error parameters include mean squared error (MSE), average root mean squared error (ARMSE), average percent error (APE), average absolute percent error (AAPE), Pearson correlation coefficient (R), squared Pearson correlation coefficient ($R^2$), standard deviation (SD), and variance (V). It is noted that variance is simply the square of standard deviation.

It is to note that $R^2$ can be used along with R to analyze the network performance. MSE and $R^2$ collectively can give an indication of the performance of the network. Generally, a $R^2$ value greater than 0.9 indicates a very satisfactory model performance, while a $R^2$ value in the range 0.8–0.9 signifies a good performance, and the value less than 0.8 indicates a rather unsatisfactory model performance (Coulibaly and Baldwin 2005).

Definitions and mathematical relations of these parameters are given in the Appendix.

## 7 Sequential Forward Selection of Input Parameters (SFS)

As said before, one of the challenges in neural network modeling is the determination of the important effective input parameters on the output. Certainly, experience and knowledge of the problem could help in determination of the effective parameters or selection criterion. However, it is better to utilize a systematic method to determine the impact of each parameter, rank them. Using sequential forward selection (SFS), each of the input parameters is considered individually. Then, the input parameters with the highest impact on the output parameters are detected in successive stages so that ranking of the parameters could be performed on the basis of highest to lowest impact. For more clarity of the procedure, an example is given below:

Suppose 5 input parameters denoted by *a* to *e* have been characterized as important parameters of a problem and pressure is the output parameter. To utilize SFS for

ranking of the parameters which have the greatest impact, in the first stage, 5 neural network models are constructed each just using one single input parameter. To compare their performance, the errors of all these single input parameters are evaluated. As it is illustrated in Fig. 14, using single parameter *c*, for instance, network creation leads to least error of all. Thus, parameter *c* is the most effective input parameter or with highest impact on output parameter which is bottom hole pressure (BHP).

In the second stage, 4 neural networks using 2 input parameters are constructed such that input parameter c is surely considered (fixed parameter) and the second parameter is changed among the 7 remaining parameters. In this way, 4 neural networks are constructed. Now, their corresponding errors are calculated. Out of these 4 networks, the second parameter of the network with the least error is selected as the second most effective parameter. For instance, this could be parameter no. *e* (Fig. 15).

At the third stage, 3 neural networks are constructed using 3 input parameters such that input parameters *c* and *e* as the first two most influential parameters are considered (fixed parameters) and the third input parameter is changed among the 3 remaining parameters. The above SFS process is continued until the ranking of all the input parameters is performed (Fig. 16). Collectively, 15 neural networks will

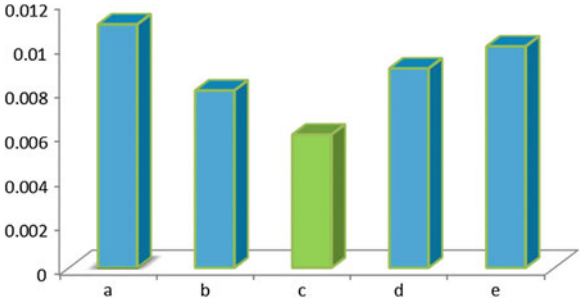


**Fig. 14** 1st stage of SFS. The MSE error corresponding to the 5 neural networks each just trained by a single input parameter (*a* to *e*). The input parameter with the most impact is *c*

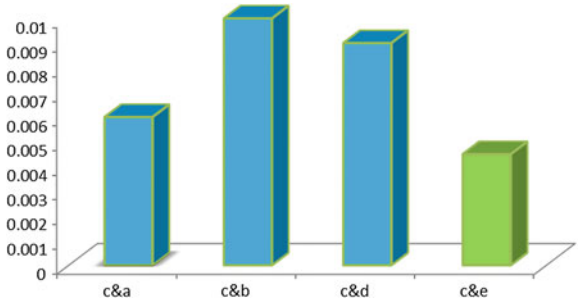


**Fig. 15** The MSE error corresponding to the 4 neural networks each just trained by a single input parameter (*a* to *e*). The second most effective parameter is *e*

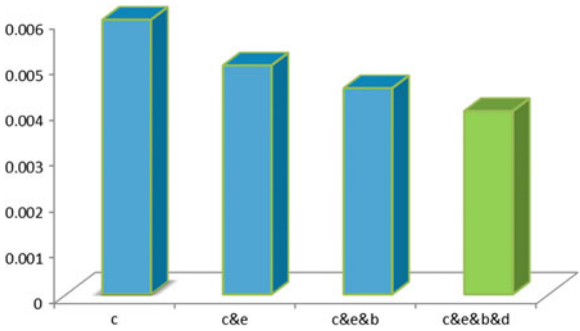


**Fig. 16** Final ranking of input parameters

have been considered to know the final ranking. After finding the input parameters with maximum to minimum effect, the errors of five neural networks corresponding to each stage of SFS (first constructed with the single most effective input parameter, second constructed with two most effective input parameter, etc.) are compared with each other. Finally, Fig. 16 is yielded which shows how adding input parameters could reduce error.

## 8 Applied Examples of ANN in Geoscience and Petroleum Engineering

Among the many applications of ANN in geoscience and petroleum engineering, some applied examples are given as follows.

### *8.1 Multiphase Flow*

Among really complex problems, multi-phase flow problems could be considered. One important parameter in two phase flow in wells is the bottom hole pressure (BHP). In underbalanced drilling (UBD), the BHP should be kept less than formation pore pressure and above wellbore collapse pressure (to prevent wellbore instability). Thus, the prediction capability of predicting BHP in UBD operations is of great importance which could leave the necessity of measuring devices bottom hole. Different approaches have their own defects. For the following reasons, the errors of the mentioned approaches are not small:

Because of the complexity of the phenomenon, indeed no analytical solutions exist for multi-phase problems. Empirical multi-phase flow correlations sometimes over-predict, make extrapolations risky, and are susceptible to uncertainties.

The use of mechanistic modeling approaches has been increasing in multi-phase flow problems in pipes; however, utilizing them in annular flow problems has not been very promising.

Taking into account the lack of analytical solutions in multi-phase problems, artificial neural network was utilized to evaluate BHP in multi-phase annular flow while UBD operations (Ashena et al. 2010). Thus, BHP was considered as the output parameter. An attempt was made to find the parameters which have influence on the output parameter (BHP) or BHP depends on. This is obtained just by a brief studying of multi-phase flow relations and also experience. Seven parameters were found as effective parameters:

Rate of liquid or diesel injection (in gallon per minute or gpm)
Rate of gas or nitrogen injection (in standard cubic foot per minute or scfm)
Measured depth or MD (in meter)
True vertical depth or TVD (in meter)
Inclination angle from the vertical (in degrees)
Well surface pressure (in psi)
Well surface temperature (in degrees celsius)

Qualitatively speaking, it must be noted that the more the value of item 1 (rate of liquid injection), the more the value of BHP. The more the value of item 2 (rate of gas injection), the less the value of BHP. Please note that as items 3–5 (measured depth, true vertical depth, and inclination angle) collectively take the hole depth in vertical and directional wells into account, they were considered as input parameters. Normally, the neural network structure is itself responsible for recognizing the extent of the input parameter strength or importance.

The number of 163 data points with the above 7 input parameters and measured BHP (the only output parameter) was collected. About 10 data points were found to cause too much error. After their deletion, the ANN performance was enhanced.

A 153 collected data points should be divided for use in training and testing phases. 60 % of them were allocated to training, 20 % to testing-1, and 20 % to validation (or testing-2) purposes. Some additional data have been used as simulation data with known measured values as well. As the number of the available collected data points was limited, only limited number of neurons and layers was tried to be utilized. Feed-forward ANN with backpropagation algorithm of learning was set as the neural network. Input data were inserted from the MATLAB workspace. All the data values were normalized (with the range between 0 and 1).

The training function was set as Levenberg–Marquardt algorithm (trainlm). It is noted that Levenberg–Marquardt algorithm makes an interpolation between the Gauss–Newton algorithm and the gradient descent method. MSE was selected as the performance or error function. The hyperbolic tangent sigmoid function (tansig) was selected as the transfer or activation function. We could have also selected the logistic sigmoid function (logsig). According to the authors' experience, it would be good to consider 2–2.5 times the number of the inputs as the number of neurons in the hidden layer as one option. In this study, it was shown that the case with 3 layers (2 hidden layers) has the least error and hence is used to simulate the input

data. However, normally 1 hidden layer can meet the engineering needs. In this study, as the number of layers was increased, the error was decreased. As can be seen from case 3 to 4 (Table 1), since the number of data points to train the network, 92, is not that many, increasing the number of neurons may not necessarily decrease the MSE. It is just to note that it is required to first reinitialize the weights at each time of running neural networks.

In Fig. 17, the value of MSE versus the number of epochs for one case of the above example has been given. It is shown that the value of error (MSE) of training, first testing (test), and second testing (as only named validation in Fig. 17) is decreasing with the number of epochs before convergence is reached. Upon convergence, the errors of one of the tests or both start increasing, while the training error still decreasing. The criterion for MSE calculation is the error value at the convergence (MSE: 0.007 for the case given).

Please note that if the starting increase of the validation (or second test) and test (or first test) MSE curves is not too much sharp at the convergence point, the neural network performance is considered to be valid. In Fig. 17, at the convergence point (epoch 3), the starting increase in the validation (second testing) MSE is not too sharp.

In Fig. 18, the values of the Pearson coefficients (R) for the training, first testing (test), and second testing (validation) have been shown. The *y*-axis and *x*-axis, respectively, show the predicted outputs by ANN and the expected real values (targets). The closer these values are to the value 1, the better the indication of convergence. In our example, these values are good enough (about 1) and thus

**Table 1** Different ANN structures used in the example and the validation error

| ANN | Type | No. of hidden layers | AAPE % | APE % | SD | *V* | *R* | $R^2$ | MSE |
|---|---|---|---|---|---|---|---|---|---|
| Case-1 | FF-BB | 1 with10 neurons | 20.99 | 20.99 | 30.46 | 927.81 | 0.8872 | 0.787124 | 0.007 |
| Case-2 | FF-BB | 1 with 20 neurons | 18.55 | 11.08 | 30.12 | 907.21 | 0.9134 | 0.8343 | 0.006 |
| Case-3 | FF-BB | 1 with 33 neurons | 26.7 | 9.73 | 93.99 | 8834.1 | 0.8982 | 0.806763 | 0.003 |
| Case-4 | FF-BB | 1 with 42 neurons | 16.19 | 12.15 | 33.06 | 1093 | 0.8625 | 0.743906 | 0.005 |
| Case-5 | FF-BB | 2 with 18 and 18 neurons | 18.87 | 18.67 | 47.2 | 2227.8 | 0.9313 | 0.86732 | 0.003 |
| Case-6 | FF-BB | 2 with 26 and 26 neurons | 15.81 | 14.59 | 36.15 | 1306.8 | 0.9337 | 0.871796 | 0.004 |
| Case-7 | FF-BB | 2 with 32 and 32 neurons | 16.99 | 16.96 | 18.97 | 359.86 | 0.9375 | 0.878906 | 0.003 |

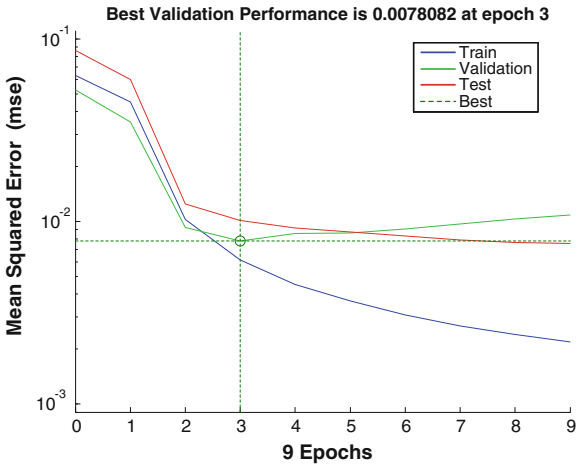


**Fig. 17** MSE versus number of epochs (MSE: 0.007)

show good convergence. This graph is an important indication of the validity of the trained neural network. Suppose only the $R^2$ corresponding to training was near to 1 (e.g. 0.9), and the first and second test values were not (e.g. 0.4). If so, the neural network performance was weak and validity of the work was under question.

To analyze the performance of different neural network models, several statistical parameters were utilized. These parameters include average absolute percent error (AAPE), average percent error (APE), average root mean squared error (ARMSE), Pearson correlation coefficient ($R$), squared Pearson coefficient ($R^2$), standard deviation (SD), and variance (V) as shown in Table 1. In the Appendix, the required statistical parameters have been described.

The trained ANN above, which has been validated successful after two testing, could be used for simulation using new input data.

## *8.2 Well Hydraulics*

Drilling hydraulics simulation is indeed a non-straightforward problem which is more sophisticated in complex wells (slim holes and extended reach wells). There are many unknowns in this problem. As simulation by hydraulics simulators is time consuming, they are not suitable for application in real-time drilling.

Because of the above reasons, Fruhwirth et al. (2006) utilized a number of 11 generations of a special ANN called by them as completely connected perceptron (CCP) for prediction of pump pressure or hydraulic pressure losses.

It is noted that CCP is a more general type of perceptron which could be considered as multilayered as seen in Fig. 19. In each generation, one hidden layer was added to the CCP (first generation without any hidden layer and the 11th generation with 10 hidden layers). Out of the available data, 50 % was devoted to

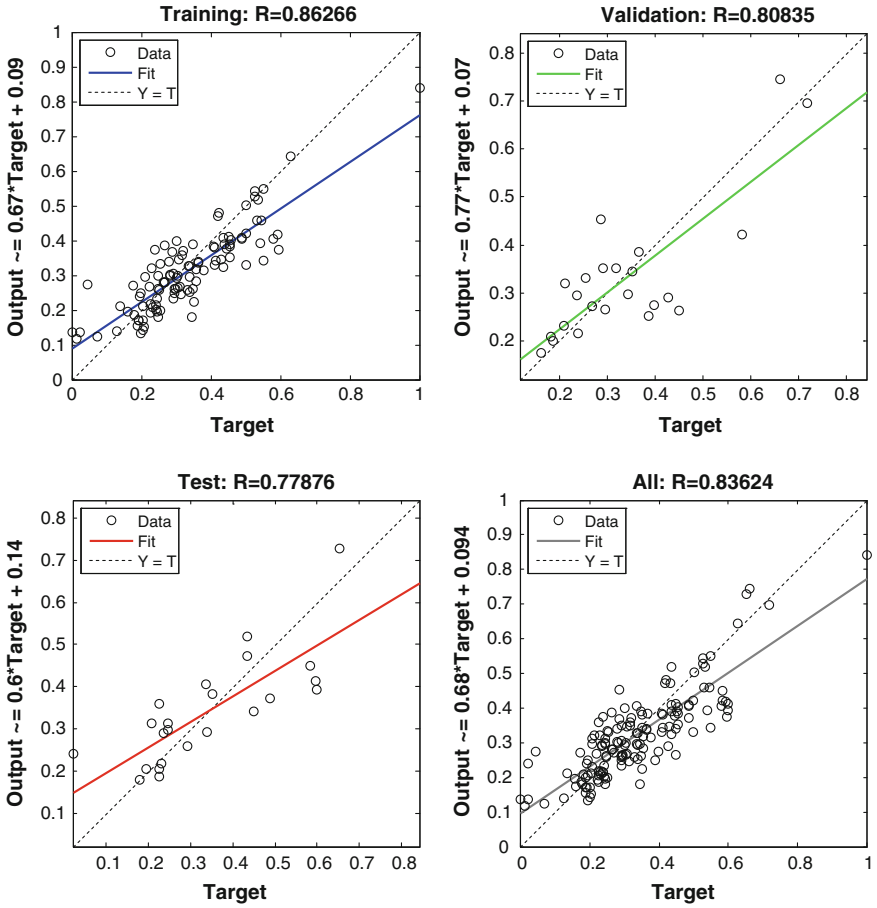


**Fig. 18** The Pearson coefficients for the training, testing 1(test), and testing 2 (validation). The *y*-axis and *x*-axis, respectively, show the predicted outputs by ANN and the expected real values (target). The values are near to 1 and thus ANN has good valid performance

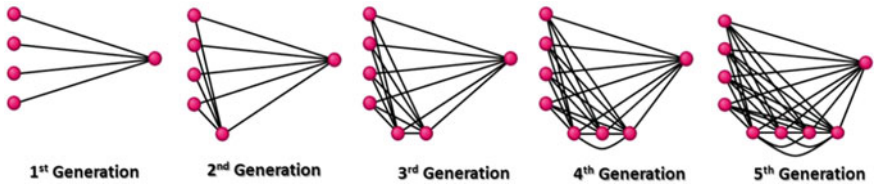


**Fig. 19** Growing completely connected perceptron (cVision Manual)

training. Then, 25 % of the data points were allocated to each test or validation. The real data of two wells were utilized for training the ANN. The input drilling parameters considered include bit measured depth (MD in m), bit true vertical depth (TVD in m), block position (in m), rate of penetration (ROP in m/hr), average mud flow rate (in $m^3/s$), average hook load (in kg), average drill string revolutions (RPM), average weight on bit (WOB in kg), and average mud weight out of hole (in $kg/m^3$). Average pump pressure (in bar) was considered as the output parameter.

Utilization of a completely connected perceptron (CCP) has the advantage of eliminating the need to find an optimal number of hidden layers (Fruhwirth et al. 2006). The schematics of different generations of ANN are shown in Fig. 20.

The results of modeling by ANN were promising. As seen in Fig. 21, the RMS error (in bar) is low enough, and there is not much change in the error from a generation on (the number of hidden layers does not have much effect from a number on). It is reminded that pump pressure is the output parameter. In Fig. 22a, the measured, calculated pump pressures by ANN (as the output) and also the corresponding error versus time have been shown for one of the wells. In Fig. 22b, a good match of the data could be observed in the cross-plot of the measured (expected real data) and the calculated (or predicted) output by ANN.

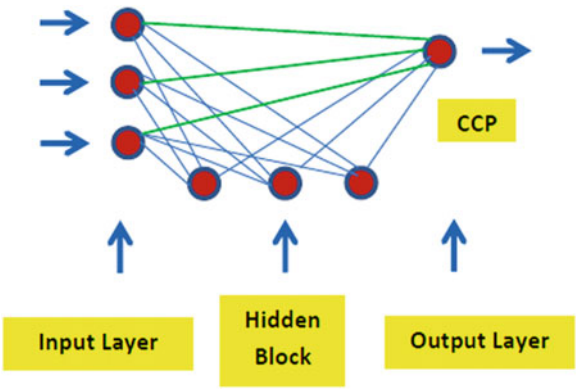


**Fig. 20** Schematics of completely connected perceptron (cVision Manual)

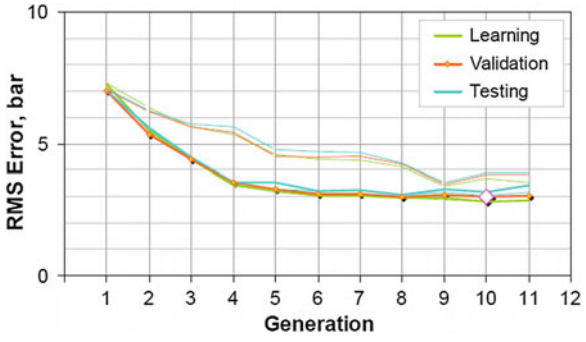


**Fig. 21** The RMS error corresponding to learning or training, test and validation phases (Fruhwirth et al. 2006)

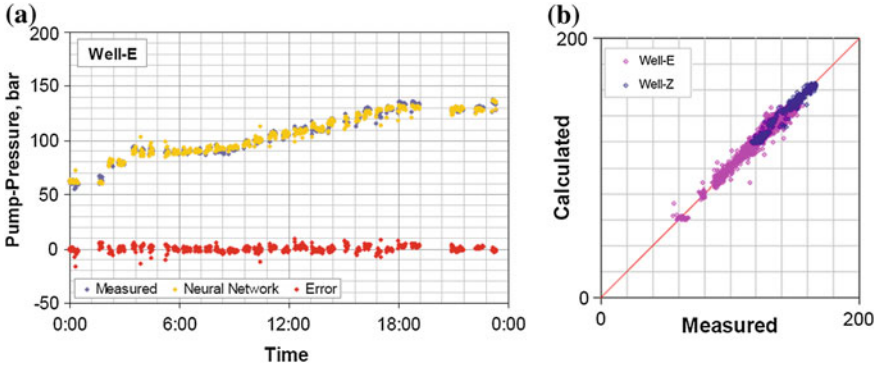


**Fig. 22** **a** Measured, calculated by ANN pump pressure and error versus time and **b** cross-plot of calculated versus measured pump pressure (Fruhwirth et al. 2006)

Although the results obtained by the authors have enough accuracy, the authors have added some more features to the input data including MD-TVD ratio, dog leg severity (DLS), sine and cosine of well inclination, and three features corresponding to Reynolds number ($N_{Re}$). Reynolds number was considered because it is related to the mud rheological properties, well geometry, and mud flow rate (which all have an effect on hydraulics).

In a further study, torque and bit measured depth have been added as additional input parameters and it has been concluded that 95 % of the theoretical predicted standpipe pressure values lie within 10 bars of the real measured data (Fig. 23). Also, in the cross-plot of predicted ($y$-axis) and measured pressure ($x$-axis) as shown Fig. 24, the squared Pearson coefficient has been found to be equal to 0.9677 (Todorov et al. 2014). Also, the simulated pressure losses clearly follow the trend of the measured standpipe pressure (Fig. 23). All the above discussions indicate the capability of ANN in handling complex hydraulics problems.

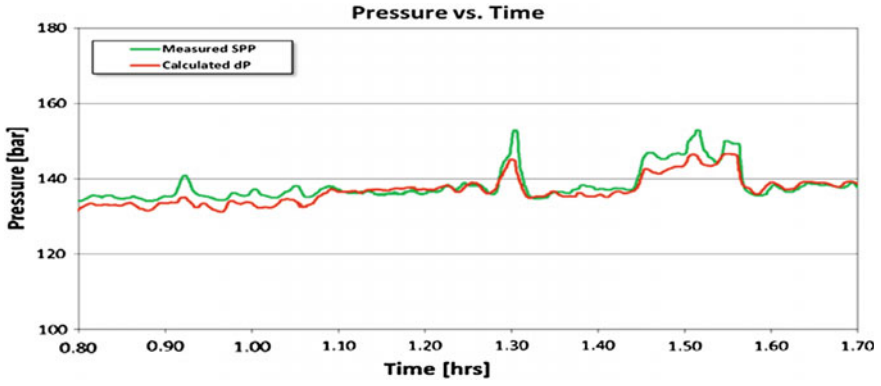


**Fig. 23** Measured standpipe pressure and calculated pressure drop by ANN versus time (Todorov and Thonhauser 2014)

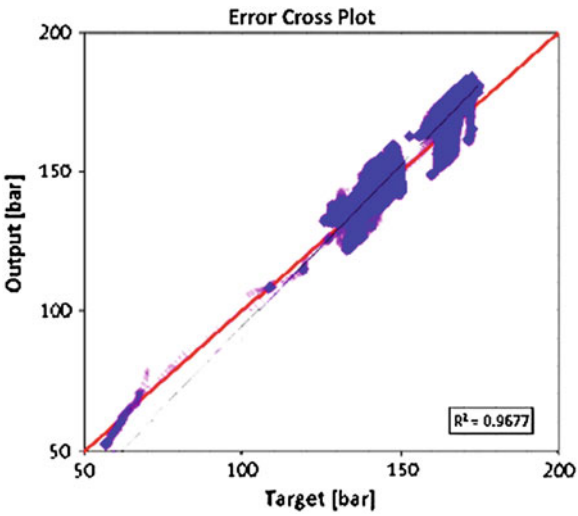


**Fig. 24** cross-plot of predicted (*y*-axis) and measured pressure (*x*-axis)

## *8.3 Drilling Optimization*

Many studies have been performed on the application of ANN for drilling optimization of rate of penetration (ROP) in the literature.

In a recent work, four ROP models with, respectively 6, 9, 15, and 18 input parameters or data channels were constructed using ANN with the objective of investigating the effect of vibration parameters on ROP (Esmaeili et al. 2012). In the first two models, the vibration parameters were not considered. In the third model, formation mechanical properties were not considered though vibration parameters were taken into consideration.

In the first ROP model, only the drilling parameters (average and standard deviation of WOB, average and standard deviation of RPM of drill string, and average and standard deviation of torque) were considered as input parameters or channels.

In the second ROP model, the drilling and mechanical properties (uniaxial compressive strength UCS, Young's modulus of elasticity, and Poisson's ratio) were considered as input parameters.

In the third ROP model, the drilling and vibration parameters were considered as input parameters. The vibration parameters include standard deviation, and first and second order frequency moment of X, Y, and Z component of vibration.

In the fourth ROP model, all the drilling, mechanical properties, and vibration parameters were considered as input parameters or channels.

In Fig. 25, the ranking of input parameters (including drilling, mechanical, and vibration) has been made for the fourth model using sequential forward selection (SFS) which shows the parameters UCS, standard deviation of Z component of vibration, average WOB, etc. that have the most impact on ROP.

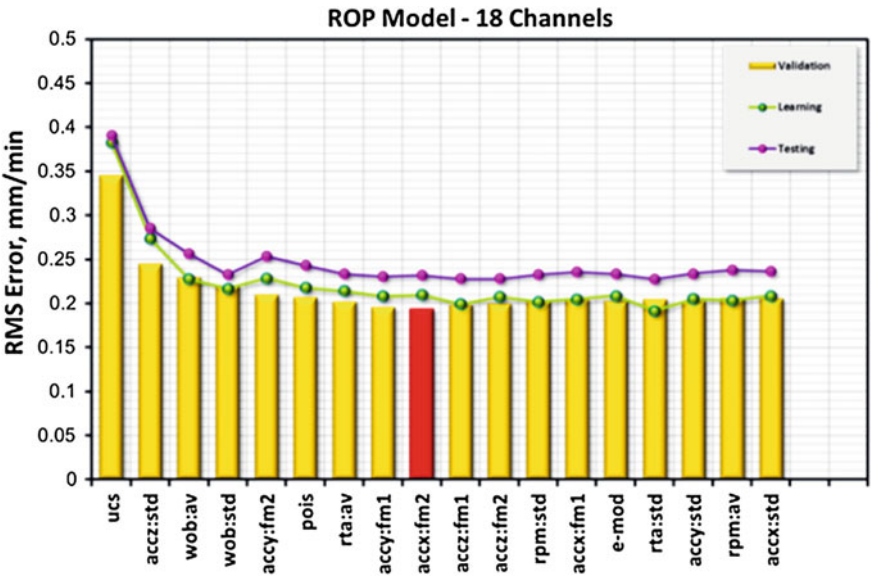


**Fig. 25** Ranking of input channels (or parameters) for the fourth ROP Model using sequential forward selection (Esmaeili et al. 2012)

**Table 2** Summary of ROP models and corresponding RMS errors (Esmaeili et al. 2012)

| Model | No. of input channels | Number of hidden units | RMS error (mm/min) |
|---|---|---|---|
| First ROP model | 6 | 6 | 0.301 |
| Second ROP model | 9 | 6 | 0.208 |
| Third ROP model | 15 | 6 | 0.231 |
| Fourth ROP model | 18 | 5 | 0.194 |

Eventually, Esmaeili et al. (2012) reached the results in Table 2.

Most of the common ROP models (Bourgoyne, Warren, Maurer, etc.), which are based on only the drilling and mechanical properties, do not consider the vibration properties. The RMS error corresponding to the second ROP model, which looks like the mentioned models, was calculated as 0.208 mm/min. Nevertheless, in the fourth ROP model wherein the vibration parameters were considered, the RMS error was calculated as 0.194 mm/min which shows vibration parameters have important effects on ROP modeling.

In Fig. 26, the cross-plot of actual ROP values (predicted by ANN) and desired ROP values (measured or expected real values) have been shown which shows a better match in the fourth model.

Some similar drilling optimization studies using neural networks to be mentioned are as follows: Gidh et al. (2012) in prediction of bit wear, Lind and Kabirova (2014) in prediction of drilling problems, and Bataee et al. (2010).

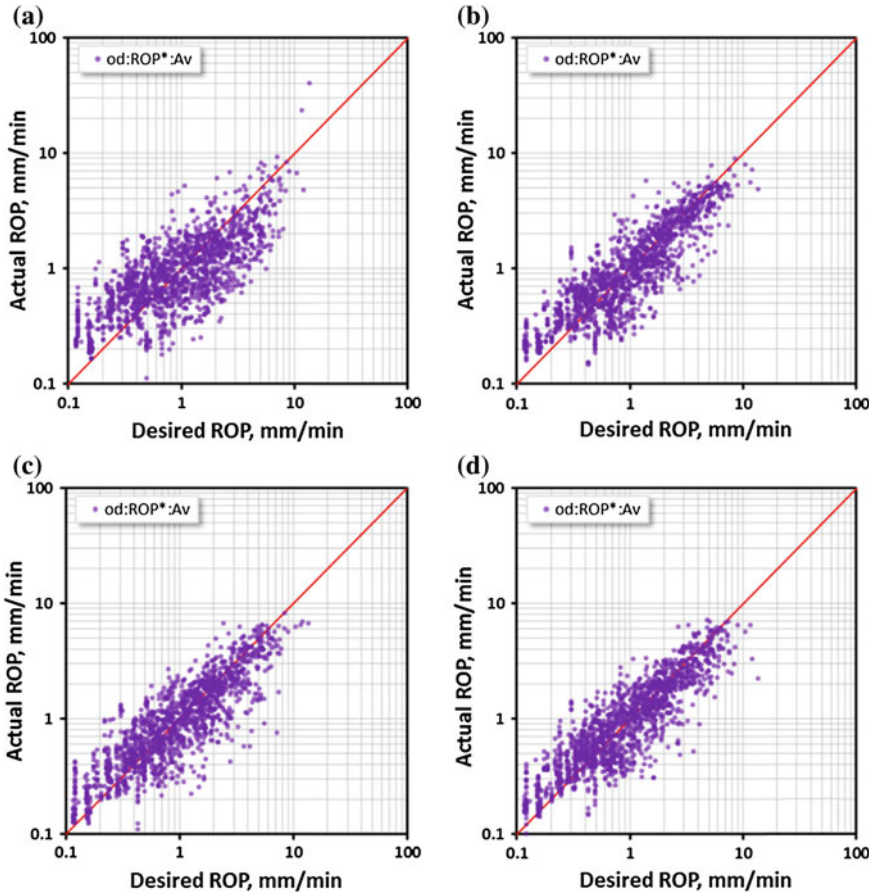


**Fig. 26** Actual ROP (predicted by ANN) versus desired ROP (expected real or measured values) for all four ROP models from **a**–**d** (Esmaeili et al. 2012)

## *8.4 Permeability Prediction*

Permeability is one of the important parameters which have been estimated using ANN in many applications.

In an application of ANN to predict permeability (Naeeni et al. 2010), the input parameters considered in the FF-ANN with backpropagation algorithm, include depth, CT (true conductivity), DT (sonic travel time), NPHI (neutron porosity), RHOB (bulk density), SGR (spectral gamma ray), NDSEP (neutron-density log separation), *northing of well*, *easting of well*, $S_{WT}$ (water saturation), and FZI (flow zone indicator). The number of three hidden layers (with 13, 10, and 1 neurons) was selected for the network. In order to estimate permeability, determination of different hydraulic flow units (HFU) is the first stage because it leads to FZI values

which are needed for the first stage. At the same time, FZI which is related to pore size and geometry has the following relation with RQI:

$$\mathrm{FZI} = \frac{\mathrm{RQI}}{\varnothing_z} \tag{19}$$

It is also noted that RQI and $\varnothing_z$ are evaluated by:

$$\mathrm{RQI}(\mu\mathrm{m}) = 0.031\sqrt{\frac{K}{\varnothing_e}} \tag{20}$$

$$\varnothing_z = \frac{\varnothing_e}{1 - \varnothing_e} \tag{21}$$

where $\varnothing_e$ is the effective porosity.

Thus, taking logarithms from both sides leads to:

$$\mathrm{Log}\ (\mathrm{RQI}) = \mathrm{Log}(\varnothing_z) + \mathrm{Log}(\mathrm{FZI}) \tag{22}$$

Therefore, Log(FZI) is the intercept of the plot of RQI versus $\varnothing_z$ in log–log scale. Thus, **first** RQI (rock quality index) values versus normalized porosity values ($\varnothing_z$) were plotted in log–log scale using core data. **Second**, RQI values versus $\varnothing_z$ were plotted in a log–log scale. **Third**, some straight lines are selected to intersect with line $\varnothing_z = 1$ such that reasonable initial guesses of intercepts are obtained as mean FZI values.

**Fourth**, the data points are assigned to the adjacent straight lines and are considered as different HFUs (clustering technique). This stage requires considerable time. **Fifth**, the intercept or FZI of each HFU is recalculated utilizing regression equations and is compared with the guess in the fourth stage. If the difference between the recalculated FZI value and the guess value in the fourth stage is considerable, it is required to come back to the fourth stage so that the guessed FZI can be updated.

Following the above procedure, finally different rock types could be determined with good accuracy (HFU determination is done) and also permeability values could be estimated as shown in Figs. 27 and 28.

As the second step, an ANN was trained with well log and RQI data as input and permeability data as output. Eventually, it is possible to simulate the ANN using well log data in uncored wells to estimate their permeability values.

As the cross-plot of predicted versus measured values of permeability is shown in Fig. 29, the performance of the ANN in permeability prediction was promising with the Pearson coefficient of 0.85 for the validation phase

In another study with a rather similar approach (Kharrat et al. 2009), the predicted permeability values show a good match with the core measured values. The well FZI profile versus depth has been shown on the left of Fig. 30 (with dots showing the values corresponding to core measured values). The predicted and core

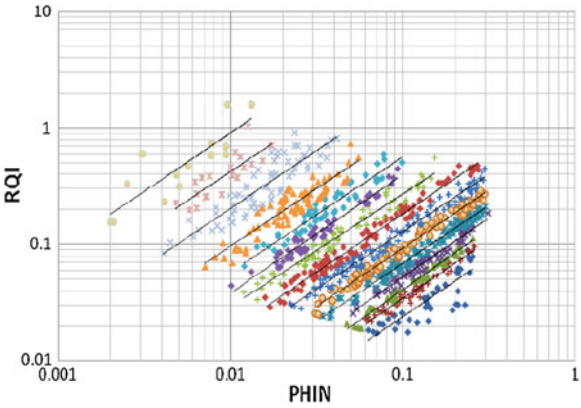


**Fig. 27** The number of 15 rock types has been identified in the collected data from the reservoir (Naeeni et al. 2010)

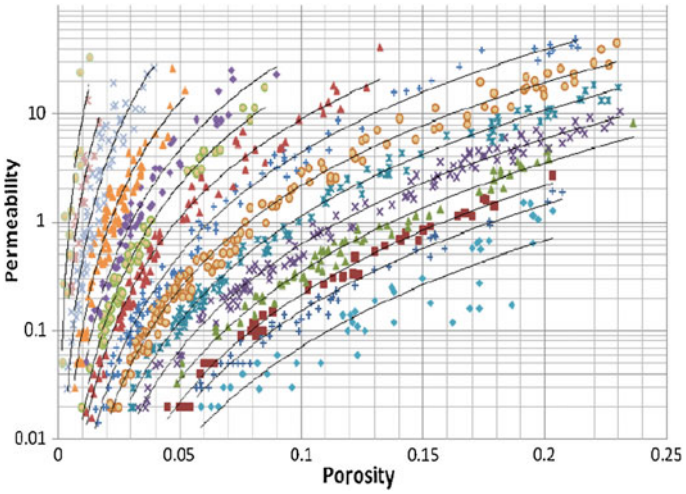


**Fig. 28** For each rock type of the reservoir, one permeability curve versus porosity has been determined (Naeeni et al. 2010)

measured permeability values versus depth have been illustrated on the right of Fig. 30.

Some similar reservoir engineering studies using neural networks are as follows: Thomas and Pointe (1995) in conductive fracture identification, Lechner and Zangl (2005) in reservoir performance prediction, Adeyemi and Sulaimon (2012) in predicting wax formation, and Tang (2008) in reservoir facies classification.

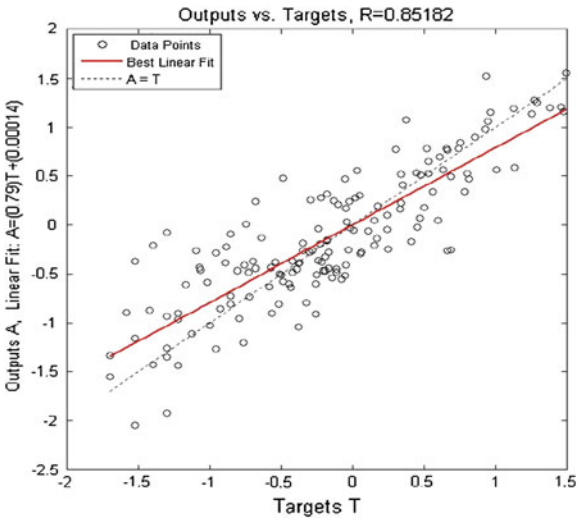


**Fig. 29** Predicted permeability values or output values as *y*-axis versus core permeability values as *x*-axis (Naeeni et al. 2010)

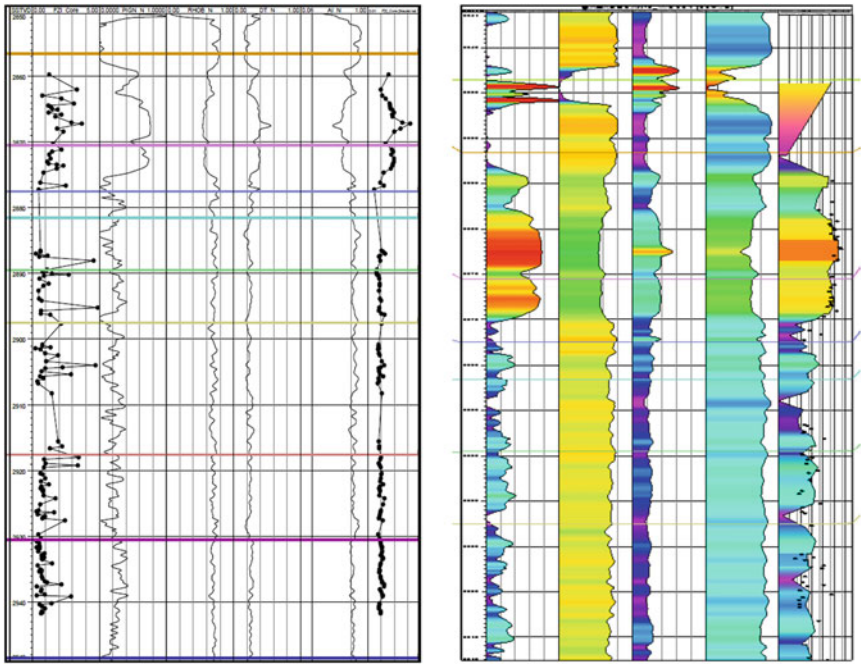

**Fig. 30** FZI profile of one well based on log and core data (*left*) and its permeability predictions (Kharrat et al. 2009)

## 9 Conclusion

Artificial neural networks (ANNs) have shown to be an effective tool to solve complex problems with no analytical solutions. In this chapter, the following topics have been covered: artificial neural networks basics (neurons, activation function, ANN structure), feed-forward ANN, backpropagation and learning, perceptrons and Backpropagation, multilayer ANNs and backpropagation algorithm, data processing by ANN (training, over-fitting, testing, validation), ANN and statistical parameters, and some applied examples of ANN in geoscience and petroleum engineering.

## Appendix: Important Statistical Parameters

The corresponding relations of a few important statistical parameters to compare performance and accuracy of different neural network models are given as follows:

1. Average percent relative error (APE):
   This error is defined as the relative deviation from the measured data.

$$\mathrm{APE} = \frac{1}{n}\sum_{i=1}^{n} E_{\mathrm{i}} \tag{23}$$

$$E_i = \left[\frac{P_{\mathrm{m}} - P_{\mathrm{e}}}{P_{\mathrm{m}}}\right]_i \quad i = 1, 2, 3, \ldots, n, \tag{24}$$

2. Average Absolute Percent Relative Error (AAPE):
   This error gives an idea of absolute relative deviation of estimated outputs from the measured or expected output data.

$$\mathrm{AAPE} = \frac{1}{n}\sum_{i=1}^{n} |e_i| \tag{25}$$

$$e_i = [P_{\mathrm{m}} - P_{\mathrm{e}}]_i \tag{26}$$

3. Mean squared error (MSE):
   This error is corresponding to the expected value of the squared error loss.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n} e_i^2 \tag{27}$$

4. Average root-mean-square error (ARMSE):
   This error is an indeed measure of scatter or lack of accuracy of the estimated data.

$$\text{ARMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2} \tag{28}$$

5. Standard deviation (SD):
   This error shows the dispersion of the values from the average value or mean.

$$\text{SD} = \left[\frac{n\sum_{i=1}^{n} E_i^2 - \left(\sum_{i=1}^{n} E_i\right)^2}{n^2}\right]^{\frac{1}{2}} \tag{29}$$

6. Variance or $V$, $\sigma^2$:
   This error is the square of the standard deviation.

$$\sigma^2 = \frac{\sum (X - M)^2}{N} \tag{30}$$

7. Correlation coefficient or Pearson coefficient ($R$):
   It represents the degree of success in reduction of the standard deviation (SD). It is normally used as a measure of the extent of the linear dependence between two variables. The nearer $R$ is to 1, the better the convergence and ANN performance is.

$$R = \frac{\sum_{i=1}^{n}\left[\left(P_{\text{m},i} - P_{m,\text{av}}\right) \times \left(P_{\text{e},i} - P_{e,\text{av}}\right)\right]}{\sqrt{\sum_{i=1}^{n}\left[\left(P_{\text{m},i} - P_{\text{m,av}}\right)^2\right] \times \sum_{i=1}^{n}\left(P_{\text{e},i} - P_{\text{e,av}}\right)^2}} \tag{31}$$

$$P_{\text{av}} = \frac{1}{n}\sum_{i=1}^{n} P_i \tag{32}$$

8. Squared Pearson coefficient: $R^2$

## References

Adeyemi BJ & Sulaimon AA (2012) Predicting wax formation using artificial neural network. In: SPE-163026-MS, Nigeria annual international conference and exhibition, Lagos, Nigeria

Ashena R, Moghadasi J, Ghalambor A, Bataee M, Ashena R, Feghhi, A (2010) Neural networks in BHCP prediction performed much better than mechanistic models. In: SPE 130095, international oil and gas conference and exhibition, Beijing, China

Bataee M, Edalatkhah S, Ashena R (2010) Comparison between bit optimization using artificial neural network and other methods base on log analysis applied in Shadegan oil field. In: SPE 132222-MS, international oil and gas conference and exhibition, Beijing, China

Bertsekas DP, Tsitsiklis JN (1996) Neuro-dynamic programming. Athena Scientific, Belmont, MA. ISBN 1-886529-10-8

Cacciola M, Calcagno S, Lagana F, Megali G, Pellicano D (2009) Advanced integration of neural networks for characterizing voids in welded strips. In: 19th international conference, Cyprus

Coulibaly P, Baldwin CK (2005) Nonstationary hydrological time series forecasting using nonlinear dynamic methods. J Hydrol 307:164–174

CVision Software Manual, NGS-Neuro Genetic Solutions GmbH

Darken C and Moody J (1992) Towards faster stochastic gradient search. In: Moody JE, Hanson SJ and Lippmann RP (eds)

Esmaeili A, Elahifar B, Fruhwirth R, Thonhauser G (2012) ROP modelling using neural network and drill string vibration data. In: SPE 163330, Kuwait international petroleum conference and exhibition

Fruhwirth R K, Thonhauser G, Mathis W (2006) Hybrid simulation using neural networks to predict drilling hydraulics in real time. In: SPE 103217, SPE annual technical conference and exhibition in San Antonia, Texas, USA

Gidh YK, Purwanto A and Ibrahim H (2012) Artificial neural network drilling parameter optimization system improves ROP by predicting/managing bit wear. In: SPE 149801-MS, SPE intelligent energy, Utrecht, The Netherlands

Kharrat R, Mahdavi R, Bagherpour M, Hejri S (2009) Rock type and permeability prediction of a heterogenous carbonate reservoir using artificial neural networks based on flow zone index approach. In: SPE 120166, SPE middle east oil and gas show and conference, Bahrain

Lechner J P and Zangl G (2005) Treating uncertainties in reservoir performance prediction with neural networks. In: SPE-94357-MS, SPE Europec/EAGE annual conference, 13–16 June, Madrid, Spain

Lind YB and Kabirova AR (2014) Artificial neural networks in drilling troubles prediction. In: SPE 171274-MS, SPE Russian oil and gas exploration and production technical conference and exhibition, Moscow, Russia

Mohaghegh S (2000) Virtual intelligence application in petroleum engineering: part I-artificial neural networks. J Pet Technol 52:64–72

Naeeni, MN, Zargari H, Ashena R, Kharrat R (2010) Permeability prediction of uncored intervals using IMLR method and artificial neural networks: a case study of Bangestan field, Iran. In: SPE 140682, 34th annual SPE international conference and exhibition, Nigeria

Tang H (2008) Improved carbonate reservoir facies classification using artificial neural network method. In: PETSOC-2008-122, Canadian international petroleum conference, Calgary, Alberta

Thomas AL and Pointe PR (1995) Conductive fracture identification using neural networks. In: ARMA-95-0627, the 35th U.S. symposium on rock mechanics (USRMS), Reno, Nevada

Todorov, D and Thonhauser G (2014) Hydraulics monitoring and well control event detection using model based analysis. In: SPE 24803, offshore technology conference Asia, Kuala lumpur, Malasia

# On Support Vector Regression to Predict Poisson's Ratio and Young's Modulus of Reservoir Rock

**A.F. Al-Anazi and I.D. Gates**

**Abstract** Accurate prediction of rock elastic properties is essential for wellbore stability analysis, hydraulic fracturing design, sand production prediction and management, and other geomechanical applications. The two most common required material properties are Poisson's ratio and Young's modulus. These elastic properties are often reliably determined from laboratory tests by using cores extracted from wells under simulated reservoir conditions. Unfortunately, most wells have limited core data. On the other hand, wells typically have log data. By using suitable regression models, the log data can be used to extend knowledge of core-based elastic properties to the entire field. Artificial neural networks (ANNs) have proven to be successful in many reservoir characterization problems. Although nonlinear problems can be well resolved by ANN-based models, extensive numerical experiments (training) must be done to optimize the network structure. In addition, generated regression models from ANNs may not perfectly generalize to unseen input data. Recently, support vector machines (SVMs) have proven successful in several real-world applications for its potential to generalize and converge to a global optimal solution. SVM models are based on the structural risk minimization principle that minimizes the generalization error by striking a balance between empirical training error and learning machine capacity. This has proven superior in several applications to the empirical risk minimization (ERM) principle adopted by ANNs that aims to reduce the training error only. Here, support vector regression (SVR) to predict Poisson's ratio and Young's modulus is described. The method uses a fuzzy-based ranking algorithm to select the most significant input variables and filter out dependency. The learning and predictive capabilities of the SVR method is compared to that of a backpropagation neural network (BPNN). The results demonstrate that SVR has similar or superior learning and prediction capabilities to that of the BPNN. Parameter sensitivity analysis was performed to

A.F. Al-Anazi · I.D. Gates (✉)
Department of Chemical and Petroleum Engineering, Schulich School of Engineering, University of Calgary, Calgary, Canada
e-mail: ian.gates@ucalgary.ca



C. Cranganu et al. (eds.), *Artificial Intelligent Approaches in Petroleum Geosciences*,
DOI 10.1007/978-3-319-16531-8_5

investigate the effect of the SVM regularization parameter, the regression tube radius, and the type of kernel function used. The result shows that the capability of the SVM approximation depends strongly on these parameters.

**Keywords** Poisson's ratio · Young's modulus · Artificial neural networks · Support vector machines · Log data · Core data

**Nomenclature**

| | |
|---|---|
| AAE | Absolute average error |
| $b$ | Bias term |
| BPNN | Backpropagation neural networks |
| $C$ | Regularization parameter |
| $E$ | Young's modulus, psi |
| $f$ | An unknown function |
| $g$ | Overburden stress, psi |
| $G$ | Shear modulus, psi |
| $h$ | Vapnik–Chervonenkis dimension |
| $K$ | Bulk modulus, psi |
| $L$ | Lagrangian equation for a dual programming problem or Loss function |
| MAE | Maximum absolute error |
| $r$ | Correlation coefficient |
| RBF | Radial basis function |
| RMSE | Root mean square error |
| $R_{\text{emp}}$ | Empirical risk |
| $R$ | Structural risk |
| $P$ | Pressure, psi |
| SVMs | Support vector machines |
| SVR | Support vector regression |
| $x$ | Input variable |
| $y$ | Output variable |
| $\hat{y}$ | Estimated output value |
| $v$ | Velocity |
| $w$ | Weight vector |

**Greek Symbols**

| | |
|---|---|
| $\alpha,\alpha^*$ | Lagrangian multiplier to be determined |
| $\varepsilon$ | Error accuracy/strain |
| $\varphi$ | Porosity, fraction |
| $\varphi$ | Mapping function from input space into a high-dimensional feature space |
| $\eta,\eta^*$ | Lagrangian multipliers |
| $\kappa$, $\vartheta$ | Sigmoid function parameters |
| $\mu$ | Poisson's ratio, dimensionless |
| $\rho$ | Density, g/cc |

$\sigma$ Stress/variance
$\sigma^2$ Standard deviation
$\xi,\xi^*$ Slack variables

**Subscripts and Superscripts**
$a$ Axial
$b$ Bulk
$h$ Horizontal
$k$, $l$ Indices
$N$ Number of samples
$n$ Input space dimension
$p$ Pore/compressional
$r$ Radial
$s$ Shear

## 1 Introduction

Elastic properties of reservoir rock are important geomechanical data required in hydraulic fracture design, wellbore stability analysis, reservoir dilation, surface heave (especially for high-pressure processes such as cyclic steam stimulation), and sand production anticipation as in cold heavy oil production. Rock properties including Poisson's ratio, $\mu$, shear modulus, $G$, Young's modulus, $E$, and bulk modulus, $K$, can be determined under static test conditions by using triaxial stress cells or under dynamic conditions by measuring compressional and shear velocities and density of reservoir core samples measured by acoustic or sonic logging tools (Gatens et al. 1990; Barree et al. 2009; Khaksar et al. 2009).

Core-measured elastic properties of reservoir rock obtained from detailed laboratory analysis are often considered to be the most direct and accurate method (Ameen et al. 2009). However, due to the costs of obtaining and handling core samples, most often, a limited number of samples are analyzed and often correlations are established between core and log data to estimate elastic properties of other reservoir rocks where core data are not available. Acoustic log measurements provide compressive and shear wave velocities which can be combined with density log and elasticity theory to estimate elastic properties of reservoir rock (Gatens et al. 1990; Abdulraheem et al. 2009). In complex reservoirs, acoustic theory may be insufficient to describe its actual behavior, thus limiting the accuracy of the derived rock correlations (Barree et al. 2009; Khaksar et al. 2009). Therefore, an integrated knowledge of the logging tool responses and understanding of underlying geology in addition to the use of advanced statistical techniques are necessary to determine an interpretation model that can effectively map the dependency between well log data and elastic rock properties. Due to several different factors that control log

responses and the construction of the mapping function between log data and the elastic properties, this can be a challenging task since the mapping can be nonlinear. To deal with this nonlinearity, artificial neural networks (ANNs), fuzzy logic (FL), and functional networks (FN) approaches to predict Poisson's ratio and Young's modulus have been used in the literature with varying degrees of success (Widarsono et al. 2001; Abdulraheem et al. 2009).

A further complication is that the data available to train artificial intelligence methods is scarce. The smaller the training data set, the greater the risk of poor estimation of rock properties. This is especially the case if data are missing from specific intervals along a well. Recently, support vector machines (SVMs) have been recognized as an efficient and accurate tool with strong learning and prediction capabilities (Al-Anazi and Gates 2010a, b, c, d). The SVM learning machine approach is novel and is based on the principle of structural risk minimization (SRM), which aims to minimize an upper bound of the generalization error. On the other hand, ANNs follow the principle of empirical risk minimization (ERM) which attempts to minimize the training error. The SRM principle is based on bounding the generalization error by minimizing the sum of the training error and a confidence interval term depending on the Vapnik–Chervonenkis (VC) dimension (Vapnik and Chervonenkis 1974; Vapnik 1982, 1995). To accomplish this, in support vector regression (SVR), a regularization term is used to determine the trade-off between the training error and VC confidence term. Consequently, the SVM approach provides a promising tool to generalize to unseen data. To capture nonlinear behavior of the mapping, kernel functions are used to project the input space into a higher dimensional feature space where a linear regression hyperplane is devised (Kecman 2005).

The accuracy and robustness of the SVM approach has been robustly demonstrated in many real-world applications; for example, face recognition, object detection, hand writing recognition, text detection, speech recognition and prediction, porosity and permeability determination from log data, and lithology classification (Li et al. 2000; Lu et al. 2001; Choisy and Belaid 2001; Gao et al. 2001; Kim et al. 2001; Ma et al. 2001; Van Gestel et al. 2001; Al-Anazi and Gates 2010a, b, c, d). In this study, the potential of SVR to establish an interpretation model that relates core and log data to elastic rock properties is evaluated from data originating from a well in a hydrocarbon reservoir. Two separate interpretation models were constructed: one for Poisson's ratio and the other for Young's modulus. The first model for Poisson's ratio was developed by using the density and compressive and shear wave velocities, whereas the second one for Young's modulus was devised by using variables chosen from a fuzzy selection scheme. For the well used in this study, the available core-derived input variables are porosity, minimum horizontal stress, pore pressure, and overburden stress, whereas the available well log data include bulk density and compressional and shear velocities.

In this research, the performance of SVM was compared with that of a back-propagation neural network (BPNN) to evaluate its potential to predict elastic rock properties under scarce data conditions. For SVM nonlinear approximation, a radial

basis function (RBF) kernel function is used. During the machine learning stage of the SVM, the kernel function parameter, insensitivity tube parameter, ε, and penalty parameter were selected through grid search and pattern search schemes. To avoid overfitting the data, a ten-fold cross-validation was used to select the optimal parameter to control the trade-off between the bias and variance of the model. Error analysis was done by examining the correlation coefficient, *r*, root mean square error (RMSE), absolute average error (AAE), and maximum absolute error (MAE) between target and predicted values. The study also investigated the impact of the penalty parameter, insensitivity tube radius, and the type of kernel function on the SVM approximation capability.

## 2 Backpropagation Neural Network (BPNN)

Backpropagation multilayer perceptron neural networks (BPNN) have been extensively used to interpret rock physical and elastic properties in hydrocarbon reservoirs (Rogers et al. 1995; Huang et al. 1996, 2001; Fung et al. 1997; Helle and Ursin 2001; Helle and Bhatt 2002; Abdulraheem et al. 2009). BPNN can approximate any continuous nonlinear function over a compact interval to any desirable accuracy depending on the number of hidden layers. They use activation functions in the processing neurons to facilitate the modeling of nonlinear mapping functions (Suykens et al. 2002). During learning, in BPNNs, the input patterns are propagated forward through hidden layers toward the output, while error is backpropagated toward the input layer. Here, a conjugate gradient algorithm is used to train the BPNN by minimizing the square of the residuals between target and training data. One well-known issue with such training algorithms is that it may become trapped in local minima since it is sensitive to the starting weight values (Hastie et al. 2001). Here, an Nguyen–Widrow algorithm was used to select the initial range of weight values and the conjugate gradient algorithm was then used to optimize the weights. Optimization is done several times with different random weight values to ensure convergence to the global optimal solution. One shortcoming of BPNNs is its susceptibility to overfit training data which results in poor generalization capability to interpret new input data. In this work, three layers (input, hidden, and output) were used with neurons being automatically optimized. A ten-fold cross-validation technique was used to stop training (DTREG v9.1 2009).

## 3 Support Vector Regression

SVMs are learning algorithms originally developed to solve classification problems. By using Vapnik's ε-insensitive loss function, SVMs can be used to solve nonlinear regression problems by using kernel functions (Vapnik 1995). Given a data set

$(y_k,\ \mathbf{x}_k)$ of dimension $N$ where $y_k$ is the output value at input variable values expressed in the vector $\mathbf{x}_k$, the relationship between the input and output variables can be expressed by the following linear regression function:

$$f(\mathbf{x}_k) = \mathbf{w}^T\mathbf{x}_k + b \tag{1}$$

where $\mathbf{w}$ is a set of weights and $b$ is an offset or bias. The empirical risk (to be minimized) can be expressed by

$$R_{\text{emp}} = \frac{1}{N}\sum_{k=1}^{N} v\left(y_k - \mathbf{w}^T\mathbf{x}_k - b\right) \tag{2}$$

where $v(\cdot)$ is the Vapnik's $\varepsilon$-insensitive loss function defined by

$$v(y - f(x)) = \begin{cases} 0 & \text{if} \quad |y - f(x)| \le \varepsilon \\ |y - f(x)| - \varepsilon & \text{otherwise} \end{cases} \tag{3}$$

The optimization problem in the primal space that has to be solved to achieve an optimal linear function in terms of the weights, $\mathbf{w}$, and bias, $b$, is given by

$$\min\ J_P(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{4}$$

$$\text{subject to} \begin{cases} y_k - \mathbf{w}^T\mathbf{x}_k - b \le \varepsilon,\ k = 1, \ldots, N \\ \mathbf{w}^T\mathbf{x}_k + b - y_k \le \varepsilon,\ k = 1, \ldots, N \end{cases}$$

The value of $\varepsilon$ in Vapnik's loss function, $v$, characterizes the radius of an approximation tube which in turn controls the accuracy of the model. Slack variables, $\xi_k, \xi_k^*$ for $k = 1, \ldots, N$, are introduced to the optimization problem given by Eq. 4:

$$\min\ J_P(\mathbf{w}, \xi, \xi^*) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{k=1}^{N}\left(\xi_K + \xi_K^*\right) \tag{5}$$

$$\text{subject to} \begin{cases} y_k - \mathbf{w}^T\mathbf{x}_k - b \le \varepsilon + \xi_k, & k = 1, \ldots, N \\ \mathbf{w}^T\mathbf{x}_k + b - y_k \le \varepsilon + \xi_k^*, & k = 1, \ldots, N \\ \xi_k, \xi_k^* \ge 0, & k = 1, \ldots, N \end{cases}$$

The regularization (penalty) constant, $C$, is positive and determines how large the deviation from the desired accuracy is tolerated. The Lagrangian form of the problem is expressed by

$$L(\mathbf{w}, b, \xi, \xi^*; \alpha, \alpha^*, \eta, \eta^*) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{k=1}^{N}\left(\xi_k + \xi_k^*\right) - \sum_{k=1}^{N}\alpha_k\left(\varepsilon + \xi_k - y_k + \mathbf{w}^T\mathbf{x}_k + b\right) - \sum_{k=1}^{N}\alpha_k^*\left(\varepsilon + \xi_k^* + y_k - \mathbf{w}^T\mathbf{x}_k - b\right) - \sum_{k=1}^{N}\left(\eta_k\xi_k + \eta_k^*\xi_k^*\right) \tag{6}$$

where $\alpha_k, \alpha_k^*, \eta_k, \eta_k^*$ are positive Lagrange multipliers. The solution is obtained by solving a saddle point problem: The Lagrangian, $L$, must be minimized with respect to $\mathbf{w}, b, \xi, \xi^*$ and maximized with respect to $\alpha, \alpha^*, \eta, \eta^*$. The following conditions are satisfied at the saddle point:

$$\begin{cases} \dfrac{\partial L}{\partial \mathbf{w}} = 0 \rightarrow \mathbf{w} = \displaystyle\sum_{k=1}^{N}\left(\alpha_k - \alpha_k^*\right)\mathbf{x}_k \\ \dfrac{\partial L}{\partial b} = 0 \rightarrow \displaystyle\sum_{k=1}^{N}\left(\alpha_k - \alpha_k^*\right)\mathbf{x}_k = 0 \\ \dfrac{\partial L}{\partial \xi_k} = 0 \rightarrow C - \alpha_\mathrm{k} - \eta_k = 0 \\ \dfrac{\partial L}{\partial \xi_k^*} = 0 \rightarrow C - \alpha_k^* - \eta_k^* = 0 \end{cases} \tag{7}$$

The primal optimization problem can be re-formulated as a dual problem as follows:

$$\max J_D(\alpha, \alpha^*) = -\frac{1}{2}\sum_{k,l=1}^{N}\left(\alpha_k - \alpha_k^*\right)\left(\alpha_l - \alpha_l^*\right)x_k^T x_l - \varepsilon\sum_{k=1}^{N}\left(\alpha_k + \alpha_k^*\right) + \sum_{k=1}^{N} y_k\left(\alpha_k - \alpha_k^*\right) \tag{8}$$

$$\text{subject to } \begin{cases} \sum_{k=1}^{N}\left(\alpha_k - \alpha_k^*\right) \\ \alpha_k, \alpha_k^* \in [0, c] \end{cases}$$

By solving Eq. 8, the optimal Lagrange multiplier pairs can be found and the linear regression is then given by

$$f(\mathbf{x}_k) = \sum_{k=1}^{N}\left(\alpha_k - \alpha_k^*\right)\mathbf{x}_k^T\mathbf{x}_k + b \tag{9}$$

with

$$\mathbf{w} = \sum_{k=1}^{N} \left(\alpha_k - \alpha_k^*\right)\mathbf{x}_k \tag{10}$$

The training points with nonzero $\alpha_k$ values allow the calculation of the bias term, $b$ (Kecman 2005). Generalization of the SVM approach to nonlinear regression estimation is accomplished by using kernel functions. Typical kernel functions that are used are linear ones, Gaussian RBF, and polynomial and sigmoid functions as listed in Table 1. In the primal weight space, the regression model is given by

$$f(\mathbf{x}) = \mathbf{w}^T\varphi(\mathbf{x}) + b \tag{11}$$

with given training data $\{\mathbf{x}_k, y_k\}_{k=1}^N$ and $\varphi(\cdot)$: $R^n \rightarrow R^{n_h}$ is a kernel mapping function which projects the input space to a higher dimensional feature space. The primal problem is then formulated as follows:

$$\min J_P(\mathbf{w}, \xi, \xi^*) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{k=1}^{N}\left(\xi_k + \xi_k^*\right) \tag{12}$$

$$\text{subject to} \begin{cases} y_k - \mathbf{w}^T\varphi(\mathbf{x}_k) - b \leq \varepsilon + \xi_k, & k = 1, \ldots, k \\ \mathbf{w}^T\varphi(\mathbf{x}_k) + b - y_k \leq \varepsilon + \xi_k^*, & k = 1, \ldots, k \\ \xi_k, \xi_k^* \geq 0, & k = 1, \ldots, N \end{cases}$$

The dual problem is then formulated as follows:

$$\begin{aligned} \max J_D(\alpha, \alpha^*) = &-\frac{1}{2}\sum_{k,l=1}^{N}\left(\alpha_k - \alpha_k^*\right)\left(\alpha_l - \alpha_l^*\right)K(\mathbf{x}_k, \mathbf{x}_l) \\ &- \varepsilon\sum_{k=1}^{N}\left(\alpha_k + \alpha_k^*\right) + \sum_{k=1}^{N} y_k\left(\alpha_k - \alpha_k^*\right) \end{aligned} \tag{13}$$

$$\text{subject to} \begin{cases} \sum_{k=1}^{N}\left(\alpha_k - \alpha_k^*\right) = 0 \\ \alpha_k, \alpha_k^* \in [0, c] \end{cases}$$

**Table 1** Common kernel function and corresponding mathematical expression

| Kernel function | Mathematical expression |
|---|---|
| Linear | $k(\mathbf{x}_i, \mathbf{x}) = \langle \mathbf{x}_i, \mathbf{x} \rangle$ |
| Gaussian radial basis function | $k(\mathbf{x}_i, \mathbf{x}) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}\|^2}{2\sigma^2}}$ |
| Sigmoid | $k(\mathbf{x}_i, \mathbf{x}) = \tanh(\kappa\langle \mathbf{x}_i, \mathbf{x} \rangle + \vartheta)$ |

**Table 2** Error measures used for accuracy assessment

| Accuracy measure | Mathematical expression |
|---|---|
| Correlation coefficient, $r$ | $\frac{\sum_{i=1}^{l}(y_i-\bar{y}_i)\left(\hat{y}_i-\bar{\hat{y}}_i\right)}{\sqrt{\sum_{i=1}^{l}(y_i-\bar{y}_i)^2\sum_{i=1}^{N_P}\left(\hat{y}_i-\bar{\hat{y}}_i\right)^2}}$ |
| Root mean square error, RMSE | $\sqrt{\frac{1}{l}\sum_{i=1}^{l}(y_i-\hat{y}_i)^2}$ |
| Average absolute error, AAE | $\frac{1}{l}\sum_{i=1}^{l}\lvert y_i-\hat{y}_i\rvert$ |
| Maximum absolute error, MAE | $\max \quad \lvert y_i-\hat{y}_i\rvert, \quad i=1,\ldots,l$ |

By setting $K(x_k, x_l) = \varphi(x_k)^T\varphi(x_l)$ for $k, l = 1, \ldots, N$, the explicit calculation of the kernel function is avoided. The nonlinear regression representation of the dual problem is given by

$$f(\mathbf{x}) = \sum_{k=1}^{N}\left(\alpha_k - \alpha_k^*\right)K(\mathbf{x}, \mathbf{x}_k) + b \tag{14}$$

where $\alpha_k, \alpha_k^*$ are the solution of Eq. 13 and the bias term $b$ is calculated as an average value over the support vectors corresponding to the training data set. The solution to Eq. 13 is unique and is a global minimum so long as the kernel function is positive definite (Suykens et al. 2002).

## 4 Bounds on the Generalization Error

The VC theory underlying the SVM formulation characterizes the generalization error instead of training (empirical) error. Given a set of functions $f(\mathbf{x}, \boldsymbol{\theta})$ characterized by different adjustable parameter vector $\boldsymbol{\theta}$ with a training data set $\{(\mathbf{x}_k, y_k)\}_{k=1}^{N}$ where $\mathbf{x}_k \in R^{N\times n}$ and $y_k \in R^n$. The empirical error is defined as follows:

$$R_{\text{emp}}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{k=1}^{N}(y_k - f(\mathbf{x}, \boldsymbol{\theta}))^2 \tag{15}$$

whereas the generalization error is defined by

$$R(\boldsymbol{\theta}) = \int (y_k - f(\mathbf{x}, \boldsymbol{\theta}))^2 p(\mathbf{x}, y)\mathrm{d}x\mathrm{d}y \tag{16}$$

measures the error over all patterns that are extracted from an underlying probability distribution $p(\mathbf{x}, y)$ which is typically unknown in practical applications. However, from Eqs. 15 and 16, the upper bound on the generalization error is given as follows:

$$R(\boldsymbol{\theta}) \leq R_{\text{emp}}(\boldsymbol{\theta}) + \left( \frac{1}{1 - c\sqrt{\frac{h(\ln(aN/h)+1) - \ln(\eta)}{N}}} \right)_+ \tag{17}$$

where $h$ is the VC dimension of the set of approximating functions and the notation $(x)_+$ indicates that $(x)_+ = x$ if $x > 0$ and 0 otherwise. The upper bound given by Eq. 17 holds for probability $1 - \eta$. This is the probability (or level of confidence) to approximate functions at which the generalization bound holds (Schölkopf and Smola 2002; Kecman 2005). The confidence term (second one in Eq. 17) also depends on the VC dimension which in turn characterizes the capacity of the set of approximating functions which in turns reflects model complexity (Vapnik 1998; Schölkopf and Smola 2002; Suykens et al. 2002). In this research, $a = c = 1$ (Cherkassky and Shao 2001).

## 5 SVR Parameter and Model Selection

To determine the optimal set of parameters ($C$, $\varepsilon$, and kernel function shape factors, e.g., the variance of the Gaussian RBF) for the SVR model, iterative grid and pattern searches are used. The grid search aims to use values extracted from a specified range controlled by geometric steps, whereas pattern search is based on the idea that a range is specified and that the search starts at the center of the range and takes trial steps in each direction for each parameter. If the parameters at the new point enhance the fit of the regression function, the search center moves to the new point and the process is repeated. Otherwise, the step size is reduced and the search is resumed. This iterative process is stopped after the step size drops below a pre-defined tolerance. Grid search is computationally demanding since the model must be evaluated at many points within the grid for each parameter. This limitation may be exaggerated if cross-validation has been adopted as a model selection technique. Pattern search requires far fewer evaluations of the model than that of grid search. However, pattern search can potentially converge to a local instead of a global optimum. Here, both search methods are used to overcome the shortcomings of each method: The optimization process starts with grid search seeking to locate a region close to the global optimal point. Next, pattern search is done over a narrow range that surrounds the best point located by the grid search.

## 6 Elastic Properties Prediction Methodology

### *6.1 Well Log and Core Data Description*

Figure 1 displays a subset of the core and log data versus depth used in this study. The input variables used to model the Poisson's ratio, $\mu$, and Young's modulus, $E$,

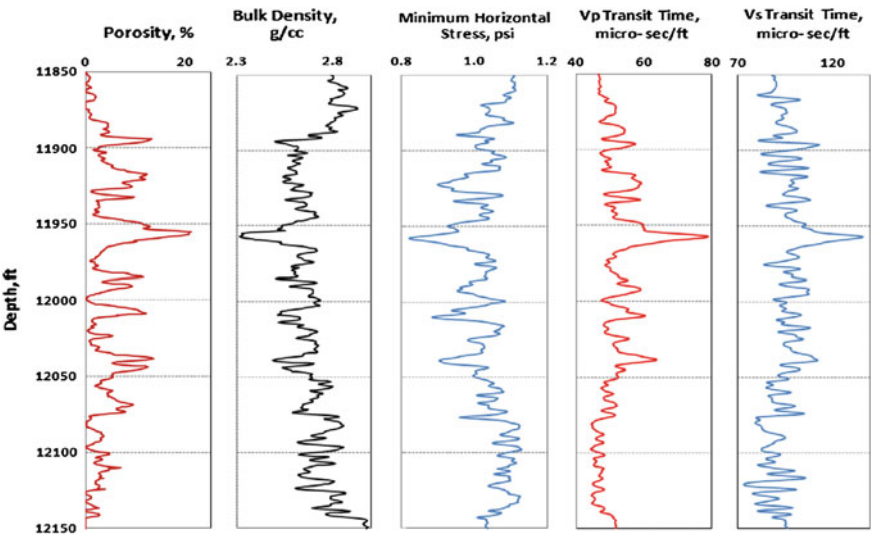


**Fig. 1** Subset of raw log data

at each depth, $\mathbf{x}_k$, are core-derived porosity $\phi$, minimum horizontal stress $\sigma_h$, pore pressure $P_p$, and overburden stress $g$, and log data including bulk density $\rho_b$, compressional wave velocity $v_p$, and shear wave velocity $v_s$. Elastic rock properties were extracted from samples tested in a laboratory-based triaxial pressure test cell. The input data set consists of 601 multidimensional data points spanning over 300 ft. of wellbore length. To evaluate the methods, the input data are randomly separated into training subsets consisting of 10, 20, 30, 40, 50, and 60 % of the total data set. The complements of the training data subsets are the testing subsets.

Acoustic logging tools measure the characteristic propagation speed of the $P$ (compression) and $S$ (shear) waves which are related to the elastic properties of the formation (Serra 1984). For core samples, elastic rock properties are determined from the stress–strain relationship that results from changes of stress with changes of the core strain (Montmayeur and Graves 1985, 1986):

$$E = \frac{\mathrm{d}\sigma_a}{\mathrm{d}\varepsilon_r}$$

where $\sigma_a$ is the axial stress applied to the core sample and $\varepsilon_r$ is the radial strain, and

$$\mu = \frac{\mathrm{d}\varepsilon_r}{\mathrm{d}\varepsilon_a}$$

where $\mu$ is the Poisson's ratio and $\varepsilon_a$ is the axial strain.

## 7 Modeling of the Poisson's Ratio, $\mu$

*Feature Selection*

Here, a model is developed that uses core-derived porosity $\phi$, minimum horizontal stress $\sigma_h$, pore pressure $P_p$, and overburden stress $g$, and log data including bulk density $\rho_b$, compressional wave velocity $v_p$, and shear wave velocity $v_s$. Following Al-Anazi and Gates (2009), a Two-stage fuzzy ranking algorithm is used to identify information-rich core and log measurements and filter out data dependencies. The result of fuzzy ranking analysis, listed in Table 3, shows that the significant input variables, in order of importance, are $\sigma_h$, $v_s$, $v_p$, $\rho_b$, $P_p$, and $\phi$. Therefore, these variables are the best correlators of Poisson's ratio and will be used in SVM model construction.

*Poisson's Ratio Training and Prediction*

The BPNN and SVR methods were compared by using different data training fractions (10, 20, 30, 40, 50, and 60 %) of the total available data to examine the impact of data scarcity on the predictive capabilities of the generated models. For example, a 10 % data training fraction means that 10 % of the total data available is selected randomly and used to train the model. The remaining 90 % of the data are used to test the correlation. For the SVR model, the Gaussian RBF kernel function was used.

Table 4 lists the error measures for the BPNN model results at different data training fractions when it is used to reproduce the training data set values. Learning performance is measured by correlation coefficient ($r$) and error statistics including RMSE, average absolute error (AAE), and MAE as defined in Table 2. The results reveal that the correlation coefficient is low and that as the data training fraction is enlarged, the capability of the BPNN to reproduce the training data set does not improve; in other words, the errors do not diminish as the training fraction grows. Table 5 lists the results of the SVR method. Even with training fraction as low as 10 %, the correlation coefficient is essentially equal to 1 and the errors are much smaller than that of the BPNN. In other words, the SVR method fully reproduces the training data set. These results demonstrate that for this data set, the SVR has superior learning capability to that of the BPNN.

Table 6 lists the error measures for the BPNN model results at different training fractions when it is used to predict the testing data set. The results show that the method does not provide a good prediction of the testing data set. The SVR results for the predicted testing data set are presented in Table 7. The results demonstrate an excellent prediction performance by the SVM revealing that the trained correlation models have captured the underlying relationships and have the potential to generalize accurately to new data. It can also be observed that consistent prediction capabilities are maintained over all data training fractions.

**Table 3** Two-stage fuzzy ranking analysis for Poisson's ratio

| Fuzzy curves ranking for Poisson's ratio data | | | | |
|---|---|---|---|---|
| Input $x^j$ | $P_{c^i}$ | $P_{c^i}/P_{c^R}$ | $P_{y_c^i}$ | $P_{v_c^i}$ |
| $\sigma_h$ | 0.6494 | 0.6512 | 0.6812 | 0.0489 |
| $v_s$ | 0.8643 | 0.8667 | 0.8776 | 0.0154 |
| $v_p$ | 0.8748 | 0.8773 | 0.8859 | 0.0127 |
| $\phi$ | 0.9194 | 0.9220 | 0.9279 | 0.0092 |
| $\rho_b$ | 0.9540 | 0.9567 | 0.9723 | 0.0192 |
| $P_p$ | 0.9740 | 0.9767 | 0.9794 | 0.0056 |
| $g$ | 0.9740 | 0.9767 | 0.9794 | 0.0056 |

| Second fuzzy surfaces ranking performance for Poisson's ratio data | | | | | |
|---|---|---|---|---|---|
| Input $x^j$ | $P_{s^{1,j}}$ | $P_{s^{1,j}}/P_{s^{1,R}}$ | $P_{s^{1,j}}/P_{c^1}$ | $P_{y_s^{1,j}}$ | $P_{v_s^{1,j}}$ |
| $v_s$ | 0.3530 | 0.5475 | 0.5436 | 0.4110 | 0.1642 |
| $\rho_b$ | 0.5759 | 0.8931 | 0.8869 | 0.6169 | 0.0711 |
| $P_p$ | 0.5997 | 0.9300 | 0.9235 | 0.6496 | 0.0831 |
| $g$ | 0.5997 | 0.9300 | 0.9235 | 0.6496 | 0.0831 |
| $\phi$ | 0.6282 | 0.9742 | 0.9674 | 0.6685 | 0.0641 |
| $v_p$ | 0.6340 | 0.9832 | 0.9763 | 0.6763 | 0.0667 |

| Third fuzzy surfaces ranking performance for Poisson's ratio data | | | | | |
|---|---|---|---|---|---|
| Input $x^j$ | $P_{s^{2,j}}$ | $P_{s^{2,j}}/P_{s^{2,R}}$ | $P_{s^{2,j}}/P_{c^2}$ | $P_{y_s^{2,j}}$ | $P_{v_s^{2,j}}$ |
| $v_p$ | 0.6325 | 0.7365 | 0.7319 | 0.6665 | 0.0538 |
| $\phi$ | 0.7135 | 0.8308 | 0.8256 | 0.7468 | 0.0467 |
| $g$ | 0.7878 | 0.9173 | 0.9116 | 0.8048 | 0.0215 |
| $P_p$ | 0.7878 | 0.9173 | 0.9116 | 0.8048 | 0.0215 |
| $\rho_b$ | 0.8085 | 0.9414 | 0.9355 | 0.8436 | 0.0434 |

| Fourth fuzzy surfaces ranking performance for Poisson's ratio data | | | | | |
|---|---|---|---|---|---|
| Input $x^j$ | $P_{s^{3,j}}$ | $P_{s^{3,j}}/P_{s^{3,R}}$ | $P_{s^{3,j}}/P_{c^3}$ | $P_{y_s^{3,j}}$ | $P_{v_s^{3,j}}$ |
| $\rho_b$ | 0.8380 | 0.9621 | 0.9578 | 0.8580 | 0.0239 |
| $P_p$ | 0.8489 | 0.9746 | 0.9703 | 0.8708 | 0.0258 |
| $g$ | 0.8489 | 0.9746 | 0.9703 | 0.8708 | 0.0258 |
| $\phi$ | 0.8495 | 0.9753 | 0.9710 | 0.8665 | 0.0200 |

| Fourth fuzzy surfaces ranking performance for Poisson's ratio data | | | | | |
|---|---|---|---|---|---|
| Input $x^j$ | $P_{s^{5,j}}$ | $P_{s^{5,j}}/P_{s^{5,R}}$ | $P_{s^{5,j}}/P_{c^5}$ | $P_{y_s^{5,j}}$ | $P_{v_s^{5,j}}$ |
| $P_p$ | 0.8789 | 0.9294 | 0.9212 | 0.9190 | 0.0457 |
| $g$ | 0.8789 | 0.9294 | 0.9212 | 0.9190 | 0.0457 |
| $\phi$ | 0.8794 | 0.9299 | 0.9218 | 0.9000 | 0.0235 |

| Fifth fuzzy surfaces ranking performance for Poisson's ratio data | | | | | |
|---|---|---|---|---|---|
| Input $x^j$ | $P_{s^{6,j}}$ | $P_{s^{6,j}}/P_{s^{6,R}}$ | $P_{s^{6,j}}/P_{c^6}$ | $P_{y_s^{6,j}}$ | $P_{v_s^{6,j}}$ |
| $\phi$ | 0.8461 | 0.8785 | 0.8687 | 0.8768 | 0.0363 |

**Table 4** Comparison of partition-based training error performance of BPNN using minimum horizontal stress, P and S wave velocities, RHOB, pore pressure, and porosity for prediction of Poisson's ratio

| | 10 % | 20 % | 30 % | 40 % | 50 % | 60 % |
|---|---|---|---|---|---|---|
| *r* | 0.6375 | 0.5394 | 0.5269 | 0.5940 | 0.5202 | 0.5628 |
| RMSE | 0.0291 | 0.0278 | 0.0289 | 0.0266 | 0.0295 | 0.0261 |
| AAE | 0.0239 | 0.0227 | 0.0224 | 0.0212 | 0.0229 | 0.0203 |
| MAE | 0.0852 | 0.0847 | 0.0984 | 0.0852 | 0.1043 | 0.1055 |

**Table 5** Comparison of partition-based training error performance of SVR using minimum horizontal stress, *P* and *S* wave velocities, RHOB, pore pressure, and porosity for prediction of Poisson's ratio

| | 10 % | 20 % | 30 % | 40 % | 50 % | 60 % |
|---|---|---|---|---|---|---|
| *r* | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| RMSE | 0.0003 | 0.0003 | 0.0002 | 0.0003 | 0.0002 | 0.0002 |
| AAE | 0.0003 | 0.0002 | 0.0002 | 0.0002 | 0.0002 | 0.0002 |
| MAE | 0.0006 | 0.0006 | 0.0008 | 0.0012 | 0.0006 | 0.0006 |

**Table 6** Comparison of partition-based testing error performance of BPNN using minimum horizontal stress, *P* and *S* wave velocities, RHOB, pore pressure, and porosity for prediction of Poisson's ratio

| | 10 % | 20 % | 30 % | 40 % | 50 % | 60 % |
|---|---|---|---|---|---|---|
| *r* | 0.5247 | 0.5330 | 0.5554 | 0.5005 | 0.5519 | 0.4997 |
| RMSE | 0.0277 | 0.0216 | 0.0271 | 0.0286 | 0.0256 | 0.0294 |
| AAE | 0.0214 | 0.0216 | 0.0213 | 0.0220 | 0.0204 | 0.0231 |
| MAE | 0.1062 | 0.1004 | 0.1050 | 0.1061 | 0.1014 | 0.1014 |

**Table 7** Comparison of partition-based testing error performance of SVR using minimum horizontal stress, *P* and *S* wave velocities, RHOB, pore pressure, and porosity for prediction of Poisson's ratio

| | 10 % | 20 % | 30 % | 40 % | 50 % | 60 % |
|---|---|---|---|---|---|---|
| *r* | 0.9992 | 0.9998 | 1.000 | 0.9999 | 1.000 | 1.000 |
| RMSE | 0.0014 | 0.0007 | 0.0003 | 0.0004 | 0.0002 | 0.0003 |
| AAE | 0.0007 | 0.0004 | 0.0002 | 0.0003 | 0.0002 | 0.0002 |
| MAE | 0.0090 | 0.0051 | 0.0021 | 0.0030 | 0.0008 | 0.0013 |

## 8 Modeling of Young's Modulus, *E*

*Feature Selection*

Similar to the analysis on Poisson's ratio described above, a two-stage fuzzy ranking analysis was done to identify the most important variables to predict Young's modulus. The results, listed in Table 8, reveal that the ranking of input data, in order of importance, are $v_s$, $\rho_b$, $v_p$, $P_p$, $\phi$, $g$, and $\sigma_h$. As expected, the ranking of the variables make it clear that the acoustic and density logs are most important with respect to the Young's modulus.

*Young's Modulus Training and Prediction*

As above, Gaussian RBFs was the kernel function used in the SVM nonlinear regression. Tables 9 and 10 list the results of the learning capabilities (the capability of the methods to reproduce the training data set) of the BPNN and SVR, respectively, at different training data fractions. For the BPNN, the results show that the correlation coefficient is high for all training data fractions. However, the RMSE and AAE do not exhibit a strong reducing trend as the training data fraction is enlarged and the MAE, in fact, shows a growth trend as the training data fraction increases. For the SVR, the correlation coefficients are high and are similar to that of the BPNN, but there is a decreasing trend of the RMSE, AAE, and MAE as the size of the training data fraction grows. Beyond 30 % training data set, all of the error measures of the SVR are lower than that of the BPNN. The results suggest that the SVM approach has higher learning capability than that of the BPNN for the data set used here.

Tables 11 and 12 list the correlation coefficient and error measures for the BPNN and SVR methods, respectively, at different training data fraction to predict the testing data subset. The results reveal excellent prediction performance can be observed by both techniques in terms of correlation coefficient. However, the analysis of the error performance indicates that the errors of the trained SVR models decline faster versus the size of the training data fraction than that of the BPNN and that the errors are lower than that of the BPNN when the training data fraction is 40 % or higher.

## 9 SVM Parameter Sensitivity Analysis

The generalization capability of SVMs depends on the regularization parameter, $C$, (see Eq. 5) that controls the trade-off between the training error (empirical error) and the VC dimension (complexity) of the regression model. If the value of $C$ is very small, the training error is the primary error that is minimized, whereas if its value is very large, the estimate on the prediction error dominates and the training error plays a lesser role to construct the SVR model. Additional inputs of the SVR model include the cost function parameter (the regression tube radius ε of the

**Table 8** Two-stage fuzzy ranking analysis for Young's modulus

| Fuzzy curves ranking for Young's modulus data | | | | |
|---|---|---|---|---|
| Input $x^j$ | $P_{c^i}$ | $P_{c^i}/P_{c^R}$ | $P_{y_c^i}$ | $P_{v_c^i}$ |
| $v_s$ | 0.2484 | 0.2507 | 0.3054 | 0.2295 |
| $v_p$ | 0.4010 | 0.4048 | 0.4620 | 0.1521 |
| $\rho_b$ | 0.4676 | 0.4720 | 0.5206 | 0.1135 |
| $\phi$ | 0.4868 | 0.4913 | 0.5468 | 0.1234 |
| $\sigma_h$ | 0.5026 | 0.5073 | 0.5589 | 0.1120 |
| $P_p$ | 0.5848 | 0.5903 | 0.6181 | 0.0571 |
| $g$ | 0.5848 | 0.5903 | 0.6181 | 0.0571 |

| Second fuzzy surfaces ranking performance for Young's modulus data | | | | | |
|---|---|---|---|---|---|
| Input $x^j$ | $P_{s^{1,j}}$ | $P_{s^{1,j}}/P_{s^{1,R}}$ | $P_{s^{1,j}}/P_{c^1}$ | $P_{y_s^{1,j}}$ | $P_{v_s^{1,j}}$ |
| $\rho_b$ | 0.1344 | 0.5432 | 0.5412 | 0.1823 | 0.3559 |
| $v_p$ | 0.1447 | 0.5847 | 0.5826 | 0.1967 | 0.3596 |
| $g$ | 0.1497 | 0.6051 | 0.6029 | 0.1993 | 0.3307 |
| $P_p$ | 0.1497 | 0.6051 | 0.6029 | 0.1993 | 0.3307 |
| $\sigma_h$ | 0.1514 | 0.6117 | 0.6095 | 0.2065 | 0.3642 |
| $\phi$ | 0.1567 | 0.6332 | 0.6309 | 0.2122 | 0.3540 |

| Third fuzzy surfaces ranking performance for Young's modulus data | | | | | |
|---|---|---|---|---|---|
| Input $x^j$ | $P_{s^{3,j}}$ | $P_{s^{3,j}}/P_{s^{3,R}}$ | $P_{s^{3,j}}/P_{c^3}$ | $P_{y_s^{3,j}}$ | $P_{v_s^{3,j}}$ |
| $v_p$ | 0.2415 | 0.5325 | 0.5166 | 0.3052 | 0.2636 |
| $P_p$ | 0.3136 | 0.6912 | 0.6706 | 0.3856 | 0.2298 |
| $g$ | 0.3136 | 0.6912 | 0.6706 | 0.3856 | 0.2298 |
| $\sigma_h$ | 0.3141 | 0.6924 | 0.6717 | 0.3842 | 0.2231 |
| $\phi$ | 0.3551 | 0.7827 | 0.7594 | 0.4259 | 0.1995 |

| Fourth fuzzy surfaces ranking performance for Young's modulus data | | | | | |
|---|---|---|---|---|---|
| Input $x^j$ | $P_{s^{2,j}}$ | $P_{s^{2,j}}/P_{s^{2,R}}$ | $P_{s^{2,j}}/P_{c^2}$ | $P_{y_s^{2,j}}$ | $P_{v_s^{2,j}}$ |
| $P_p$ | 0.2770 | 0.6983 | 0.6909 | 0.3477 | 0.2551 |
| $g$ | 0.2770 | 0.6983 | 0.6909 | 0.3477 | 0.2551 |
| $\phi$ | 0.3054 | 0.7698 | 0.7616 | 0.3808 | 0.2469 |
| $\sigma_h$ | 0.3436 | 0.8661 | 0.8569 | 0.4229 | 0.2307 |

| Fourth fuzzy surfaces ranking performance for Young's modulus data | | | | | |
|---|---|---|---|---|---|
| Input $x^j$ | $P_{s^{6,j}}$ | $P_{s^{6,j}}/P_{s^{6,R}}$ | $P_{s^{6,j}}/P_{c^6}$ | $P_{y_s^{6,j}}$ | $P_{v_s^{6,j}}$ |
| $\phi$ | 0.2909 | 0.5072 | 0.4975 | 0.3645 | 0.2531 |
| $\sigma_h$ | 0.3599 | 0.6276 | 0.6155 | 0.4370 | 0.2140 |
| $g$ | 0.5175 | 0.9023 | 0.8850 | 0.5717 | 0.1046 |

| Fourth fuzzy surfaces ranking performance for Young's modulus data | | | | | |
|---|---|---|---|---|---|
| Input $x^j$ | $P_{s^{4,j}}$ | $\frac{P_{s^{4,j}}}{P_{s^{4,R}}}$ | $\frac{P_{s^{4,j}}}{P_{c^4}}$ | $P_{y_s^{4,j}}$ | $P_{v_s^{4,j}}$ |
| $g$ | 0.2909 | 0.6115 | 0.5977 | 0.3645 | 0.2531 |
| $\sigma_h$ | 0.3595 | 0.7556 | 0.7385 | 0.4371 | 0.2160 |

| Fifth fuzzy surfaces ranking performance for Young's modulus data | | | | | |
|---|---|---|---|---|---|
| Input $x^j$ | $P_{s^{7,j}}$ | $P_{s^{7,j}}/P_{s^{7,R}}$ | $P_{s^{7,j}}/P_{c^7}$ | $P_{y_s^{7,j}}$ | $P_{v_s^{7,j}}$ |
| $\sigma_h$ | 0.3599 | 0.6276 | 0.6155 | 0.4370 | 0.2140 |

**Table 9** Comparison of partition-based training error performance of BPNN using minimum horizontal stress, *P* and *S* wave velocities, pore pressure, overburden stress, and porosity for prediction of Young's modulus

| | 10 % | 20 % | 30 % | 40 % | 50 % | 60 % |
|---|---|---|---|---|---|---|
| *r* | 0.9995 | 0.9995 | 0.9996 | 0.9995 | 0.9995 | 0.9995 |
| RMSE | 0.2336 | 0.2415 | 0.2635 | 0.2705 | 0.2864 | 0.2594 |
| AAE | 0.1903 | 0.1806 | 0.1990 | 0.2032 | 0.2181 | 0.1792 |
| MAE | 0.5223 | 0.9198 | 1.2780 | 1.2645 | 1.4517 | 2.0796 |

**Table 10** Comparison of partition-based training error performance of SVR using minimum horizontal stress, *P* and *S* wave velocities, pore pressure, overburden stress, and porosity for prediction of Young's modulus

| | 10 % | 20 % | 30 % | 40 % | 50 % | 60 % |
|---|---|---|---|---|---|---|
| *r* | 0.9944 | 0.9993 | 0.9999 | 0.9999 | 0.9999 | 1.0000 |
| RMSE | 0.9389 | 0.3098 | 0.1619 | 0.1390 | 0.0954 | 0.0759 |
| AAE | 0.5270 | 0.1584 | 0.0845 | 0.0764 | 0.0460 | 0.0419 |
| MAE | 4.6358 | 1.7137 | 0.9692 | 1.0244 | 1.1114 | 0.4942 |

**Table 11** Comparison of partition-based testing error performance of BPNN using minimum horizontal stress, *P* and *S* wave velocities, pore pressure, overburden stress, and porosity for prediction of Young's modulus

| | 10 % | 20 % | 30 % | 40 % | 50 % | 60 % |
|---|---|---|---|---|---|---|
| *r* | 0.9981 | 0.9989 | 0.9997 | 0.9987 | 0.9994 | 0.9994 |
| RMSE | 0.5426 | 0.4209 | 0.249 | 0.4658 | 0.2922 | 0.3159 |
| AAE | 0.3421 | 0.2731 | 0.1964 | 0.2609 | 0.2337 | 0.2002 |
| MAE | 4.0785 | 3.2514 | 1.3903 | 3.4581 | 1.5281 | 2.0031 |

**Table 12** Comparison of partition-based testing error performance of SVR using minimum horizontal stress, *P* and *S* wave velocities, pore pressure, overburden stress, and porosity for prediction of Young's modulus

| | 10 % | 20 % | 30 % | 40 % | 50 % | 60 % |
|---|---|---|---|---|---|---|
| *r* | 0.9905 | 0.9986 | 0.9996 | 0.9994 | 0.9998 | 0.9999 |
| RMSE | 1.5067 | 0.5016 | 0.2405 | 0.3153 | 0.1547 | 0.1445 |
| AAE | 0.922 | 0.2747 | 0.1337 | 0.1378 | 0.0764 | 0.0786 |
| MAE | 7.5939 | 3.596 | 1.9554 | 2.2864 | 1.4729 | 1.0244 |

ε-insensitivity cost function) and the type of kernel function used and its associated input parameters (for example, the variance of the Gaussian RBF kernel function). In the above analysis, these parameters were chosen by using the cross-validation method. Here, the impact of these parameters on the SVR model is investigated. Here, 10 % data training fraction subset (with 90 % remaining for testing) is used.

## 10 Impact of *C* on SVM Regression Performance

The effect of the value of *C* on the SVR training regression model for Poisson's ratio and Young's modulus were investigated with RBF kernel function with both approximation tube radius ε and the variance of the kernel function $\sigma$ kept constant at values selected in the above analysis. Tables 13 and 14 list the training performance as examined by correlation coefficient and error statistics to predict Poisson's ratio and Young's modulus. The results indicate that the constructed regression model fits the data perfectly as the values of the regularization parameter, *C*, increases. The selection of higher values of *C*, however, does not always improve performance of the SVR especially if the data itself does not describe the underlying function that relates the input vector to the output scalar as would be the case with data polluted with severe noise. In this case, the SVR would model the noise rather than the true relationship between the input and output data.

## 11 Impact of $\varepsilon$ on SVM Regression Performance

The radius of the regression tube, $\varepsilon$, within which the regression function must lay, is a measure of the error tolerance of the predictive capability of the regression model. If a predicted value lies within the tube radius, that is, its absolute value is less than $\varepsilon$ and the loss (error) is set equal to zero. For a predicted value lying outside the $\varepsilon$-tube, the loss (error) equals the difference between the predicted value and the radius of the tube $\varepsilon$. Here, to investigate the effect of the regression tube radius on the capability of the SVR to generalize to unseen data, the regularization

**Table 13** Comparison of partition-based training error performance of SVM over different regularization constant, *C* values for prediction of Poisson's ratio ($\sigma = 0.1877$, $\varepsilon = 9 \times 10^{-5}$)

| | $1 \times 10^{-4}$ | $1 \times 10^{-3}$ | $1 \times 10^{-2}$ | $1 \times 10^{-1}$ | 1 | $1 \times 10^{1}$ | $1 \times 10^{2}$ |
|---|---|---|---|---|---|---|---|
| *r* | 0.6849 | 0.6849 | 0.7297 | 0.9865 | 0.9994 | 0.9999 | 1.0000 |
| RMSE | 0.0356 | 0.0351 | 0.0312 | 0.0092 | 0.0014 | 0.0004 | 0.0003 |
| AAE | 0.0287 | 0.0283 | 0.0250 | 0.0067 | 0.0009 | 0.0003 | 0.0003 |
| MAE | 0.0964 | 0.0959 | 0.0895 | 0.0292 | 0.0067 | 0.0010 | 0.0006 |

**Table 14** Comparison of partition-based training error performance of SVM over different regularization constant, *C* values for prediction of Young's modulus ($\sigma = 0.1208$, $\varepsilon = 4 \times 10^{-5}$)

| | $1 \times 10^{3}$ | $1 \times 10^{4}$ | $1 \times 10^{5}$ | $1 \times 10^{6}$ | $1 \times 10^{7}$ | $1 \times 10^{8}$ | $1 \times 10^{9}$ |
|---|---|---|---|---|---|---|---|
| *r* | 0.0000 | 0.7908 | 0.8246 | 0.9331 | 0.9944 | 0.9995 | 1.0000 |
| RMSE | 7.5444 | 7.4695 | 6.8751 | 3.8890 | 0.9389 | 0.2379 | 0.0432 |
| AAE | 5.8183 | 5.7562 | 5.2528 | 2.8527 | 0.5270 | 0.1476 | 0.0262 |
| MAE | 25.0131 | 24.7374 | 21.9113 | 12.1491 | 4.6358 | 0.9137 | 0.2444 |

**Table 15** Comparison of partition-based training error performance of SVM over different approximation tube radius, ε values for prediction of Poisson's ratio (σ = 0.1877, $C$ = 45.66)

| | $1 \times 10^{-4}$ | $1 \times 10^{-3}$ | $1 \times 10^{-2}$ | $1 \times 10^{-1}$ |
|---|---|---|---|---|
| $r$ | 1.0000 | 0.9997 | 0.9907 | 0.000 |
| RMSE | 0.0003 | 0.0009 | 0.0060 | 0.0378 |
| AAE | 0.0003 | 0.0007 | 0.0051 | 0.0319 |
| MAE | 0.0005 | 0.0014 | 0.0104 | 0.0776 |

**Table 16** Comparison of partition-based training error performance of SVM over different approximation tube radius, ε values for prediction of Young's modulus (σ = 0.1208, $C = 1 \times 10^7$)

| | $1 \times 10^3$ | $1 \times 10^4$ | $1 \times 10^5$ | $1 \times 10^6$ | $1 \times 10^7$ |
|---|---|---|---|---|---|
| $r$ | 0.9944 | 0.9945 | 0.9934 | 0.9155 | 0.0000 |
| RMSE | 0.9389 | 0.9228 | 0.9868 | 3.8853 | 7.9617 |
| AAE | 0.5270 | 0.5303 | 0.7088 | 3.2992 | 6.0465 |
| MAE | 4.6358 | 4.5669 | 4.2223 | 8.4958 | 22.5317 |

parameter, $C$, and the variance, $\sigma$, of the RBF kernel function were fixed at the values determined above. The results listed in Tables 15 and 16 reveal that the constructed training model of Poisson's ratio and Young's modulus fits perfectly the data as the values of radius of the regression tube decreases as indicated by the correlation coefficient and error statistics. On the contrary, as the size of the insensitivity tube increases, the accuracy of the prediction model drops. The input data for the Poisson's ratio and Young's modulus data are quite clustered, and there are few outliers. As a result, when the radius of the regression tube enlarges, a fewer number of training points is used, thus the number of support vectors drop, and thus, the quality of the SVR model degrades. A further increase in the radius causes the SVR model to overshoot the test data. The results reveal that an increase in the radius of the insensitivity tube has a smoothing effect on modeling, whereas a decrease in the size may lead to overfitting the data.

## 12 Impact of Kernel Function on SVM Regression Performance

The kernel function is an integral part of the SVM formulation because it is necessary to solve nonlinear regression problems. The kernel function maps the nonlinear input space into a high-dimensional feature space where a linear SVM formulation can be applied. As such, the kernel function is assumed to have the capability to provide or approximately provide the nonlinear mapping. Thus, the choice of the kernel function will depend on the nature of the regression problem being solved. Here, three kernel functions are investigated including the linear,

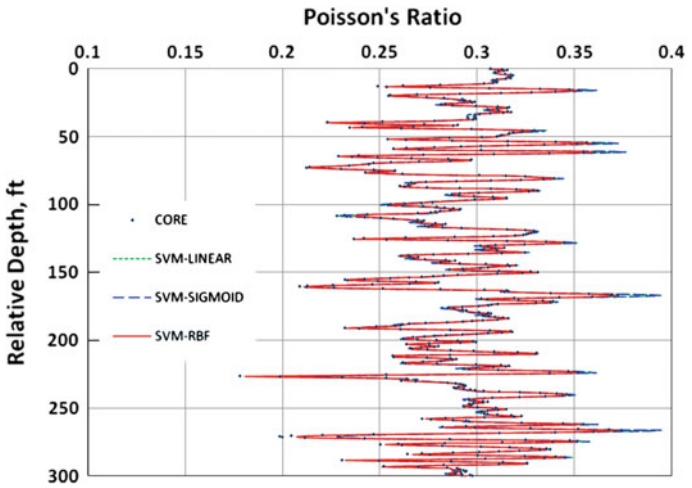


**Fig. 2** Comparison of SVM-LINEAR, SVM-SIGMOID, and SVM-RBF predictions of Poisson's ratio using 10 % of the data for training and 90 % for testing. The dots are data obtained from triaxial tests of core samples

sigmoid, and Gaussian RBF functions. The prediction capabilities of SVR approximation models based on these kernels to predict Poisson's ratio and Young's modulus are compared are shown in Figs. 2 and 3, respectively. The corresponding correlation coefficients and error statistics are listed in Tables 17 and 18. The results reveal that all three kernel functions exhibit reasonably good capabilities to

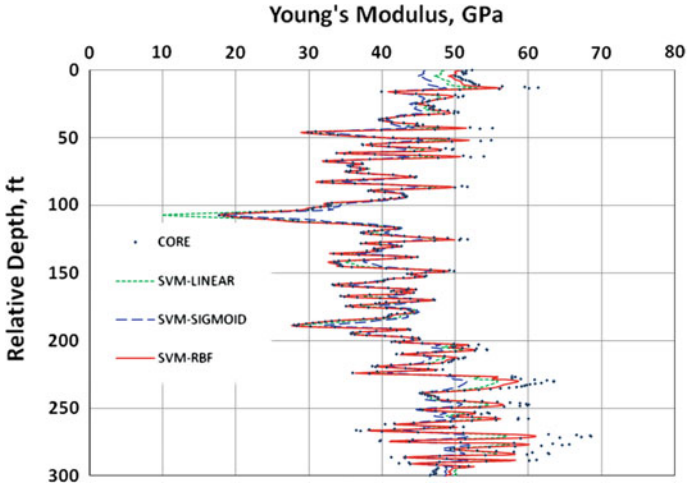


**Fig. 3** Comparison of SVM-LINEAR, SVM-SIGMOID, and SVM-RBF predictions of Young's modulus using 10 % of the data for training and 90 % for testing. The dots are data obtained from triaxial tests of core samples

**Table 17** Comparison of correlation coefficient, RMSE, AAE, and MAE of the errors between SVM-LINEAR, SVM-SIGMOID, and SVM-RBF predictions of Poisson's ratio using 10 % of the data for training and 90 % for testing

| | Linear | Sigmoid | RBF |
|---|---|---|---|
| *r* | 0.9935 | 0.9937 | 1.0000 |
| RMSE | 0.0040 | 0.0040 | 0.0003 |
| AAE | 0.0029 | 0.0029 | 0.0003 |
| MAE | 0.0193 | 0.0184 | 0.0006 |

**Table 18** Comparison of correlation coefficient, RMSE, AAE, and MAE of the errors between SVM-LINEAR, SVM-SIGMOID, and SVM-RBF predictions of Young's modulus using 10 % of the data for training and 90 % for testing

| | Linear | Sigmoid | RBF |
|---|---|---|---|
| *r* | 0.9612 | 0.9203 | 0.9905 |
| RMSE | 2.8075 | 4.5299 | 1.5067 |
| AAE | 1.9659 | 3.3711 | 0.9219 |
| MAE | 11.6000 | 16.8300 | 7.6000 |

approximate the core-measured Poisson's ratio values. The best results are achieved with the RBF kernel function. Similarly, all three kernel functions performed well to predict the core-measured Young's modulus, but the RBF function performs slightly better than the linear and sigmoid kernel functions.

## 13 Conclusions

SVMs integrated with a fuzzy-based curve and surface input variable ranking analysis has demonstrated its potential applicability to develop interpretation models of the Poisson's ratio and Young's modulus under limited core data conditions. The main conclusions are as follows:

1. The SVM model is easier to construct than that of an artificial neural network model.
2. The SVMs formulation facilitates unique global solution compared to BPNN which often suffers multiple local minima.
3. The SVMs formulation offers natural means to deal with sparse data given that the number of support vectors used is equal to the number of training data points.
4. The SVMs shows high learning capability for both Poisson's ratio and Young's modulus under the presence of limited core data.
5. The results show that SVR yields a better model to predict Poisson's ratio than a backpropagation neural networks model.

6. The results demonstrate that the prediction errors for the Young's modulus obtained from the SVR decrease faster as the training data size grows than that of the backpropagation neural network.
7. Linear, sigmoid, and Gaussian RBF kernel functions show high prediction generalizability for Poisson's ratio and Young's modulus. The RBF function exhibits slightly better performance than the other two kernel functions.

## References

Abdulraheem A, Ahmed M, Vantala A, Parvez T (2009) Prediction of rock mechanical parameters for hydrocarbon reservoirs using different artificial intelligence techniques. Paper SPE 126094 presented at the SPE Saudi Arabia section technical symposium, Alkhobar, Saudi Arabia, 9–11 May 2009

Al-Anazi A, Gates ID (2009) Fuzzy logic data-driven permeability prediction for heterogeneous reservoirs. Paper SPE 121159 presented at the 2009 SPE EUROPEC/EAGE annual conference and exhibition, Amsterdam, The Netherlands, 8–11 June 2009

Al-Anazi A, Gates ID (2010a) On the capability of support vector machines to classify lithology from well logs. Nat Resour Res 19(2):125–139. doi:10.1007/s11053-010-9118-9

Al-Anazi A, Gates ID (2010b) Support vector regression for permeability prediction in a heterogeneous reservoir: a comparative study. SPE Res Eval Eng 13(3):485–495. SPE-126339-PA. doi:10.2118/126339-PA

Al-Anazi A, Gates ID (2010c) A support vector machine algorithm to classify lithofacies and model permeability in heterogeneous reservoirs. Eng Geol 114:267–277. doi:10.1016/j.enggeo.2010.05.005

Al-Anazi A, Gates ID (2010d) Support vector regression for porosity prediction in a heterogeneous reservoir: a comparative study. Comput Geosci 36(12):1494–1503

Ameen MS, Smart Brian GD, Mc J, Somerville Sally Hammilton, Naji Nassir A (2009) Predicting rock mechanical properties of carbonates from wireline logs (a case study: Arab-D reservoir, Ghawar field, Saudi Arabia). Mar Petrol Geol 26(4):430–444

Barree RD, Gilbert JV, Conway MW (2009) Stress and rock property profiling for unconventional reservoir stimulation. Paper SPE 118703 presented at the SPE hydraulic fracturing technology conference, The Woodlands, Texas, 19–21 Jan 2009

Cherkassky V, Shao X (2001) Signal estimation and denoising using VC-theory. Neural Netw 14:37–52

Choisy C, Belaid A (2001) Handwriting recognition using local methods for normalization and global methods for recognition. In: Proceedings of sixth international conference on document analysis and recognition, pp 23–27

DTREG (2009) Predictive modeling software user's manual, version 9.1 (http://www.dtreg.com)

Fung C, Wong K, Eren H (1997) Modular artificial neural network for prediction of petrophysical properties from well log data. IEEE Trans Instrum Measure 46(6):1295–1299

Gao D, Zhou J, Xin L (2001) SVM-based detection of moving vehicles for automatic traffic monitoring. IEEE Intell Transp Syst 745–749

Gatens JM III, Harrison CW III, Lancaster DE, Guldry FK (1990) In-situ stress tests and acoustic logs determine mechanical properties and stress profiles in the devonian shales. SPE Formation Eval 5(3):248–254

Hastie T, Tibshirani R, Friedman J (2001) The elements of statistical learning: data mining, inference, and prediction. Springer, New York

Helle H, Bhatt A (2002) Fluid saturation from well logs using committee neural networks. Petrol Geosci 8:109–118

Helle H, Ursin B (2001) Porosity and permeability prediction from wireline logs using artificial neural networks: a North Sea case study. Geophys Prospect 49:431–444

Huang Y, Gedeon TD, Wong PM (2001) An integrated neural-fuzzy-genetic-algorithm using hyper-surface membership functions to predict permeability in petroleum reservoirs. J Eng Appl Artif Intell 14:15–21

Huang Z, Shimeld J, Williamson M, Katsube J (1996) Permeability prediction with artificial neural network modelling in the venture gas field, offshore eastern Canada. Geophysics 61:422–436

Kecman V (2005) Support vector machines—an introduction. In: Wang L (ed) Support vector machines: theory and applications, Chap. 1. Springer, Berlin, pp 1–47

Khaksar A, Taylor PG, Fang Z, Kayes T, Salazar A, Rahman K (2009) Rock strength from core and logs, where we stand and ways to go. Paper SPE 121972 presented at the EUROPEC/EAGE conference and exhibition, Amsterdam, The Netherlands, 8–11 June 2009

Kim K, Jung K, Park S, Kim HJ (2001) Support vector machine-based text detection in digital video. Pattern Recognit 34:527–529

Li Z, Weida Z, Licheng J (2000) Radar target recognition based on support vector machine. In: Proceedings of 5th international conference on signal processing, vol 3, pp 1453–1456

Lu J, Plataniotis K, Ventesanopoulos A (2001) Face recognition using feature optimization and v-support vector machine. IEEE neural networks for signal processing, vol 11, pp 373–382

Ma C, Randolph M, Drish J (2001) A support vector machines-based rejection technique for speech recognition. In: Proceedings of IEEE international conference on acoustics, speech, and signal processing, vol 1, pp 381–384

Montmayeur H, Graves RM (1985) Prediction of static elastic/mechanical properties of consolidated and unconsolidated sands from acoustic measurements: basic measurements. Paper SPE 14159 presented at the 60th SPE annual technical conference and exhibition, Las Vegas, NV, 22–25 Sept 1985

Montmayeur H, Graves RM (1986) Prediction of static elastic/mechanical properties of consolidated and unconsolidated sands from acoustic measurements: correlations. Paper SPE 15644 presented at the 61st SPE annual technical conference and exhibition, Orleans, LA, 5–8 Oct 1986

Rogers SJ, Chen HC, Kopaska-Merkel DC, Fang JH (1995) Predicting permeability from porosity using artificial neural networks. AAPG Bull 79:1786–1797

Schölkopf B, Smola AJ (2002) Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT Press, Cambridge, MA

Serra O (1984) Fundamentals of well-log interpretation. Elsevier, Amsterdam

Suykens JAK, Van Gestel T, Brabanter J, De Moor B, Vandewalle J (2002) Least squares support vector machines. World Scientific, Singapore

Van Gestel T, Suykens J, Baestaens D, Lambrechts A, Lanckriet G, Vandaele B, De Moor B, Vandewalle J (2001) Financial time series prediction using least squares support vector machines within the evidence framework. IEEE Trans Neural Netw 12(4):809–821

Vapnik V, Chervonenkis A (1974) Theory of pattern recognition [in Russian]. Nauka, Moscow (German trans: Wapnik W, Tscherwonenkis A., Theorie der Zeichenerkennung, Akademie, Berlin, 1979)

Vapnik VN (1982) Estimation of dependences based on empirical data. Springer, Berlin

Vapnik V (1995) The nature of statistical learning theory. Springer, New York

Vapnik V (1998) Statistical learning theory. Wiley, New York

Widarsono B, Wong PM, Saptono F (2001) Estimation of rock dynamic elastic property profiles through a combination of soft computing, acoustic velocity modeling and laboratory dynamic test on core samples. Paper SPE 68712 presented at the SPE Asia Pacific oil and gas conference and exhibition, Jakarta, Indonesia, 17–19 Apr 2001

# Use of Active Learning Method to Determine the Presence and Estimate the Magnitude of Abnormally Pressured Fluid Zones: A Case Study from the Anadarko Basin, Oklahoma

**Constantin Cranganu and Fouad Bahrpeyma**

**Abstract** We discuss active learning method (ALM) as an artificial intelligent approach for predicting a missing log (DT or sonic log) when only two other logs (GR and REID) are present. Applying ALM approach involves three steps: (1) supervised training of the model, using available GR, REID, and DT logs; (2) confirmation and validation of the model by blind-testing the results in a well containing both the predictors (GR, REID) and the target (DT) values; and (3) applying the predicted model to wells containing the predictor data and obtaining the synthetic (simulated) DT values. Our modeling approach indicates that the performance of the algorithm is satisfactory, while the time performance is significant. The quality of our simulation procedure was assessed by three parameters, namely mean square error (MSE), mean relative error (MRE), and Pearson product–momentum correlation coefficient ($R$). The values obtained for these three quality-control parameters appear congruent, with the exception of MRE, regardless of the training set used (*reduced* vs. *complete*). ALM performance was measured also by the time required to attain the desirable outcomes: five depth levels of investigation took a little more than one minute of computing time during which MSE dropped significantly. We performed twice the regression analysis: with and without normalization of input data sets (training well and validation well) using the procedure indicated by previous works. The results show minimum differences in quality assessment parameters (MSE, MRE, and $R$), suggesting that data normalization is not a necessary step in all regression algorithms. We employed both the measured and simulated sonic logs DT to predict the presence and estimate

C. Cranganu (✉)
Department of Earth and Environmental Sciences, Brooklyn College, The City University of New York, 2900 Bedford Avenue, New York City, NY 11210, USA
e-mail: cranganu@brooklyn.cuny.edu

F. Bahrpeyma
Faculty of Electrical and Computer Engineering, Shahid Beheshti University G.C., Tehran, Iran



C. Cranganu et al. (eds.), *Artificial Intelligent Approaches in Petroleum Geosciences*,
DOI 10.1007/978-3-319-16531-8_6

the depth intervals where overpressured fluid zone may develop in the Anadarko Basin, Oklahoma. Based on our interpretation of the sonic log trends, we inferred that overpressure regions are developing between ~1250 and 2500 m depth and the overpressured intervals have thicknesses varying between ~700 and 1000 m. These results match very well our previous results reported in the Anadarko Basin, using the same wells, but different artificial intelligent approaches.

**Keywords** Active learning method · Well logs · Sonic log · Abnormally pressured zones · Anadarko Basin

## 1 Introduction

In petroleum geosciences, it is necessary most of the time to characterize pore-fluid pressures, rock lithologies, and a large number of petrophysical properties of reservoir rocks (porosity, permeability, water/oil saturation, etc.). Those parameters are best determined by in situ and core measurements. When such measurements are not available, due to financial, technical, or other impediments, the next best available way of obtaining those data is using a suite of geophysical and petrophysical logs.

Common recorded logs, such as gamma ray (GR), dual induction, deep resistivity (REID), density porosity, photoelectric absorption factor (PEF), or self-potential (SP), provide information about a specific physical characteristic of the rocks penetrated by and surrounding the borehole (e.g., natural radioactivity content, electric resistivity, density, mineralogy, etc.). Among various logs recorded by petroleum industry, the compressional acoustic or sonic log (DT) has been used many times to predict rock porosity (so-called "acoustic porosity", Ellis and Singer (2007)), to evaluate petrophysical properties or to characterize areas containing pore fluids with abnormal pressure (overpressurized) (Hearst et al. 2000; Cranganu 2007; Cranganu and Bautu 2010; Cranganu and Breaban 2013).

Despite its proven value in estimating rock porosity, or studying and estimating the presence of abnormally pressured pore fluids in sedimentary basins containing oil and gas reservoirs, the sonic log is not always a part of a commonly recorded well logs. The reasons include lacking a full suite of logs, lacking of data due to incomplete logging, instrument failure (damage or faulty), or poor-quality recordings (Bahrpeyma et al. 2013b; Cranganu and Bautu 2010; Cranganu and Breaban 2013). As a result, some wells may lack the sonic log entirely or the log is only partially recorded.

To overcome such situations, we propose using a soft computing method—active learning method (ALM)—to synthesize missing sonic (DT) logs when only other common logs (e.g., GR and REID) are present. Thus, we will be able to more effectively map porosity variations, to detect changes in pore-fluid pressure, and to increase the density of control data in wells without recorded DT.

As hinted above, a major use of DT logs is determining the presence and estimating the amplitude of areas with abnormal pore-fluid pressures via effective rock stress estimations. In other words, the sonic log can be used as a predictor of pore-fluid pressures because it responds to changes in porosity or compaction produced when abnormal pore-fluid pressures are present in a sedimentary basin. A case study from the Anadarko Basin, Oklahoma, will be presented to illustrate this use of DT log.

## 2 Active Learning Method—Background

ALM was originally introduced by Shouraki and Honda (1999) as a soft computing tool for modeling unknown multiple inputs-single output (M.I.S.O.) systems. ALM performs simulation based on the intelligent information-handling processes existing inside the human brain. The engine of ALM is the ink drop spread (IDS) (Saeed Bagheri and Honda 1997) operator, which acts as a fuzzification operator. IDS works based on a non-exact processing procedure without suffering from complex formulas (Murakami and Honda 2005a, b, c).

ALM has proved its ability for estimating missing logs in hydrocarbon reservoirs (Bahrpeyma et al. 2013a), modeling chlorophyll and pigment retrieval (Shahraini et al. 2005), modeling geophysical variables (Shahraiyni et al. 2007), and controlling design problems (Behnia et al. 2011; Ghorbani et al. 2010; Sakurayi et al. 2003).

In ALM, the behavior of the output is depicted (projected) onto a two-dimensional (2D) surface with respect to each input parameter in order to reduce the complexity of the modeling procedure for M.I.S.O. systems. In fact, the projected 2D planes are single input-single output (S.I.S.O.) candidate models for the given output parameter. The fuzzy interpolation is then responsible for aggregating the candidate S.I.S.O. models into a unified model according to the degree of relativity the candidate models exhibit for describing the behavior of the output (Fig. 1).

As shown in Fig. 1, ALM breaks an M.I.S.O. system into some S.I.S.O. subsystems in order to cope with less computational complexities when trying to extract the relationships. The idea is originated from the fact that handling the projected Input-Output 2D surfaces reduces the complexity imposed by extraction of multi-dimensional relations existing between the dimensions inside an M.I.S.O. system. Therefore, complexity is meaningfully reduced. At the end, the fuzzy interpolation revives the impact of each input parameter on the output parameter.

The flowchart of ALM is illustrated in Fig. 2.

At the 2D surface of each S.I.S.O. sub-system, the general behavior of the output with respect to the input is called the narrow path (NP). Figure 3 illustrates the process of NP extraction through IDS and COG (Bahrpeyma et al. 2013a; Saeed Bagheri and Honda 1997) operators in an operation board.

The NP is a continuous and function-like path representing the output as the function of the input in a 2D space. The function-like path, as a model of an S.I.S.

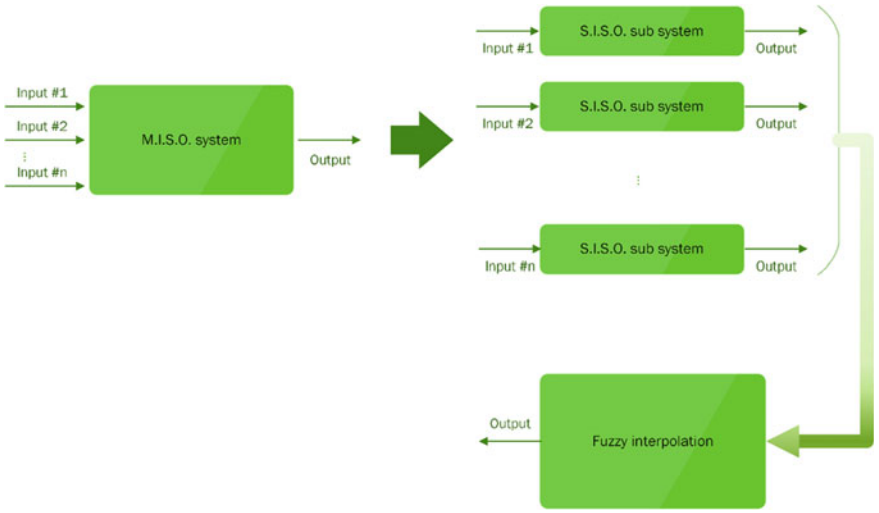


**Fig. 1** Handling complexity by breaking the M.I.S.O. system into some S.I.S.O. sub-systems and aggregating them into a unified model inside ALM

O. system, should be extracted from a continuous path to provide generalization ability for unseen data. However, the pure S.I.S.O. sub-system consists of some discrete points in a 2D surface (Fig. 3a, b) which display no aspect of continuity.

The IDS operator is used inside ALM as a fuzzification method to extract fuzzy continuity from a set of discrete points inside a 2D surface. Thus, inside the 2D surface of a S.I.S.O. sub-system, for each projected data point, the IDS operator propagates membership values to the neighborhood in an attenuated pattern. After application of IDS operator on each data point in the 2D surface, the outcome is a fuzzy continuous area that is called the IDS plane (Fig. 3c). The fuzzy continuous area is then converted to a narrow function-like path by the application of the COG operator. The outcome is a function that represents the general behavior of the output with respect to the input, i.e., the NP (Fig. 3d).

To provide an environment for implementing the IDS operator, Bahrpeyma et al. (2013a) used a $d\%$ margined operation board (Fig. 3), insuring that membership propagation can be simulated through the neighborhood of rooms inside a 2D board.

The domain of $i$th S.I.S.O. sub-system ($X_i - Y$) is defined as:

$$\begin{cases} D_{X_i} : x | \min(X_i) < x < \max(X_i) \\ D_Y : y | \min(Y) < y < \max(Y) \end{cases} \quad (1)$$

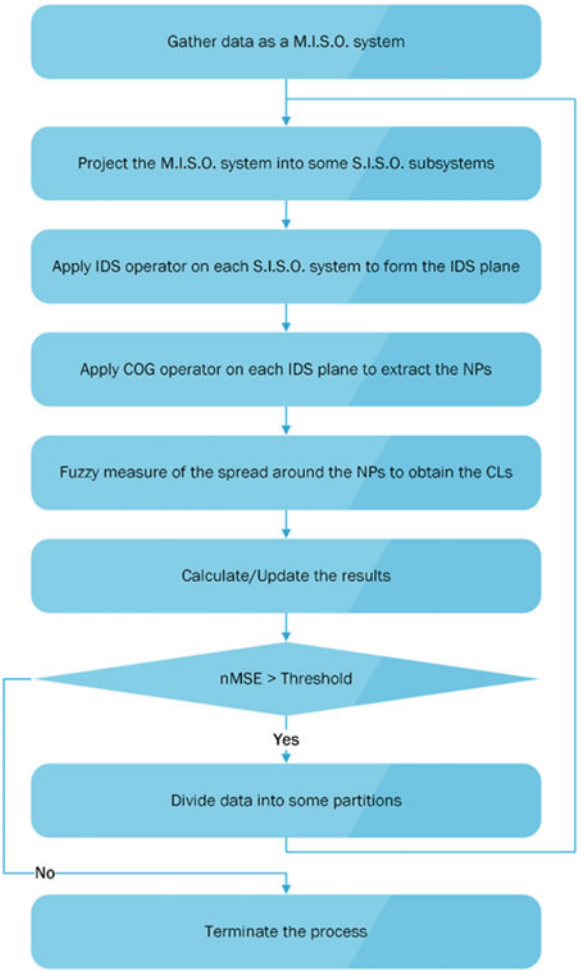


**Fig. 2** The flowchart of ALM

The domain of $d\%$ margined domain is considered as:

$$\begin{cases} D_{X_i}^{\frown} : x | \min(X_i) - \text{margin}_{X_i}^{d\%} < x < \max(X_i) + \text{margin}_{X_i}^{d\%} \\ D_{Y}^{\frown} : y | \min(Y) - \text{margin}_{Y}^{d\%} < y < \max(Y) + \text{margin}_{Y}^{d\%} \end{cases} \tag{2}$$

where $\text{margin}_{X}^{d\%}$ is calculated by:

$$\text{margin}_{X}^{d\%} = [\max(X_i) - \min(X_i)] \times \frac{d}{100} \tag{3}$$

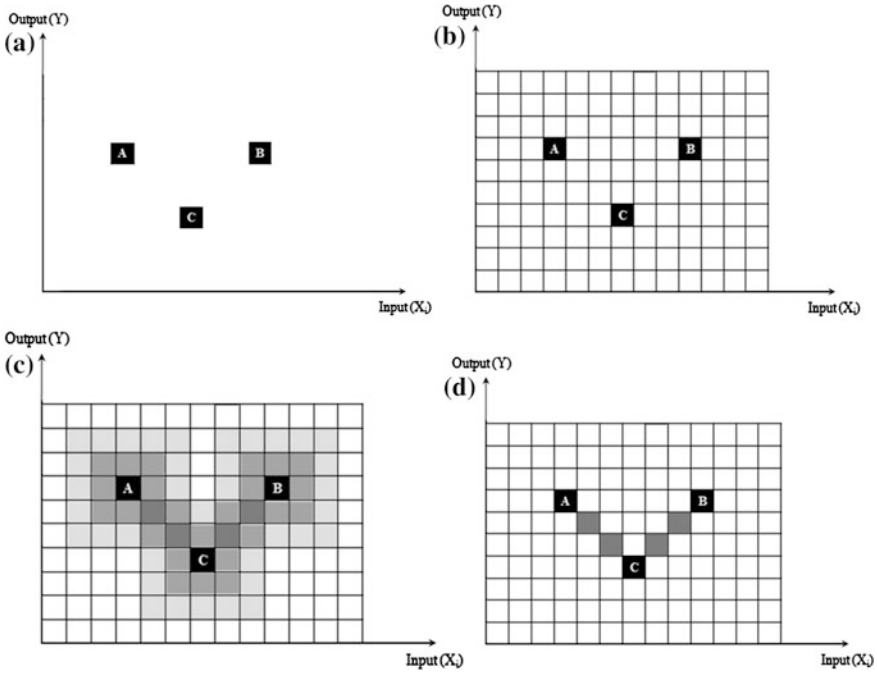


**Fig. 3** Extracting the NPs by applying IDS and COG operators. **a** Projection of the M.I.S.O. system onto a 2D surface. **b** Mapping the observations onto an operation board. **c** Applying IDS operator on observed data points to form IDS planes. **d** Applying COG operator on IDS planes to extract the NPs

Finally, the size of each unit $\left[U^{M}_{X_i}, U^{M}_{Y}\right]$ in $i$th S.I.S.O. sub-system for an $M \times M$ board is:

$$\begin{cases} U^{M}_{X_i} = \dfrac{\max\left(\widehat{X}_i\right) - \min\left(\widehat{X}_i\right)}{M} \\ U^{M}_{Y} = \dfrac{\max\left(\widehat{Y}\right) - \min\left(\widehat{Y}\right)}{M} \end{cases} \tag{4}$$

Propagation of fuzzy membership values is attenuated by distance to provide more membership values for the closer neighbors. Regarding a linear function for attenuation, a room inside the board receives $\mu$ from a projected point:

$$\mu = R - \sqrt{u^2 + v^2} + 1; \quad -R \le u, v \le R,$$
$$\Delta d(x_s + u, y_s + v) \Rightarrow \begin{cases} \mu; & \text{if } \mu > 0 \\ 0; & \text{otherwise} \end{cases} \tag{5}$$

where $R$ is the radius of the IDS operator, ($x_s$, $y_s$) are the coordinates of the propagator, ($u$, $v$) is the distance between the receiver and:

$$\Delta d(x_s + u, y_s + v) = \sqrt{u^2 + v^2} \tag{6}$$

After forming the IDS planes, to extract the NP $\psi(x)$, the COG operator is applied on the IDS plane:

$$\psi(x) = \frac{\sum_{j \in Y(x)} \mu_j Y_j}{\sum_{j \in Y(x)} Y_j} \tag{7}$$

where $Y$ is the output axis of the board.

After extracting NPs, the NPs of multiple S.I.S.O. systems are unified into a single model through a fuzzy interpolation. The fuzzy interpolation is a weighted averaging mechanism in which each S.I.S.O. system participates in the aggregation with a weight called the confidence level (CL). For each S.I.S.O. system, the spread of data is measured around the corresponding NP:

$$\text{Spread}_k = \frac{1}{n} \sum_{i=1}^{n} \left( \psi_k \left( x_i^k \right) - y_i \right)^2 \tag{8}$$

where $x_i^k$ and $y_i$ are the coordinates of $i$th data point and $\psi_k$ is the projection of the $i$th point on $k$th NP.

The interpolation mechanism for an $m$ inputs, single output system is implemented by:

$$y_{\text{final}} = \frac{\sum_{k=1}^{m} w_k y_k}{\sum_{k=1}^{m} w_k}, \tag{9}$$

where $w_k$ (CL) is the weight of $k$th candidate model:

$$w_k = \frac{1}{\text{Spread}_k} \tag{10}$$

An example of the NPs for a three inputs-single output system in presented in Fig. 4.

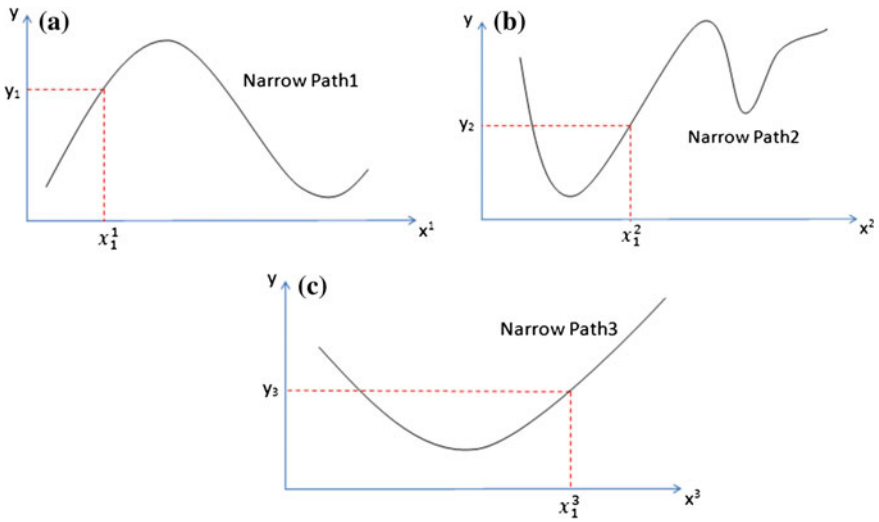


**Fig. 4** The narrow paths of a rule for a 3 inputs-single output system

For the example in Fig. 4, the output of the unseen sample $x_1 = [x_1^1, x_1^2, x_1^3]$ is calculated by:

$$\mathrm{Out}\left(x_1^1, x_1^2, x_1^3\right) = \frac{C_1 \times \psi_1\left(x_1^1\right) + C_2 \times \psi_2\left(x_1^2\right) + C_3 \times \psi_3\left(x_1^3\right)}{C_1 + C_2 + C_3} \tag{11}$$

In the end, ALM recursively localizes the entire modeling process in order to improve the accuracy of the algorithm. The localization process, which is performed by recursively partitioning the domain of the M.I.S.O. system into some sub-domains, is performed to algorithmically model the local behaviors of the system that is damped by the general IO behavior extraction procedure. Localization can help including more local behaviors and improving the accuracy of the entire process. Therefore, after localization, multiple localized M.I.S.O. systems are processed by information-handling procedure of ALM (Fig. 5).

The recursive partitioning procedure inside ALM is similar to the training process of a decision tree. Thus, to avoid overfitting, early stopping (which is one of the well-known solutions to overfitting) is performed by ALM. Early stopping inside ALM includes measuring the overall generalization error through cross-validation. Thus, if the generalization error, which is measured via normalized nMSE, is less than the threshold, partitioning will not happen. nMSE is calculated by:

$$\mathrm{nMSE} = \frac{\sum_{i=1}^{N} (Y_i - T_i)^2}{\sum_{i=1}^{N} (Y_i - \bar{Y})^2} \tag{12}$$

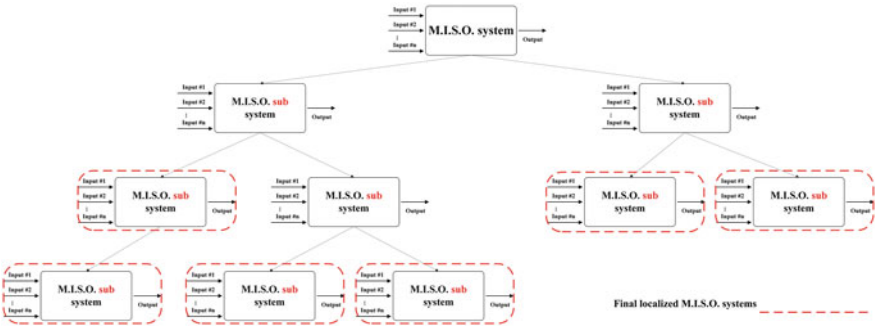


**Fig. 5** Partitioning the M.I.S.O. domain into some sub-domains and modeling based on localized M.I.S.O. sub-models

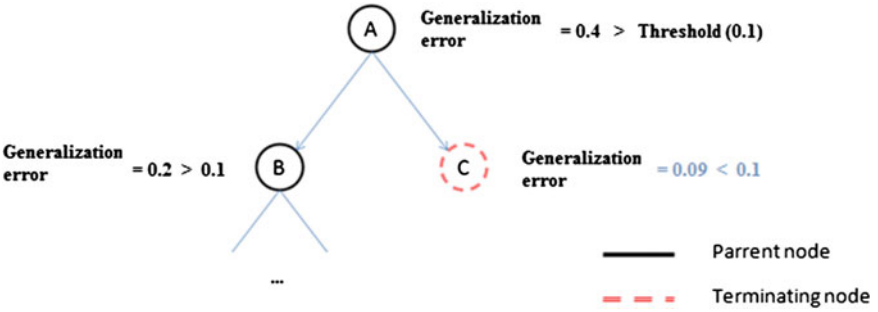


**Fig. 6** Choosing from inputs to divide the data at each node in ALM tree based on the spread

Figure 6 depicts the decision process of partitioning for *Threshold* = 0.1; partitioning stops at the terminating node.

In this paper, if the decision implies to carry on partitioning in the current stat (node), the data are divided into two partitions from the midpoint of the input axis inside the S.I.S.O. system (2D surface), which has gained the greatest weight in the interpolation mechanism. Greater weight/confidence level implies on that the underlying input parameter has more impact on determination of the behavior of the output. Therefore, dividing the data based on this S.I.S.O. system helps preserving the greatest CL, while the CLs of the other S.I.S.O systems may be improved in aspect of accuracy. This approach is a heuristic, although the heuristic search is a guided search and does not guarantee to present the optimal solution; it can cause to reach to satisfactory solutions (Abbas et al. 2002).

## 3 Data Sets and Features

In order to use ALM to estimate missing sonic logs in the Anadarko Basin, Oklahoma, we employed a 3-step approach: supervised training to create a simulation model, verification and confirmation of the model, and application of the model to new wells. More details about this approach can be found in our previous work (Cranganu 2007; Cranganu and Bautu 2010; Cranganu and Breaban 2013).

The data sets are represented by GR[1], deep resistivity[2] (REID), sonic[3] (DT), and caliper[4] (CAL) logs obtained from four wells drilled in the Anadarko Basin, Oklahoma. The CAL log is not participating directly in the modeling process, because it is not related unequivocally to a physical property. Rather, CAL log serves as an indicator of the borehole environmental "health": it points out to the presence of such drilling problems as caverns, mud cake, wash out zones, etc. The use of CAL log and the selection of most suitable depth intervals for simulation (*reduced data set* vs. *complete data set*) are fully described in Cranganu (2005, 2007), Cranganu and Bautu (2010), and Cranganu and Breaban (2013).

## 4 Evaluation Measures

The quality of the training and verification models was assessed by using the following parameters:

The MSE was computed as the average over all squared deviations of the prediction from the real values:

$$\mathrm{MSE}(f) = \frac{1}{n}\sum_{i=1}^{n}\left(f\left(x^{(i)}\right) - y^{(i)}\right)^2 \tag{13}$$

The mean relative error (MRE) was computed as the average deviation reported to the real value:

$$\mathrm{MRE}(f) = \frac{1}{n}\sum_{i=1}^{n}\frac{f\left(x^{(i)}\right) - y^{(i)}}{y^{(i)}} \tag{14}$$

The *Pearson product–moment correlation coefficient, R*, is also used throughout this paper to measure the linear correlation between the measured and predicted values.

[1] Natural gamma radiation, in uAPI.

[2] Electric resistivity variation, in Ωm.

[3] Propagation time of seismic waves in and around a borehole, in μs/ft.

[4] Measurement of the borehole diameter variations, in in.

## 5 Training, Validation, and Application

We used a 3-step approach to perform our ALM simulation:

(a) *Training*: In this step, we computed the regression function based on a training set consisting of selected values of GR, REID, and DT from well T&T 1–10 (aka the *training well*). Because in ALM, the learned regression function depends on the values of input parameters and the procedural steps described in Sect. 2, we had to run several configurations of the learning pattern. The best model was considered the one that achieved the most accurate predictions (equivalent to the lowest MSE) (Fig. 7).

(b) *Validation*: In this step, the ALM algorithm created during training phase was evaluated on a data set from a *validation well*. This last data set consisted from a suite of GR, REID, and DT log values, but only GR and REID were used to simulate the DT values. Then, the simulated values were compared to the recorded ones. This step is meant to verify and validate the algorithm ability to simulate a DT log. The *validation well* was Whittenberg 3–29 (Fig. 8).

(c) *Application*: Once the ALM algorithm was verified and validated, it was applied to two *application wells* (Ledbetter 1–18, Fig. 9, and Smith 1–13, Fig. 10) where only GR and REID were recorded.

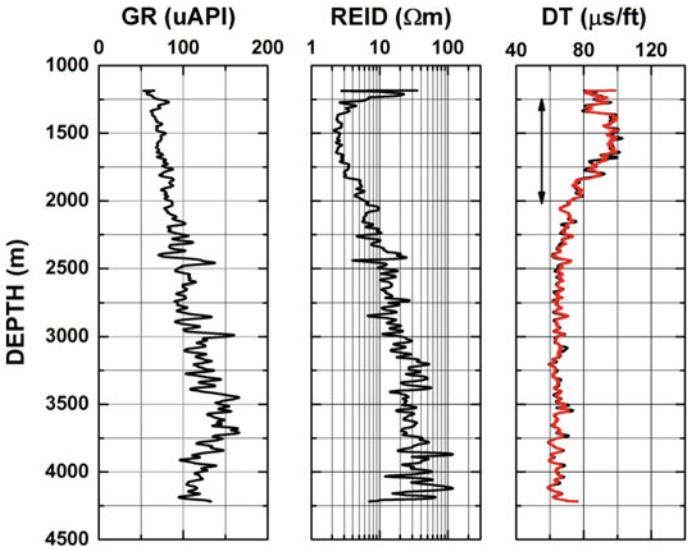


**Fig. 7** Training results using T&T 1–10 well data set. The *red DT curve* is the simulated one. The *arrow* indicates the location and depth extension of the overpressure

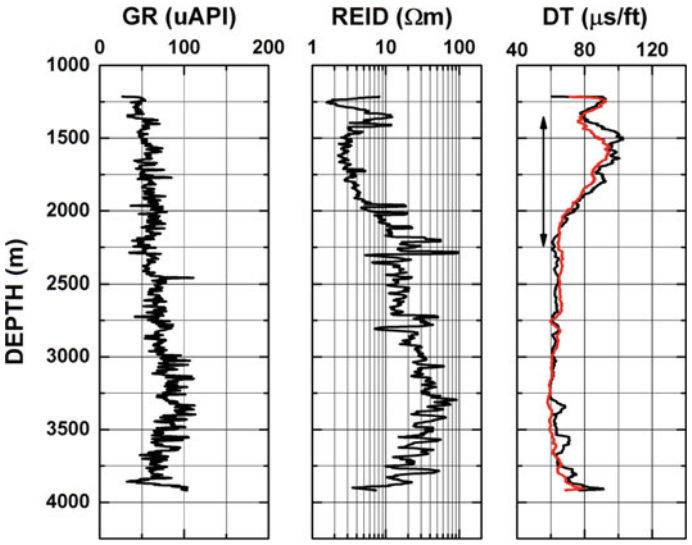


**Fig. 8** Verification and validation results using Whittenberg 3–29 well data set. The *red DT curve* is the simulated one. The *arrow* indicates the location and depth extension of the overpressure

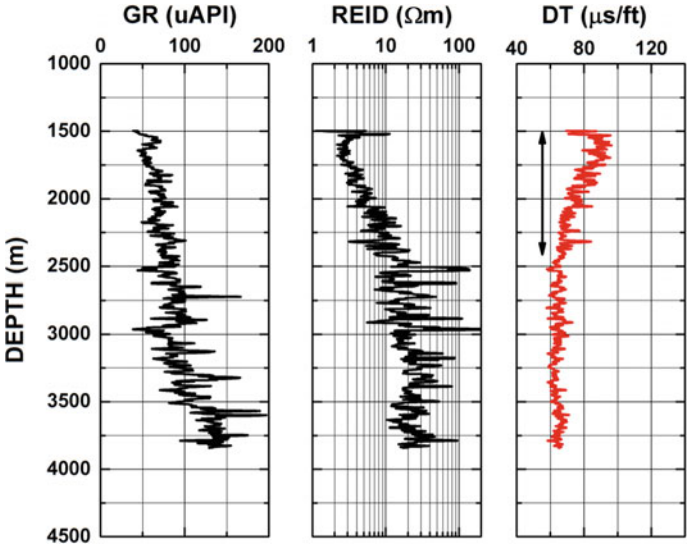


**Fig. 9** Predicted DT values for application well Smith 1–13. The *arrow* indicates the location and depth extension of the overpressure

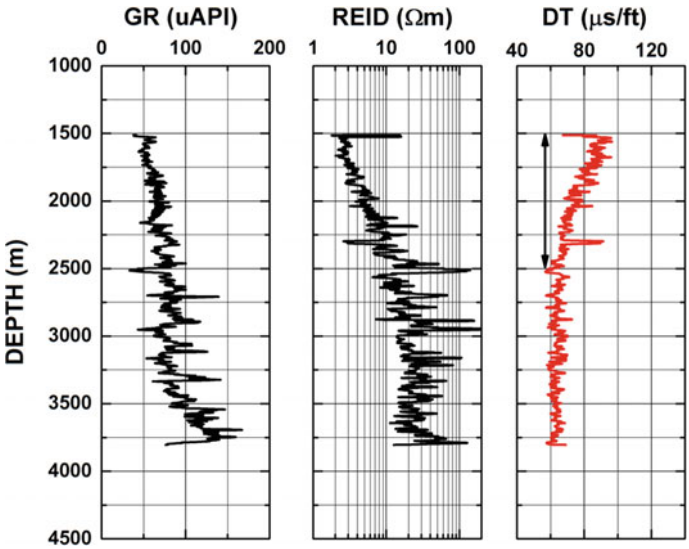


**Fig. 10** Predicted DT values for application well Ledbetter 1–18. The *arrow* indicates the location and depth extension of the overpressure

## 6 Data Normalization

When doing data analysis, normalization is a popular pre-processing step. One main goal of data normalization is speeding up of the convergence of the gradient descent algorithm, which represents the basis of many machine learning jobs.

When using support vector regression, Cranganu and Breaban (2013) normalized all their input data sets and found that normalization yielded significant impacts on accuracy of predictions and a better compression of the training model.

However, data normalization is not a necessary step in all regression algorithms. For example, while performing symbolic regression with genetic programing, (Cranganu and Bautu 2010) found that data normalization does not have a significant influence on the final results of regression analysis.

In this paper, we performed twice the regression analysis: with and without normalization of input data sets using the procedure indicated by Cranganu and Breaban (2013). The results are presented in Table 1.

## 7 Results

ALM took 67.09 s to perform five-level localization through the domain of the problem (Fig. 11). At the end, results of validation in the Whittenberg well including MSE = 90.99, MRE = −0.01251 and $R = 0.8032$, indicate that the performance of the

**Table 1** The impact of normalization procedure on the prediction accuracy of the trained model (DT = f(GR, REID)) measured by MSE, MRE, and *R* on the training and validation sets

| | T&T 1–10 | | | Whittenberg 3–29 | | |
|---|---|---|---|---|---|---|
| | MSE | MRE | R | MSE | MRE | *R* |
| Without normalization | 45.1622 | 0.0000886 | 0.8900 | 90.9971 | −0.01251 | 0.8032 |
| With normalization | 46.2572 | 0.0000980 | 0.8890 | 91.6786 | 0.00049 | 0.8020 |

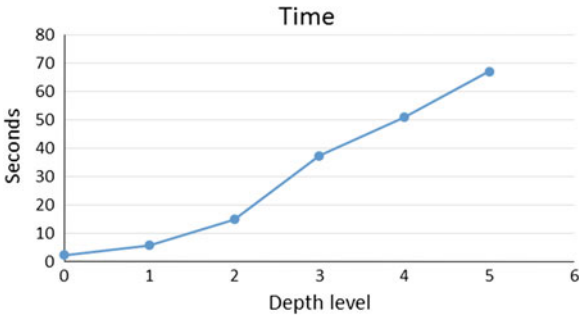


**Fig. 11** Time necessary for reaching the depth levels of localization process

algorithm is satisfactory while the time performance is significant. Table 2 illustrates the results of the ALM process for training, and validation tests.

Table 3 and Fig. 12 illustrate the process of ALM for reaching to the desired performance in the localized sub-models at different localization depth levels.

## 8 Estimating the Presence and Depth Extension of Overpressured Zones in the Anadarko Basin, Based on Sonic Logs

As detailed in our previous work (Cranganu 2007; Cranganu and Bautu 2010; Cranganu and Breaban 2013) and references therein, many sedimentary basins may experience pore-fluid pressures that differ from the hydrostatic ("normal") pressure:

**Table 2** Results obtained when the regression function (DT = f(GR, REID)) was trained by ALM on the T&T 1–10 well and validated on Whittenberg 3–29 well

| | MSE | MRE | *R* |
|---|---|---|---|
| **Training set**: T&T 1–10 Reduced | 44.8996 | 0.0095 | 0.9032 |
| **Test set**: T&T 1–10 Complete | 45.1622 | 0.0000886 | 0.8900 |
| **Validation set**: Whittenberg 3–29 | 90.9971 | −0.01251 | 0.8032 |

**Table 3** The performance of ALM for time and accuracy in the process of localization in different depth levels

| Depth level | Time | MSE |
|---|---|---|
| 0 | 2.3784 | 207.534 |
| 1 | 5.8774 | 114.947 |
| 2 | 14.9953 | 97.838 |
| 3 | 37.337 | 52.005 |
| 4 | 51.0026 | 46.885 |
| 5 | 67.0907+ | 43.345 |

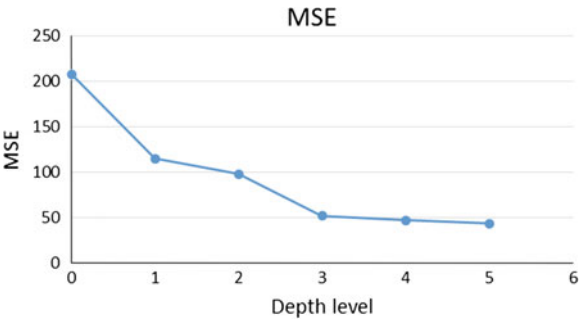


**Fig. 12** MSE in the process of localization in different depth levels

*overpressure* (pore-fluid pressure exceeds the hydrostatic pressure) or *underpressure* (pore-fluid pressure is lower than hydrostatic pressure).

The existence of overpressured fluid zones, their depth locations, along with estimated magnitude of overpressure, represents important decision factors when drilling through those zones. The main danger is that unknowingly penetrating an overpressured zone can create potential hazards, such as blowouts.

Along with other information (mud weight, repeat formation testing, drill-stem testing), the sonic log (DT) proved to be a reliable predictor of overpressure zone because it responds to changes in porosity or compaction generated by abnormal pore-fluid pressure. It seems that DT log is preferred in oil industry because it is accurate and sensitive to changes in rock stress subject to overpressured pore fluids, especially in sequences dominated by shales (Asquith and Gibson 2004; Hearst et al. 2000). Determining trends in DT log variations with depth is a conventional method to predict the presence and estimate the depth intervals of overpressured zones (Cranganu and Bautu 2010; Cranganu and Breaban 2013). Because the acoustic transit time (recorded by DT log) in sedimentary rock is dependent on the effective rock stress, an increase in DT values is indicating a lower effective rock stress and, consequently, a zone with overpressure.

On Figs. 7, 8, 9 and 10, we predicted the existence of overpressured pore-fluid zones in four wells drilled in the Anadarko Basin, Oklahoma. The vertical arrows in those figures suggest that overpressure regime is developing between $\sim$1250 and 2500 m depth and the overpressured intervals have thicknesses varying between $\sim$700 and 1000 m. These results match very well with previous results

reported in the Anadarko Basin, using the same wells, but different artificial intelligent approaches (Cranganu 2007; Cranganu and Bautu 2010; Cranganu and Breaban 2013).

## 9 Conclusions

This paper discusses Active Learning Method as a tool for predicting a missing log (DT or sonic log) when only two other logs (GR and REID) are present. Applying ALM approach involves three steps: (1) supervised training of the model, using available GR, REID, and DT logs; (2) confirmation and validation of the model by blind-testing the results in a well containing both the predictors (GR, REID) and the target (DT) values; and (3) applying the predicted model to wells containing the predictor data and obtaining the synthetic (simulated) DT values.

Our modeling approach indicates that the performance of the algorithm is satisfactory, while the time performance is significant. The quality of our simulation procedure was assessed by three parameters, namely MSE, MRE, and Pearson product–momentum correlation coefficient ($R$).

The values obtained for these three quality-control parameters appear congruent, with the exception of MRE), regardless of the training set used (*reduced* vs. *complete*): MSE values are 44.8996 and 45.1622, respectively, while $R$ values are 0.9032 and 0.8900, respectively. MRE values are more different: 0.0095 and 0.0000886, respectively, and we suppose that the difference is attributable to way MRE formula works.

ALM performance was measured also by the time required to attain the desirable outcomes: five depth levels of investigation took a little more than one minute of computing time during which MSE dropped from 207.534 to 43.345.

We performed twice the regression analysis: with and without normalization of input data sets (training well and validation well) using the procedure indicated by Cranganu and Breaban (2013). The results show minimum differences in quality assessment parameters (MSE, MRE, and $R$), suggesting that data normalization is not a necessary step in all regression algorithms. For example, while performing symbolic regression with genetic algorithm programing, Cranganu and Bautu (2010) found that data normalization does not have a significant influence on the final results of regression analysis.

We employed both the measured and simulated sonic logs DT to predict the presence and estimate the depth intervals where overpressured fluid zone may develop in the Anadarko Basin, Oklahoma. Based on our interpretation of the sonic log trends, we inferred that overpressure regions are developing between $\sim$1250 and 2500 m depth and the overpressured intervals have thicknesses varying between $\sim$700 and 1000 m. These results match very well our previous results reported in the Anadarko Basin, using the same wells, but different artificial intelligent approaches (Cranganu 2007; Cranganu and Bautu 2010; Cranganu and Breaban 2013).

ALM has the following advantages:

- ALM diminishes the heavy load of computational complexities, which naturally exist in approximations methods like ANN, multiple regression, and TSK-FIS. The computational complexities are always due to complex formulas.
- ALM has fast computational speed, which introduces itself as a proper suit for real-time applications.
- Because ALM devotes confidence levels to the input parameters for aggregation, the method can identify and exclude the unimportant inputs parameters from the modeling process. Therefore, extra parameters cannot mislead the algorithm to non-optimal partitioning and inference solutions.

ALM has the following disadvantages:

- Since the sub-domains are declared for NP extraction, ALM is limited to the domain of the training data. Therefore, any test data outside of the training boundaries can be denied by the algorithm and ALM cannot provide the estimation.
- The radius of the IDS operator should match the nature of the training data, such that the continuity of the IDS plane is guaranteed. Therefore, constant radius for IDS cannot provide the best optimality.
- ALM uses a heuristic for partitioning, and heuristics cannot guarantee that the best solution will be found.

## References

Abbas HA, Sarker RA, Newton CS (2002) Data mining: a heuristic approach. Idea Group Publishing, Hershey

Asquith GB, Gibson CR (2004) Basic well log analysis: AAPG methods in exploration series, vol ix. American Association of Petroleum Geologists, Tulsa, Okla, 244 p

Bahrpeyma F, Golchin B, Cranganu C (2013a) Fast fuzzy modeling method to estimate missing logs in hydrocarbon reservoirs. J Pet Sci Eng 112:310–321

Bahrpeyma F, Golchin B, Cranganu C (2013b) Fast fuzzy modeling method to estimate missing logs in hydrocarbon reservoirs. J Pet Sci Eng 112:310–321

Behnia H, Arab-Khaburi D, Ejlali A (2011) Enhanced direct torque control for doubly fed induction machine by active learning method. In: Proceedings of 46th international universities' power engineering conference, UPEC 2011, Soest-Germany, VDE VERLAG GMBH, Berlin, Offenbach, pp 1–6

Cranganu C (2005) Using artificial neural networks to predict abnormal pressures in the Anadarko Basin, Oklahoma. J Balkan Soc 8:343–348

Cranganu C (2007) Using artificial neural networks to predict abnormal pressures in the Anadarko Basin, Oklahoma. Pure Appl Geophys 164:2067–2081

Cranganu C, Bautu E (2010) Using gene expression programming to estimate sonic log distributions based on the natural gamma ray and deep resistivity logs: a case study from the Anadarko Basin, Oklahoma. J Pet Sci Eng 70:243–255

Cranganu C, Breaban M (2013) Using support vector regression to estimate sonic log distributions: a case study from the Anadarko Basin, Oklahoma. J Pet Sci Eng 103:1–13

Ellis DV, Singer JM (2007) Well logging for earth scientists. Springer, The Netherlands, 692 p
Ghorbani MJ, Akhbari M, Mokhtari H (2010) Direct torque control of induction motor by active learning method. In: Proceedings of 1st power electronic and drive systems and technologies conference. IEEE, Tehran, Iran, pp 267–272
Hearst JR, Nelson PH, Paillet FL (2000) Well logging for physical properties: a handbook for geophysicists, geologists, and engineers: Chichester, vol viii. Wiley, New York, p 483
Murakami M, Honda N (2005a) Classification performance of the IDS method based on the two-spiral benchmark. IEEE Int Conf Syst Man Cybern 3797–3803
Murakami M, Honda N (2005b) A Comparative study of the IDS method and feedforward neural networks. In: International joint conference on neural networks, Montreal, Canada, pp 1776–1781
Murakami M, Honda N (2005c) A study on the modeling ability of the IDS method: a soft computing technique using pattern-based information processing. Int J Approx Reason 45:470–487
Saeed Bagheri S, Honda N (1997) A new method for establishment and saving fuzzy membership functions. In: Proceedings of 13th fuzzy symposium, Toyama, Japan, pp 91–94
Sakurayi Y, Honda N, Nishino J (2003) Acquisition of control knowledge of nonholonomic system by active learning method. IEEE Int Conf Syst Man Cybern 2400–2405
Shahraini HT, Bagheri S, Fell F, Abrishamchi A, Tajrishy M, Fischer J (2005) Investigating the ability of active learning method for chlorophyll and pigment retrieval in case I waters using SeaWiFS wavelengths. In: International conference on advanced remote sensing, Riyadh, Kingdom of Saudi Arabia
Shahraiyni TH, Schaale M, Fell F, Fischer J, Preusker R, Vatandoust M, Shouraki BS, Tajrishy M, Khodaparast H, Tavakoli A (2007) Application of the active learning method for the estimation of geophysical variables in the Caspian Sea from satellite ocean colour observations. Int J Remote Sens 28:4677–4684
Shouraki SB, Honda N (1999) Recursive fuzzy modeling based on fuzzy interpolation. J Adv Comput Intell 3:114–125

# Active Learning Method for Estimating Missing Logs in Hydrocarbon Reservoirs

**Fouad Bahrpeyma, Constantin Cranganu and Behrouz Zamani Dadaneh**

**Abstract** Characterization and estimation of physical properties are two of the most important key activities for successful exploration and exploitation in the petroleum industry. Pore-fluid pressures as well as estimating permeability, porosity, or fluid saturation are some of the important example of such activities. Due to various problems occurring during the measurements, e.g., incomplete logging, inappropriate data storage, or measurement errors, missing data maybe observed in recorded well logs. This unfortunate situation can be overcome by using soft computing approximation tools that will estimate the missing or incomplete data. Active learning method (ALM) is such a soft computing tool based on a recursive fuzzy modeling process meant to model multi-dimensional approximation problems. ALM breaks a multiple-input single-output system into some single-input single-output sub-systems and estimates the output by an interpolation. The heart of ALM is fuzzy measuring of the spread. In this paper, ALM is used to estimate missing logs in hydrocarbon reservoirs. The regression and normalized mean squared error (MSE) for estimating density log using ALM were equal to 0.9 and 0.042, respectively. The results, including errors and regression coefficients, proved that ALM was successful on processing the density estimation. ALM is illustrated by an example of a petroleum field in the NW Persian Gulf.

**Keywords** Active learning method (ALM) · Ink drop spread (IDS) · Hydrocarbon reservoir · Well logs

F. Bahrpeyma (✉) · B.Z. Dadaneh
Faculty of Electrical and Computer Engineering, Shahid Beheshti University G.C., Tehran, Iran
e-mail: f.bahrpeyma@mail.sbu.ac.ir

C. Cranganu
Department of Earth and Environmental Sciences, Brooklyn College of the City University of New York, Brooklyn, NY, USA
e-mail: cranganu@brooklyn.cuny.edu



C. Cranganu et al. (eds.), *Artificial Intelligent Approaches in Petroleum Geosciences*,
DOI 10.1007/978-3-319-16531-8_7

## 1 Introduction

In the petroleum industry, petrophysical and geophysical logs are considered as important tools for reservoir evaluation, and they are frequently used to obtain reservoir parameters. Loss of data is a common characteristic of the logging procedures which is commonly due to incomplete logging and other failures which are addressed in Bahrpeyma et al. (2013), Cranganu and Bautu (2010), Cranganu and Breaban (2013), El-Sebakhy et al. (2012), Rezaee et al. (2008), and Saramet et al. (2008).

Soft computing tools have recently become very useful in the field of petroleum industry because of their capability to estimate petrophysical and geophysical parameters of oil wells (Cranganu and Bautu 2010; Cranganu and Breaban 2013).

Soft computing is a set of computational techniques for finding solutions to problems for which there is no way to solve them analytically, or problems which could be solved theoretically but practically impossible, due to the necessity of huge resources and/or enormous time required for computation. For these problems, methods inspired by nature are sometimes efficient and effective. Although the solutions obtained by these methods are not always identical to analytical solutions, a near optimal solution is sometimes enough in most practical purposes. The main goal of soft computing tools is to support systems which need dealing with huge amounts of data storage, complexity, and uncertainties. Time performance is another factor which demands soft computing tools to improve their performance wherever time is an important factor.

Active learning method (ALM) (Saeed Bagheri and Honda 1999) is a fuzzy-based soft computing tool inspired from the ability of the human brain to model the systems it faces. ALM as a macro-level brain imitation has been inspired by some behavioral specification of brain and human active learning ability. The process of ALM focuses on the remarkable human ability to effortlessly deal with intricate information.

Taheri-Shahraiyni et al. (2011) used ALM for simulating runoff in the Karoon basin. Taheri-Shahraiyni et al. (2007), showed how ALM can be used for the estimation of geophysical variables in the Caspian Sea employing satellite ocean color observations.

Bahrpeyma et al. (2013), used two ALM operators to model physical properties of hydrocarbon reservoirs. Although they used the IDS method for modeling purposes, they just used one step modeling procedure. This paper presents the complete implementation of the ALM method to accurately synthesize missing log data. In fact, this paper is an improvement on Bahrpeyma et al. (2013), to provide more accurate and reliable modeling procedure by recursively localizing the modeling process inside the problem domain.

## 2 Methodology

Active learning method (ALM) is a powerful soft computing tool inspired by the way the human brain deals with complexity when solving problems (Saeed Bagheri and Honda 1997a). The human brain can solve modeling problems without dealing with huge amount of computational complexities. Therefore, simulating the way the human brain models a system enables us to achieve a modeling technique with low complexity.

ALM uses different types of devoting membership functions called ink drop spread (IDS), which acts as a powerful engine. IDS method (Saeed Bagheri and Honda 1997a) is analogous to fuzzy logic since it is algorithmically modeled on the processes of the information handling in the human brain (Afrakoti et al. 2013; Sagha et al. 2008). Considering that conventional fuzzy logic uses linguistic and logical processing, the IDS method employs intuitive image information. This concept is derived from the hypothesis that humans interpret information as images rather than in numerical or logical forms (Murakami and Honda 2004). The main idea behind ALM is to break a multiple-input single-output (M.I.S.O) system into some single-input-single-output (S.I.S.O) sub-systems and aggregate the behavior of sub-systems to obtain the final output. This idea resembles the brain activity which stores the *behavior of data* instead of their *exact values*. Each S.I.S.O sub-system is expressed as a data plane (called the IDS plane), resulted from the projection of the gathered data on each input–output plane (Murakami and Honda 2005; Saeed Bagheri and Honda 1997b).

Two types of information can be extracted from an IDS plane. One is the behavior of the output with respect to each input variable. The other is the confidence level for each input variable which is proportional to the reciprocal of variance of data around the behavior of the output with respect to each input. Narrow paths (NP) are estimated by applying IDS on data points and Center of Gravity (COG) on data planes. IDS and COG are the first two main operators of ALM (Saeed Bagheri and Honda 1999).

The flowchart of the modified ALM presented in this paper is illustrated in Fig. 1.

The algorithm includes four steps:

**Step 1: Projecting data on each $X_i$–$Y$ plane**

To observe the behavior of the target variable (as output of the system), the given data which is a M.I.S.O system is projected into some two-dimensional (2D) input–output systems. In fact, an M.I.S.O. system is broken into some S.I.S.O sub-systems (Fig. 2).

The benefits of projection are as follows: (a) Observing and analyzing the behavior of output with respect to each input; (b) Recognizing the most effective inputs, and (c) Easing the processes of identification and computation.

The major problems of this projection are as follows: (a) Loosing the relationship between inputs, and (b) Calculating the final output.

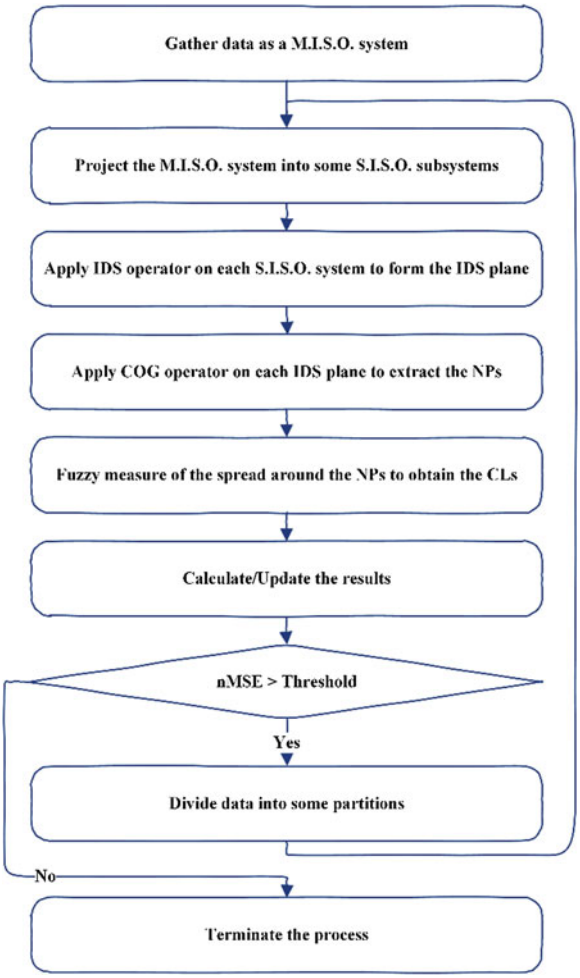


**Fig. 1** The flowchart of the proposed ALM

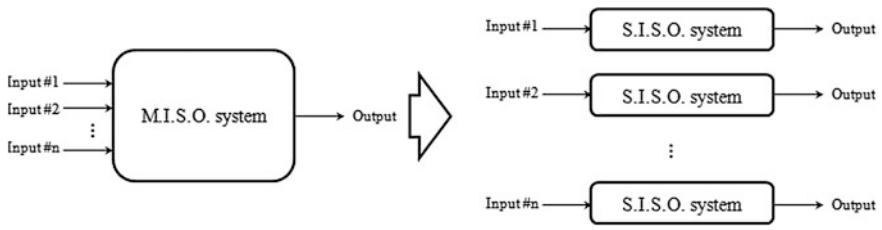


**Fig. 2** Projecting an M.I.S.O system into some S.I.S.O sub-systems

### Step 2: Finding the NPs for each 2D plane

To observe the behavior of the output with respect to each input, ALM extracts a function from each projected data samples. The function, called the NP, is a curve which shows just a unique value for each value in the domain of the input. The goal of extracting the NP function (as illustrated in Fig. 2) is to achieve a single model of the output based on each input parameter, separately:

$$y_i^{\text{out}} = \psi_i(x_i) \tag{1}$$

where $i$, $1 \leq i \leq n$ specifies the input index for $n$ system inputs, $y_i^{\text{out}}$ is the model of the output of the model constructed based on $i$th input, $\psi_i$ is the NP of the $i$th input, and $x_i$ is a value in the domain of $i$th input.

Extraction of the NP in ALM is a two-stage process: *Thickening* and *Thinning*. First, a thickening operator is applied on data samples in order to make a continuous path (fuzzification). The process of thickening is a fuzzification operation on the 2D plane of a S.I.S.O system. While observed data samples on input$_i$–output plane are set of discrete points (Fig. 3a), the fuzzification operator thickens the 2D planes such as to extract continuity (Fig. 3b) as a requirement of the generalization ability of a model. The fuzzification operation in ALM is performed through IDS operator (Bahrpeyma et al. 2013; Saeed Bagheri and Honda 1997a).

The implementation of propagating membership function for IDS operator with a radius $R$ is performed through Eq. 2. The amount of propagated darkness $\mu$ is calculated by (Murakami and Honda 2006):

$$\begin{aligned} &\tau = R - \Delta d + 1, \ -R \leq u, \ v \leq R; \\ &\Delta d(x_s + u, y_s + v) = \sqrt{u^2 + v^2}; \\ &\mu = \begin{cases} \tau, & \text{if } \tau > 0 \\ 0, & \text{otherwise,} \end{cases} \end{aligned} \tag{2}$$

where $(x_s, y_s)$ are the coordinates of the propagator, which is an observed point in the training dataset, $u$ and $v$ are the distances of $x$ and $y$ axes from the propagation point, and $\Delta d$ is the euclidean distance between the membership receiver and the propagator.

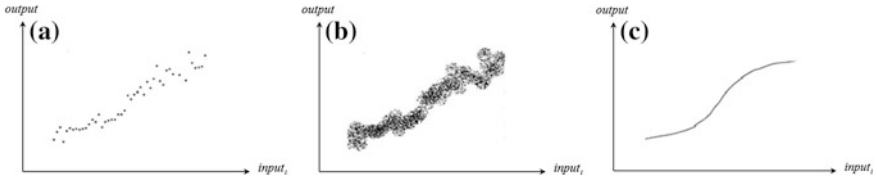


**Fig. 3** Extracting an NP. **a** Discrete observations. **b** Thickening operation. **c** Thinning operation

Second, to extract a function like curve $\psi$, the continuous area obtained by IDS operation on the IDS plane should be thinned. The thinning operation in ALM (Fig. 3c) is performed as a defuzzification operation on the IDS plane by COG operator (Bahrpeyma et al. 2013; Saeed Bagheri and Honda 1997a) to extract the NP i.e. $\psi$ (Eq. 3):

$$\psi(x) = \frac{\sum_{j \in Y(x)} \mu_j Y_j}{\sum_{j \in Y(x)} Y_j} \tag{3}$$

**Step 3: Calculating the spread of each $X_i$–$Y$ plane**

By extracting the NPs from the IDS planes, there are multiple S.I.S.O models for the given M.I.S.O. model which express the output separately. ALM uses a criterion to specify a confidence level for each model and uses the confidence level as a fuzzy weight to calculate the model output. In ALM, the confidence level of input$_i$–output plane is the reciprocal value of the spread of data samples around the extracted NP of that plane. The confidence level indicates how much the NP is proper to show the behavior of the output with respect to the underlying input. As the spread increases, the variation of the data samples around the NP increases, too. The spread of $i$th S.I.S.O. plane is calculated by Eq. (4):

$$v_i = \frac{1}{n} \sum_{k=1}^{n} (\psi_i(x_k) - y_k)^2 \tag{4}$$

where $n$ is the number of data points, and $(x_k, y_k)$ is the coordinate of the $k$th data sample of $i$th S.I.S.O. plane.

Thus, the confidence level is calculated by Eq. (5):

$$\theta_i = \frac{1}{v_i} \tag{5}$$

**Step 4: Evaluation of the model and dividing the data samples into several partitions**

The spread of data calculated by Eq. 3 is a measure to evaluate the accuracy of the model. When the spread is big, it means that the model is not reliable enough. ALM uses division of data samples in order to reduce the spread of data and extract more accurately localized NPs. The evaluation of the models can be performed by the means of measuring the generalization error which can be implemented though a cross-validation operation (Jack 1983; Mori and Stephen 1991). To measure the generalization error, the output is calculated by an interpolation mechanism:

$$\text{output}_k = \frac{\sum_{i=1}^{n} \theta_i \times \psi_i(x_k)}{\sum_{i=1}^{n} \theta_i} \tag{6}$$

The generalization error is calculated by FVU relation (Murakami and Honda 2006):

$$\text{FVU} = \frac{\sum_{k=1}^{K} \left(\hat{y}(x_k) - y_k\right)^2}{\sum_{k=1}^{K} \left(y_k - \bar{y}\right)^2} \tag{7}$$

where $\hat{y}$ is the output of the model, $K$ is the number of test points and $\bar{y}$ is:

$$\bar{y} = \frac{1}{K} \sum_{k=1}^{K} y_k \tag{8}$$

If FVU satisfies the error threshold of ALM $(\text{FVU} \leq \text{Err}_{\text{threshold}})$, the division is not happening because the model is satisfactory; otherwise, ALM divides data samples into partitions in order to gain more accurately localized sub-models.

According to the algorithm of ALM (Fig. 1), dividing the data samples into partitions is performed in this step. An approach is to divide data based on a point in one the projected S.I.S.O planes. A heuristic to choose a proper plane is dividing the plane with the smallest spread, i.e., choosing the plane that has a higher degree of confidence. The main reason to choose this heuristic is that at least it holds the plane with the greatest degree of confidence unchanged while changing the others (Taheri-Shahriyani et al. 2006). Choosing a proper place to divide the chosen IDS plane requires another heuristic. Takagi and Sugeno (1985) used the midpoint to divide a plane. The heuristic search is a guided search, and it does not guarantee an optimal solution. However, it can often find satisfactory solutions (Takagi and Sugeno 1985).

Figure 4 illustrates how ALM divides data into two parts at each stage:

In fact, ALM decides on dividing data samples based on a tree structure which is implied on the recursive nature of ALM. Figure 5 illustrates the decision tree in which its nodes are labeled by their generalization error.

Each node in the decision tree of ALM correspondents to a sub-model constructed by a partition of data samples. Figure 6 is an example decision making about division in ALM for reaching to a state with value of 0.1 for generalization error.

In the example of Fig. 6, the node labeled by “terminating node” corresponds to a state satisfying the threshold of generalization error for stop division. At each state which corresponds to a node in ALM decision tree, the generalization error of the node is compared with the threshold so as to decide on continuing or stop dividing.

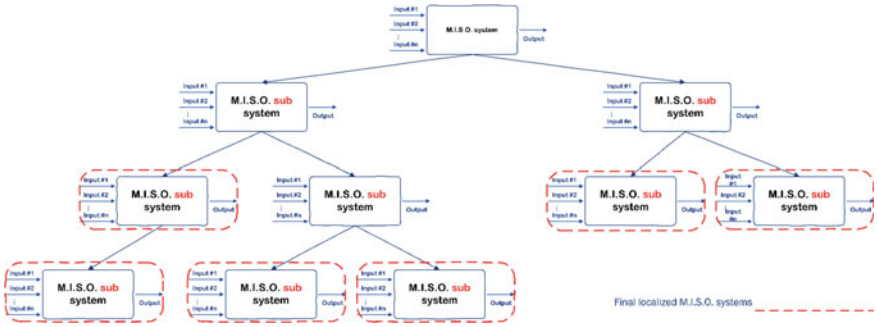


**Fig. 4** Localized M.I.S.O sub-systems resulted by recursive partitioning

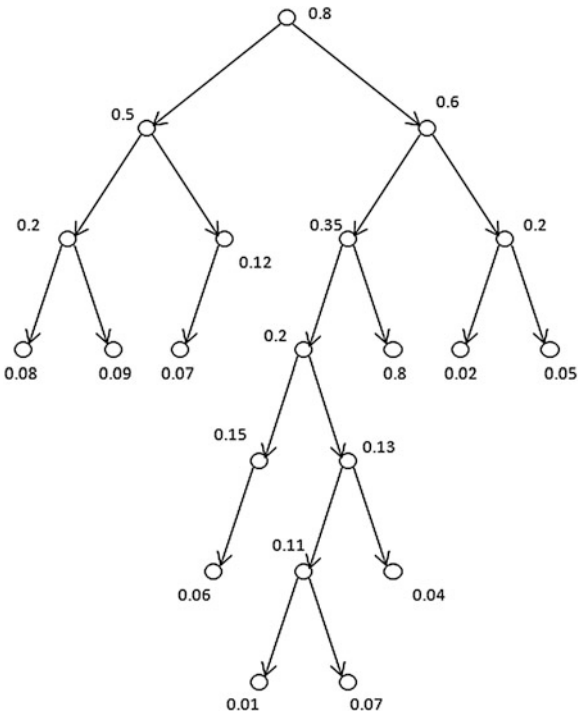


**Fig. 5** An example of ALM decision tree in which the nodes are labeled by generalization error

## 3 Experimental Results

In this paper, the experimental results are provided through modeling the log density based on photoelectric log, neutron log, and sonic log.

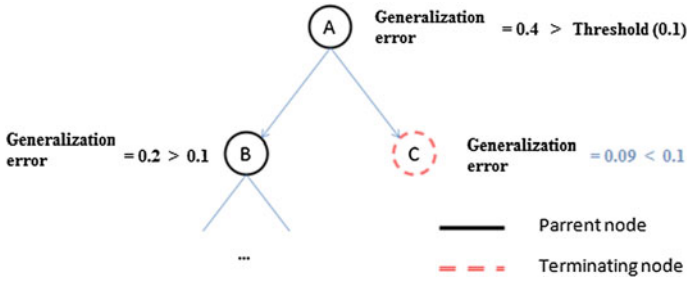


**Fig. 6** An example of decision making about division in ALM at each state

## 3.1 A Brief Overview of Petrophysical Logs

***Density log (RHOB)***: The bulk density of the formation is measured by recording the number of returning gamma rays affected by Compton scattering; this is proportional to the electron density of the formation (Rider 2002). RHOB is used to estimate porosity (so-called *density porosity*), identify mineral lithology (mainly evaporite minerals), detect gas-bearing zones, determine hydrocarbon density, evaluate source rocks, and recognize fractures.
***Photoelectric log (PEF)***: The photoelectric (Litho-Density) log is a continuous record of Pe (the cross-section index of effective photoelectric absorption) related to the formation composition (Rider 2002). Lithology and mineral identification is the major usage of this log.
***Neutron log (NPHI)***: The hydrogen density of the formation is measured via the neutron log. Neutron log is used as a hydrocarbon indicator for estimation of porosity and lithology identification.
***Sonic log (DT)***: Sonic log is the log which is the result of sending acoustic signal through the rocks. DT is mainly used for lithology identification.

The dataset of this study is the collection of logs from a petroleum field in Southern Iran (NW Persian Gulf), also used in (Bahrpeyma et al. 2013). The dataset contains 4500 records which is separated into training and test sets in which 500 records are randomly selected as the test set, and the remaining are used as the training set.

## 3.2 Modeling RHOB

Based on the methodology presented in Sect. 2, the collected data have to be expressed as an M.I.S.O system. ALM projects an M.I.S.O system into some S.I.S.O sub-systems. In this paper, the goal is to estimate RHOB. Therefore, inside each recursion, first the M.I.S.O system {***DT, NPHI, PEF*** → ***RHOB***} is expressed as three S.I.S.O sub-systems: {***DT-RHOB***}, {***NPHI-RHOB***}, and {***PEF-RHOB***}.

The stage of fuzzification and extracting the NPs begins by applying IDS operator on the data for each IDS plane which is shown in Fig. 7.

The process of estimation using ALM needs numbers of iterations which are the results of dividing planes/sub-planes. At each stage, the recursive algorithm of ALM divides data into two parts, and the process is performed on data for each part. Then, the algorithm estimates the output for the test data. The results of matching measured and estimated data for 4 arbitrary exploration levels (3rd, 6th, 9th, and 12th levels) are shown in Fig. 8.

In this paper, the performance of the method is measured via two popular performance measures: normalized mean squared error (nMSE) and regression ($R$). $R$ is the reliability which ranges from −1 to 1 to show how much linear relationship exists between the measured and the simulated values of the output. $R$ is calculated by:

$$R = \frac{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \bar{T})(T_i - \bar{Y})}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \bar{T})^2} \times \sqrt{\frac{1}{N}\sum_{i=1}^{N}(T_i - \bar{Y})^2}} \tag{9}$$

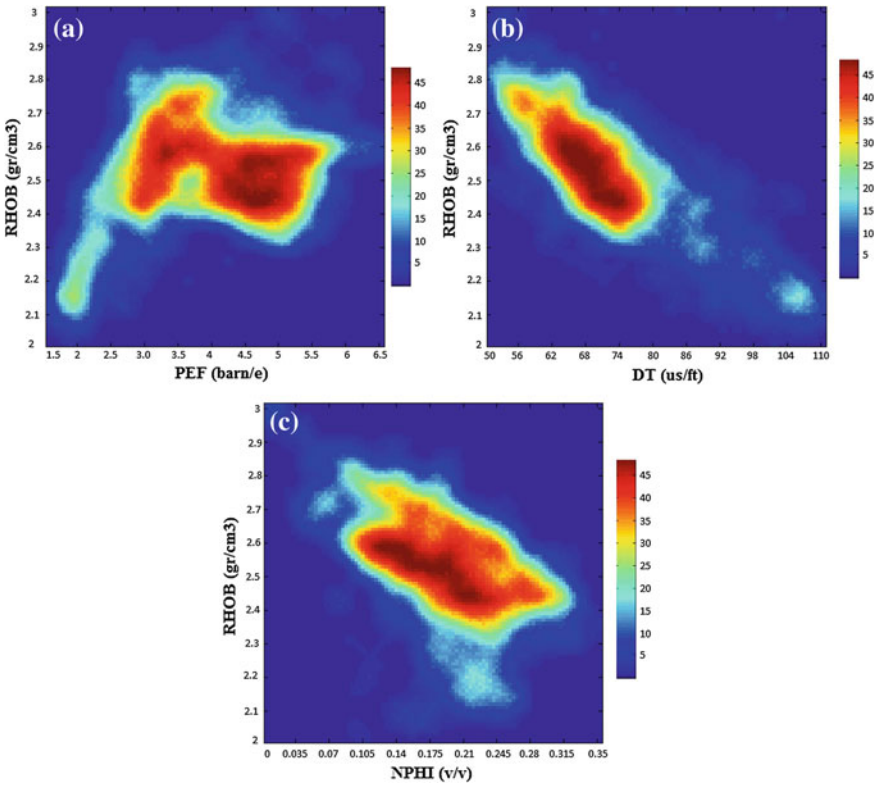


**Fig. 7** Applying IDS operator as the thickening operator on data: **a** IDS plane of PEF-RHOB, **b** IDS plane of DT-RHOB, **c** IDS plane of NPHI-RHOB

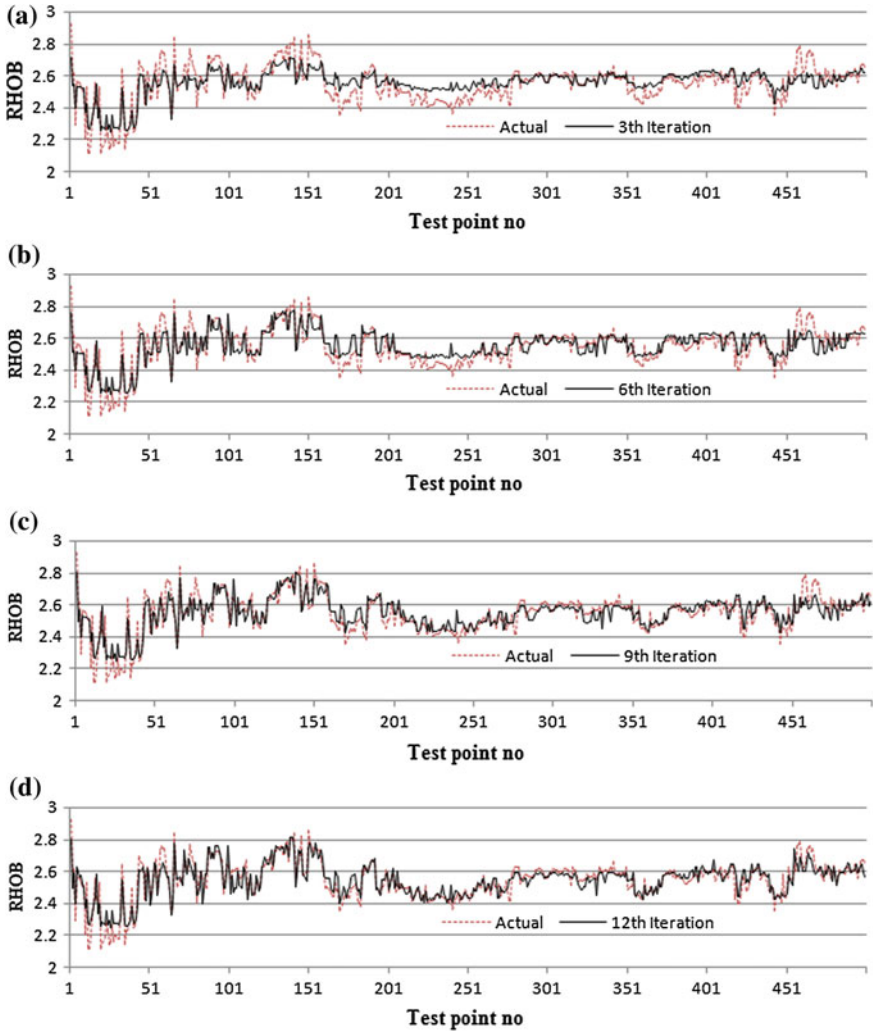


**Fig. 8** Comparison between measured and estimated outputs. **a** 3rd level of exploration; **b** 6th level of exploration; **c** 9th level of exploration; **d** 12th level of exploration

And nMSE is defined as:

$$\text{nMSE} = \frac{\sum_{i=1}^{N} (Y_i - T_i)^2}{\sum_{i=1}^{N} (Y_i - \bar{Y})^2} \tag{10}$$

$R$ for four depth levels (3rd, 6th, 9th, and 12th levels) is shown in Fig. 9.

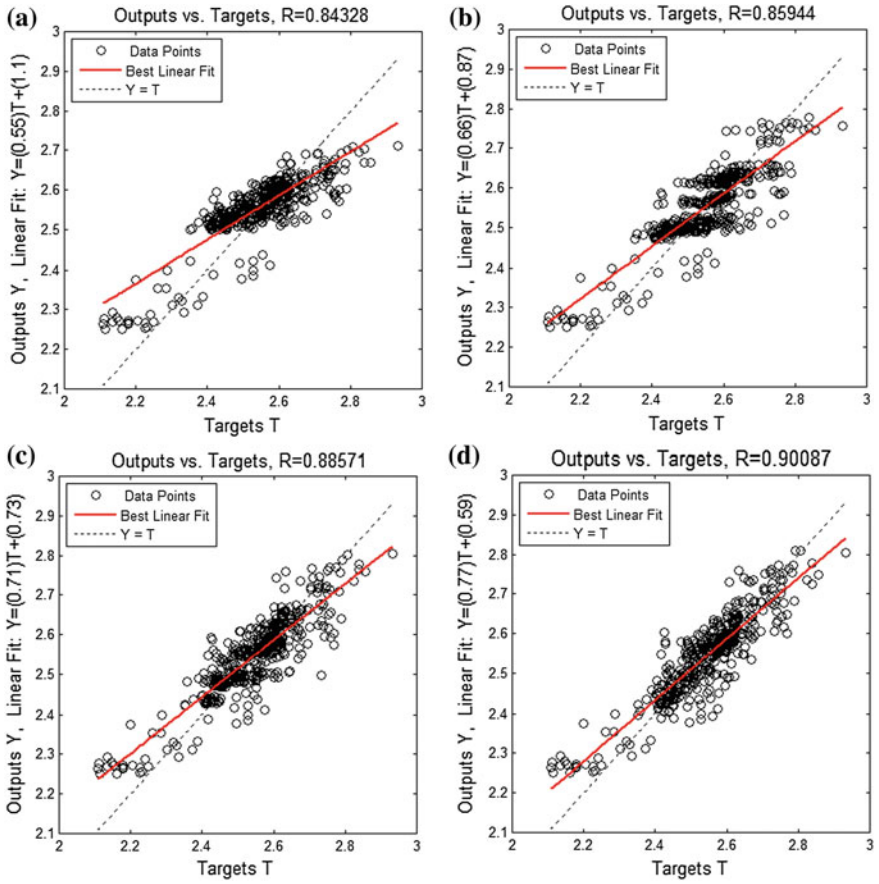


**Fig. 9** *R* for measured and estimated outputs. **a** *R* at 3rd level of exploration; **b** *R* at 6th level of exploration; **c** *R* at 9th level of exploration; **d** *R* at 12th (final) level of exploration

Figure 9 shows that ALM tries to model data more accurately when divides data at each stage and converges better when models data locally on the divided data samples.

The process of modeling depends on the accuracy of the modeling which needs to be illustrated to show how much ALM was successful on estimation. Figure 10a illustrates the sequences of resulted *R* between measured and estimated data. Figure 10b also illustrates ALM tries to converge nMSE to zero which is the most desired output of any types of estimating algorithm.

The most important part of ALM is the process of convergence to a stable state which is required for the validation of the algorithm. This goal is achieved as a result of the value of the given error. Results (Fig. 10) illustrate that ALM is converged to the stable state (no further exploration is observed in the process).

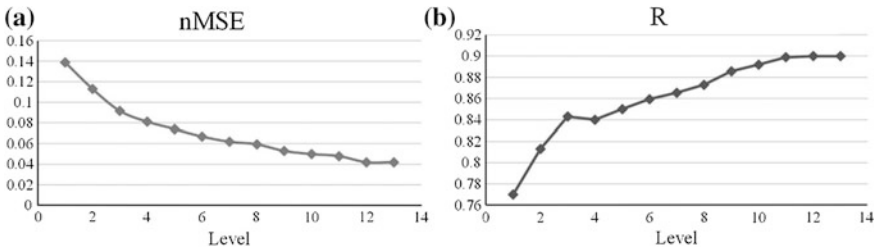


**Fig. 10** The resulted *R* and nMSE gained by ALM at each depth level. **a** nMSE for each exploration level of ALM; **b** *R* of measured and estimated outputs for each exploration level of ALM

The stable state is the final depth level of the ALM tree. The depth level indicates the maximum number of recursions that are allowed to hierarchically localize inside the parameter domain.

## 4 Discussion

Based on the nature of ALM, there are several advantages for ALM when it is compared with conventional estimation methods. In fact, the main characteristic of ALM is the way it deals with the complexities of estimation when it receives its raw data. Unlike other estimation methods, ALM does not tune weights or parameters for foreseeing the behavior it sees from the training data; instead, it extracts the general behavior of data locally and uses an aggregation mechanism to preserve the structure of the effectiveness of each input on the output. Then, speaking about ALM and its algorithm, there are some different points when compared with other methods. One is the overfitting, a major problem of conventional methods. Based on the learning methods and specifically, more related ones, like decision tree characteristics, ALM uses threshold as the early stopping factor to avoid overfitting (Jiang et al. 2009; Liu et al. 2008; Rynkiewicz 2012; Schittenkopf et al. 1997).

Figure 10 illustrates that the depth level of ALM decision tree is equal to 12, and the algorithm has stopped at level 12. It means that the most localized sub-domain required 12 level divisions to gain the required accuracy.

The results of applying ALM and different modeling techniques are illustrated in Table 1 which includes comparison between the performance of artificial neural network ***ANN*** (as a 3)20)1 trained by Levenberg–Marquardt learning algorithm), fuzzy logic ***FL*** (a Takagi-Sugeno-Kang Fuzzy Inference System TSK-FIS), radial basis function network ***RBFN***, Neuro-fuzzy ***NF*** (fuzzy neural network), FFMM (Bahrpeyma et al. 2013), and the ALM; except for FFMM and ALM, the other methods are provided by the MATLAB 2008 toolbox.

Figure 11 illustrates visualized comparison between the performance of popular modeling methods and ALM for modeling RHOB.

**Table 1** The results of applying ALM and popular modeling techniques for estimating RHOB

| | FFMM | ALM | ANN | FL | RBFNN | NF |
|---|---|---|---|---|---|---|
| nMSE | 0.14 | **0.042** | 0.057 | 0.063 | 0.073 | 0.053 |
| *R* | 0.77 | **0.90** | 0.86 | 0.85 | 0.83 | 0.87 |

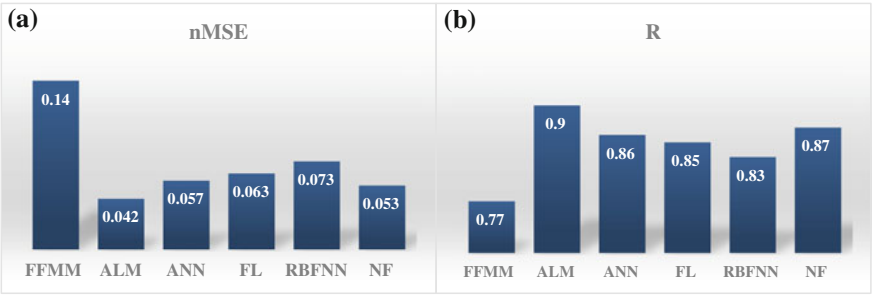


**Fig. 11** Comparison between the performance of popular modeling methods (ANN, FL, RBFNN, and NF) and ALM for modeling RHOB. **a** nMSE, **b** *R*

Comparison between the results gained by modeling RHOB via ALM and other modeling techniques (illustrated in Fig. 11) proves that ALM has acceptable performance and performs well in comparison with popular modeling techniques.

As a result of dealing with computational complexities naturally exist in the popular modeling methods through complex formulas, they generally suffer from lowness of the modeling speed. In this paper, ALM is presented as a modeling method that does not suffer from huge computational complexities. Therefore, ALM is expected to show an acceptable time performance. Figure 12 compares time performance between ALM and some popular modeling methods employed in this paper.

As shown in Fig. 12, ALM was successful to model RHOB without dealing with huge computational complexity which results in reduction in the time required for

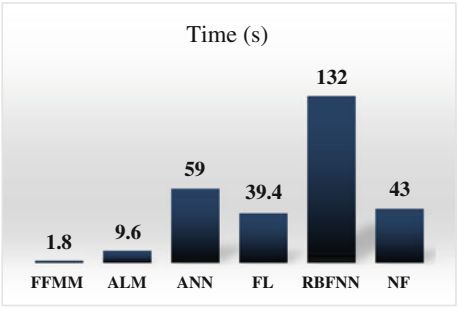


**Fig. 12** Time performance in seconds (s) for modeling RHOB, comparison between ALM and some popular modeling methods addressed in this paper

modeling process. Although FFMM (Bahrpeyma et al. 2013) with the best time performance may be considered as the most successful method among the employed methods, the nMSE and $R$ for FFMM are not so satisfying comparing to other employed methods in this paper.

## 5 Conclusions

In petroleum industry, the problems of incomplete data acquisition and loss of data in the process of measurement require supporting methods which are able to accurately estimate the unmeasured/lost data. This paper introduced a new soft computing tool for the application of estimating missing/unmeasured data called ALM. ALM is inspired from the ability of the human brain to model a multiple-input multiple-output system actively. Results (0.042 and 0.9 for nMSE and $R$, respectively) illustrate that ALM has excellent ability for modeling RHOB and can be used as a reliable approximator in petroleum industry. Comparison between the results obtained by modeling RHOB via ALM and other modeling techniques proves that ALM has acceptable performance and performs well in comparison with other popular modeling techniques.

## References

Afrakoti IEP, Ghaffari A, Shouraki SB (2013) Effective partitioning of input domains for ALM algorithm. In: First Iranian conference on pattern recognition and image analysis (PRIA), 2013. IEEE

Bahrpeyma F, Golchin B, Cranganu C (2013) Fast fuzzy modeling method to estimate missing logs in hydrocarbon reservoirs. J Petrol Sci Eng 112:310–321

Cranganu C, Bautu E (2010) Using gene expression programming to estimate sonic log distributions based on the natural gamma ray and deep resistivity logs: a case study from the Anadarko Basin Oklahoma. J Petrol Sci Eng 70:243–255

Cranganu C, Breaban M (2013) Using support vector regression to estimate sonic log distributions: a case study from the Anadarko Basin, Oklahoma. J Petrol Sci Eng 103:1–13

El-Sebakhy EA, Asparouhov O, Abdulraheem AA, Al-Majed AA, Wu D, Latinski K, Raharja I (2012) Functional networks as a new data mining predictive paradigm to predict permeability in a carbonate reservoir. Expert Syst Appl 39(12):10359–10375

Jack PCK (1983) Cross-validation using the t statistic. Eur J Oper Res 13(2):133–141

Jiang Y, Zur RM, Pesce LL, Drukker K (2009) A study of the effect of noise injection on the training of artificial neural networks. In: International joint conference on neural networks, 2009 (IJCNN 2009). IEEE, pp 1428–1432

Liu Y, Starzyk JA, Zhu Z (2008) Optimized approximation algorithm in neural networks without overfitting. IEEE Trans Neural Networks 19(6):983–995

Mori M, Stephen J (1991) A note on generalized cross-validation with replicates. Stoch Process Appl 38(1):157–165

Murakami M, Honda N (2004) Hardware for a new fuzzy-based modeling system and its redundancy. In: 23rd international conference of the North American fuzzy information processing society (NAFIPS'04), pp 599–604

Murakami M, Honda N (2005) A comparative study of the IDS method and feedforward neural networks. In: International joint conference on neural networks, Montreal, Canada

Murakami M, Honda N (2006) Model accuracy of the IDS method for three-input systems and a basic constructive algorithm for structural optimization. In: International joint conference on neural networks, 2006 (IJCNN'06). IEEE, pp 2900–2906

Rezaee MR, Kadkhodaie-Ilkhchi A, Alizadeh PM (2008) Intelligent approaches for the synthesis of petrophysical logs. J Geophys Eng 5:12–26

Rider M (2002) The geological interpretation of well logs. Whittles Publishing, Malta

Rynkiewicz J (2012) General bound of overfitting for MLP regression models. Neurocomputing 90:106–110

Saeed Bagheri S, Honda N (1997a) A new method for establishment and saving fuzzy membership functions. In: 13th fuzzy symposium, Toyama, Japan, pp 91–94

Saeed Bagheri S, Honda N (1997b) Outlines of a living structure based on biological neurons for simulating the active learning method. In: 7th intelligent systems symposium, pp 183–188

Saeed Bagheri S, Honda N (1999) Recursive fuzzy modeling based on fuzzy interpolation. J Adv Comput Intell 3:114–125

Sagha H, Shouraki SB, Beigy H, Khasteh H, Enayati E (2008) Genetic ink drop spread. In: Second international symposium on intelligent information technology application, 2008 (IITA'08), vol 2, pp 603–607

Saramet M, Gavrilescu G, Cranganu C (2008) Quantitative estimation of expelled fluids from Oligocene rocks, Histria Basin, Western Black Sea. Mar Pet Geol 25(6):544–552

Schittenkopf C, Deco G, Brauer W (1997) Two strategies to avoid overfitting in feedforward networks. Neural Networks 10:505–516

Taheri-Shahriyani H et al (2006) Investigating the ability of active learning method for chlorophyll and pigment retrieval in case-I waters using seawifs wavelengths. Int J Remote Sens 28(20):4677–4683

Taheri-Shahraiyni H et al (2007) Application of the active learning method for the estimation of geophysical variables in the Caspian Sea from satellite ocean colour observations. Int J Remote Sens 28(20):4677–4683

Taheri-Shahraiyni H, Mohammad RG, Bagheri-Shouraki S, Saghafian B (2011) A new fuzzy modeling method for the runoff simulation in the Karoon Basin. Int J Water Resour Arid Environ 1(6):440–449

Takagi T, Sugeno M (1985) Identification of systems and its application to modeling and control. IEEE Trans Syst Man Cybern 15(1):116–132

# Improving the Accuracy of Active Learning Method via Noise Injection for Estimating Hydraulic Flow Units: An Example from a Heterogeneous Carbonate Reservoir

**Fouad Bahrpeyma, Constantin Cranganu and Bahman Golchin**

**Abstract** Due to many reasons, in many occasions, reservoir engineers should analyze the reservoirs with small sets of measurements; this problem is known as the small sample size problem. Because of small sample size problem, modeling techniques commonly fail to accurately extract the true relationships between the inputs and the outputs used for reservoir properties prediction or modeling. In this paper, small sample size problem is addressed for modeling carbonate reservoirs by the active learning method (ALM). In this paper, noise injection technique, which is a popular solution to small sample size problem, is employed to recover the impact of separating the validation and test sets from the entire sample set in the process of ALM. The proposed method is used to model hydraulic flow units (HFUs). HFUs are defined as correlatable and mappable zones within a reservoir which control fluid flow. This study presents quantitative formulation between flow units and well logs data in one of the heterogeneous carbonate reservoir in Persian Gulf. The results for R and nMSE are equal to 85 % and 0.0042 which reflect the ability of the proposed method when facing with sample size problem.

**Keywords** Active learning method · Noise injection · Overfitting · Hydraulic flow unit · Ink drop spread · Carbonate reservoir

F. Bahrpeyma
Faculty of Electrical and Computer Engineering, Shahid Beheshti University G.C., Tehran, Iran

C. Cranganu (✉)
Department of Earth and Environmental Sciences, Brooklyn College of the City University of New York, Brooklyn, NY, USA
e-mail: cranganu@brooklyn.cuny.edu

B. Golchin
Department of Geology, Shahid Beheshti University, Tehran, Iran



C. Cranganu et al. (eds.), *Artificial Intelligent Approaches in Petroleum Geosciences*,
DOI 10.1007/978-3-319-16531-8_8

## 1 Introduction

A fundamental approach to decrease uncertainty in estimation of reservoir properties (which is due to heterogeneity in carbonates) is rock typing based on petrophysical characteristics. One of the popular methods to rock classification from geology framework and physics of flow is through pore network scale (pore throat geometric properties). Generally, the pore geometry is controlled by mineralogy (type, abundance, location) and texture (grain size, grain shape, sorting, packing). Different combinations of these properties can lead to identify distinct flow groups which have similar fluid transport properties. Each group is referred to as a hydraulic flow unit (HFU).

HFU is defined classically as the representative volume of total reservoir rock within which geological and petrophysical properties that control the fluid flow. They are internally consistent and predictably different from properties of other rocks (Ebanks 1987; Jude et al. 1993). Briefly, it can be identified as a zone in a reservoir where the similar flow of the oil or gas is continuous laterally and vertically. It relates to geological facies distributions, however, do not necessarily coincide with facies boundaries.

To identify HFU for formation evaluation, reservoir description, and characterization using flow zone indicator (FZI), permeability and porosity of the reservoir rock have always been considered as two of the most important parameters. FZI, which is measured based on core porosity and permeability, characterizes each flow unit, and it is popular to derive the FZI value and its corresponding flow unit for each sample. Conventional studies are executed on the core data to determine the FZI values, however, such works are expensive, time-consuming, and limited. To solve this problem, researchers use soft computing's modeling techniques such as artificial neural network (ANN), fuzzy logic (FL), support vector machine (SVM), and committee machines (CM) to estimate FZI and other reservoir parameters by well logs data. Kadkhodaei-Ilkhchi and Amini (2009) developed a FL model to determine FZI from well logs at un-core wells. Ghiasi-Freez et al. (2012) compared two types of CMs to estimate FZI. In this study, we developed new soft computing tool called the active learning method (ALM), which is as fast and accurate modeling technique, for determining HFUs in heterogeneous carbonate reservoir in Persian Gulf as one of the world's largest non-associated gas accumulation. ALM (Saeed Bagheri and Honda 1999) is a solution to model multi-dimensional systems without dealing with huge computational complexities exist in traditional modeling techniques.

The data set in this paper contains small set of samples that charges ALM with small sample size problem. Due to many reasons, small sample size problem is one of the common concerns in petroleum industry when trying to estimate the properties in the carbonate reservoirs. In many occasions, this is not possible to provide adequate sample size for the modeling techniques; it may degrade the performance of the estimation process. The problem is more crucial when, to provide valid modeling procedure, the modeling techniques split the entire sample set into three subsets: 1-the training set, 2-the validation set, and 3-the test set. The training set is used for

constructing the model. The validation set is used for modeling inter-level validation. And the test set is used to evaluate the performance of the constructed model. More importantly, for small data sets, splitting the data samples into training, validation, and test sets separates some of the useful information from the training set which meaningfully degrades the generalization ability of the modeling procedure.

One of the popular solutions to small sample size problem is to improve the generalization ability of the model by reviving the separated information through noise injection (Rifai et al. 2011). In fact, noise injection can be regarded as a regularization method. The problem of regularization has been broadly studied in the context of learning methods specifically in the field of ANNs. Regularization is commonly used by learning methods to improve the generalization ability for small sample sets or when data samples are contaminated with noise, meaningfully. In the context of ANNs, weight decay and output smoothing that are used to avoid overfitting during the training of the considered model are the other popular methods (Yulei Jiang et al. 2009).

Noise injection has been broadly studied in literature. In Raudys (2003, 2006), the impact of using noise injection technique to fight with sample size problem is studied. Rifai et al. (2011) used noise injection to the input with a regularized objective to improve generalization performance in the ANNs. Yulei Jiang et al. (2009) studied the effect of noise injection on the training of ANNs. Liu and Janusz (2008) proposed an optimized approximation algorithm which employed noise injection in ANNs to overcome overfitting. Skurichina et al. (2000) used K-nearest neighbors algorithm to direct noise injection in multilayer perceptron training. Piotrowski and Napiorkowski (2013) compared popular methods, such as noise injection, for avoiding overfitting in ANNs training in the case of catchment runoff modeling.

In addition to the mentioned ability which noise injection provides, the smoothness of the curve which provides more generalization ability is another benefit of noise injection technique; an example of smoothness provided by additional noise is illustrated in Fig. 1.

The rest of the paper is organized as follows: the next section is on overview on ALM. In Sect. 3, the proposed methodology for employing noise injection in ALM

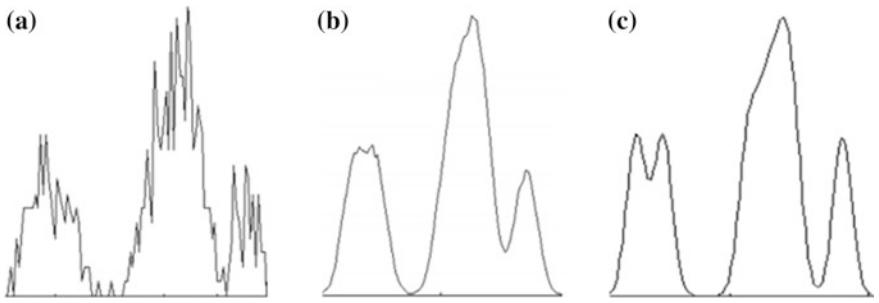


**Fig. 1** Additive noise for improving the smoothness and penalization ability of a curve gained by training samples. **a** Actual arbitrary data curve, **b** Adding 5 % noisy samples to the data samples, **c** Adding 10 % noisy samples to the data samples

is described. The experimental results are illustrated and discussed in Sect. 4. Finally, Sect. 5 states conclusions derived from the work.

## 2 An Overview on Active Learning Method

ALM algorithmically models a system similar to intelligent information management procedure of the human brain. ALM works fundamentally based on the hypothesis that the human brain observes an information system as an image in which the information is stored as patterns (Saeed Bagheri and Honda 1999). The main advantage of ALM is its ability to model multi-dimensional systems without dealing with computational complexities.

ALM uses a recursive approach in order to model multi-dimensional systems without dealing with computational complexities. This special approach is characterized by the usage of a fuzzification engine called ink drop spread (IDS) which is inspired by the way the drops of ink can create a continuous path that expresses the overall continuity instead of dealing with a discrete appearance (Saeed Bagheri and Honda 1997; Shouraki 2000). The main idea behind ALM is the projection of a multiple-input single-output (M.I.S.O.) system into some single-input single-output (S.I.S.O.) sub-systems and unifying the outcomes by an interpolation mechanism to generate the final output of the modeling system in a recursive manner.

### *2.1 The ALM Algorithm*

A practical implementation of ALM can be performed through a recursive process which divides data hierarchically into small and smaller partitions, while proper partitioning is the goal of the entire process. Recursive partitioning enables to intuitively model a system based on the locality of the data distribution within the data domains. Hierarchical partitioning helps figuring out the locality imagination with the stopping parameter. The flowchart of ALM is illustrated in Fig. 2.

In the flowchart of ALM illustrated in Fig. 2, at each recursion, the IDS modeling method is responsible to model the corresponding data domain. Then, if the normalized mean-squared error (nMSE) of the model is greater than the threshold, the recursive process of ALM divides the corresponding data and partitioning does not continue for the corresponding data domain.

In Fig. 3, an example of applying ALM on a 2-input system is illustrated showing how the hierarchical division mechanism divides the domain of inputs in order to satisfy the predefined generalization threshold in a sub-domain.

In this example, starting from the entire domain, ALM divides training data into sub-domains and for each sub-domain, dividing is stopped when the terminating condition (nMSE's threshold) is satisfied. A particular tail of the hierarchy in Fig. 3 starts from the entire 2-D domain (A0) to reach A6 area, which satisfied the

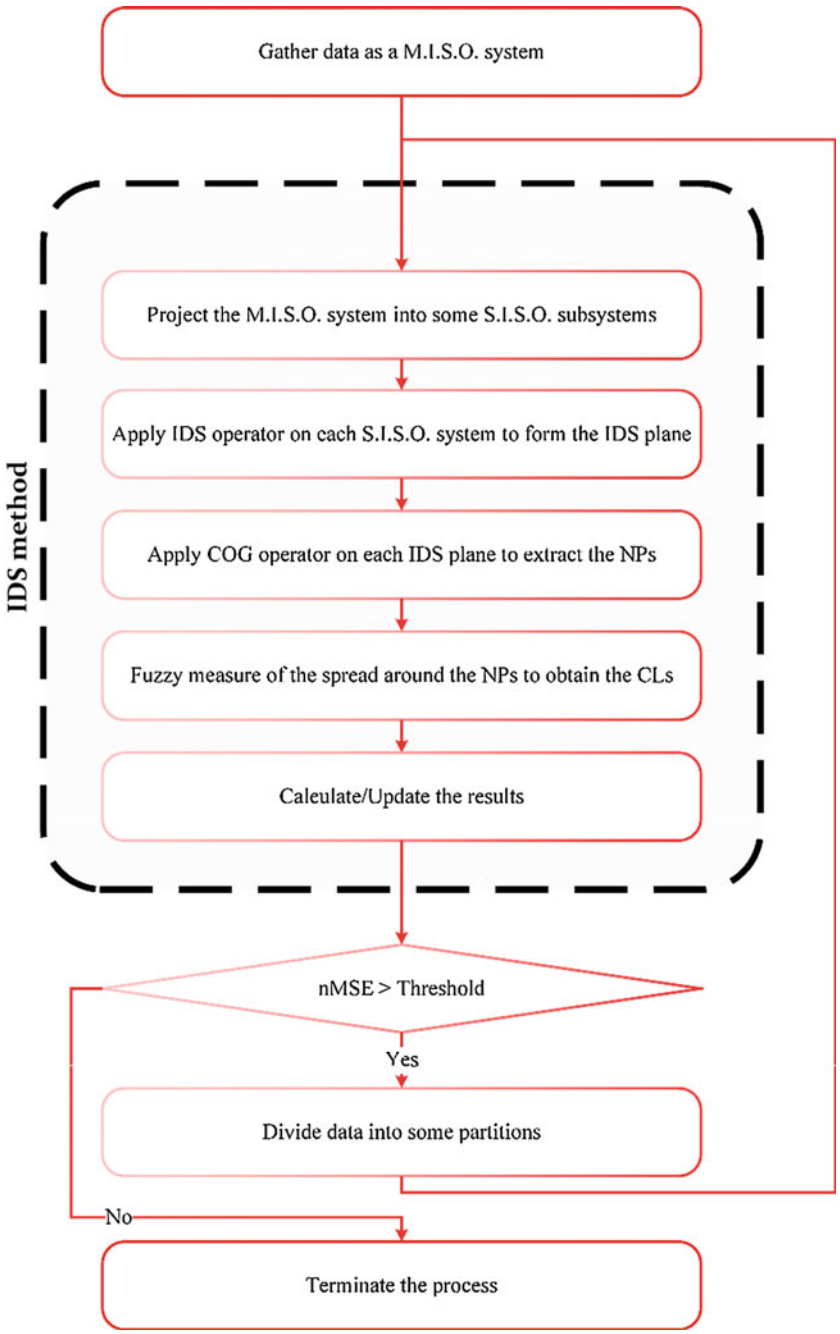


**Fig. 2** The flowchart of ALM (Shouraki 2000)

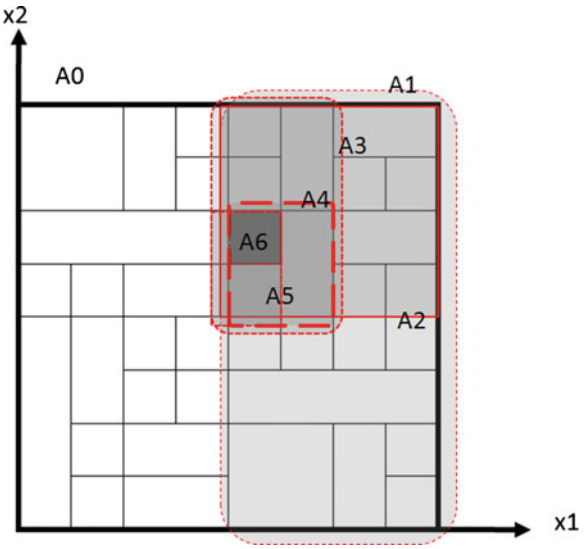


**Fig. 3** 2-Dimensional space of inputs in a 2-input single-output system

terminating condition. Figure 4 illustrates that how modeling procedure (performed by IDS) is localized through the recursive process of ALM. In this paper, at each recursion, ALM divides data sub-domain into two partitions, i.e., the branching factor is 2.

As illustrated in Fig. 4, IDS separately models each localized sub-domain which is introduced by ALM.

## *2.2 IDS Modeling Method*

As the engine of ALM, IDS modeling method models a multiple-input single-output (M.I.S.O.) system (or sub-system when the system, as a domain, is divided

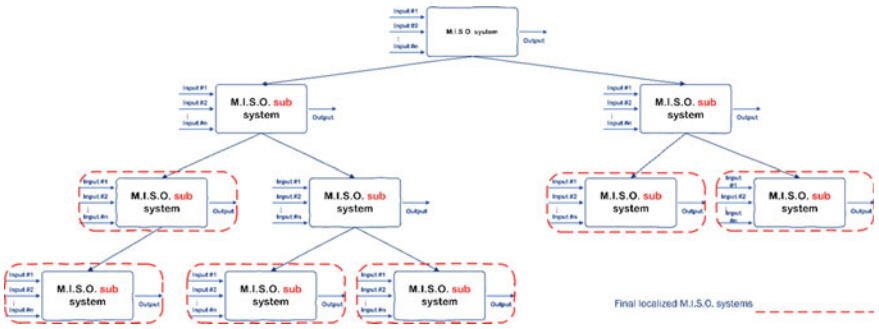


**Fig. 4** Stages of recursively dividing data planes

into some partitions) based on projection of the system into single-input single-output (S.I.S.O.) sub-systems and then unifies the S.I.S.O. models into a single model through an aggregation mechanism. IDS models each S.I.S.O. sub-system separately through IDS and COG operators. Therefore, after projection, the process of IDS method includes application of IDS and COG operators, (consequently) on each S.I.S.O. sub-system and then unifying them into a single model.

- Applying IDS operator on the projected data planes (fuzzification)
  IDS operator roles as a fuzzification operator. The goal of IDS operator is to provide continuous paths between data samples (in 2D spaces of the S.I.S.O. systems). IDS uses a different approach to devote membership values to the elements in a data set. IDS operator extracts the relationships between data points intuitively which are discrete in details, but the entire set is treated as a continuous pattern. Figure 5 illustrates the application of IDS operator on a 2D data plane.
  As shown in Fig. 5, while the midpoints between A, B, and C, which are the actual data samples, are not observed as the data samples, application of IDS operator on the 2D domain of the S.I.S.O. system creates the sense of continuity in an arbitrary unity. Therefore, propagation of membership values to the neighbors from each actual data point which is attenuated by distance and reinforced by other actual neighbors creates the sense of continuity in the 2D input–output (I-O) domain. As Fig. 5 shows that, while A, B, and C are the actual data samples and the domain formed by discrete units, quantization of the domain by the means of a board and propagation of the membership values by IDS extracts the continuity between actual data samples in each 2D I-O system.
- Applying center of gravity operator on the IDS planes (defuzzification)
  Defuzzification in ALM is performed by COG operator (Saeed Bagheri and Honda 1997). COG defuzzifies the results gained by IDS operator so as to make a crisp world induction. The outcomes of COG operator on the IDS planes are some curves called narrow paths (NPs) which express the overall behavior of the output (parameter) with respect to the input (parameter) in each S.I.S.O. system. This behavior ought to be unique for the input domain (acting like a function of input domain).

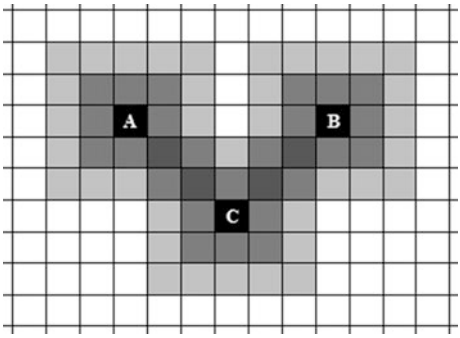


**Fig. 5** Extracting the midpoints relationships between observed data samples in a S.I.S.O. system (Bahrpeyma et al. 2013)

- Aggregating the S.I.S.O. candidate models
  The NPs are the models which are extracted by the IDS method for each S.I.S.O. sub-system in the current M.I.S.O. sub-domain, separately. Each NP introduces a particular candidate S.I.S.O. model which has to be unified to a unique M.I.S.O. model. Therefore, at the final step of the modeling process in the IDS method, the NPs are aggregated through an interpolation mechanism to form a single M.I.S.O. model.

## 3 The Proposed Methodology

A very popular approach to deal with small sample size problem is the noise injection technique. Noise injection is known as a practical technique to improve generalization ability of many different regions such as classifiers, approximators, data-dependent analysis, and especially in NNs.

In this section, since the literature has not studied on the generalization ability and overfitting in ALM, we briefly explore the topics and then the proposed methodology is described.

### *3.1 Generalization in ALM*

Inside ALM, generalization, which is the main goal of the entire modeling process, is evaluated after modeling each data sub-domain in order to recognize whether overfitting is occurred or the desired generalization ability (measured by nMSE) is provided for the sub-domain. The goal of generalization is to provide a model so as to be used to approximate unseen data samples. Generalization in a modeling method is measured by the generalization error. The generalization error is a criterion that measures how good a modeling method generalizes to unseen data. Generally, the generalization error is measured as the distance between the error on the training set and the test set in the process of learning which can be regarded locally (for ALM) or in the entire training set.

In ALM, the recursive localization continues only if the generalization error (which is measured by nMSE) is greater than a predetermined threshold. If the generalization error is less than the threshold, ALM stops localization. In other words, ALM tries to reach a certain value of generalization performance to model the system and to avoid overfitting stops partitioning the data in the current level.

One of the most important methods for measuring the generalization error is the cross-validation technique. Using cross-validation for measuring the generalization error requires separating the entire training data set into two different sub-sets of data: *Training set* and *Validation set*. This separation reduces the number of actual training data which regarding that the entire data set is formerly separated into the

training and test data obviously reduce the generalization ability of the trained model. This issue becomes very more important when the size of data samples is insufficient of the generalization faces with lack of enough data.

## 3.2 Overfitting in ALM

One of the most challenging problems with modeling techniques, which has an important impact on the performance of the modeling process, is the problem of overfitting and underfitting. While modeling techniques construct models from a set of training samples, identifying the structure and the effective characteristics of data used for constructing the models are the matter of concern. Overfitting is one of the problems which is faced by the modeling techniques and is due to the data structure and characteristics. Overfitting happens when the learning phase of the modeling process converges to influence from a very specific part or characteristic of the samples belonged to the training data which meaningfully degrades the generalization ability. As a result, when overfitting happens, the output cannot be guaranteed to exhibit the true formulation of what is actually expected from the learning process through training samples and the output is unreliable.

One of the reasons for overfitting is due overtraining the model from the training samples called overlearning. At the occurrence of overlearning, model trains until almost all the model parameters converge to express characteristics of the training samples (or specific parts).

Based on the modeling techniques, many different methods have been developed for preventing the modeling process from overfitting to the training samples such as early stopping, weight decay (in ANNs), and so on.

ALM uses the method called early stopping so as to prevent from overfitting. As the flowchart of Fig. 2 exhibits, there is a level in the process of ALM for stopping the localization (more localized learning) process. This level includes evaluating the performance of the model and comparing the accuracy (nMSE) against a predefined threshold of error, and stopping the learning/localization process for the corresponding data sub-domain if a predetermined accuracy is reached.

One of the most popular approaches for avoiding overfitting is to divide the entire data set into two separated sets: a set for training and another set for validation. This approach is called *early stopping*. Then, modeling and training are performed just by the training set and evaluation of model or the performance of the system is performed by the validation set. The validation data are independent of the training data. This is due to have a proper measure for generalization which is the goal of training. So, until the learning process continues, performance on the validation set improves with training.

Schematic view of the learning process illustrating error through the training and validation sets is shown in Fig. 6. To avoid overfitting, training stops at time $t$ where performance on the validation set is optimal.

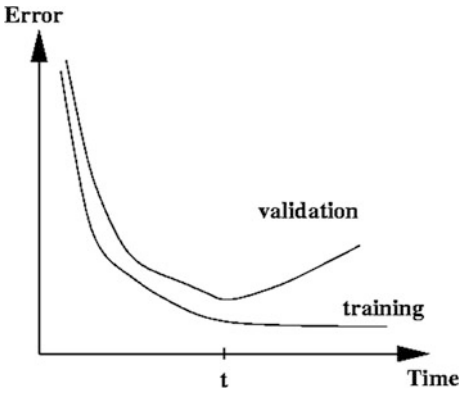


**Fig. 6** Overfitting avoidance by stopping learning process at time *t*

Choosing a threshold of error origins from the generalization disability of learning method to identify well-sampled data and noise locations in overall view of data and to prevent the learning process from minimization of overtraining from a particular part of data samples. This leads to choose from several methods so as not to lower the performance of the modeling process due to overlearning or overfitting.

One of the disadvantages of the early stopping approach is that commonly, most part of data are not used for training which is broadly studied in the field of ANNs (Rynkiewicz 2012; Schittenkopf et al. 1997; Liu and Janusz 2008; Yulei Jiang et al. 2009). Another possibility for early stopping detection is to use the whole part of the data for training and perform validation on some other data samples which is not always accessible.

## *3.3 Noise Injection in ALM*

This section describes where and how to use noise inside the ALM process to improve the generalization ability. Due to the need for independent training, validation, and test data sets, the early stopping used in ALM is only useful in a data-rich situation. The main idea behind using noise injection in this paper is to provide pseudo-training set for ALM in order to properly regularize the model when facing with small sample size problem.

In the process of ALM, noise injection should be addressed inside the IDS method which is responsible to construct the model inside the ALM process. The modified flowchart of the IDS process with the aid of noise injection technique is illustrated in Fig. 7.

According to the flowchart of Fig. 7, first, the data of the current localized M.I.S.O. system (stated in the current recursion) divided into two subsets: the training set and the validation set.

Based on the algorithm of the IDS method, the M.I.S.O. system is then projected into some S.I.S.O. systems with respect to the number of the input parameters.

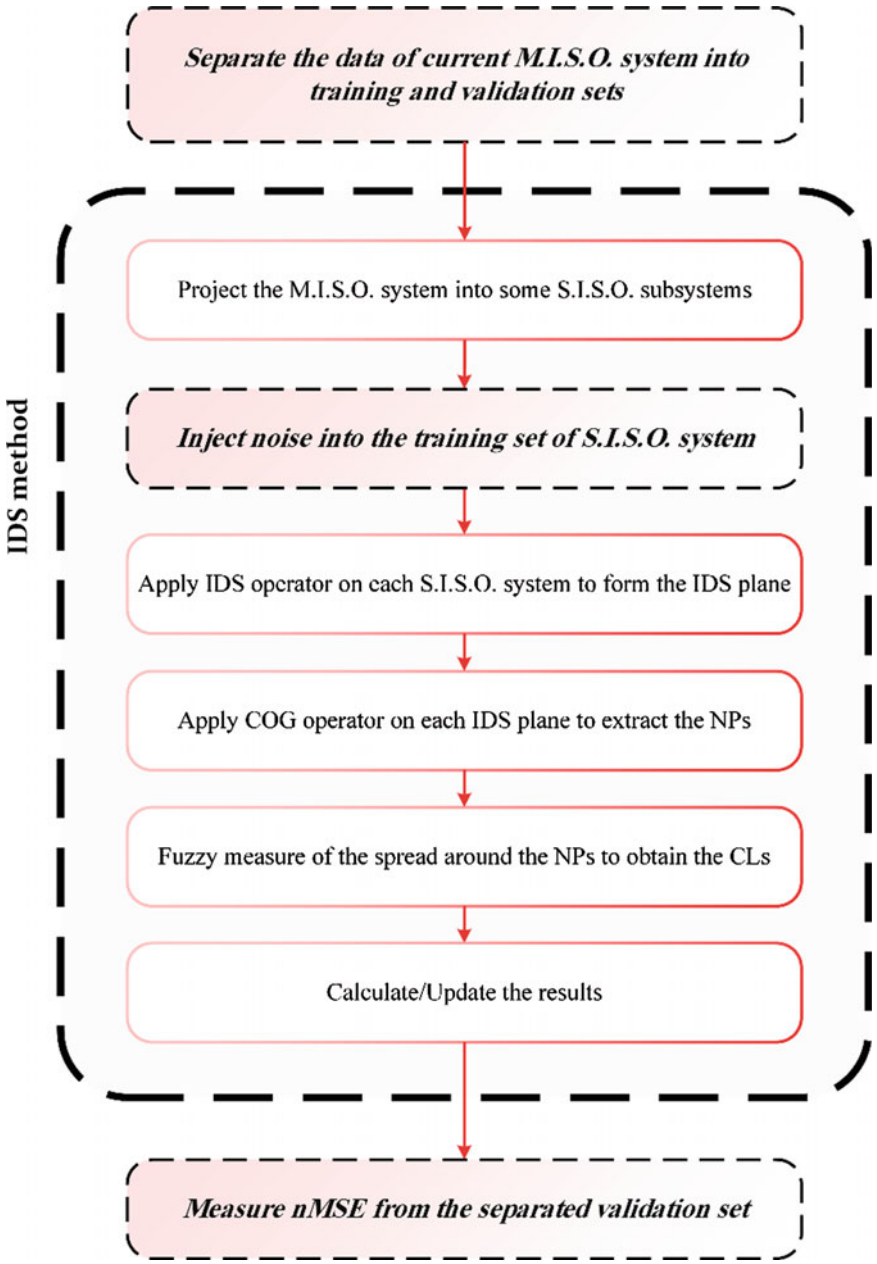


**Fig. 7** Noise injection phase added to the algorithm

For each S.I.S.O. system, noise is added to the actual data m times independently to increase the number of training samples and to smooth the 2D plane of the S.I.S.O. system through scattered noise around the actual samples. At each time the IDS

method is addressed, noise is added separately in random fashion to eliminate possible memory of noise so as not to degrade the training performance acquired by the pure behavior of the actual samples.

One of the most important characteristics which is required for preserving the pure behavior of the actual samples inside the training phase is the zero mean of the noise (Fig. 1).

This is important to note that the noisy data will not be participated to measure generalization ability (nMSE). The noisy data are just involved in the process of training, and the validation set is selected from the actual data in order to guarantee true measurement.

Noise is mainly characterized by three characteristics: 1-mean, 2-variance, 3-density, and 4-fashion. In this paper, the noise is characterized by zero mean value and scattered through normal distribution around each actual sample, independent from the other actual samples.

Noise injection is employed to produce pseudo-training or pseudo-validation data sets for enriching the sample sets. It can be done by adding random zero mean vectors to training data set (Yulei Jiang et al. 2009). According to (Yulei Jiang et al. 2009), to make pseudo sets, a zero mean and small covariance noise vector can be added to the training data set:

$$S_{NI} = S_P + N \tag{1}$$

where $S_P$ is the actual training samples, $N$ is the noisy data added to the actual training samples, and $S_{NI}$ the increased training sample set polluted by noise.

From now on, the entire process before measuring nMSE is performed on the noisy data as the pseudo-training set. Implementation of the IDS operator requires regarding a $d$ % margined operation board to provide continuity through fuzzy membership propagation. The operation board is a 2D board which enables the IDS operator to propagate fuzzy membership from actual data sample around to make a continuous fuzzy area.

The domain of $i$th S.I.S.O. sub-system for IDS operation board is defined as:

$$\begin{cases} D_{X_i} : x | \min(X_i) < x < \max(X_i) \\ D_Y \ : y | \min(Y) < y < \max(Y) \end{cases}, \tag{2}$$

where $D_{X_i}$ and $D_Y$ are the domains of the input and the output of $i$th the S.I.S.O. system ($X_i$-$Y$).

Therefore, the domain of $d$% margined domain is regarded through:

$$\begin{cases} D_{X_i}^{\frown} : x | \min(X_i) - \text{margin}_{X_i}^{d\%} < x < \max(X_i) + \text{margin}_{X_i}^{d\%} \\ D_{Y}^{\frown} : y | \min(Y) - \text{margin}_{Y}^{d\%} < y < \max(Y) + \text{margin}_{Y}^{d\%} \end{cases}, \tag{3}$$

where $\text{margin}_X^{d\%}$ is calculated by:

$$\text{margin}_X^{d\%} = [\max(X_i) - \min(X_i)] \times \frac{d}{100}. \tag{4}$$

Finally, for an $M \times M$ board, each unit $\left[U_{X_i}^{M}, U_{Y}^{M}\right]$ in $i$th S.I.S.O. sub-system is calculated by:

$$\begin{cases} U_{X_i}^{M} = \dfrac{\max(\widehat{X}_i) - \min(\widehat{X}_i)}{M} \\ U_{Y}^{M} = \dfrac{\max(\widehat{Y}) - \min(\widehat{Y})}{M} \end{cases}. \tag{5}$$

For applying the IDS operator on the 2D planes of each S.I.S.O. system (which is polluted by noise), the fuzzy membership $\mu$ which is propagated by each sample to the neighborhood is attenuated by distance. In this paper, attenuation is implemented through a linear function:

$$\begin{aligned} \mu &= R - \sqrt{u^2 + v^2} + 1; -R \le u, v \le R, \\ \Delta d(x_s + u, y_s + v) &\Rightarrow \begin{cases} \mu; & \text{if } \mu > 0 \\ 0; & \text{otherwise} \end{cases}, \end{aligned} \tag{6}$$

where $R$ is the radius of the IDS operator, $(x_s, y_s)$ is the coordinates of the propagator, $(u, v)$ is the distance between the receiver and:

$$\Delta d(x_s + u, y_s + v) = \sqrt{u^2 + v^2}. \tag{7}$$

After forming the IDS planes, to extract the NP $\psi(x)$, the COG operator is applied on the IDS plane:

$$\psi(x) = \frac{\sum_{j \in Y(x)} \mu_j Y_j}{\sum_{j \in Y(x)} Y_j}, \tag{8}$$

where $Y$ is the output axis of the operation board.

The NPs are in fact the candidate S.I.S.O. models for the current (localized) M.I.S.O. system which should be unified/aggregated to provide a true M.I.S.O. model. Therefore, a weighted averaging is employed as an interpolation mechanism to aggregate the candidate S.I.S.O. models:

$$y_{\text{final}} = \frac{\sum_{k=1}^{m} w_k y_k}{\sum_{k=1}^{m} w_k}, \tag{9}$$

In the weighted averaging of Eq. 9, the weight, as confidence level (CL), devoted to each S.I.S.O. candidate model corresponds to the reciprocal value of the spread:

$$w_k = \frac{1}{\text{Spread}_k}. \tag{10}$$

The Spread is calculated by

$$\text{Spread}_k = \frac{1}{n}\sum_{i=1}^{n}(\psi_k(x_i^k) - y_i)^2, \tag{11}$$

where $x_i^k$ and $y_i$ are the coordinates of $i$th data point and $\psi_k$ is the projection of the $i$th point on $k$th NP.

At the end of IDS modeling process, if nMSE is greater than the predetermined threshold, the recursive algorithm of ALM divides data into two partitions from the midpoint of the input sub-domain in which its CL has the greatest value. This heuristic helps at least preserving the most reliable S.I.S.O. model for the two resulting subsets.

## 4 Experimental Results

This section describes and discusses the experimental results of employing noise injection for improving the generalization ability of ALM when is used for estimating FZI with the small sample size problem.

### *4.1 Data Preparation*

Jude et al. (1993) presented a theoretical methodology to identify the flow units. He defined a concept called FZI, which is a unique and useful value to quantify the flow character of a reservoir and one that offers a relationship between petrophysical properties at small scale, such as core plugs, and large scale, such as well bore level. FZI is defined by the following equation:

$$\text{FZI} = \frac{1}{\left(\sqrt{F_\text{S}}\right)\tau S_\text{gv}} = \frac{\text{RQI}}{\phi_Z}, \tag{12}$$

where $F_\text{S}$ is the pore throat shape factor, $\tau$ is the tortuosity, and $S_\text{gv}$ is the effective surface area per unit grain volume. RQI is the reservoir quality index defined below.

$$\mathrm{RQI} = 0.0314\sqrt{\frac{K}{\phi_Z}}, \tag{13}$$

where $K$ is permeability and $\phi_Z$ is the pore volume ($\phi$) to grain volume ratio defined as

$$\phi_Z = \frac{\phi}{1-\phi}, \tag{14}$$

The above parameters were derived from a modified form of the Kozeny–Carmen relation.

The present study used available data sets of two wells with name of well A and well B from a carbonate reservoir in Persian Gulf. These wells contain both log and core data. Core and log porosity values were plotted with respect to depth in order to check whether the data need any depth shifts. Three most commonly logs were selected as input to estimate FZI. These logs are neutron porosity (NPHI), sonic (DT), and density (RHOB). Data from well A were used for the training the model (with 720 data samples) while data from well B were used for testing the reliability of the model (with 85 data points).

## 4.2 Numerical Results

In this paper, two statistical evaluation criteria were used to assess the model performance including correlation coefficients ($R$) and mean square error (MSE):

1. The correlation coefficient ranges between 0 and 1 which defined as:

$$R = \frac{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \bar{T})(T_i - \bar{Y})}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \bar{T})^2} \times \sqrt{\frac{1}{N}\sum_{i=1}^{N}(T_i - \bar{Y})^2}} \tag{15}$$

where $T_i$ and $u_i$ represents the measured and estimated FZI for training data or testing data $i$, respectively, while $T$ and $u$ are the mean value of measured and estimated FZI, respectively, and $n$ is the number of data in the training or testing data set.

Higher values of R indicate the better performance of the model. Legates and McCabe (David and Gregory 1999) argued that this indicator should not be applied as fitness measure alone, and it is appropriate to quantify the error in the same unit as for the variables.

(2) The mean square error (MSE) is one of the most commonly used measures of success for numeric estimation, computed by taking the average of the squared differences between each estimated FZI and its corresponding measured FZI. It is defined as:

$$n\text{MSE} = \frac{\sum_{i=1}^{N} (Y_i - T_i)^2}{\sum_{i=1}^{N} (Y_i - \bar{Y})^2}, \tag{16}$$

Model performance increases as MSE decreases.

For implementation of the IDS operator, the operation board is considered as a 256 × 256 board with the radius of 8 for fuzzy membership propagation. For the experiments, noise injection has been characterized by zero mean value and variance of 8 units in the operation board for the I–O domain of the S.I.S.O. systems. The final result is the average of 5 experiments in order to reduce the possible impact of noise in training process. At each stage of recursion, noise is added independently to each projected S.I.S.O. so as to eliminate the impact of "the history of noise" in the modeling process.

According to the algorithm, through the modeling process of IDS method, for each recursion, FZI should be projected into some 2D planes with respect to the input parameters (DT, NPHI, and RHOB). Therefore, the three-input single-output system is converted into three S.I.S.O. sub-systems. The first application of

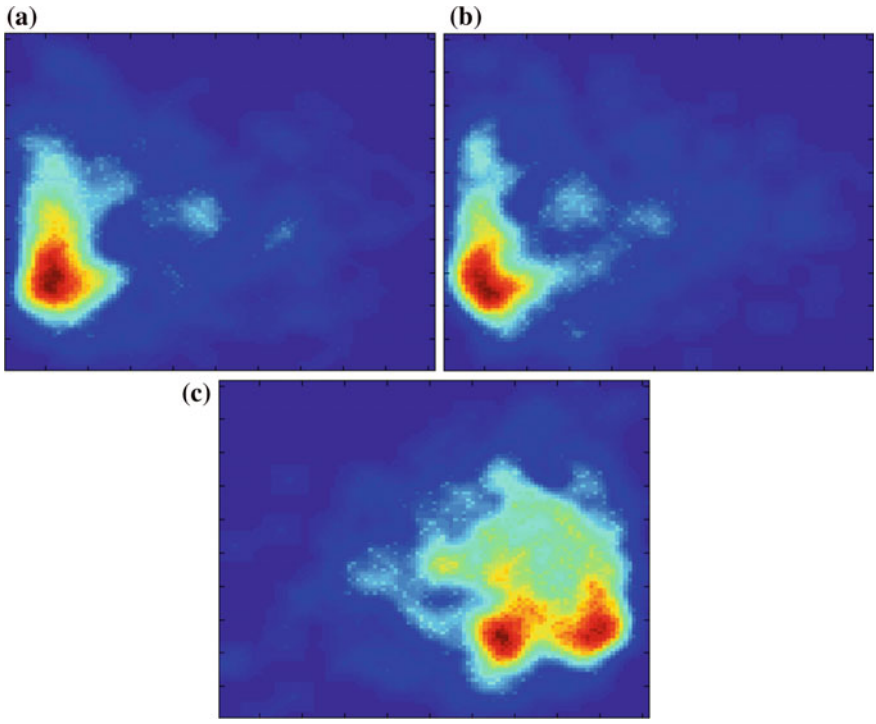


**Fig. 8** The application of IDS operator on the projected $X_i$–$Y$ data planes. **a** Applying IDS operator on DT-FZI plane **b** Applying IDS operator on NPHI-FZI plane **c** Applying IDS operator on RHOB-FZI plane

IDS operator (in the first recursion) on the projected noisy data planes is illustrated in Fig. 8.

ALM tries to reach a predetermined value of generalization error, and until the acquisition of this purpose, the process continues recursively and localized the IDS modeling procedure. In this experiment, ALM took 12 localization levels and noise-injected ALM (NIALM) took 13 localization levels to converge into a stable state. After convergence when no changes in the performance is observed, ALM stops the recursion and returns the simulated outputs.

Figure 9 illustrates the convergence process of ALM and NIALM convergence to the final state for R and nMSE when regarding maximum of 20 levels for localization.

Figure 10 illustrates R for the final results of ALM and noise-injected ALM (NIALM).

Figure 11 illustrates a comparison between the performance of some popular methods such as ANN, RBFN, FL, and Neuro-Fuzzy (which are provided by the MATLAB toolboxes) with ALM and NIALM to demonstrate the effectiveness of noise injection into ALM process.

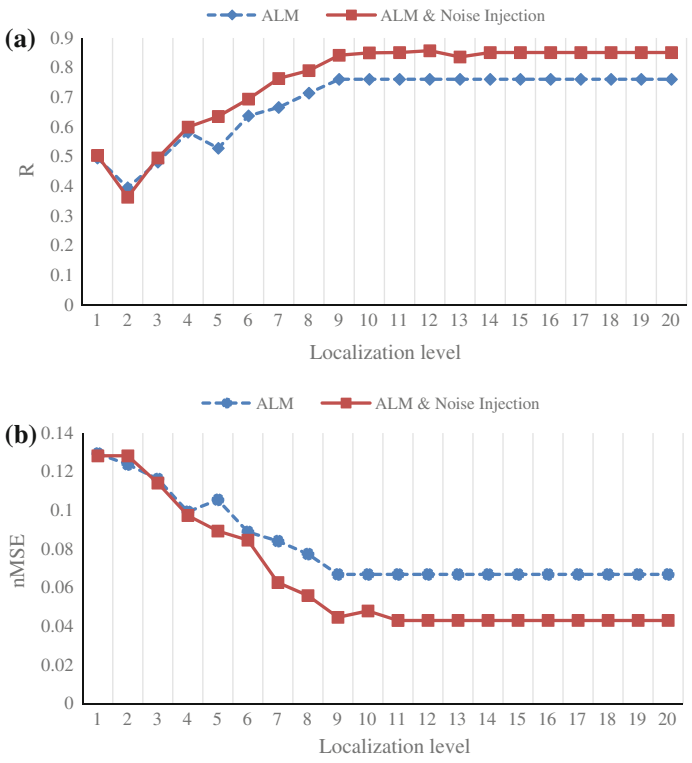


**Fig. 9** The learning procedure of ALM and reaching the final state for R and nMSE

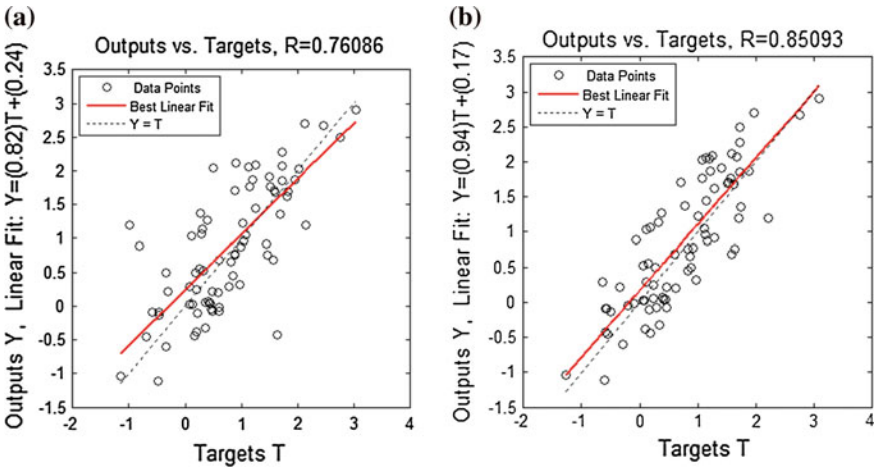


**Fig. 10** R for ALM and NIALM. **a** R of ALM, **b** R of NIALLM

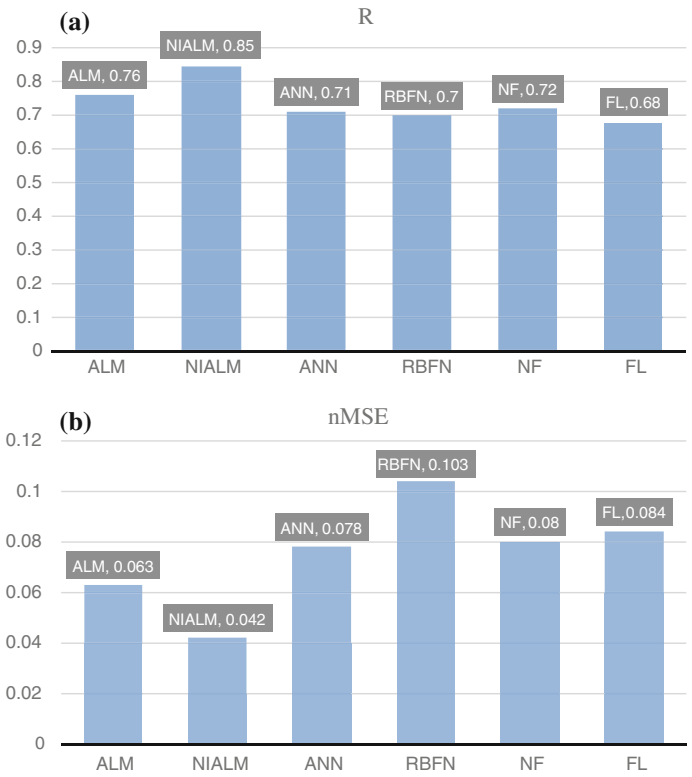


**Fig. 11** Comparison between ALM, NIALM and some popular methods. **a** R, **b** nMSE

Results illustrated in Fig. 11 shows noise injection improves the performance of ALM while both ALM and NIALM express better modeling ability for FZI in comparison with conventional modeling techniques.

## 5 Conclusions

In this paper, utilization of noise injection technique is addressed to improve the performance of ALM for estimating HFUs in a small sample set. While early stopping, which is used by ALM as the overfitting avoidance strategy, suffers from low performance when insufficiency in data samples is observed, noise injection technique helps improving the performance of early stopping when separation of data samples set into training, validation, and test sets degrades the generalization ability of ALM modeling process as a learning technique. Results exhibit satisfactory performance in comparison with use of original ALM and other conventional modeling technique such as ANN, FL, NF, and RBFNN. Increasing nMSE and R by 20 and 10 %, respectively, is a significant improvement which is the result of employing noise injection inside the modeling process of ALM.

## References

Bahrpeyma F, Golchin B, Cranganu C (2013) Fast fuzzy modeling method to estimate missing logs in hydrocarbon reservoirs. J Pet Sci Eng 112:310–321

David RL, Gregory JM (1999) Evaluating the use of goodness-of-fit measures in hydrologic and hydroclimatic model validation. Water Resour Res 35(1):233–241

Ebanks W (1987) Flow unit concept-integrated approach to reservoir description for engineering projects. AAPG Meet Abstr 1(5):521–522

Ghiasi-Freez J, Kadkhodaie-Ilkhchi A, Ziaii M (2012) Improving the accuracy of flow units prediction through two committee machines, South Pars Gas Field, Persian Gulf Basin, Iran. Comput Geosci 46:10–23

Jude OA, Altunbay M, Tiab D, Kersey DG, Keelan DK (1993) Enhanced reservoir description: using core and log data to identify hydraulic (flow) units and predict permeability in uncored intervals/wells. In: 68th annual technical conference and exhibition, Houston

Kadkhodaie-Ilkhchi A, Amini A (2009) A fuzzy logic approach to estimating hydraulic flow units from well log data: a case study from the Ahwazoil field, South Iran. J Pet Geol 32(1):67–78

Liu YS, Janusz AZ (2008) Optimized approximation algorithm in neural networks without overfitting. IEEE Trans Neural Netw 19:983–995

Piotrowski AP, Napiorkowski JJ (2013) A comparison of methods to avoid overfitting in neural networks training in the case of catchment runoff modelling. J Hydrol 476:97–111

Raudys S (2003) Experts' boasting in trainable fusion rules. IEEE Trans Pattern Anal Mach Intell 25:1178–1182

Raudys S (2006) Trainable fusion rules. II. Small sample-size effects. Neural Netw 19:1517–1527

Rifai S, Glorot X, Bengio Y, Vincent P (2011) Adding noise to the input of a model trained with a regularized objective. Technical report dept. IRO, Universite de Montreal. Montreal (QC), H3C 3J7, Canada

Rynkiewicz J (2012) General bound of overfitting for MLP regression models. Neurocomputing 90:106–110

Saeed Bagheri S, Honda N (1997) A new method for establishment and saving fuzzy membership functions. In: 13th fuzzy symposium, Toyama, Japan 1997, pp 91–94

Saeed Bagheri S, Honda N (1999) Recursive fuzzy modeling based on fuzzy interpolation. J Adv Comput Intell 3:114–125

Schittenkopf C, Deco G, Brauer W (1997) Two strategies to avoid overfitting in feedforward networks. Neural Netw 10:505–516

Shouraki SB (2000) A novel fuzzy approach to modeling and control and its hardware implementation based on brain functionality and specification. The University of Electro-Communication, Chofu-Tokyo

Skurichina M, Raudys S, Duin RPW (2000) K-nearest neighbors directed noise injection in multilayer perceptron training. IEEE Trans Neural Netw 11:504–511

Yulei Jiang RMZ, Lorenzo LP, Karen D (2009) A study of the effect of noise injection on the training of artificial neural networks. In: Proceedings of international joint conference on neural networks, Atlanta, Georgia, USA

# Well Log Analysis by Global Optimization-based Interval Inversion Method

**Mihály Dobróka and Norbert Péter Szabó**

**Abstract** Artificial intelligence methods play an important role in solving an optimization problem in well log analysis. Global optimization procedures such as genetic algorithms and simulated annealing methods offer robust and highly accurate solution to several problems in petroleum geosciences. According to experience, these methods can be used effectively in the solution of well-logging inverse problems. Traditional inversion methods are used to process the borehole geophysical data collected at a given depth point. As having barely more types of probes than unknowns in a given depth, a set of marginally over-determined inverse problems has to be solved along a borehole. This single inversion scheme represents a relatively noise-sensitive interpretation procedure. For the reduction of noise, the degree of over-determination of the inverse problem must be increased. To fulfill this requirement, the so-called interval inversion method is developed, which inverts all data from a greater depth interval jointly to estimate petrophysical parameters of hydrocarbon reservoirs to the same interval. The chapter gives a detailed description of the interval inversion problem, which is solved by a series expansion-based discretization technique. Different types of basis functions can be used in series expansion depending on the geological structure to treat much more data against unknowns. The high degree of over-determination significantly increases the accuracy of parameter estimation. The quality improvement in the accuracy of estimated model parameters often leads to a more reliable calculation of hydrocarbon reserves. The knowledge of formation boundaries is also required for reserve calculation. Well logs do contain information about layer thicknesses, which cannot be extracted by the traditional local inversion approach. The interval inversion method is applicable to derive the layer boundary coordinates and certain zone parameters involved in the interpretation problem automatically. In this chapter, it is analyzed how to apply a fully automated procedure for the determination of rock interfaces and petrophysical parameters of hydrocarbon formations. Cluster analysis of well-logging data is performed as a preliminary data processing

M. Dobróka (✉) · N.P. Szabó
Department of Geophysics, University of Miskolc, H-3515 Miskolc-Egyetemváros, Miskolc, Hungary
e-mail: dobroka@uni-miskolc.hu



C. Cranganu et al. (eds.), *Artificial Intelligent Approaches in Petroleum Geosciences*,
DOI 10.1007/978-3-319-16531-8_9

step before inversion. The analysis of cluster number log allows the separation of formations and gives an initial estimate for layer thicknesses. In the global inversion phase, the model including petrophysical parameters and layer boundary coordinates is progressively refined to achieve an optimal solution. The very fast simulated re-annealing method ensures the best fit between the measured data and theoretical data calculated on the model. The inversion methodology is demonstrated by a hydrocarbon field example, which shows an application for shaly sand reservoirs. The theoretical part of the chapter gives a detailed mathematical formulation of the inverse problem, while the case study focuses on the practical details of its solution by using artificial intelligence tools.

**Keywords** Well-logging · Interval inversion · Global optimization · Simulated annealing · Cluster analysis · Calculation of hydrocarbon reserves · Hungary

## 1 Introduction

Geophysical surveying methods with their measuring and evaluation results support the exploration of the Earth and its outer environment. Borehole geophysics is abounding in observed information on the geological formations that are intersected by the drill hole. Well-logging data measured by different probes are recorded along depth in the form of well logs. The processing of open-hole logging data enables to determine some geometrical (e.g., thickness or dip of layers) and petrophysical properties such as porosity, water saturation, composition of rock matrix, and permeability that form an integral part of geological interpretation. Nowadays, there is an ever-increasing claim to the quality of well logs and interpretation results. This is especially important in oil field applications, where a precise calculation of hydrocarbon reserves should be made in complex geological environments.

The advent of inverse modeling (abbreviated as inversion)-based data processing methods was facilitated by the quick evolution of well-logging interpretation systems in the 1980s. In the early years, deterministic techniques solving linear sets of equations or using cross-plot-based graphical methods were applied. These methods gave a solution in several consecutive steps at which the petrophysical parameters were extracted one by one in different procedures (Serra 1984). It was the increased storage capacity and processor speed of computers that promoted the use of simultaneous processing of well logs. The benefit of using the data and petrophysical parameters as statistical variables was unequivocal in the improvement of the quality of interpretation results. Nowadays, the inversion methods are widely used in the petrophysical practice as they give a quick, largely automatic and reliable estimate to the vertical distributions of petrophysical parameters and their estimation errors. The biggest service companies offer inversion-based well-logging interpretation systems, e.g., Global by Schlumberger (Mayer and Sibbit 1980),

Ultra by Gearhart (Alberty and Hashmy 1984), or Optima by Baker Hughes (Ball et al. 1987). The development of these methods is strongly focused in scientific research, too.

Local inversion is the most commonly used technique for the evaluation of borehole geophysical data. Several implementations used in the oil and gas industry are well-known. They have in common that a local value of any petrophysical property is estimated to one depth point using the data measured by different probes in the same depth. In the terminology of geophysical inversion, it is a narrow type of over-determined inverse problem, where the total number of data is barely more than that of the unknown model parameters. The data and the model are connected by probe response functions that are used to calculate theoretical logs in the forward modeling phase of the inversion procedure. By assuming a petrophysical model, one can calculate theoretical well logs, which are then compared with real measurements. The actual model is progressively refined until a proper fit is achieved between the predictions and observations. Local data processing comprises a set of separate inversion runs in adjacent measuring points for the logging interval. It is a general experience that in the inversion of small number of observations, the inversion result is strongly influenced by the uncertainty of measured data. The noise of data highly affects the quality of parameter estimation; thus, the accuracy and reliability of local inversion results are relatively limited. The measurement accuracy of logging tools is prescribed that can be improved seldom with the use of any data processing method. It is a fundamental task to reduce the amount of estimation errors of inversion parameters. In one hand, one can develop more realistic probe response functions. This also means that one tends to set a model approximating the geological structure better. As a result, one can calculate such data by the response functions that are closer to the real observations. Petrophysical research deals with the development of these types of procedures that reduces the model errors. By the above contexture, it is unequivocal that another alternative to improve the quality of parameter estimation can only be facilitated by the further development of the inversion procedures. The most important requirement of the development is the improvement of accuracy and reliability of parameter estimation. For this purpose, the most essential task is the increase of data used in one interpretation procedure. In the framework of local inversion, it leads to the expansion of log types, which is of course restricted and implies additional charges. There is a more effective technique to increase the number of data without extra cost. In the so-called interval inversion procedure, all data of a longer logging interval are processed jointly to determine the characteristic values of petrophysical parameters of several rock units. As a result of the formulation of the interval inversion problem, at least one order of magnitude higher number of data than unknowns can be processed together compared with local inversion. This bears great influence on the accuracy and reliability of the extracted petrophysical parameters. The interval inversion method was introduced in Dobróka (1995), where depth-dependent probe response functions were used (instead of local ones) in the forward problem to give an estimate to the vertical distributions of petrophysical parameters for the entire logging interval. The interval inversion procedure

allows to treat increasing number of inversion unknowns without significant decrease of over-determination (data-to-unknowns) ratio. As a new feature, additional unknowns can be determined together with conventional petrophysical parameters in the same inversion procedure. In Dobróka and Szabó (2012), the possibilities of the determination of formation thicknesses were studied, where the starting model for layer-boundaries was set by external procedure. In this chapter, we suggest a fully automatic inversion strategy using a series expansion-based parameter discretization scheme to estimate the formation boundary coordinates and petrophysical parameters in one inversion procedure for a more objective calculation of hydrocarbon reserves.

## 2 Inverse Problem of Borehole Geophysics

In well-logging inversion, the model parameters of the geological structure are determined in the knowledge of measurement data and approximate formulae of response functions. The aim of interpretation was the lithological separation of formations and the estimation of layer thicknesses and petrophysical properties of formations such as effective porosity, water and hydrocarbon saturation, shale content, mineral volumes, and permeability to infer the quantity and quality of mineral resources. Among them, only those parameters can be determined by inversion, which are contained explicitly in the set of probe response functions and to which almost all types of data are sufficiently sensitive.

The inverse problem of borehole geophysics is classically a joint inversion problem with the particular feature that the quantities included in probe response functions can be divided into two groups. The first group comprises the so-called zone parameters, which are either constants or varying slowly over a longer depth interval (e.g., pore-water resistivity and cementation exponent). The layer parameters form the second group that are nearly constant in a given layer (e.g., porosity and mineral volume). In the practice of well-logging inversion, the zone parameters are treated as external constants that are a priori given in the inversion procedure. This simplification is compulsory in local inversion because the total number of suitable well logs is no more than 10–12, which sets a limit to the number of designated unknown quantities. If the zone parameters were treated as unknowns, an underdetermined (ambiguous) inverse problem would be encountered. In the $k$th local response equation

$$\varphi^{(k)} = g^{(k)}(m_1, \ldots, m_P, M_1, \ldots, M_L) \quad (k = 1, 2, \ldots, S) \tag{1}$$

the layer parameters $(m_1, \ldots, m_P)$ are only determined by inversion, while the zone parameters $(M_1, \ldots, M_L)$ are fixed during the procedure. On the left side of Eq. (1), the calculated value of the $k$th logging data can be found ($S$ is the number of applied probes). As $\varphi^{(k)}$ normally represents a nonlinear functional relationship, thus a nonlinear over-determined inverse problem is posed in the case of $S > P$.

## 2.1 Theory of Local Inversion

In formulating the local inverse problem, all data measured in a given depth point are collected in a column vector

$$\mathbf{d} = \{d_1, \ldots, d_N\}^{\mathrm{T}}, \tag{2}$$

where $(d_1, d_2, d_3 \ldots)$ represent different types of logs such as natural gamma-ray intensity, neutron porosity, and density (T is the symbol of matrix transpose). The theoretical values of the above data can be calculated by the response equations defined in Eq. (1). Let the computed data be represented by vector

$$\boldsymbol{\varpi} = \{\boldsymbol{\varphi_I}, \ldots, \boldsymbol{\varphi_N}\}^{\mathrm{T}}, \tag{3}$$

where the $k$th response equation is as follows:

$$\varphi^{(k)} = g^{(k)}(m_1, \ldots, m_P). \tag{4}$$

The nonlinear functional relationship $g^{(k)}$ can be approximated by its Taylor series truncated at the first order

$$\varphi^{(k)} = \varphi^{(k)}(\mathbf{m_0}) + \sum_{i=1}^{P} \left(\frac{\partial \varphi^{(k)}}{\partial m_i}\right)_{mo} \delta\mathbf{m}, \tag{5}$$

where the series expansion is performed around point $\mathbf{m_0}$, which denotes the vector of initial model parameters. Equation (5) is expressed in vector representation

$$\boldsymbol{\varpi} = \boldsymbol{\varpi}^{(\mathrm{o})} + \mathbf{G}\delta\mathbf{m}, \tag{6}$$

where $\boldsymbol{\varpi}^{(\mathrm{o})} = \boldsymbol{\varpi}(\mathbf{m_0})$ and $G_{ki} = \left(\partial\varphi_k/\partial m_i\right)_{mo}$ is the Jacobi's (parameter sensitivity) matrix. The parameter correction vector $\delta\mathbf{m}$ is estimated by the damped least squares method, which minimizes the Euclidean norm of the following deviation vector

$$\mathbf{e} = \mathbf{d} - \boldsymbol{\varpi}^{(o)} - \mathbf{G}\delta\mathbf{m} \tag{7}$$

with a side condition that $|\delta\mathbf{m}|^2$ is minimal. The objective function of the inverse problem is as follows:

$$E = \sum_{k=1}^{N} e_k^2 + \lambda \sum_{i=1}^{P} \delta m_i^2, \tag{8}$$

where $\lambda$ is a positive damping factor. With the substitution of $\delta\mathbf{d} = \mathbf{d} - \boldsymbol{\varpi}^{(o)}$, the following solution is derived

$$\delta\mathbf{m} = \left(\mathbf{G}^{\mathrm{T}}\mathbf{G} + \lambda\mathbf{I}\right)^{-1}\mathbf{G}^{\mathrm{T}}\delta\mathbf{d}. \tag{9}$$

By solving Eq. (9), the inversion procedure is continued in a given point of the model space

$$\mathbf{m} = \mathbf{m_0} + \delta\mathbf{m} \tag{10}$$

until a stopping criterion is met. The local inversion procedure differs from the Levenberg–Marquardt algorithm only in the consideration of a priori knowledge. Thence some criteria for the lower and upper bounds of the unknowns as well as for the sum of the specific volumes of rock constituents (material balance equation) must be fulfilled. Besides applying these constrains, another program development question is that any parameter may be set fixed in the iteration procedure. The third group of unknowns of the well-logging interpretation problem is formed by the layer boundary coordinates or layer thicknesses. Their role in local inversion is unique, because they are not contained explicitly in the probe response equations. Thus, their estimation by local inversion is out of the question. The measurement data set does contain information on the boundaries that are of great interest in oil field applications, e.g., in the estimation of hydrocarbon reserves. The determination of layer-boundaries is realized commonly in well log analysis not within the inversion procedure.

## 2.2 Depth-Dependent Response Functions

For the calculation of layer thicknesses and zone parameters, a new inversion strategy called interval inversion was developed. Consider the petrophysical (layer) parameters $(m_1, \ldots, m_P)$ as the function of depth. Based on Eq. (4), the $k$th depth-dependent response function is as follows:

$$\varphi^{(k)}(z) = g^{(k)}(m_1(z), \ldots, m_P(z)). \tag{11}$$

In the general case, Eq. (11) contains also the functions of zone parameters $(M_1, \ldots, M_L)$, which can be determined by the interval inversion method (Dobróka and Szabó 2011). The discretization of model parameters $m_1(z), \ldots, m_P(z)$ can be performed by several manners. In the case of layerwise homogeneous model, a series expansion technique with proper basis functions including the coordinates of boundaries answers the purpose. Let $\left(B_1^{(i)}, \ldots, B_{Q_i}^{(i)}\right)$ be the series expansion coefficients of the $i$th model parameter $m_i(z)$. The response function in Eq. (11) takes the form as follows:

$$\varphi^{(k)}(z) = g^{(k)}\left(B_1^{(1)}, \ldots, B_{Q_1}^{(1)}, \ldots, B_1^{(P)}, \ldots, B_{Q_P}^{(P)}, Z_1, \ldots, Z_R, z\right), \tag{12}$$

where $(Z_1, \ldots, Z_R)$ denote the coordinates of layer-boundaries ($Q_i$ is the requisite number of expansion coefficients describing the relevant model parameter). The above response function is valid in the entire interval, in which the series expansion coefficients must be chosen in such a way that the values of $\varphi^{(k)}(z)$ in each depth fit to measurement data $d^{(k)}(z)$ with the highest possible accuracy. The aim of the inversion procedure was the estimation of coefficients $B$, in which all data of the observed interval are inverted. This inverse problem is highly over-determined, because the number of data is several times higher than that of the unknown expansion coefficients. In local inversion, the over-determination ratio is at the best two. On the contrary, in interval inversion the same ratio may reach 50–60. Under this circumstance, the boundary coordinates $(Z_1, \ldots, Z_R)$ can be treated also as inversion unknowns to determine them with the expansion coefficients without significant reduction of the over-determination ratio. The above procedure is called interval inversion including the depth interval where the series expansion is applied for the model parameters (Dobróka 1995). It is assumed that $d^{(k)}$ in a given depth represents a punctual data, i.e., the linear dimensions of the observed volume are smaller than the thickness of layers.

## 3 The Theory of Interval Inversion Method

In geophysical data processing, the term of joint inversion is used when different types of data sets are inverted together in one interpretation procedure. The data sets are measured either by different physical principles or by the same principle but in various measurement arrays. All data measured at different spread layouts carry information on the same geological structure. The theoretical values of data sets integrated into the joint inversion procedure are calculated in the knowledge of all model parameters by a proper forward modeling algorithm, that is, the data may depend on each model parameter. The more the parameters of the geological structure appear in the determination of different data sets, the more successful the solution to the inverse problem can be given. The use of such data sets that are depending only on separated groups of model parameters is unbeneficial compared with independent inversion. In the latter case, the solution will not be more accurate or reliable at all. The local inversion of well-logging data utilizes several data sets based on different physical principles (e.g., nuclear, acoustic, and electric methods), where each datum in the inversion procedure is acquired from the same depth. The observed datum does not depend on the parameters of outlying layers. In this case, therefore, the term of joint inversion can be used only in a restricted sense. It can readily be understood that in the interval inversion approach, it is easy to develop such procedures that allow to exploit all the advantages of joint inversion.

For approximating the depth variations of petrophysical parameters $m_1(z), \ldots, m_P(z)$, a series expansion technique is suggested as follows:

$$m_i(z) = \sum_{q=1}^{Q_i} B_q^{(i)} \psi_q(z, Z_1, \ldots, Z_R), \tag{13}$$

where $B_q^{(i)}$ are expansion coefficients, $\psi_q(z, Z_1, \ldots, Z_R)$ are properly chosen (known) depth-dependent basis functions including the layer-boundaries ($Q_i$ is the requisite number of expansion coefficients describing the $i$th model parameter). Combining Eqs. (12) and (13), the total number of unknowns is $\sum Q_i$, while that of the data is $\sum N_k$. Let us define the data vector of the $k$th well log as follows:

$$d^{(k)} = \left\{ d_1^{(k)}, \ldots, d_{N_k}^{(k)} \right\}^{\mathrm{T}}. \tag{14}$$

The $k$th data in the $j$th depth is calculated by

$$\varphi_j^{(k)} = \varphi^{(k)}(z_j) = g^{(k)}\left( B_1^{(1)}, \ldots, B_{Q_1}^{(1)}, \ldots, B_1^{(P)}, \ldots, B_{Q_P}^{(P)}, z_j \right) \tag{15}$$

and the vector of calculated data is as follows:

$$\boldsymbol{\varpi}^{(k)} = \left\{ \varphi_1^{(k)}, \ldots, \varphi_j^{(k)}, \ldots, \varphi_{N_j}^{(k)} \right\}^{\mathrm{T}}. \tag{16}$$

The data vector of the joint inversion problem including $S$ number of well logs is as follows:

$$\mathbf{d} = \left\{ d_1^{(1)}, \ldots, d_{N_1}^{(1)}, d_1^{(2)}, \ldots, d_{N_2}^{(2)}, \ldots, d_1^{(S)}, \ldots, d_{N_s}^{(S)} \right\}^{\mathrm{T}}, \tag{17}$$

and the vector of all calculated data analogously is as follows:

$$\boldsymbol{\varpi} = \left\{ \boldsymbol{\varphi}^{(1)}, \ldots, \boldsymbol{\varphi}^{(S)} \right\}^{\mathrm{T}}. \tag{18}$$

The series expansion coefficients in Eq. (13) represent the unknowns of the joint inversion problem, thus the combined parameter vector is as follows:

$$\mathbf{m} = \left\{ B_1^{(1)}, \ldots, B_{Q_1}^{(1)}, \ldots, B_1^{(P)}, \ldots, B_{Q_P}^{(P)} \right\}^{\mathrm{T}}, \tag{19}$$

with which the forward problem can be written as follows:

$$\boldsymbol{\varpi} = \boldsymbol{g}(\mathbf{m}) = \left\{ g_1^{(1)}(\mathbf{m}), \ldots, g_{N_1}^{(1)}(\mathbf{m}), \ldots, g_1^{(S)}(\mathbf{m}), \ldots, g_{N_s}^{(S)}(\mathbf{m}) \right\}^{\mathrm{T}}. \tag{20}$$

The measurement data are always contaminated with some amount of noise. On the other hand, the data calculated by Eq. (20) contain modeling errors resulting from discretization and other physical simplifications. The overall error between the two quantities is defined as follows:

$$\mathbf{e} = \mathbf{d} - \mathbf{g}(\mathbf{m}), \tag{21}$$

which is not a zero vector. Inversion methods find a solution at the minimum of some norm of deviation vector **e**. The response functions are usually nonlinear, thence the inverse problem can be solved by either global or linearized inversion methods.

In case of using linearized inversion, the starting point $\mathbf{m_0}$ in model space is considered not too remote from the solution **m**. The vector of parameter corrections in Eq. (10) is sought. The calculated data can be approximated by Eq. (5). The $k$th element of the deviation vector defined in Eq. (21) is as follows:

$$e_k = d_k - \varphi_k(o) - \sum_{i=1}^{P} \left( \frac{\partial g_k}{\partial m_i} \right)_{m_o} \delta m_i, \tag{22}$$

where $\varphi_k(o) = g_k(\mathbf{m_0})$. By introducing the notation $\delta d_k = d_k - \varphi_k(o)$ and $G_{ki} = \left( {}^{\partial g_k}/_{\partial m_i} \right)_{mo}$, the previous vector is as follows:

$$\mathbf{e} = \delta\mathbf{d} - \mathbf{G}\delta\mathbf{m}. \tag{23}$$

The determination of $\delta \boldsymbol{m}$ ensures to reach a closer point to the solution. The optimal estimate can be extracted by an iterative method. The correction of the actual model in the $l$th step is as follows:

$$\mathbf{m}^{(l)} = \mathbf{m}^{(l-1)} + \delta\mathbf{m}^{(l)}, \quad \delta d_k = d_k - \varphi_k(\mathbf{m}^{(l-1)}), \quad G_{ki} = \left( \frac{\partial g_k}{\partial m_i} \right)_{m^{(l-1)}}. \tag{24}$$

If the solution is bound to the minimum of the $L_p$ norm of the deviation vector given in Eq. (23)

$$E = \sum_{k=1}^{N} \left| \delta d_k - \sum_{i=1}^{P} G_{ki} \delta m_i \right|^p, \tag{25}$$

then the undermentioned set of conditions

$$\frac{\partial L_p}{\partial m_h} = 0 \quad (h = 1, 2, \ldots, P) \tag{26}$$

must be fulfilled. As a result of derivation, a nonlinear set of equations is obtained

$$\delta \mathbf{m} = \left(\mathbf{G}^{\mathrm{T}}\mathbf{W}\mathbf{G}\right)^{-1}\mathbf{G}^{\mathrm{T}}\mathbf{W}\delta\mathbf{d}, \tag{27}$$

which can be solved by the iteratively reweighted least squares (IRLS) method that re-calculates the diagonal elements $W_{ks} = |e_k|^{p-2}\delta_{ks}$ of the weighting matrix $\mathbf{W}$ in each iteration step.

## 3.1 Basis Functions Used in Interval Inversion

The selection of basis functions is not strictly limited, but the finding of suitable ones may greatly improve the accuracy and reliability of the inversion result. In case of proper basis functions, a relatively small number of additive terms are enough to be used, because the effect of truncation in Eq. (13) is negligibly small. One should tend to reduce the number of expansion coefficients to maintain the numerical stability of the inversion procedure.

In geophysical inversion, there are several applications of using simple models. In borehole geophysics, the layerwise homogeneous model is of high importance. This situation can be described easily by substituting a combination of Heaviside basis functions into Eq. (13)

$$m_i(z) = \sum_{q=1}^{Q_i} B_q^{(i)}\psi_q(z) = \sum_{q=1}^{Q_i} B_q^{(i)}\left[u(z - Z_{q-1}) - u(z - Z_q)\right], \tag{28}$$

where $Z_q$ is the depth coordinate of the $q$th layer and $Q$ is the number of homogeneous layers. The basis function $\psi_q$ introduced in Eq. (28) is always zero except in the $q$th layer, which is an element of an orthogonal sequence of functions

$$\int_0^{z_{\max}} \Psi_q \Psi_{q'}, \quad \mathrm{d}z = \begin{cases} 0, & \text{if } q \neq q' \\ Z_q - Z_{q-1}, & \text{if } q = q' \end{cases}. \tag{29}$$

It arises that the series expansion coefficient in the $q$th layer equals to the petrophysical parameter in the same layer, that is, $B_q^{(i)} = m_i(Z_{q-1} < z < Z_q)$. The series in Eq. (13) can be rewritten as follows:

$$m_i(z) = \sum_{q=1}^{Q_i} m_q^{(i)} \Psi_q(z), \tag{30}$$

where $m_q^{(i)}$ is the value of the $i$th parameter in the $q$th layer. It is obvious that the inverse problem can be solved by the smallest possible number of unknowns. On the other hand, the layer-boundaries appear in the argument of the basis function $\psi_q$, which can be extracted by the interval inversion method.

The variation of petrophysical parameters within the layer can be approximated by polynomial series expansion

$$m_i(z) = \sum_{q=1}^{Q_i} B_q^{(i)} P_q(z), \tag{31}$$

where $P_q(z)$ represents some polynomial, for instance Legendre polynomials. The meaning of expansion is not demonstrative than in Eq. (30). Similarly, the selection of parameter $Q$ is less unequivocal. The number of unknowns can be much higher than in the homogeneous case. A trade-off must be taken between the vertical resolution of model parameters and the stability of the inversion procedure. To relieve the task, in some practical cases the combination of Eqs. (30) and (31) is used to get an adequate solution. If the layerwise homogeneous model contains an inhomogeneous layer, the following series expansion can be used

$$m_i(z) = \sum_{\substack{q=1 \\ q \neq q'}}^{Q_i} m_q^{(i)} \Psi_q(z) + \sum_{u=1}^{U} B_u P_u\left(z - Z_{q'-1}\right), \tag{32}$$

where $U$ is the number of additive terms used in the approximation of variation in the $q'$th layer. The above problem was solved by Dobróka and Szabó (2005) using a combined inversion algorithm based on the subsequent use of global and linearized optimization methods.

## 3.2 Layer-Thickness Determination by Interval Inversion

The greatly over-determined interval inversion method allows to treat increasing number of inversion unknowns without significant decrease of accuracy in parameter estimation. Some groups of the inversion unknowns are contained in local response functions (e.g., zone parameters); some are not included (layer thicknesses). The latter can be determined by the interval inversion of the combined data set defined in Eq. (17). Consider the model vector of the inverse problem including the layer boundary coordinates

$$\mathbf{m} = \left\{ B_1^{(1)}, \ldots, B_{Q_1}^{(1)}, \ldots, B_1^{(P)}, \ldots, B_{Q_P}^{(P)}, Z_1, \ldots, Z_R \right\}^{\mathrm{T}}. \tag{33}$$

The number of unknowns is $R + \sum Q_i$ that are substituted into Eq. (12) to calculate theoretical well logs in the forward problem. The number of data is specified in Eqs. (17) and (18). The connection between model and data $\boldsymbol{\varpi} = \mathbf{g}(\mathbf{m})$ contains the layer boundary coordinates; therefore, Eq. (5) modifies as follows:

$$\varphi_k = g_k(\mathbf{m_0}) + \sum_{i=1}^{P} \left( \frac{\partial g_k}{\partial m_i} \right)_{\boldsymbol{mo}} \delta m_i + \sum_{r=1}^{R} \left( \frac{\partial g_k}{\partial Z_r} \right)_{\boldsymbol{mo}} \delta Z_r, \tag{34}$$

where $\delta Z_r = Z_r - Z_r^{(0)}$ is an element of the model correction vector $\delta\mathbf{m}$. In Eq. (23), the following Jacobi's matrix is used

$$G_{ki} = \begin{cases} \left( \frac{\partial g_k}{\partial m_i} \right)_{mo}, & \text{if } \; i = 1, 2, \ldots, P \\ \left( \frac{\partial g_k}{\partial Z_i} \right)_{mo}, & \text{if } \; i = P+1, \ldots, P+R \end{cases}. \tag{35}$$

The minimization of the $L_p$ norm of the deviation vector leads to the solution of the inverse problem given by Eq. (27). The iterative method gives an estimate for the series expansion coefficients and layer boundary coordinates. The determination of layer-boundaries can be made easily by using a series expansion based on Eq. (28). However, if it is required, the method can be combined with the scheme of polynomial discretization. The estimation of layer-boundaries can be performed most efficiently by using a global optimization method.

## 4 Global Inversion by Simulated Annealing Method

The performance of inversion methods highly depends on how successfully the optimum of the objective function defined in Eq. (25) is found. Conventional interpretation systems offer linear optimization tools that give quick and satisfactory results in case of having a suitable initial model. The weakness of these gradient-based searching methods is that they tend to find a solution at a local optimum of the objective function. This problem can be avoided by using a global optimization method, which finds the absolute optimum of the same function. There is another typical problem of linear interval inversion. In case of linear optimization, the partial derivatives with respect to depth in the Jacobi's matrix can only be determined in a rough approximation, because the difference quotient with a depth difference being equal to the distance between two measuring points. This may lead to a numerically instable inversion procedure. Global optimization does not require the computation of derivatives. For the solution of global inverse problems,

artificial intelligence tools can be used effectively. Szabó and Dobróka (2013) published previously a float-encoded genetic algorithm-based interval inversion algorithm for oil field and hydrogeological applications. In this study, a fast simulated annealing (SA) method is suggested for the determination of layer parameters and formation thicknesses.

The conventional SA method was developed by Metropolis et al. (1953). In metallurgy, the removal of work-hardening is realized by a slow cooling manipulation from the temperature of liquid alloy state. This process reduces progressively the kinetic energy of a large number of atoms with high thermal mobility, which is followed by the starting of crystallization. Theoretically, the perfect crystal with minimal overall atomic energy can be produced by an infinitely slow cooling schedule. This is analogous with the stabilization of the inversion procedure at the global optimum of the objective function. A quick cooling process causes grating defects and the solid freezes in imperfect grid at a relatively higher energy state. It is similar to the trapping of the inversion procedure in a local minimum. However, the atoms may escape from the high-energy state owing to a special process called annealing to achieve the optimal crystal grating by a slower cooling process. The SA algorithm employs this technology to search the global optimum of the objective (in the terminology energy) function such as $E$ defined in Eq. (25). At first, the components of the model vector defined in Eq. (33) are modified properly. The modification of the $i$th model parameter in the $l$th iteration step is as follows:

$$m_i^{(l+1)} = m_i^{(l)} + b, \tag{36}$$

where $b < b_{\max}$ is a perturbation term ($b_{\max}$ is decreased appropriately as the iteration procedure progresses). During the random seeking, the energy function $E(\boldsymbol{m})$ is calculated and compared with the previous one in every iteration step. The acceptance probability of the new model depends on the Metropolis criteria

$$P(\Delta E, T) = \begin{cases} 1, & \text{if } \Delta E \leq 0 \\ \mathrm{e}^{-\Delta E/T}, & \text{otherwise} \end{cases}, \tag{37}$$

where the model is always accepted when the value of energy function is lower in the new state than that of the previous one. If the energy of the new model increased, there is also some probability of acceptance depending on the values of energy $E$ and control temperature $T$. If $P(\Delta E) \geq \alpha$ fulfills, the new model is accepted, else it is rejected ($\alpha$ is a random number generated with uniform probability from the interval of 0 and 1). These criteria assure the escape from the local minima. Geman and Geman (1984) proved that the following cooling schedule is the necessary condition to find the global optimum

$$T(l) = \frac{T_0}{\ln(l)} \quad (l > 1), \tag{38}$$

where $T_0$ is a properly chosen initial temperature. The SA algorithm is very effective, but the logarithmic reduction of temperature in Eq. (38) is rather time-consuming. Several attempts were made to shorten the CPU time. Ingber (1989) proposed a modified SA algorithm called very fast simulated re-annealing (VFSR). Consider different ranges of variation for each model parameter

$$m_i^{(\min)} \leq m_i^{(l)} \leq m_i^{(\max)}. \tag{39}$$

The perturbation of the $i$th model parameter at iteration $(l + 1)$ is as follows:

$$m_i^{(l+1)} = m_i^{(l)} + y_i\left(m_i^{(\max)} - m_i^{(\min)}\right), \tag{40}$$

where $y_i$ is a random number between $-1$ and 1 generated from a specified non-uniform probability distribution function. The global optimum is guaranteed when the decrease of the $i$th individual temperature follows

$$T_i^{(l)} = T_{0,i}\mathrm{e}^{\left(-c_i\sqrt[P]{l}\right)} \tag{41}$$

Equation (41) specifies different temperature to each model parameter, where $T_{0,i}$ is the initial temperature of the $i$th model parameter, $c_i$ is the $i$th control parameter, and $P$ is the number of model parameters. The acceptance rule of the VFSR algorithm is the same as that used in Metropolis SA method, but the exponential cooling schedule assures much faster convergence to the global optimum than the logarithmic one suggested in Eq. (38).

## 5 Selection of Initial Model by Cluster Analysis

Multivariate statistical methods such as regression, factor, and cluster analyses help to find similarities between petrophysical properties of rocks, reduce problem dimensionality or explore non-measurable (latent) information from the observations, and arrange data into groups to reveal different lithological characteristics of the investigated formations. This a priori information can be useful in petrophysical modeling, facies, or trend analysis and in geophysical inversion to set an initial model as input for the inversion procedure. Clustering methods are applicable to sort data into groups in such a way that the $S$ dimensional objects specified by well logs measured from given depths are more similar than other ones observed from different depths. From the point of the interval inversion method, it is of great importance that objects connected to the same cluster define approximately the same lithological character, while other clusters represent dissimilar ones.

Agglomerative cluster analysis builds a hierarchy from the observations by progressively merging clusters. At the beginning, we have as many clusters as individual elements. In the first step, the closest points are coupled together to form

a new cluster. In each following step, the distances between objects are re-calculated and the procedure is continued until all elements are grouped into one cluster. Several distance definitions can be used as a measure of dissimilarity between the pairs of observed objects such as Euclidean ($L_2$ norm based), Manhattan ($L_1$ norm based), or Mahalanobis (sample covariance based). During the procedure, the distances between the elements of the same group are minimized, while they are maximized between the clusters simultaneously. For the reconnection of clusters, the Ward's linkage criterion is followed that minimizes the deviances of $(x_i - C)$, where $x_i$ is the $i$th object and $C$ is the centroid (average of elements) of the given cluster (Ward 1963). The result of cluster analysis is a dendrogram that shows the hierarchy of clusters and the connections between them at different distances.

In this study, cluster analysis is used as a preliminary data processing step before inverse modeling. It is shown in Fig. 1 that clustering makes use of the complete wellbore data set originated from the entire logging interval. By finding the similarities between the well logs, the objects are grouped into clusters. The log of clusters correlates well with the lithology variation along a borehole. The change in the group number of clusters appearing on the log gives the positions of layer-boundaries, which can be read automatically by computer processing. The estimated layer boundary coordinates as important a priori information for constructing the initial model serve as input for the interval inversion procedure. In an earlier study, the layer-boundaries extracted by cluster analysis were fixed during the interval inversion procedure (Szabó et al. 2013). However, similar to the layer parameters, the layer boundary coordinates may be treated as unknowns in the interval inversion procedure.

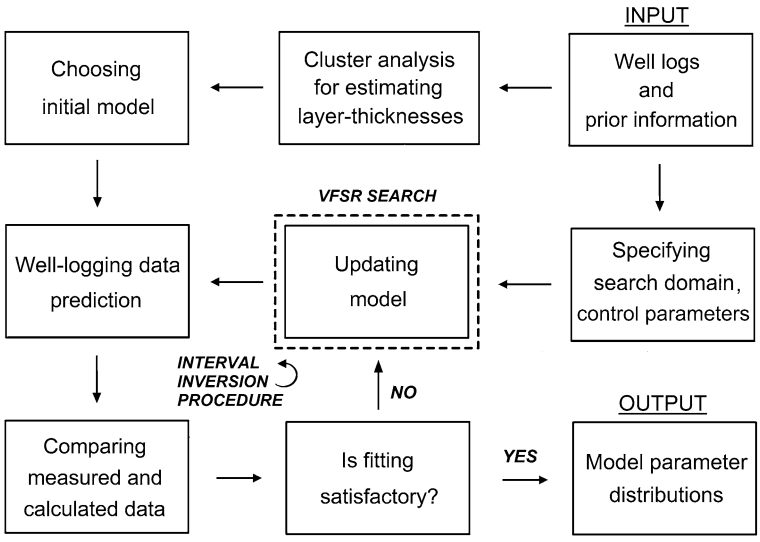


**Fig. 1** Scheme of cluster analysis-assisted global inversion procedure

# 6 Oil field Application

## *6.1 Results of Cluster Analysis*

In a Hungarian hydrocarbon borehole (Well No. 1), nine well logs were used for testing the interval inversion method. The following log types formed the input of clustering such as caliper (CAL), compensated neutron porosity (CN), gamma–gamma density (DEN), acoustic (primary wave) interval time (AT), natural gamma-ray intensity (GR), deep resistivity (RD), microlaterolog resistivity (RMLL), shallow resistivity (RS), and spontaneous potential (SP). In the first step of the procedure, hierarchical cluster analysis was applied to find a proper initial model for inversion processing (Fig. 1). In the processed interval, a sedimentary complex made up of seven unconsolidated shaly sandy beds were deposited. Three lithological categories were specified, namely shale, shaly sand, and sand. At this stage of interpretation, this lithological resolution was enough for finding the layer-boundaries, because the relative volumes of rock matrix and shale could be estimated in the inversion phase. The standardized Euclidean distance evaluating each datum in the sum of squares inversely weighted by the sample variance was used for measuring the distance between data objects. A hierarchical cluster tree was created by using the Ward's linkage algorithm (Fig. 2a). In Fig. 2b, the ordinal numbers of leaf nodes can be seen that were assigned to each object. Since some leaf nodes corresponded to multiple objects, the total number of nodes was 30. Three clusters can be separated if the tree is cut at centroid distance 1. The layer-boundaries can be traced out in the well log of cluster numbers. According to traditional interpretation, the inflection points of high amplitudes in the GR log indicate rock interfaces. The layer boundary coordinates can be well approximated by the steps between the clusters of sand and shale, i.e., cluster 2 and cluster 3. The depth coordinates indicated by black arrows in Fig. 2c were chosen as initial model parameters for the subsequent interval inversion procedure.

The combination of well logs usually gives useful information on lithology, petrophysical, and zone parameters. In Fig. 3, the three-dimensional cross-plots of clustered well-logging data can be seen which specify several site-specific constants for calculating data in forward modeling. These constants can be used directly in the probe response functions. For instance in Fig. 3a, the neutron porosity of sand (13 %) and shale (25 %) and the natural gamma-ray intensity of sand (45 API) and shale (140 API) can be chosen for the given hydrocarbon zone. Several observed data types show strong correlation with each other. In Fig. 3c, d, the nonlinear connection between natural gamma-ray intensity and resistivity is eye-catching. The detailed list of correlation relationships between the data is contained by the correlation matrix including the Pearson's correlation coefficients (Table 1). The highest correlations are between lithology logs (GR and CAL, SP and GR) and saturation logs (RS and RD). Porosity-sensitive logs also show strong correlations with lithology logs (DEN and CAL, SP and CAL, SP and CN, GR and DEN, CN and GR). The negative elements of the correlation matrix show inverse proportionality between the data variables (GR and SP).

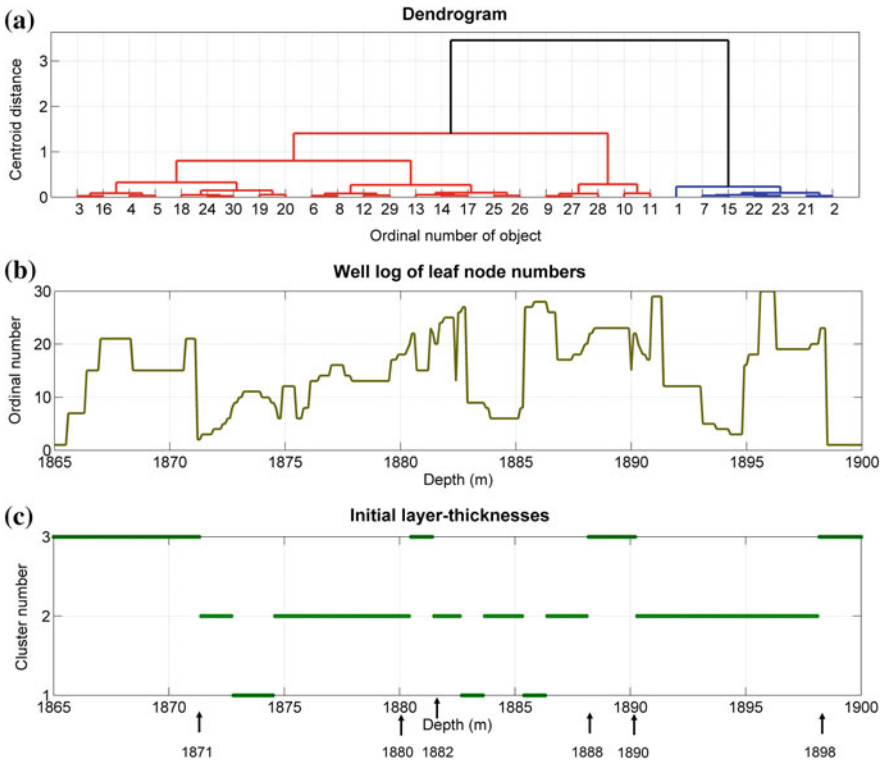


**Fig. 2** Results of cluster analysis in Well No. 1. **a** Dendrogram. **b** Well log of leaf node numbers. **c** Initial layer-thicknesses

## *6.2 Forward Problem of Well-Logging*

The mathematical relationships between the petrophysical properties and well-logging data are dominantly empirical. In the case study, the parameters of the initial model are effective porosity (POR), shale volume (VSH), water saturation of the invaded zone flooded by drilling mud (SX0), water saturation of the virgin zone occupied by original pore fluid (SW), and sand volume (VSD = 1 − POR − VSH). These parameters and other derived ones underlie the calculation of hydrocarbon reserves. The following set of response functions was used to approximate observable data

$$\begin{aligned}\mathrm{DENTH} = \mathrm{POR}[(\mathrm{SX0}\cdot\mathrm{DEMF}) + (1-\mathrm{SX0})\mathrm{DEHC}] \\ + \mathrm{VSH}\cdot\mathrm{DESH} + \mathrm{VSD}\cdot\mathrm{DESD},\end{aligned} \tag{42}$$

$$\mathrm{GRTH} = \mathrm{GRSD} + \frac{1}{\mathrm{DENTH}}\begin{pmatrix}\mathrm{VSH}\cdot\mathrm{GRSH}\cdot\mathrm{DESH} \\ + \mathrm{VSD}\cdot\mathrm{GRSD}\cdot\mathrm{DESD}\end{pmatrix}, \tag{43}$$

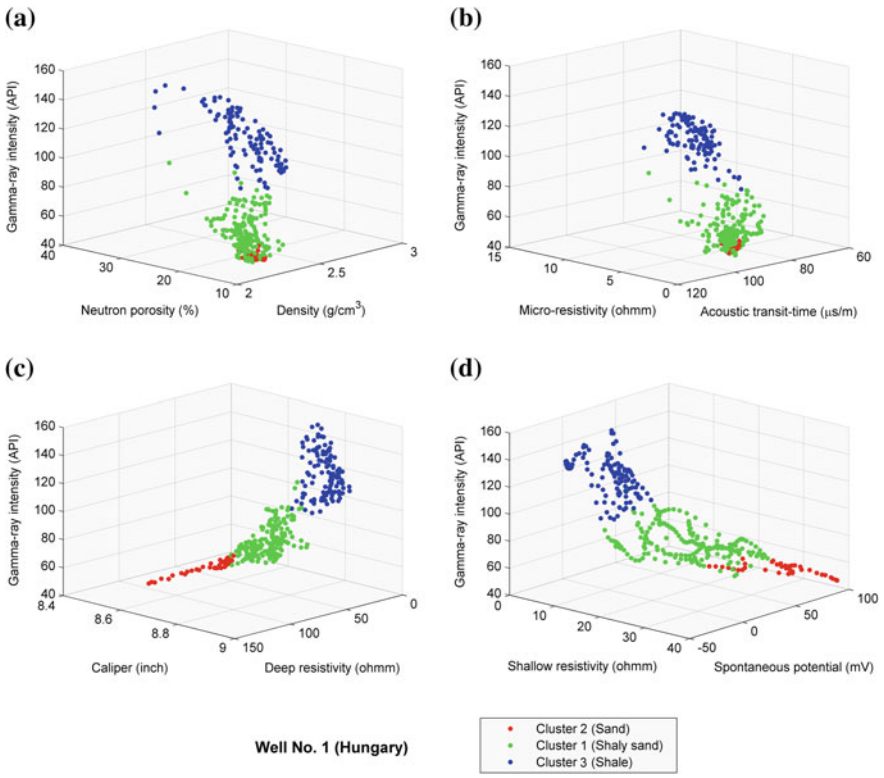


**Fig. 3** Clustered data represented in the form of cross-plots in Well No. 1

**Table 1** Correlation matrix of well logs measured in Well No. 1

| | CAL | CN | DEN | AT | GR | RD | RMLL | RS | SP |
|---|---|---|---|---|---|---|---|---|---|
| CAL | 1.0 | 0.67 | 0.89 | −0.34 | 0.83 | −0.53 | 0.82 | −0.52 | −0.74 |
| CN | 0.67 | 1.0 | 0.68 | 0.16 | 0.88 | −0.59 | 0.47 | −0.56 | −0.76 |
| DEN | 0.89 | 0.68 | 1.0 | −0.51 | 0.84 | −0.58 | 0.82 | −0.59 | −0.73 |
| AT | −0.34 | 0.16 | −0.51 | 1.0 | −0.01 | 0.16 | −0.51 | 0.19 | 0.10 |
| GR | 0.83 | 0.88 | 0.84 | −0.01 | 1.0 | −0.59 | 0.68 | −0.61 | −0.81 |
| RD | −0.53 | −0.59 | −0.58 | 0.16 | −0.59 | 1.0 | −0.41 | 0.92 | 0.68 |
| RMLL | 0.82 | 0.47 | 0.82 | −0.51 | 0.68 | −0.41 | 1.0 | −0.41 | −0.59 |
| RS | −0.52 | −0.56 | −0.59 | 0.19 | −0.61 | 0.92 | −0.41 | 1.0 | 0.71 |
| SP | −0.74 | −0.76 | −0.73 | 0.10 | −0.81 | 0.68 | −0.59 | 0.71 | 1.0 |

$$\begin{aligned} \mathrm{CNTH} = \mathrm{POR}[(\mathrm{SX0} \cdot \mathrm{CNMF}) + (1 - \mathrm{SX0})\mathrm{CNHC}] \\ + \mathrm{VSH} \cdot \mathrm{CNSH} + \mathrm{VSD} \cdot \mathrm{CNSD}, \end{aligned} \tag{44}$$

$$\begin{aligned} \mathrm{ATTH} = \mathrm{POR}[(\mathrm{SX0} \cdot \mathrm{ATMF}) + (1 - \mathrm{SX0})\mathrm{ATHC}] \\ + \mathrm{VSH} \cdot \mathrm{ATSH} + \mathrm{VSD} \cdot \mathrm{ATSD}, \end{aligned} \tag{45}$$

$$\frac{1}{\sqrt{\mathrm{RDTH}}} = \left[\frac{\mathrm{VSH}^{(1-\mathrm{VSH}/2)}}{\sqrt{\mathrm{RSH}}} + \frac{\left(\sqrt{\mathrm{POR}}\right)^{\mathrm{BM}}}{\sqrt{\mathrm{BA} \cdot \mathrm{RW}}}\right]\left(\sqrt{\mathrm{SW}}\right)^{\mathrm{BN}}, \tag{46}$$

$$\frac{1}{\sqrt{\mathrm{RSTH}}} = \left[\frac{\mathrm{VSH}^{(1-\mathrm{VSH}/2)}}{\sqrt{\mathrm{RSH}}} + \frac{\left(\sqrt{\mathrm{POR}}\right)^{\mathrm{BM}}}{\sqrt{\mathrm{BA} \cdot \mathrm{RMF}}}\right]\left(\sqrt{\mathrm{SX0}}\right)^{\mathrm{BN}}. \tag{47}$$

On the left side of Eqs. (42)–(47), the theoretical (TH) values of the well-logging data stand, while on the right, the layer parameters (POR, VSH, SX0, SW, and VSD) and zone parameters can be found. The latter represent the physical properties of mud filtrate (MF), water (W), hydrocarbon (HC), shale (SH), and sand (SD). They are treated as unvarying quantities known from cluster analysis (Fig. 3) or other a priori information (laboratory and well-site reports). The textural constants, such as cementation exponent (BM), saturation exponent (BN), and tortuosity factor (BA), can be estimated from the literature, laboratory data, or the interval inversion method (Dobróka and Szabó 2011). In complex reservoirs, the rock matrix may be composed of several mineral components. Depending on the interpretation problem, the relative volumes of rock constituents can also be extracted within the interval inversion procedure (Dobróka et al. 2012). By using the response Eqs. (42)–(47), an estimate for the model (layer) parameters can be given by the inversion procedure.

### *6.3 Results of Interval Inversion*

The interval inversion procedure was performed on suitable well logs (CN, DEN, AT, GR, RD, and RS) of Well No. 1. Based on the results of cluster analysis, the following depth coordinates were chosen as initial model parameters for interval inversion: 1871, 1880, 1882, 1888, 1890, and 1898 m. The last (seventh) coordinate was the depth at the bottom of the logging interval. As a result, seven shaly sandy layers were traced out. Within three permeable intervals, the separation between DEN and CN logs confirmed the presence of hydrocarbons. In the model approximation, constant layer parameters (POR, VSH, SX0, and SW) were assumed (VSD was calculated deterministically in every iteration steps), which represented a high over-determination ratio (62 with 2100 data, 28 layer parameters, and 6 boundary coordinates) and very stable inversion procedure.

The unknowns of the inverse problem were the series expansion coefficients given in Eq. (28) and the layer boundary coordinates. The VFSR algorithm was used to give a quick estimate to the global optimum of the $L_2$-norm-based energy function. The maximal number of iteration steps was set to 10,000. The logarithmic cooling process based on Eq. (38) was applied with an initial temperature of 0.01. The lower and upper limits of layer parameters were 0 and 1, respectively. The layer-boundaries were allowed to vary within 0 and 10 m. The rate of convergence of the inversion procedure was smooth and progressive as it is seen in Fig. 4, where the root mean squared errors of the relative differences between the measured and calculated data were plotted. The data distance of the final result is influenced by the data noise and the model approximation. In Fig. 5, the change of layer thicknesses calculated from the boundary coordinates can be followed. Until the 7000th iteration step, all thicknesses had attained to their optima. Then, only the layer parameters showed considerable variation.

In the depth scale of Fig. 6, the boundary coordinates estimated by the interval inversion procedure are marked. The method distinguished the permeable and non-permeable intervals within the hydrocarbon zone and gave a proper estimate to rock interfaces as they correlate well with the layer-boundaries inferred from the GR log. The well logs of estimated layer parameters are in tracks 6–8. The pore space was divided into two separate parts filled with salty water and hydrocarbons. The movable and irreducible hydrocarbon saturation were derived from the inversion results by basic equations (Movable HC = SX0 − SW, Irreducible HC = 1 − SX0). The hydrocarbon reserves are estimated from the movable hydrocarbon saturation, porosity, and the total volume of the reservoir rock. The latter can be estimated

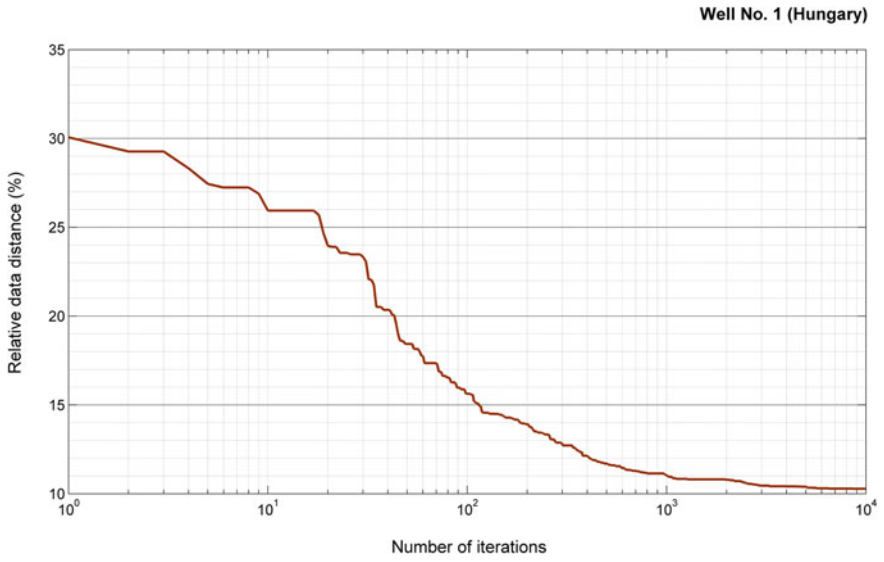


**Fig. 4** The convergence of the interval inversion procedure

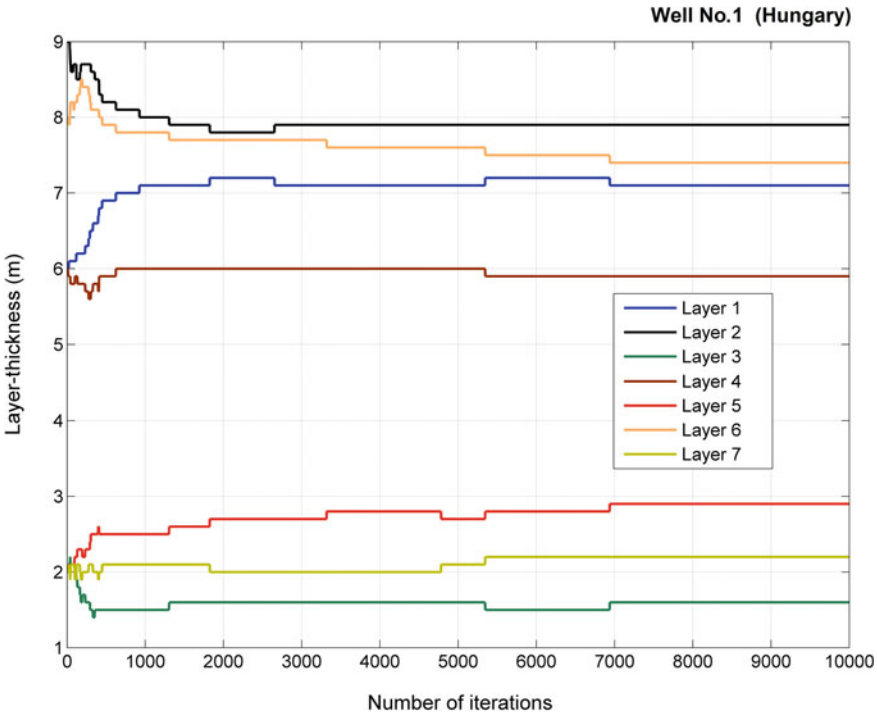


**Fig. 5** The variation of layer-thicknesses during the interval inversion procedure

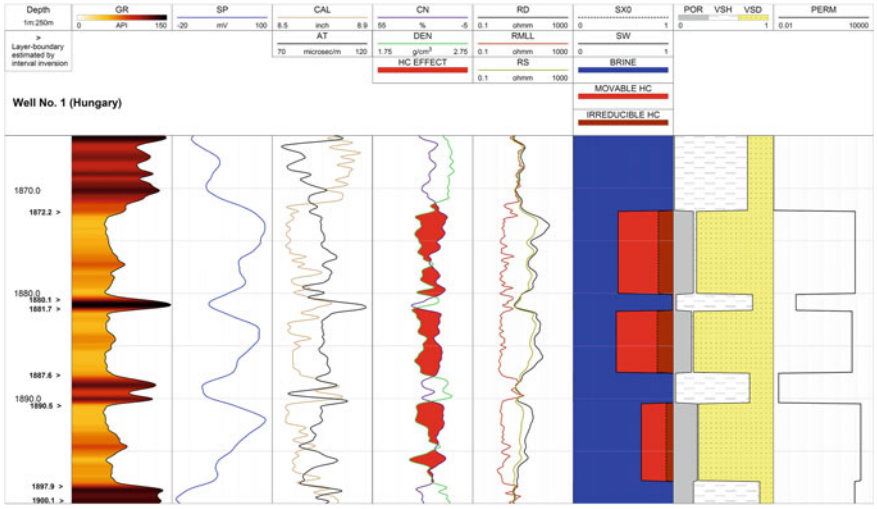


**Fig. 6** The well logs of observed data and interval inversion results

from ground geophysical surveys (e.g., seismic) and multi-borehole data. A model reliable estimate of oil and gas reserves is supported by the results of the interval inversion procedure. Szabó and Dobróka (2013) made a comparison between the local and interval inversion methods. It was shown that the estimation error of porosity and water saturation can be reduced significantly by the interval inversion method. This improvement bears influence on the calculation of hydrocarbon reserves. The unit volume of rock was composed of porosity, shale content, and sand volume in different proportions along the interval (track 7). The absolute permeability shown in track 8 for each formation was calculated by the knowledge of porosity and bound water saturation (chosen as 0.1) in the hydrocarbon reservoirs (Timur 1968).

## 7 Conclusions

Advanced data processing methods are essential to extract reliable petrophysical information from geophysical data sets. An intensive research of inversion methods is being made worldwide in all fields of geophysics. In oilfield applications, the proper interpretation of in situ borehole logging data is especially important, because these methods lay the foundations to hydrocarbon reserve calculations. In this chapter, a new inversion methodology was presented, which is now fully automatized by giving an estimate to petrophysical parameters and layer-boundaries in a joint inversion procedure. The cluster analysis-assisted interval inversion method assures a greatly over-determined inverse problem to give accurate and reliable solution along the entire borehole. The specialty of the method is that the basis functions of series expansion can be chosen arbitrarily. The optimal set of basis functions to be in use depends on the variation of lithology and pore fluid types along a borehole. In this study, a layerwise homogeneous model was chosen to reduce the number of inversion unknowns as far as possible. This approach keeps the numerical stability of the inversion procedure in view. However, there is nothing to prevent from the improvement of the vertical resolution of the interval inversion method, but it goes with the relative increase of the number of series expansion coefficients. As the problem is highly over-determined, it can be allowed to some extent. Practically, a trade-off must be taken between the number of unknowns (resolution) and stability of the inversion (unique solution) procedure as they are inversely proportional. An adequate solution is to choose orthogonal basis functions such as Legendre polynomials in the development of series, in which case the correlation between the estimated model parameters and the parameter estimation errors is relatively the lowest. It is suggested to apply preliminary cluster analysis to find lithological similarities in the data set, which separates such intervals where the polynomial discretization can be performed most effectively. This technical solution leads to the reduction of data and model distances. The inverse problem also affected by the quality of forward problem solution. In the probe response equations, there are several zone parameters that could be chosen

properly for the given well-site. An objective solution can be given by the interval inversion procedure as the zone parameters most sensitive to data can be treated as inversion unknowns. Another strength of the method is the possibility to extend the inverse modeling to multi-borehole applications by expanding the model parameters into series of bivariate basis functions. All of the above properties confirm the feasibility of the interval inversion methods and the use of global optimization techniques preferably very fast simulated re-annealing and float-encoded genetic algorithm in petroleum geoscience applications.

**Acknowledgments** The first author as the leading researcher of Project No. K 109441 thanks to the support of the Hungarian Scientific Research Fund. The second author as the leading researcher of Project No. PD 109408 thanks to the support of the Hungarian Scientific Research Fund. The second author also thanks to the support of the János Bolyai Research Fellowship of the Hungarian Academy of Sciences. The authors thank the Hungarian Oil and Gas Company's (MOL) contribution to the research work and the long-term cooperation. The authors also thank Hajnalka Szegedi for improving the manuscript formally.

## References

Alberty M, Hashmy KH (1984) Application of ULTRA to log analysis. SPWLA 25th annual logging symposium, New Orleans, Paper Z, 10–13 June 1984

Ball SM, Chace DM, Fertl WH (1987) The well data system (WDS): an advanced formation evaluation concept in a microcomputer environment. In: Proceedings of SPE eastern regional meeting, Paper 17034, pp 61–85

Dobróka M (1995) The introduction of joint inversion algorithms into well log interpretation (in Hungarian). Scientific report, Geophysical Department, University of Miskolc

Dobróka M, Szabó NP (2005) Combined global/linear inversion of well-logging data in layer-wise homogeneous and inhomogeneous media. Acta Geod Geophys Hung 40:203–214

Dobróka M, Szabó NP (2011) Interval inversion of well-logging data for objective determination of textural parameters. Acta Geophys 59(5):907–934

Dobróka M, Szabó NP (2012) Interval inversion of well-logging data for automatic determination of formation boundaries by using a float-encoded genetic algorithm. J Petrol Sci Eng 86–87:144–152

Dobróka M, Szabó NP, Turai E (2012) Interval inversion of borehole data for petrophysical characterization of complex reservoirs. Acta Geod Geophys Hung 47(2):172–184

Geman S, Geman D (1984) Stochastic relaxation, Gibbs' distribution and Bayesian restoration of images. IEEE Trans PAMI 6:721–741

Ingber L (1989) Very fast simulated reannealing. Math Comput Model 12(8):967–993

Mayer C, Sibbit A (1980) GLOBAL, a new approach to computer-processed log interpretation. In: Proceedings of SPE annual fall technical conference and exhibition, Paper 9341, pp 1–14

Metropolis N, Rosenbluth A, Rosenbluth M, Teller A, Teller E (1953) Equation of state calculations by fast computing machines. J Chem Phys 21:1087–1092

Serra O (1984) Fundamentals of well-log interpretation. Elsevier, Amsterdam

Szabó NP, Dobróka M (2013) Float-encoded genetic algorithm used for the inversion processing of well-logging data. In: Michalski A (ed) Global optimization: theory, developments and applications. Mathematics research developments, computational mathematics and analysis series. Nova Science Publishers Inc., New York

Szabó NP, Dobróka M, Kavanda R (2013) Cluster analysis assisted float-encoded genetic algorithm for a more automated characterization of hydrocarbon reservoirs. Intell Control Autom 4(4):362–370

Timur A (1968) An investigation of permeability, porosity and residual water saturation relationships for sandstone reservoirs. Log Analyst 9:8–17

Ward JH (1963) Hierarchical grouping to optimize an objective function. J Am Stat Assoc 58(301):236–244

# Permeability Estimation in Petroleum Reservoir by Meta-heuristics: An Overview

**Ali Mohebbi and Hossein Kaydani**

**Abstract** Proper permeability distribution in reservoir models is very important in the determination of oil and gas reservoir quality. In fact, it is not possible to have accurate solutions in many petroleum engineering problems without having accurate values for this key parameter of hydrocarbon reservoir. Permeability estimation by individual techniques within the various porous media can vary with the state of in situ environment, fluid distribution, and the scale of the medium under investigation. Recently, attempts have been made to utilize artificial intelligent methods for the identification of the relationship which may exist between the well log data and core permeability. This study overviews the different artificial intelligent methods in permeability prediction with advantage of each method. Finally, some suggestions and comments to choose the best method are introduced.

## 1 Introduction

Permeability is defined as a measure of the ability of a porous material to allow fluids to pass through it (Ahmed 2001). The concept of permeability is vital in determining precise reservoir description and simulation, which are the most important means for reservoir management. Permeability is also essential in overall reservoir management and development for choosing the optimal drainage points and production rate, optimizing completion and perforation design, and devising EOR patterns and injection conditions. Therefore, before any modeling or calculation, this parameter must be determined (Biswas et al. 2003). The most exact method in permeability prediction is core analysis in a laboratory by application of Darcy's law, which is expensive and time consuming (Timur 1968; Saemi et al. 2007). The well testing analysis is another expensive, time-consuming method in

A. Mohebbi (✉) · H. Kaydani
Department of Chemical Engineering, Faculty of Engineering, Shahid Bahonar University of Kerman, Kerman, Iran
e-mail: amohebbi2002@yahoo.com; amohebbi@uk.ac.ir



C. Cranganu et al. (eds.), *Artificial Intelligent Approaches in Petroleum Geosciences*,
DOI 10.1007/978-3-319-16531-8_10

permeability prediction, which gives the average permeability of the porous media around the wellbore (Mohebbi et al. 2012).

To overcome these obstacles, different methods for permeability estimation have been proposed. The oldest method in permeability prediction was empirical correlations between permeability and other petrophysical properties (Timur 1968). These correlations have been used with some success in sandstone reservoirs (Weber and Van Geuns 1990); however, for heterogeneous formations, they cannot be applied (Molnar et al. 1994).

In any oil or gas field, all wells are logged using electrical tools to measure geophysical parameters. This availability leads the attempts that have been applied to predict permeability from well log data (Mohaghegh et al. 1994, 1996; Huang et al. 1996). The complexity, vagueness, and uncertainty existence, in addition to nonlinear behavior of most reservoir parameters, require a powerful tool to find relationship between permeability and petrophysical properties of reservoir rock. Therefore, intelligent techniques or meta-heuristics, such as artificial neural networks (ANNs), fuzzy logic (FL), support vector machine (SVM), and genetic algorithms (GAs), has played a noticeable part in permeability prediction from well log data. Previous investigations (Mohaghegh and Ameri 1995; Aminzadeh et al. 1999; Mohaghegh et al. 2001; Saemi et al. 2007; Saemi and Ahmadi 2008; Karimpouli et al. 2010; Al-Anazi and Gates 2010) indicated that intelligent techniques are superior to statistical methods in predicting permeability from well log data because of their excellent pattern recognition ability. In this chapter, an overview on different meta-heuristics in permeability prediction was done.

## 2 Permeability Estimation

Various techniques are available that provide the reservoir permeability estimation in porous media. There are two generally reliable ways of acquiring knowledge on rock permeability. These are (1) direct measurements of rock sample (cores) and analyzing well test data and (2) estimation models that relate permeability to other petrophysical rock properties. The first methods (e.g., coring and well testing) are very useful, but they are not sufficient to show the heterogeneity of reservoir, because, due to intensive time demanding and high cost, it is possible to drill only limited wells. Also, another restriction can be due to unavailability of cores, missed cores in certain intervals, etc.

The well testing method for permeability determination is *pressure transient analysis*, which provides a volumetrically averaged permeability for the volume of the reservoir that has been investigated during the test. Tests should be designed so that they are long enough to achieve reliable and usable data. On the other hand, the longer the test time, the larger the volume represented by the calculated permeability. In any oil or gas field, all wells are logged using electrical tools to measure geophysical parameters. This availability leads the attempts that have been applied to predict permeability from well log data.

The oldest method in permeability prediction was based on empirical correlations between permeability and other petrophysical properties. These correlations have been used with some success in sandstone reservoirs. The complexity, vagueness, and uncertainty existence, in addition to nonlinear behavior of most heterogeneous reservoir parameters, require a powerful tool to overcome these challenges. In recent years, meta-heuristics such as ANNs, FL, and GAs has played a noticeable part in reservoir engineering applications. Each of these methods is explained briefly below.

## 2.1 Core Analysis

Coring is an essential part of the reservoir life cycle process, with cored wells selected to verify or provide maximum information for the geological, engineering, or production model of the reservoir (Levorsen 1996). Permeability is one of the parameters that is generally measured in the laboratory on the cored rocks taken from the reservoir. In this method, permeability is determined primarily by flowing nitrogen, air, or any nonreactive fluid through the sample. The core plug was inserted in special holding device such as illustrated in Fig. 1. The permeability can be determined from obtained data at several flow rates by applying Darcy's law. Figure 2 shows typical plotting results with either liquid or gas.

It should be mentioned that a viscous flow is the best satisfied condition in permeability measurements. Although gas permeability measurements are significantly faster and less expensive, the industry is still debating the validity and utility of gas permeability data for liquid-producing reservoirs because of effect of gas slippage (or Klinkenberg effect) on permeability measurement (Amyx et al. 1960; Norman 1984). The effect of overburden on the permeability measurement is also important. This effect is more pronounced in unconsolidated, low-permeability, and

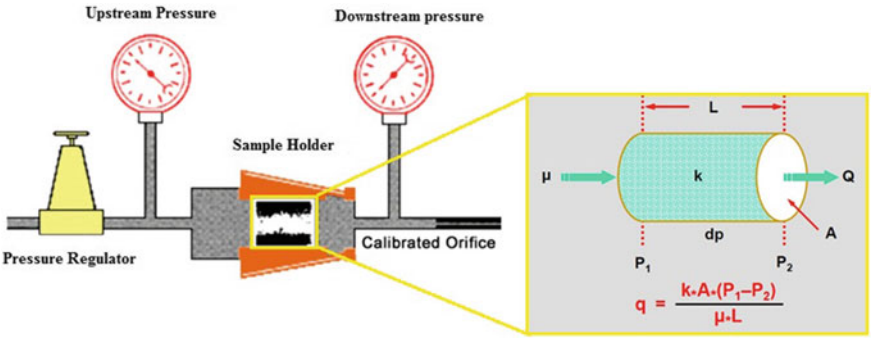


**Fig. 1** A core plug in special holding device illustration for permeability measurement

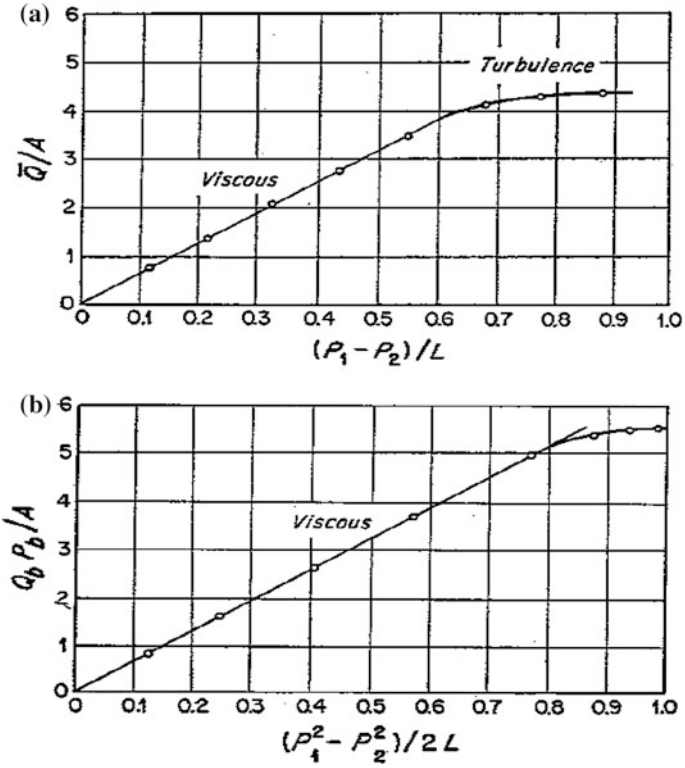


**Fig. 2** A typical plotting results in permeability measurement with **a** liquid and **b** gas

fractured samples. Consolidated-sample ambient-pressure permeability can be corrected for the effect of overburden by applying a correction factor determined by measuring a few samples at both overburden and ambient conditions (Tiab and Donaldson 2004; Amyx et al. 1960).

Among different techniques, the permeability obtained from core analysis in laboratory is more valid than that obtained by the other methods. But, this technique cannot be widely used from an economic point of view, because of its high cost and being time consuming. However, the results from this reliable technique were used as target data to validate other estimation models (Saemi et al. 2007).

## 2.2 Well Test Analysis

Another popular way to obtaining reservoir permeability is *well test analysis*. The data that become available after a carefully designed well test help petroleum engineers to calculate a volumetric average of the formation permeability, among

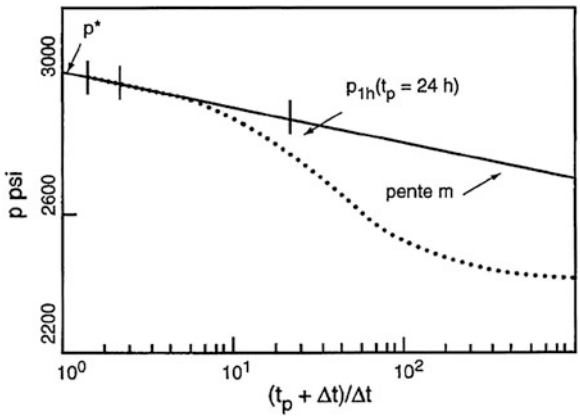


**Fig. 3** A typical pressure curve response in buildup well testing

other parameters such as skin factor and wellbore storage (Prasad et al. 1996; Jeirani and Mohebbi 2006). The basic of well test technique is to create a pressure drop in a bottom hole pressure, which causes reservoir fluid to flow in a certain rate from the reservoir rock to wellbore, followed by shut-in period. The production period is generally referred as pressure drawdown, whereas the shut-in period is called pressure buildup. By utilizing some analytical equations, which are solutions of diffusivity equations, from the response of the pressure versus time (pressure curve), some of rock properties such as porosity and permeability can be obtained (Earlougher 1977; Horne 1995). A typical pressure curve response in buildup test is showed in Fig. 3.

Although it is a valuable and necessary procedure, well testing is not a viable procedure for any developed reservoirs. Due to its cost of performing the test, in addition to the loss of production during the test and its limited radius of the formation investigated around the wellbore, this worthwhile method cannot be widely used. More information about well test analysis techniques can be found elsewhere (Matthewe and Russell 1967; Earlougher 1977; Horne 1995).

## *2.3 Empirical Correlation*

The first equation relating petrophysical rock properties to permeability was proposed by Kozeny (1927). The lack of global applicability of his model has led researchers to modify it, and other parameters in this equation were considered. This equation was modified by Carman (1937) and expressed as follows:

$$k = \phi^3/[5S_o^2(1-\phi)^2] \tag{1}$$

where $S_o$ is surface area of grains exposed to fluid per unit volume of solid rock and $\phi$ is porosity of the rock. The surface area parameter is the major drawback of this formula, because it can be determined only by core analysis and obtained only with special equipment. Tixier (1949) represented a mathematical correlation, which indicated the influence of resistivity gradients, water saturation, and capillary pressure on permeability of rock. However, his model is physically limited in scope by the relative paucity of logs exhibiting valid oil water contacts and the necessity for estimating the hydrocarbon density as it exists in the reservoir. Wyllie and Rose (1950) expanded the empirical relationship proposed by Tixier and investigated the effects of irreducible water saturation and tortuosity on rock permeability. Timur (1968) derived a similar expression that related permeability to porosity and water saturation in a generalized form as:

$$k = A\frac{\phi^B}{S_{\mathrm{wi}}^C} \tag{2}$$

where $A$, $B$, and $C$ are parameters that should be determined statistically. Also, a similar empirical correlation was proposed by some other investigators (Pirson 1963; Coates and Dumanoir 1974; Coates and Denoo 1981; Bloch 1991). Ahmed et al. (1991) presented some graphs used for permeability estimation from the various correlations (Fig. 4). Empirical equations have been used with some successes in some sandstone and fracture reservoirs.

Flow zone indicator (FIZ) is another way to estimate rock permeability (Prasad 1999; Perez et al. 2005; Uguru et al. 2005; Bagheripour and Shabaninejad 2011). FZI is a factor that includes the geological and petrophysical properties for permeability prediction. Low FZI can be a sign of fine-grained, poorly sorted sands, whereas high FZI can indicate clean, coarse-grained and well-sorted sands. Different depositional environments and diagenetic processes control the geometry of the reservoir and consequently the flow zone index (Amaefule et al. 1993).

Among the available permeability correlations, the hydraulic unit concept is widely used in permeability prediction. This technique is based on the porosity–permeability empirical relationship, modified by Amaefule et al. (1993). The hydraulic unit permeability formula can be expressed as follows:

$$\log \mathrm{RQI} = \log \phi_n + \log \mathrm{FZI} \tag{3}$$

$$\mathrm{RQI} = 0.314\sqrt{\frac{k}{\phi_e}} \tag{4}$$

$$\phi_n = \frac{\phi_e}{1-\phi_e} \tag{5}$$

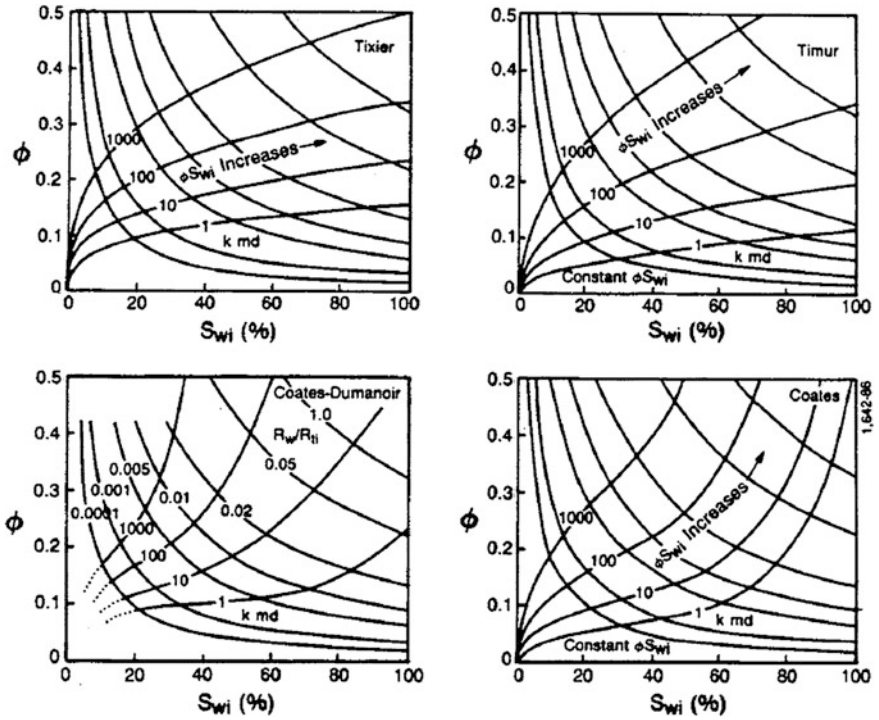


**Fig. 4** Graphs used for permeability estimation from the various correlations presented by Ahmed et al. (1991)

where FZI is flow zone indicator (μm), $\phi_n$ is normalized porosity index, RQI is reservoir quality index (μm), $\phi_e$ is effective porosity, and $k$ is permeability (md). This equation yields a straight line on a log–log plot of RQI versus $\phi_n$ with a unit slope. The intercept of this line is flow zone indicator. Samples with different FZI values will lay on parallel lines. By estimation of FZI, the permeability of reservoir can be calculated according to the following equation:

$$k = 1014(\mathrm{FZI})^2 \left[\frac{\phi_e^3}{(1-\phi_e)^2}\right] \tag{6}$$

Ghafoori et al. (2008) used FZI method for permeability estimation in one of the Iranian carbonate reservoirs and concluded that this method fails to predict permeability over the high permeable intervals.

Although this technique seems to be an ideal tool in permeability estimation, the vagueness and uncertainty existence, in addition to complex behavior of most reservoir parameters, caused this method to not be considered as powerful tool to overcome these challenges (Molnar et al. 1994; Saemi et al. 2007).

### 2.4 Meta-heuristics Approaches

Recently, artificial intelligent methods (meta-heuristics) have become a notable part of petroleum engineering problems. The major reason for this rapid growth and application of meta-heuristics is their ability to approximate any function in a stable and efficient way by an inexpensive approach.

ANNs have been increasingly applied to predict reservoir properties using well log data. Moreover, previous investigations have indicated that ANNs can predict formation permeability even in highly heterogeneous reservoirs using geophysical well log data with good accuracy (Mohaghegh et al. 1994; Saemi et al. 2007). In spite of the wide range of applications, ANNs are still designed through a time-consuming iterative trial-and-error approach. This leads to a significant amount of time and effort being expended to find the optimum or near-optimum structure for a neural network for the desired task. In order to mitigate these deficiencies, design of neural networks using GAs has been proposed (Dehghani et al. 2008; Kaydani et al. 2011). Fuzzy modeling, as a powerful artificial intelligent method, can model highly complex nonlinear problems (Taghavi 2005; Ilkhchi et al. 2006). In the next section, permeability prediction by different artificial intelligent methods is discussed briefly.

## 3 Artificial Intelligent Approaches in Permeability Prediction

Availability of the well log and coring data for the wells in any oil and gas reservoir leads to attempts that have been applied to predict permeability from them (Mohaghegh et al. 1994; Malki et al. 1996; Wong et al. 1998; Kumar et al. 2000). The prediction of permeability in heterogeneous formations from well log data poses a difficult and complex problem (Saemi et al. 2007). A comprehensive approach for correlating permeability with geophysical well log data in heterogeneous formations was developed by Molnar et al. (1994). This approach combined gamma ray, deep induction, and compensated bulk density well log responses and detailed core analysis to subdivide the formation into several zones. Then, a reliable statistical correlation between permeability and bulk density was developed for each zone.

Alternatively, ANNs have been increasingly applied to predict reservoir properties using well log data (Mohaghegh et al. 1994; Mohaghegh and Ameri 1995; Wiener 1995; Boadu 1997; Arpat et al. 1998; Jamialahmadi and Javadpour 2000; Chang et al. 2000). ANNs are computing systems based on the interaction of large numbers of simple processing units, which are called nodes. A typical multilayer ANN consists of different layer of nodes as shown in Fig. 5. Mohaghegh et al. (1996) indicated that neural network is a powerful tool for identifying the relationship

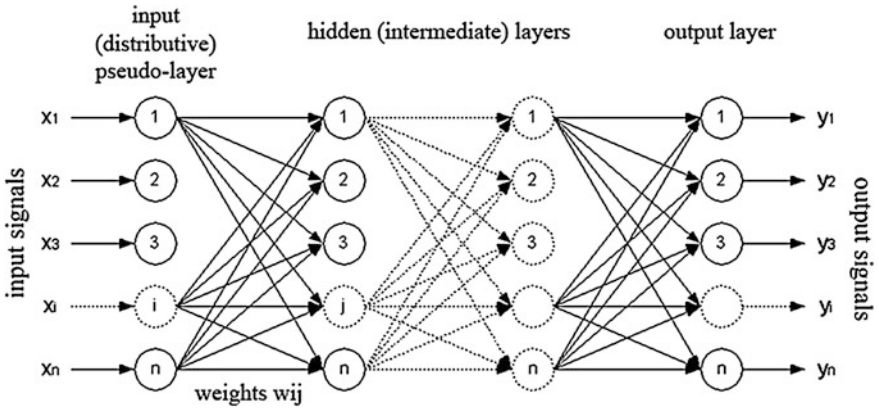


**Fig. 5** A typical multilayer ANN with different layer of nodes

among permeability and geophysical well log data. Moreover, Aminian et al. (2000, 2001) indicated that ANNs can be applied to predict formation permeability even in highly heterogeneous reservoirs with good accuracy.

In spite of the wide range of applications, neural networks are still designed through a time-consuming approach. This leads to a significant amount of time and effort being expended to find the optimum or near-optimum structure for a neural network in a desired task (Niculescu 2003; Saxena and Saad 2006; Dehghani et al. 2008). In order to mitigate these deficiencies, automatic designs of neural networks have been proposed by Boozarjomehry and Svrcek (2001). However, these methods have been applied only for the design of neural networks used for simple tasks and not for more complex problems.

The researches by Van Rooij et al. (1996) and Vonk et al. (1997) have proposed the use of evolutionary computation techniques such as GAs in the field of ANNs to generate an optimal ANN architecture. Huange et al. (2001) and Chena and Lina (2006) applied this method in permeability estimation from log data and showed that it is highly effective to apply integrated GAs to ANNs in permeability prediction. However, these works did not cover the optimization of ANN parameters using GAs. Saemi et al. (2007) proposed a new method, whose design of topology and parameters of the neural networks as decision variables was done by using GAs in order to improve the effectiveness of forecasting when ANN is applied to a permeability predicting problem by a case study in South Pars gas field in Persian Gulf. Tables 1 and 2 show their results by two methods: trial-and-error approach and optimization with GA method, respectively. It can be found from these tables that GA was a good alternative over the trial-and-error approach to determine the optimal ANN architecture and internal parameters quickly and efficiently. Moreover, Kaydani et al. (2011) estimated permeability based on reservoir zonation by a hybrid neural GA in one of the Iranian heterogeneous oil reservoirs. They showed that permeability prediction based on designing separate networks for each zone is more accurately than designing single network design for all of zones. Tahmasebi

**Table 1** The performance of Saemi et al. (2007) testing data set by trial-and-error approach

| Parameter | Performance |
|---|---|
| MSE | 0.3783 |
| NMSE | 0.2437 |
| MAE | 0.2272 |
| Min absolute error | 0.0011 |
| Max absolute error | 4.0766 |
| *r* | 0.8579 |

**Table 2** The performance of Saemi et al. (2007) testing data set by ANNs optimization with GA

| Parameter | Performance |
|---|---|
| MSE | 0.1197 |
| NMSE | 0.0453 |
| MAE | 0.0681 |
| Min absolute error | 0.0001 |
| Max absolute error | 1.0864 |
| *r* | 0.989 |

and Hezarkhani (2012) proposed a method along four different neural network architectures to predict the permeability, and the obtained results were compared statistically. According to their results, they showed a modular neural network (MNN) as a new method, which had a very low computational time with high learning capacity and affordability for permeability prediction.

Kaydani and Mohebbi (2013) presented a comparison study of using optimization algorithms and ANNs for predicting permeability. They proposed a novel approach to estimate permeability by combining Cuckoo Optimization Algorithm (COA), particles swarm, and Imperialist Competitive Algorithms (ICA) with Levenberg–Marquardt (LM) neural network algorithm in one of the heterogeneous oil reservoirs in Iran. Figure 6 shows the proposed flowchart of the optimized LM neural network modeling with optimization algorithms for permeability prediction in their work. They concluded from a testing data set that the trained COA–LM neural model can efficiently accomplish permeability prediction. Also, the comparison of COA with particle swarm optimization and ICA showed the superiority of COA on fast convergence and best optimum solution achievement (see Table 3). COA is a new evolutionary algorithm, proposed by Rajabioun (2011), and was inspired from special lifestyle of cuckoo birds. COA mimics the breeding behavior of cuckoos, where each individual searches the most suitable nest to lay an egg in order to maximize the egg's survival rate, which is an efficient search pattern. Application of the COA in different optimization problems has proven its capability to deal with difficult optimization problems, especially in multi-dimensional problems. More information about this optimization algorithm is available in literature (Rajabioun 2011).

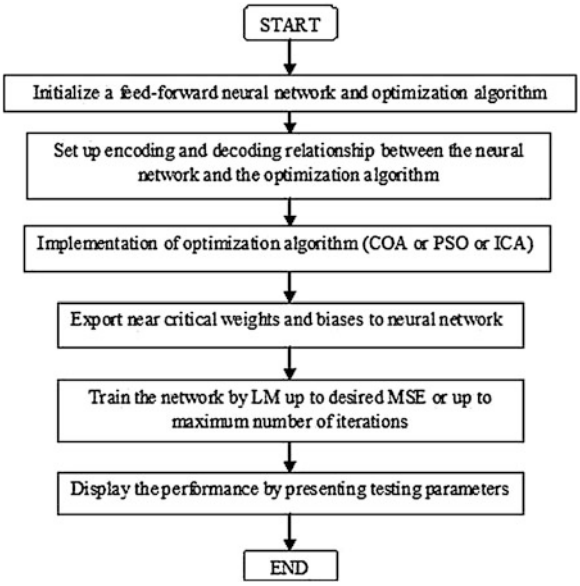


**Fig. 6** Flowchart of the hybrid optimization—LM neural network proposed by Kaydani and Mohebbi (2013)

**Table 3** Performance of neural network optimum model by different optimization methods

| Parameter | COA-LM | ICA-LM | PSO-LM |
|---|---|---|---|
| AAD | 0.099 | 0.104 | 0.119 |
| NMSE | 0. 012 | 0.017 | 0.020 |
| *R* | 0.978 | 0.961 | 0.943 |
| *R*-square | 0.957 | 0.924 | 0.889 |

Fuzzy set theory, a method to distribute linguistic fuzzy information by mathematics, distributes a set by using a membership function and extends the concepts of classical set theory (Zadeh 1965; Klir and Yuan 1995; Zeng and Singh 1996). Fuzzy modeling as a powerful meta-heuristics can model highly complex nonlinear systems, such as multi-input and multi-output problems. It is an established fact that geosciences disciplines are not clear-cut and, most of the time, are associated with uncertainties. So, FL can be applied successfully in permeability prediction of porous media (Cuddy and Putnam 1998; Hambalek and Reinaldo 2003; Taghavi 2005; Lim 2005; Ali et al. 2006; Abdullraheem et al. 2007; Nashawi and Malallah 2010; Olatunji et al. 2011). Ilkhchi et al. (2006) used data from three wells of the Iran offshore gas field for construction of FL models of the reservoir, and a fourth well was used as a test well to evaluate the reliability of the models. Their results showed that FL approach was successful for the prediction of permeability in rocks of the gas field.

Combination of the explicit knowledge representation of FL and the learning power of neural nets yields adaptive neural fuzzy inference system (ANFIS), which can be more useful in prediction of model. Nowadays, neural fuzzy systems have become more versatile approach to the problem in petroleum engineering

(Nowroozi et al. 2009). One example for using this technique in permeability prediction was the work done by Gedeon et al. (1997). They incorporated fuzzy IF-THEN rules into neural networks to interpolate the reservoir properties. Kaydani et al. (2012) developed a neural fuzzy system for prediction of permeability from wireline data based on fuzzy clustering in one of the Iranian carbonate reservoirs. They showed that by using a fuzzy c-means cluster technique in neuro-fuzzy model, the prediction of permeability in porous media can be improved.

Recent works on artificial intelligence techniques have led to introduce a robust machine learning methodology, called SVM. SVMs are supervised learning models with associated learning algorithms that analyze data and recognize patterns. SVMs, based on the structural risk minimization (SRM) principle (Stitson et al. 1999), seem to be a promising method for data mining and knowledge discovery. It was introduced in the early decade of 2000 as a nonlinear solution for classification and regression tasks (Burbidge et al. 2001; Jeng et al. 2003; Trontl et al. 2007). This technique aimed at predicting the permeability of the hydrocarbon reservoir (Al-Anazi and Gates 2010, 2012). As an example, Gholami et al. (2012) utilize the SVM for predicting the permeability of three gas wells in the Southern Pars field in Iran. Their results showed that the correlation coefficient between core and predicted permeability is 0.96 by using the SVM in testing data set, which illustrated in Fig. 7. Also, comparing the results of SVM with those of a general regression neural network (GRNN) revealed that the SVM approach is faster and more accurate than the GRNN in prediction of hydrocarbon reservoirs permeability.

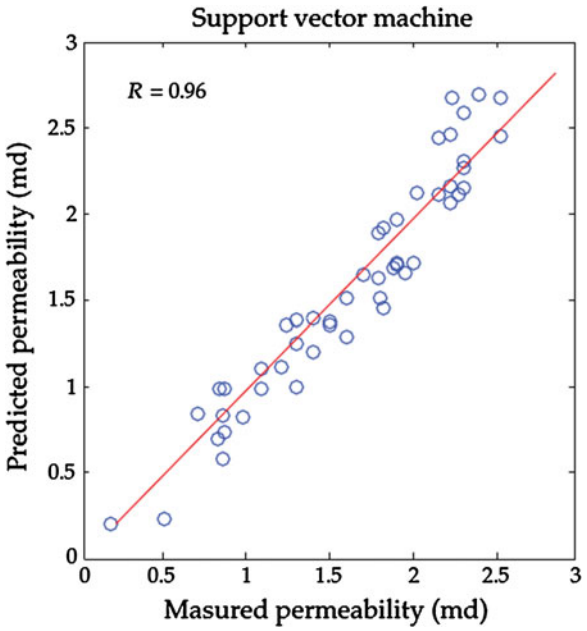


**Fig. 7** SVM result for permeability estimation in three gas wells (Gholami et al. 2012)

## 4 Techniques Interrelationships and Comparisons

The ways for permeability determination can be categorized in two major groups: conventional and estimation methods. The conventional methods are core analysis and well test techniques. These methods are very expensive and time consuming. However, the essential information by conventional methods can be used in estimation methods for permeability determination. The oldest method for permeability determination is empirical correlations, which related permeability with other petrophysical properties of reservoir rock such as porosity and water saturation. Although this technique seems to be an ideal tool in permeability estimation in sandstone reservoirs, it fails to predict permeability over the high permeable intervals and heterogeneous formations.

Availability of the well logging data for the most wells in any oil reservoir motivates the researcher to predict permeability from them by using the multi-linear regression and artificial intelligent approaches. In these methods for estimation of permeability, logs and cores data were used as input and target data, respectively. Intelligent systems, such as neural networks and FL, have much better solutions than multi-linear regression technique in permeability prediction. Previous investigations indicated that artificial intelligence provides powerful tools for identifying the relationship among permeability and geophysical well log data even in highly heterogeneous reservoirs with good accuracy. In using intelligent techniques, the validations of coring data are essential and coring operation must be done more accurate in wells drilled in the reservoir.

In spite of the wide range of applications, a significant amount of time and effort is being expended to find the optimum or near-optimum structure for artificial system such as ANNs or ANFIS for the desired task. To mitigate these deficiencies, design of them using optimization algorithms such as GA, PSO, and COA has been proposed. Application of the COA in different optimization problems has proven its capability to deal with difficult optimization problems, especially in multi-dimensional problems rather than other optimization algorithms (Rajabioun 2011). Moreover, zoning the reservoir according to geology characteristics and sorting the data in the same manner have been made to improve the proficiency of artificial intelligent results in permeability prediction especially in high heterogeneous reservoirs.

## 5 Conclusion

Permeability is one of the most important parameters in reservoir characterization, playing a major role in reservoir simulation, enhanced oil recovery, or well completion design. Therefore, before any field exploitation and development strategies design, this vital parameter must be determined. Core analysis provides the best fine scale permeability measurements. Nevertheless, the coring operations are very costly and time consuming and impractical to perform in all wells especially in

horizontal wells. Also, permeability is commonly determined from transient well test analyses. However, well tests yield an average permeability value for the entire drainage area of the well.

Empirical correlations were the oldest estimation method for permeability prediction, which seem to be an ideal tool for homogeneous formations that have fairly constant porosity and grain size. But the complex behavior of most reservoir parameters caused this method to not be considered as a powerful tool in permeability prediction especially in carbonate formations.

Fortunately, well log responses are widely available in any oil and gas field. When these data are properly analyzed, wireline logs have the advantage of providing continuous permeability traces as opposed to scanty, discrete core data. Artificial intelligent approaches, such as ANNs and FL, are common methods that use wireline data in permeability prediction, even in high heterogeneous reservoirs, with good accuracy. The ability of soft computing in approximating virtually any function in a stable and efficient way caused a rapid growth in recent decade in permeability prediction. ANNs have been applied more than other soft computing techniques to predict reservoir performance. But optimum design structure of neural networks can be obtained with optimization algorithm especially COA, which has fast convergence in global optima achievement than other optimization algorithms.

Also, permeability prediction based on designing separate networks for each zone of reservoir according to geology characteristics is more accurate than designing single network design for all of zones of reservoir.

## References

Abdullraheem A, Sabakhi E, Ahmed M (2007) Estimate of permeability from wireline logs in middle eastern carbonate reservoir using fuzzy logic. SPE paper 105350

Ahmed T (2001) Reservoir engineering handbook, 2nd edn. Gulf Professional Publishing, Houston

Ahmed U, Crary SF, Coates GR (1991) Permeability estimation: the various sources and their interrelationships. JPT 43(5):578–587

Al-Anazi AF, Gates ID (2010) A support vector machine algorithm to classify lithofacies and model permeability in heterogeneous reservoirs. Eng Geol 114(3–4):267–277

Al-Anazi AF, Gates ID (2012) Support vector regression to predict porosity and permeability: effect of sample size original research article. Comput Geosci 39:64–76

Ali KI, Mohammadreza R, Seyed AM (2006) A fuzzy logic approach for estimation of permeability and rock type from conventional well log data: an example from the Kangan reservoir in the Iran offshore gas field. J Geophys Eng 3:356–369

Amaefule JO, Altunbay M, Tiab D (1993) Enhanced reservoir description: using core and log data to identify hydraulic flow units and predict permeability in uncored intervals well. SPE paper 26436

Jamialahmadi M, Javadpour FG (2000) Relationship of permeability, porosity and depth using an artificial neural network. J Petrol Sci Eng 26:235–239

Aminian K, Bilgesu HI, Ameri S et al (2000) Improving the simulation of water flood performance with the use of neural networks. SPE paper 65630, pp 105–110

Aminian K, Thomas B, Bilgesu HI et al (2001) Permeability distribution prediction. In: Proceedings of the SPE eastern regional conference, Oct 2001
Aminzadeh F, Barhen J, Toomarian NB (1999) Estimation of reservoir parameter using a hybrid neural network. J Pet Sci Eng 24(1):49–56
Amyx JW, Bass DM, Whiting RL (1960) Petroleum reservoir engineering: physical properties. McGraw-Hill Book Co., New York
Arpat GB, Gumrah F, Yeten B (1998) The neighborhood approach to prediction of permeability from wireline logs and limited core plug analysis data using back-propagation artificial neural networks. J Pet Sci Eng 20:1–8
Bagheripour HM, Shabaninejad M (2011) A permeability predictive model based on hydraulic flow unit for one of iranian carbonate tight gas reservoir. SPE paper 142183
Biswas D, Suryanarayana PV, Frink PJ et al (2003) An improved model to predict reservoir characteristics during underbalanced drilling. SPE paper 84176
Bloch S (1991) Empirical prediction of porosity and permeability in sandstones. Am Assoc Petrol Geol Bull 75(7):1145
Boadu FK (1997) Rock properties and seismic attenuation: neural network analysis. Pure Appl Geophys 149:507–524
Boozarjomehry RB, Svrcek WY (2001) Automatic design of neural network structures. J Comput Chem Eng 25:1075–1088
Burbidge R, Trotter M, Buxton B et al (2001) Drug design by machine learning: support vector machines for pharmaceutical data analysis. Comput Chem 26(1):5–14
Carman PC (1937) Fluid flow through granular beds. Trans Inst Chem Eng 15:150–166
Chang HC, Kopaska-Merkel DC, Chen HC et al (2000) Lithofacies identification using multiple adaptive resonance theory neural networks and group decision expert system. J Comput Geosci 26:591–601
Chena C, Lina L (2006) A committee machine with empirical formulas for permeability prediction. Comput Geosci 32:485–496
Coates G, Denoo S (1981) The producibility answer product. Tech Rev 29(2):55–63
Coates GR, Dumanoir JL (1974) A new approach to improved log-derived permeability. Log Anal 15(1):17
Cuddy SJ, Putnam TW (1998) Litho-facies and permeability prediction from electrical logs using fuzzy logic. SPE paper 49470
Earlougher RC (1977) Advances in well test analysis, 2nd edn. Society of Petroleum Engineers of AIME, New York
Gedeon TD, Wong PM, Huang Y et al (1997) Two dimensional neural-fuzzy interpolations for spatial data. In: Proceedings of GIS geo-informatics, vol 1, Taipei, Taiwan, pp 159–166
Ghafoori MR, Roostaeian M, Sajjadiain VA (2008) A state of the art permeability modeling using fuzzy logic in a heterogenous carbonate (an iranian carbonate reservoir case study). IPTC paper 12019, Kuala Lumpur, Malaysia
Gholami R, Shahraki AR, Jamali Paghaleh M (2012) Prediction of hydrocarbon reservoirs permeability using support vector machine. Math Probl Eng. doi:10.1155/2012/670723
Hambalek N, Reinaldo G (2003) Fuzzy logic applied to lithofacies and permeability forecasting. SPE paper 81078
Horne RN (1995) Modern well test analysis: a computer-aided approach, 2nd edn. Petroway Inc, Palo Alto
Huang Z, Shimeld J, Williamson M et al (1996) Permeability prediction with artificial neural network modeling in the Venture gas field, offshore eastern Canada. Geophysics 61(2):422–436
Huange Y, Gedeonb T, Wongc P (2001) An integrated neural-fuzzy genetic-algorithm using hyper-surface membership functions to predict permeability in petroleum reservoirs. J Pet Sci Eng 14:15–21
Ilkhchi AK, Rezaee M, Moallemi SA (2006) A fuzzy logic approach for estimate of permeability and rock type from conventional well log data: an example from the Kangan reservoir in the Iran offshore gas field. J Geophys Eng 3:356–369

Jeirani Z, Mohebbi A (2006) Estimating the initial pressure, permeability and skin factor of oil reservoirs using artificial neural networks. J Petrol Sci Eng 50:11–20
Jeng JT, Chuang CC, Su SF (2003) Support vector interval regression networks for interval regression analysis. Fuzzy Sets Syst 138(2):283–300
Karimpouli S, Fathianpour N, Roohi J (2010) A new approach to improve neural networks algorithm in permeability prediction of petroleum reservoirs using supervised committee machine neural network (SCMNN). J Petrol Sci Eng 73:227–232
Kaydani H, Mohebbi A (2013) A comparison study of using optimization algorithms and artificial Neural networks for predicting permeability. J Pet Sci Eng 112:17–23
Kaydani H, Mohebbi A, Baghaie A (2012) Neural fuzzy system development for the prediction of permeability from wireline data based on fuzzy clustering. J Pet Sci Eng 30(19):2036–2045
Kaydani H, Mohebbi A, Baghaie A (2011) Permeability prediction based on reservoir zonation by a hybrid neural genetic algorithm in one of the Iranian heterogeneous oil reservoirs for permeability prediction. J Pet Sci Eng 86–87:118–126
Klir G, Yuan B (1995) Fuzzy sets and fuzzy logic: theory and applications. Prentice-Hall, Englewood Cliffs
Kozeny J (1927) Uber Kapillare Leitung des Wassers im Boot Sitzungsberichte, vol 136. Royal Academy of Science, Vienna, Paris. Class I, pp 271–306
Kumar N, Hughes N, Scott M (2000) Using well logs to infer permeability. Center for Applied Petrophysical Studies, Texas Tech University
Levorsen AI (1996) Geology of petroleum, 2nd edn. Freeman and Company Publishing, New York
Lim JS (2005) Reservoir properties determination using fuzzy logic and neural networks from well data in offshore Korea. J Pet Sci Eng 49:182–192
Malki HA, Baldwin JL, Kwari MA (1996) Estimating permeability by use of neural networks in thinly bedded shaly gas sands. SPE Comput Appl 8:58–62
Matthewe CS, Russell DG (1967) Pressure buildup and flow tests in wells. SPE, Dallas (Monograph series)
Mohaghegh S, Ameri S (1995) Artificial neural network as a valuable tool for petroleum engineers. SPE paper 29220
Mohaghegh S, Arefi R, Ameri S et al (1994) Design and development of an artificial neural network for estimation of formation permeability. SPE paper 28237
Mohaghegh S, Balan B, Ameri S (1996) State-of-the-art in permeability determination from well log data. SPE paper 30979
Mohaghegh S, Gaskari R, Popa A et al (2001) Identifying best practices in hydraulic fracturing using virtual intelligence techniques. In: Proceedings of 2001 SPE eastern regional conference and exhibition, SPE 72385, Oct 17–19, North Canton, Ohio
Mohebbi A, Kamalpour R, Keyvanloo K et al (2012) The prediction of permeability from well logging data based on reservoir zoning, using artificial neural networks in one of an Iranian heterogeneous oil reservoir. J Petrol Sci Tech 30(19):1998–2007
Molnar D, Aminian K, Ameri S (1994) The use of well log data for permeability estimation in a heterogeneous reservoir. In: Proceedings of SPE eastern regional conference, SPE 29175, pp 167–180
Dehghani SAM, Vafaie Sefti M, Ameri A (2008) Minimum miscibility pressure prediction based on a hybrid neural genetic algorithm. J Chem Eng Res 86:173–185
Nashawi IS, Malallah A (2010) Permeability prediction from wireline well logs using fuzzy logic and discriminated analysis. SPE paper 133209
Niculescu SP (2003) Artificial neural networks and genetic algorithms in QSAR. J Mol Struct Theochem 622:71–83
Norman JH (1984) Geology for petroleum drilling and production. McGraw-Hill Inc, New York
Nowroozi S, Ranjbar M, Hashemipour H et al (2009) Development of a neural fuzzy system for advanced prediction of dew point pressure in gas condensate reservoirs. J Fuel Process Technol 90:452–457

Olatunji SO, Selamat A, Abdulraheem A (2011) Modeling the permeability of carbonate reservoir using type-2 fuzzy logic systems. Comput Ind 62:147–163
Perez H, Gupta D, Misra S (2005) The role of electrofacies, lithofacies and hydraulic flow units in permeability predictions from well logs: a comparative analysis using classification trees. SPE Reservoir Eng Eval 8(2):143–155
Pirson SJ (1963) Handbook of well log analysis. Prentice-Hall Inc, Englewood Cliffs
Prasad M (1999) Correlating permeability with velocity using flow zone indicators. SEG conference Houston, Texas, paper ID 1999-0184
Prasad RS, Al-Attar EH, Al-Jasmi AK (1996) Reservoir permeability upscaling indicators from welltest analysis. SPE paper 36175
Rajabioun R (2011) Cuckoo optimization algorithm. Appl Soft Comput 11:5508–5518
Saemi M, Ahmadi M (2008) Integration of genetic algorithm and a coactive neuro-fuzzy inference system for permeability prediction from well logs data. Trans Porous Med 71:273–288
Saemi M, Ahmadi M, Yazdian A (2007) Design of neural networks using genetic algorithm for the permeability estimation of the reservoir. J Pet Sci Eng 59:97–105
Saxena A, Saad A (2006) Evolving an artificial neural network classifier for condition monitoring of rotating mechanical systems. J Appl Soft Comput 7:1568–4946
Stitson M, Gammerman A, Vapnik V et al (1999) Advances in kernel methods-support vector learning. MIT Press, Cambridge
Taghavi AA (2005) Improved permeability estimation through use of fuzzy logic in a carbonate reservoir from southwest Iran. SPE paper 93269
Tahmasebi P, Hezarkhani A (2012) A fast and independent architecture of artificial neural network for permeability prediction. J Pet Sci Eng 86–87:118–126
Tiab D, Donaldson EC (2004) Petrophysics, theory and practice of measuring reservoir rock and fluid transport properties, 2nd edn. Gulf Professional Publishing, Elsevier, USA, p 889
Timur A (1968) An investigation of permeability, porosity, and water saturation relationship for sandstone reservoirs. Log Analyst 9(4)
Tixier MP (1949) Evaluation of permeability from electric-log resistivity gradients. Oil Gas J 48:113
Trontl K, Smuc T, Pevec D (2007) Support vector regression model for the estimation of γ-ray buildup factors for multi-layer shields. Ann Nucl Energy 34(12):939–952
Uguru CI, Onyeagoro UO, Lin J et al (2005) Permeability prediction using genetic unit averages of flow zone indicators (FZIs) and neural networks. SPE paper 98828
Van Rooij AJF, Jain LC, Johnson RP (1996) Neural network training using genetic algorithms. World Scientific Publishing Co. Pvt. Ltd, Singapore
Vonk E, Jain LC, Johnson RP (1997) Automatic generation of neural network architecture using evolutionary computation. World Scientific Publishing Co. Pvt. Ltd, Singapore
Weber KJ, Van Geuns LC (1990) Framework for constructing clastic reservoir simulation model. JPT 42(10):1–248
Wiener J (1995) Predict permeability from wireline logs using neural networks. Pet Eng Int 68:18–24
Wong PM, Henderson DJ, Brooks LJ (1998) Reservoir permeability determination from well log data using artificial neural networks: an example from the Ravva field, offshore India. In: Proceedings of SPE Asia Pacific oil and gas conference, Kuala Lumpur, Malaysia, SPE paper 38034
Wyllie MRJ, Rose WD (1950) Some theoretical consideration related to quantitative evaluation of physical characteristics of reservoir rock from electrical log data. Trans AIME 189:105–118
Zadeh LA (1965) Fuzzy set. Inf Control 8:338–353
Zeng X, Singh MG (1996) Approximation accuracy analysis of fuzzy system as function approximators. IEEE Trans Fuzzy Syst 4:44–63

# Index


**A**
Abnormally pressured zones, 192
Accuracy, 14
Active learning, 2
Active learning method (ALM), 192, 225
Adaptive neural fuzzy inference system (ANFIS), 279
Agglomerative cluster analysis, 258
Alleles, 65
ALM algorithm, 228
Anadarko Basin, 192
A posteriori probabilities, 5
Applications of GP techniques, 122
Artificial neural network (ANN), 28, 128, 129, 154, 165, 221, 226, 276

**B**
Backpropagation, 129, 133
Backpropagation algorithm, 35
Bayes law, 5
Bayes classifiers, 5
Boosting, 12
Borehole geophysics, 246

**C**
Carbonate reservoir, 225
Center of gravity (COG), 211
Chromosome, 65
Classifiers, 3
Closed segment, 40
Closure property, 103
Cluster analysis, 259
Clustering, 2, 258
COA-LM neural model, 278
COG operator, 194, 231
Collective intelligence, 60
Committee machines (CM), 226
Completeness property, 104
Constraint handling, 61
Convex function, 40
Core analysis, 269, 272, 281
Correlations, 281
Crossover, 65, 69, 77
Crowding, 81
Cuckoo Optimization Algorithm (COA), 278, 281

**D**
Darcy's law, 271
Data normalization, 203
Decision trees, 10
Deep resistivity (REID), 192
Defuzzification, 231
Dendrogram, 259
Density log (RHOB), 217
Density porosity, 192
Depth-dependent probe response functions, 247
Depth-dependent response function, 250
Depth variations of petrophysical parameters, 252
Dew point pressure, 123
Differential evolution, 60, 66, 75
Diffusivity equations, 273
Dual Induction, 192

**E**
Early stopping, 233
Editing, 110
Effect of overburden, 271
Elitist replacement strategy, 78
Empirical correlations, 270, 271, 274, 282
Encapsulation, 110
Encoding, 67, 76
Encoding for candidate solutions, 67
Error rate, 14
Euclidean distance, 260
Evolutionary algorithms, 65




C. Cranganu et al. (eds.), *Artificial Intelligent Approaches in Petroleum Geosciences*,
DOI 10.1007/978-3-319-16531-8

Evolutionary programing, 66
Evolutionary strategies, 66
Experiment design, 62
Expression parse tree, 111

**F**
False negative cases, 14
False positive cases, 14
Feasability, 61
Feasible region, 44
Feed-forward artificial neural network (FF-ANN), 128
Fitness, 107
Fitness function, 65, 68, 114
Fitness sharing, 80
Flow Zone Indicator (FZI), 226, 274, 275
Forward modeling, 251
Forward problem, 247, 252, 256
Full method, 106
Fuzzification, 194, 213, 218, 228, 231
Fuzzy c-means cluster, 280
Fuzzy logic (FL), 226, 279, 281
Fuzzy modeling, 276, 279
Fuzzy set theory, 279
FZI method, 275

**G**
Gamma Ray (GR), 192
Gene expression programming, 111
Generalization, 2
Generation of GP individuals, 105
Generations, 65
Genetic Algorithms (GAs), 58, 66, 67, 277
Genetic programming, 59
Genotype, 65
GEP genes, 112
Global inverse problems, 256
Gradient descent, 137
Graphical user interface of RGP, 119
Grow method, 106

**H**
Heaviside basis functions, 254
Heuristic, 56
Hierarchical cluster analysis, 260
Hill climbing, 56
Hybridisation, 62
Hydraulic Flow Unit (HFU), 225
Hydraulic unit, 274
Hyper-heuristics, 58
Hyper-parameters, 90
Hyperplane, 39

**I**
IDS operator, 194
Imperialist Competitive Algorithms (ICA), 278
Impurity, 11
Including the depth interval, 251
Independent inversion, 251
Individual, 65
Information gain, 11
Initialization, 68
Ink Drop Spread (IDS), 225, 228
Interval inversion, 247, 250, 251
Interval inversion method, 247
Inverse modeling, 246
Inverse problem, 248
Inverse problem of borehole geophysics, 248
Inversion, 246
Island model, 81

**J**
Jacobi's, 249
Joint inversion, 248, 251, 252

**K**
K-Nearest Neighbors algorithm, 227
Karoon basin, 210
Kernel function, 22
Klinkenberg effect, 271

**L**
Layer parameters, 248
Layer-thicknesses, 248, 250, 264
Learning classifier systems, 89
Levenberg–Marquardt algorithm, 250
Levenberg–Marquardt (LM) neural network algorithm, 278
Levenberg–Marquardt learning algorithm, 221
Linear interval inversion, 256
Linearized inversion, 253
Linear regression, 24
Local inverse problem, 249
Local inversion, 247, 248

**M**
Machine learning, 1
Main GP operators, 108
Matrix, 249
Maximum a posteriori class, 5
Mean squared error, 107, 218
Meta-heuristic, 56
Meta-parameters, 90
Metropolis criteria, 257
Model selection, 90

Modular Neural Network (MNN), 278
Multigenic, 112
Multilayer ANN, 138, 141
Multi linear regression, 281
Multi-modal environments, 80
Multi-modal optimization, 81
Multi-objective optimization, 81
Multilayer perceptron training, 227
Multiple input single-output (M.I.S.O), 211
Multiple input single output (M.I.S.O.) system, 228
Mutation, 65, 70, 76, 108

**N**
Naïve Bayes classifiers, 5
Narrow paths (NP), 211, 231
Negative closed half-space, 39
Negative open half -space, 39
Neural networks, 281
Neuro-fuzzy NF (Fuzzy neural network), 221
Neutron log (NPHI), 217
Neutron porosity (NPHI), 239
Niching, 80
No Free Lunch Theorem, 54
Noise injection, 225, 227, 234
Nonlinear over-determined inverse problem, 248
Normalized mean squared error (nMSE), 228

**O**
Optimization problem, 44, 55
Over-determination ratio, 251
Over-determined inverse problem, 247
Over-fitting, 2, 141, 145, 225, 233
Over-training, 141

**P**
Parameter control, 60, 85
Parent, 65
Pareto optimal solution, 81
Particle, 83
Particles swarm, 278
Particle Swarm Optimization (PSO), 60, 82, 83
Pearson's correlation coefficients, 260
Perceptron, 28
Perceptron ANN, 138
Permeability, 269, 274, 276, 281
Permeability estimation, 270
Permutation, 109
Persian Gulf, 209
Petrophysical properties, 246, 248, 281
Phenotype, 65
Photoelectric absorption factor (PEF), 192
Photoelectric log (PEF), 217
Population, 65
Porosity, 274, 281
Porous media, 270
Positive closed half-space, 39
Positive open half-space, 39
Precision, 15
Premature convergence, 68, 85
Pressure curve, 273
Pressure transient analysis, 270
Probe response functions, 247
Problem-solving, 54
Protected functions, 104

**R**
R, 2
Ramped half-and-half method, 106
R code, 63
R package called DEoptim, 78
R package called GA, 70
R package pso, 86
Rate of penetration, 158
Regression, 24
Reinforcement learning, 2
Representation of GEP individuals, 111
Representation of individuals, 102
Reservoir characterization, 123
Response equation, 248, 249
Response functions, 248, 261
RGP, 116
Rock dataset, 119
Roulette-wheel selection, 68

**S**
Selection, 65
Selection for variation, 68, 76
Selection pressure, 68
Self-potential (SP), 192
Semi-supervised learning, 2
Sensitivity, 15
Simulated annealing (SA), 57, 257
Single input-single output (S.I.S.O.), 193
Single-input single-output (S.I.S.O.) sub-systems, 211, 228
Soil–water characteristic curve, 123
Sonic (DT), 239
Sonic log (DT), 192, 217
Specificity, 14
Splitting criterion, 11
Spring, 65
Standard symbolic regression problem, 117
Structural Risk Minimization (SRM) principle, 280

Supervised learning, 1
Support Vector Machine (SVM), 16, 226, 280
Swarm intelligence, 82
Symbolic regression, 118
Syntax trees, 103

**T**
Taylor series, 249
Testing, 144, 147
Training, 144
Training process, 145
Training set, 232
Transfer function, 32
True negative cases, 14
True positive cases, 14

**U**
Underdetermined, 248
Underfitting, 233
Unsupervised learning, 2

**V**
Validation, 144, 148
Validation error, 146
Validation set, 232
Very fast simulated re-annealing (VFSR), 258
VFSR algorithm, 264

**W**
Ward's linkage algorithm, 260
Ward's linkage criterion, 259
Water saturation, 274
Weak methods, 54
Well logs, 192
Well test analysis, 272, 273
Well testing, 273
Well testing analysis, 269

**Z**
Zone parameters, 248, 250