UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Laurea Specialistica in Ingegneria Informatica per la Gestione d'Azienda

A single-objective and a multi-objective genetic algorithm to generate accurate and interpretable fuzzy rule-based classifiers for the analysis of complex financial data

Supervisors

Prof. Francesco Marcelloni
Dipartimento di Ing. dell'Informazione
Università di Pisa

Prof.ssa Beatrice Lazzerini
Dipartimento di Ing. dell'Informazione
Università di Pisa

Candidate

Simona Palmeri

ACADEMIC YEAR 2012-2013

Table of contents

Abstract

Nowadays, organizations deal with rapidly increasing amount of data that is stored in their databases. It has therefore become of crucial importance for them to identify the necessary patterns in these large databases to turn row data into valuable and actionable information. By exploring these important datasets, the organizations gain competitive advantage against other competitors, based on the assumption that the added value of Knowledge Management Systems is first and foremost to facilitate the decision making process. Especially if we consider the importance of knowledge in the 21st century, data mining can be seen as a very effective tool to explore the essential data that foster competitive gain in a changing environment.

The overall aim of this study is to design the rule base component of a fuzzy rule-based system (FRBS) through the use of genetic algorithms. The main objective is to generate accurate and interpretable models of the data trying to overcome the existing tradeoff between accuracy and interpretability. We propose two different approaches: an accuracy-driven single-objective genetic algorithm, and a three-objective genetic algorithm that produces a Pareto front approximation, composed of classifiers with different tradeoffs between accuracy and complexity. The proposed approaches have been compared with two other systems, namely a rule selection single-objective algorithm, and a three-objective algorithm. The latter has been developed by the University of Pisa and is able to generate the rule base, while simultaneously learning the definition points of the membership functions, by taking into account both the accuracy and the interpretability of the final model.

1 Data mining as a tool for data analysis

1.1 Introduction

Our age is popularly known as the "*Information age*" due to the widespread diffusion of the information technology (IT) and the availability of cutting-edge and more effective tools for the collection and storage of data that have increasingly facilitated the access to all kind of information we need, with improving levels of accuracy and rapidity.

During the '90s, the amount of data generated and collected in all sort of fields, from business to medicine, science and so on, grew so exponentially as to make it very challenging to find new and intelligent ways to extract useful information (or knowledge) from this potential source of knowledge. [Bal, 2011].

In fact, traditional methods of data analysis (linear and multiple regression analysis, correlation analysis, principal component analysis and many other statistical techniques) have proved to be largely inadequate in achieving this goal, mainly due to an intrinsic limitation: conventional methods let the user interact directly with the data. This involves a twofold drawback: first, the user needs to know what to look for and will make use of these methods to prove it; second, the user is not able to handle properly large datasets. In this scenario, turning data into knowledge is not a trivial task and this is why alternative data mining techniques have been explored with the overall objective of responding to the new emerging needs.

The first step required to approach the study of these techniques is to provide a definition of data mining. This thesis will adopt the definition provided by Han and

Kamber [Han, 2001], namely the "*process of discovering **interesting** patterns and knowledge from large amounts of data*".

The keyword of this definition is "interesting" which means that the main aim of data mining techniques is to shed the light on non-trivial, implicit and previously unknown patterns within the data. This explains why a simple search or a query processing cannot be considered data mining.

While data mining techniques help establishing a model reflecting the data, the latter are only one ring of that chain process that, starting from raw data, generates useful knowledge. The process is known as Knowledge Discovery from Data (KDD) and consists of an iterative sequence of five steps, as described below (see figure 1.1).

1) Data ***selection***:

Data are selected among the wealth of information collected according to their relevance to the aim of the analysis, previously identified.

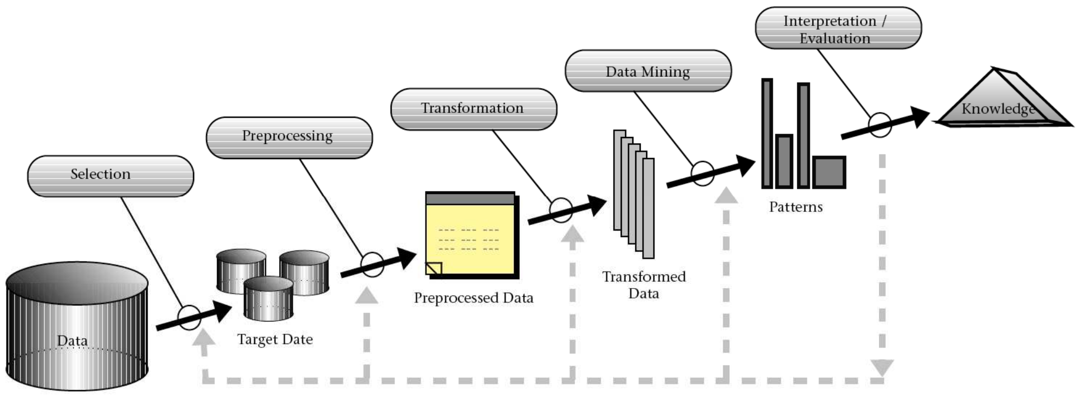


Figure 1.1: An overview of the steps that compose the KDD process [Fayyad, 1996]

2) Data ***preprocessing***:

The data stored can be affected by noise, inconsistency or be incomplete (missing data). Data processing therefore refers to the strategy adopted for dealing with these problems.

3) Data ***transformation:***

Performing complex analysis of cumbersome amount of data is computationally expensive and extremely time-consuming. Techniques of data reduction are applied to the original data in order to reduce the amount of data to a representative sample, while retaining their characteristics. More specifically, data reduction strategies operate:

- dimensionality reduction, i.e. removing the least important attributes using PCA or feature selection techniques;
- numerosity reduction i.e. removing samples using regression, histograms, clustering or sampling;
- data compression i.e. data are compressed by using, for instance, discretization techniques.

4) ***Data mining:***

At this stage, data are ready to be analysed using one of the data mining strategy to search for patterns (classification, clustering and association rules). The quality of the patterns so obtained is closely linked to the correct performance in the previous steps.

5) ***Interpretation/evaluation***

Finally, the patterns extracted are evaluated to identify the most interesting ones, which are usually stored into a knowledge base.

This study will focus on the data mining techniques. The main kind of analysis that they support are:

- ***Classification/Regression***

The aim of classification techniques is to classify the data into a set of pre-defined classes. They employ a set of labelled examples called training set, to learn and build a model (i.e. neural networks, rules, decision trees) of the data. Then, this model is used to classify new objects. Typical problems faced by classification techniques are: optical character recognition (OCR), which allows automating the document digitalization process, medical image analysis, fraud detection and so on. The classification paradigm will be described more in detail in the following.

Regression techniques are similar to classification ones, but the output variable can take continuous values instead of discrete ones.

- ***Clustering***

The aim of clustering algorithms is to group input data into subsets, called clusters. Each cluster contains data objects sharing some similarities, which are dissimilar to the data objects belonging to other clusters. These techniques are very effective only if the distribution of data objects is clustered, otherwise the model built performs poorly. Cluster analysis is widely used in many applications: information retrieval to cluster documents; customer relationship management (CRM) for the clustering of a large number of customers into few categories and then develop a common business strategy for each of them; and so on.

- ***Association***

These techniques are used to reveal association relationships and correlations among data. They are often formulated as rules in the form X −> Y which means that all data satisfying X are likely to satisfy Y. Association techniques are largely applied in the business sector, more specifically in the so called market basket analysis, whose aim is to analyse and understand the buying habits of customers.

From the above overview of the different strategies used in data mining techniques, it is clear that these can be applied to a variety of sectors, from bioinformatics to telecommunication, from business and finance to medicine.

Indeed, thanks to its wide application data mining has become one of the most interesting and popular research areas.

The paragraph that follows will give an account of the main applications of data mining in business and finance, including some operational examples.

1.2 Data mining in business and finance: motivations and examples

Organizations need to be proactive if they want to have a chance to survive in today's competitive and dynamic environment, and information plays a pivotal role in supporting the achievement of such a goal. In fact, it is only by disposing of the right information at the right time that managers can timely expand their knowledge and strengthen their decision making process, by ultimately increasing their responsiveness to the market demands.

During their daily working routine, organizations produce and store cumbersome amount of data. Filtering this data through traditional data analysis tools would entail an equally cumbersome amount of work and a time-consuming process.

Here data mining techniques reveal their strength, meaning their ability to unlock and unveil the potential hidden behind mountains of data . [Singh, 2012]. The information extracted can then be used to create new knowledge within a very short time, which in turn supports the organizations in elaborating better action-oriented strategies with the potential to generate added value in terms of customer satisfaction, internal processes performance (elimination of inefficiencies), money and time savings, increased productivity and related revenue.

This is what in literature is usually defined the virtuous cycle of data mining [Berry, 1997 in Bal, 2011]. Briefly, the information acquired through data mining give organizations a competitive advantage against their competitors and, by strengthening the knowledge management process, become a critical success factor. [Bal, 2011].

Some of the most common applications of data mining techniques in financial area are: stock forecasting, prediction of bankruptcy and detections of financial fraud. In the following sections these areas of application are analysed more in detail.

The forecast of market stocks is very important for investors, since their priority is to buy and sell at the most convenient moment. However, the non-linearity of the price variation makes it almost impossible to foresee when this is going to happen. It follows that traditional analysis techniques like regressions, producing only linear models, are not the best placed to forecast future trends. The most used technique to deal with these kind of problems is neural network modelling. [Zhang, 2004]

The prediction of bankruptcy is an important issue in finance, because it can prevent huge economic losses for companies, stockholders and also employees. In this field, it is of utmost importance to rely on a model for prediction as well as to dispose of a good-enough explanation of how this prediction has been reached. The main techniques used in this field include genetic algorithms and decision trees for classification and regression problems. [Zhang, 2004]

Another big issue of our time is credit card frauds detection. The number of credit card transactions increases every day, therefore is more and more important to prevent this misconduct. The task is not easy , first of all due to the nature of the problem: data distribution is highly skewed because of the huge amount of valid transactions against the slim number of fraudulent transactions. Secondly, it is not easy to deal with such large datasets. The main data mining techniques used to face these problems are neural networks and classifier systems specialized in the recognition of the fraudulent pattern. [Zhang, 2004].

The following paragraph will explore the main features of financial data, by highlighting the most common drawbacks that emerge when dealing with this kind of data.

1.3 Main issues in financial data analysis

In the financial sector, the analysis of data can be particularly onerous, as emphasized by the study on the main applications of data mining in this field. This is due to several factors. First of all, financial data are often in the form of **non-linear** time series. A time series is a sequence of data that represents values of a real variable measured at equal temporal intervals, such as daily, weekly, monthly and so on [Zhang, 2004]. In order to perform an analysis of a time series, the sequence of the data is important. The primary purpose of the analysis is to predict the future trend of the phenomenon under observation. The main characteristic of financial time series is their nonlinearity, which represents a big issue in the analysis task. In fact, traditional tools of analysis, such as correlation and linear regressions, can predict only linear patterns [Taylor, 1986].

Another important issue is represented by the **unbalanced** frequencies of financial data. If we recall at the example of fraud detection, it is clear that there are a huge number of legitimate transactions and only a few of fraudulent ones. Thus, the training set used to create the model has a skewed distribution of the data which makes it difficult both the setting up of the model and its evaluation. In fact, it is clear that missing a fraud is more expensive than a false alarm, so the cost per error is not uniform and this should be taken into consideration in the evaluation stage. How to handle unbalanced datasets is an open research area and some approaches from literature will be discussed in chapter 3.

In addition, datasets used for financial applications are usually huge and with lots of attributes. Statisticians usually consider the data as a matrix, whose rows represent the different observations and the columns the variables measured. For instance, in credit card frauds detection, transactions of a lot of customers are recorded and, for each of them, several variables are available. Or if we consider the forecast of exchange rates, the columns could represent the various currencies, while the rows the time sampling which should refer to a long period of time to obtain an accurate prediction. Data matrixes with lots of records and attributes are called **large** and **high dimensional** datasets. A systematic search in a high dimensional space is highly time demanding and requires a discrete amount of computational resources. For instance, consider the optimization problem of a function over a continuous space of 10 variables and suppose to face this problem by discretizing the search space using a grid of ten intervals for each variable. In this case, we should consider 10^{10} different points. If we increase the number of dimensions to 20, then we would have 10^{20} points and, generally speaking, let n be the number of dimensions, 10^{n} points. Therefore, linear increasing of the number of dimensions results in an exponential growth of the search space. This is known as the "*curse of dimensionality*" problem. [Bellman, 1961 in Donoho, 2000]. Then, in order to reduce the complexity of the task, techniques of dimensionality reduction are often used. The main idea underlying these techniques is that only some of the variables measured are responsible for the overall behaviour. One of the techniques employed to reduce the number of dimensions is the Principal Component Analysis (PCA), which is based on the concepts of covariance matrix and eigenvectors. It aims at transforming the data by projecting them onto a lower-dimensional space that preserves as much data variation as possible.

Thus, the techniques used to analyse large and high dimensional datasets should take scalability and efficiency issues into account. Dealing with high dimensional datasets is one of the most challenging areas of research.

Moreover, once the model is built, its **interpretability** is of great value. In fact, as mentioned before, the final goal of the application of the various techniques of data analysis is to extract useful knowledge in order to ease the decision making process. It is of crucial importance that the model can be easily understood and interpreted in order to be trusted. The flip side of the coin is that interpretability, being a subjective concept, makes it not easy to find a measure to assess it.

Last but not least, another characteristic of financial data is their **uncertainty**. Decisions in the financial field are usually related to risks, such as credit, liquidity risks and input data that are often inaccurate. Thus, uncertainty needs to be part of the model. Although traditionally the probabilistic approach has been chosen to address this problem, such approach is often too difficult to handle. [Guerra, 2012]

1.4 Approaches to financial data analysis issues

A variety of techniques for data analysis have been used in financial applications. The aim of this paragraph is to provide an overview of the most common approaches identified and described in the literature.

As stated by [Robles-Granda, 2010], in the area of financial data analysis many studies demonstrate that machine learning techniques often outperform classical statistical methods, especially in the classification task. For this reason, the first section of this study will focus on neural networks modelling, which is considered the classical machine learning approach in financial data analysis. Then, rule induction methods will be investigated. Finally, the last section will approach a comparative analysis of these techniques in order to stress strength and weakness of each of them.

1.4.1 Artificial neural networks

Artificial neural networks are mathematical models that try to mimic the structure of the human brain. The basic element of the brain is the neuron, a special type of cell with the capacity to link to other neurons. The incredible number of connections that may be established among neurons is the power of the human brain. Similarly, the artificial neural networks consist of a set of elementary processing elements called artificial neurons whose structure is shown in the figure below.

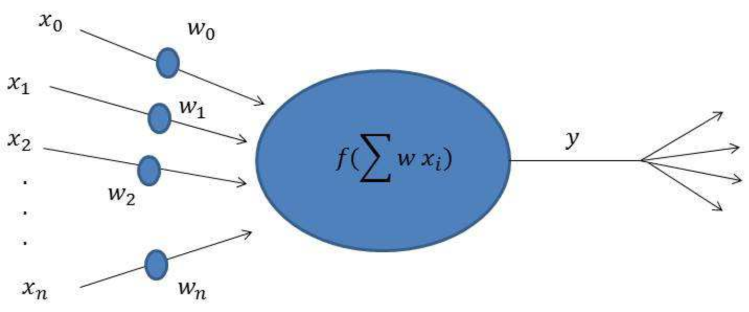


Figure 1.2: Artificial neurons structure

As illustrated in figure 1.2, a neuron has n input variables and each of them has a connectivity coefficient called weight which is a real number representing the strength of the connection. The output y of the neuron is calculated by applying a transfer function f to the weighted sum of the inputs.

$$y = f(a) = f\left(\sum_{i=0}^{n} w_i x_i\right) \tag{1.1}$$

Some of the most used transfer functions are: linear, Sigmoid, step and piecewise linear. Each neuron receives the input from many neurons and produce a single output which is communicated to other neurons. The connections among this simple units create a network with a precise topology. There are different architectures, the most common is the multilayered feed-forward, illustrated in figure 1.3.

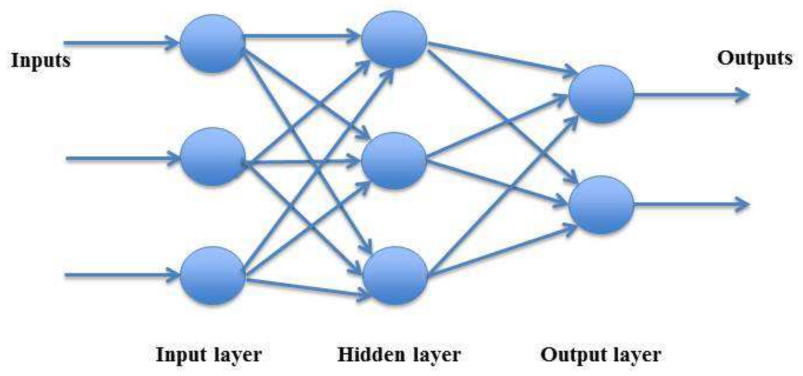


Figure 1.3: Multilayer feed-forward neural network

This kind of networks consists of three layers: input, hidden (one or more) and output layer. Each neuron of a layer is connected to all the neurons of the following layer, while there are not connections among neurons in the same level. The number of layers and the number of neurons for each layer depend on the specific problem to be solved. Starting from a defined structure, training samples are presented to the network and the weights of the connections are iteratively adjusted in order to produce the desired output. In this way, artificial neural networks can learn from the input data and approximate any function, both linear and nonlinear. Many algorithms to perform the learning task have been developed and the most famous one is called backpropagation. It is a supervised learning algorithm used for classification and numeric prediction problems. In the former task the output of the network is a class label, while in the latter is a continuous value. Typically, starting from a random initialization of the weights, a dataset of training examples is presented to the network and the final output for each input data is calculated. Then, the mean squared error is computed by comparing the prediction of the network with the actual known target value. The weights of the connections are adjusted, in order to minimize the overall error, by propagating it backward, from the output layer to the input one. The process is iteratively repeated unless a convergence is reached, i.e. the error on the training set is less than a pre specified threshold. Then, the training phase is over and the model created can be evaluated against previously unseen examples, called test set [Han, 2001].

Artificial neural networks have been the most used technique in the field of financial data analysis because of their ability to create a nonlinear input-output mapping which can model very complex behaviours.

To sum up, ***advantages*** of artificial neural networks include:

- Nonlinear mapping ability
- high tolerance of noisy data
- well suited to handle with continuous values output (unlike decision trees algorithms).

On the other hand, neural networks have also many ***disadvantages*** which include:

- long time to build the model
- lots of empirical parameters, such as number of layers, number of neurons for each layer, best transfer function and so on
- black box model, then poor interpretability

The main drawback of neural networks is their poor interpretability, due to their knowledge representation. They can be considered like black boxes, which means that users can be aware of inputs and outputs, but they cannot understand how the model works because it is difficult to figure out a clear interpretation of what weights and hidden layers represent. [Han, 2001]

1.4.2 Decision trees

Decision tree induction algorithms aim to generate tree structures, called decision trees which represent the knowledge acquired directly from the data. An example of decision tree is shown in the following figure. In this example, the task of the analysis is to predict whether a customer is likely to buy a computer or not, considering his age, credit rating and employment.

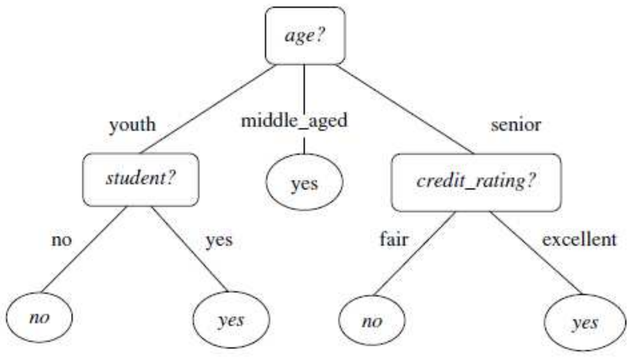


Figure 1.4: Decision tree [Han, 2001]

Generally speaking, the leaves of the tree represent the class labels, while the other nodes denote a test on an attribute. Each branch represents one of the possible outcomes of the test. The tree is constructed using a top down strategy, then starting from the root, at each step, one of the attributes is selected and the training set is partitioned according to the split criterion. A greedy approach is used to choose the attribute for the split, in fact, at each stage the attribute that can generate the purer partitions of the training set is chosen. A partition is considered pure if all the examples belonging to it have the same class label and statistical measures (information gain, gain ratio or Gini index) are used to determine the degree of purity of the partitions . When a pure partition is generated a leaf is added to the tree. The most common algorithms used to generate decision trees are CART and C4.5. [Han, 2001]

Decision trees are widely used in many application areas, such as finance, medicine, biology, to solve classification problems. Their main advantage is the possibility to extract rules which clearly explain how the label has been assigned to the examples, that is how the classification task is performed. More in depth, one rule can be extracted following a path from the root to a leaf node. It is clear that decision trees can be considered a white box because they allow to know exactly how the classification task has been performed.

The main ***advantages*** of decision trees include:

- to construct a decision tree there is no need to set parameters or have domain knowledge
- they can handle high dimensional datasets
- the learning process is simple and fast but also accurate
- white box approach

On the other hand, decision trees have also some ***drawbacks***:

- they perform well only if there is a minimum number of relevant attributes, otherwise repetition and replication issues are likely to occur. Since the tree is constructed using a greedy approach, an attribute can be tested more than once in the same branch (repetition) or equal sub-trees can be generated (replication).

- they are not scalable, so they are inefficient with large datasets. This is why, if the training set is too big to fit in memory, frequently swaps are necessary
- they can be deeply influenced by noise and outliers in the training set.

1.5 Discussion

In this chapter two of the most common techniques to analyse data for classification purposes have been studied. Now we want to investigate their ability to deal with the main issues of financial data analysis outlined in paragraph 1.3.

The table below shows a brief summary of the comparison between the two techniques.

	Neural networks	**Decision trees**
Nonlinearity	Good	Good
Unbalanced	Good	Bad
Scalability	Good	Bad
High dimensionality	Good	Bad
Interpretability	Bad	Good
Uncertainty	**Bad**	**Bad**

Table 1.1: Neural networks versus decision trees

As the table above shows, the two methods are very different, particularly from an interpretability point of view. In fact, neural networks can be considered black box models because they do not give explanations of how the results are generated. On the contrary, decision trees are white box models because following the paths within the tree it is possible to have a clear idea of the reasoning process generating the results.

Since the aim of the financial data analysis is to support the decision making process, interpretability is a key issue. Therefore it is advisable to use white box systems in order to gain decision makers' trust on the model constructed.

Finally, the main weakness of both systems resides in their limited capacities in dealing with the uncertainty which is an intrinsic feature of financial data. Moreover models should not only handle the uncertainty but also include it in the model.

These findings suggest that other kind of systems, namely the fuzzy rule based classifiers, should be used in order to benefit from their strengths while at the same time overcoming their drawbacks.

Fuzzy rule based classifiers create a model made up of simple ***interpretable*** rules and take into account the ***uncertainty*** of data included in the model through the application of the fuzzy sets theory.

2 Fuzzy Logic Theory

2.1 Introduction

In this chapter the basic principles of fuzzy logic theory will be discussed and, more specifically, the capacity of this framework to introduce uncertainty into the models will be clarified.

The theory of fuzzy sets was proposed by Zadeh back in 1965 [Zadeh, 1965]. The aim of this theory is to create a model that simulates human reasoning processes in solving complex problems. The main concept is that humans do not think using numbers, but labels of fuzzy sets, which are defined as "classes of objects in which transition from membership to non-membership is gradual rather than abrupt". [Zadeh, 1973] Furthermore, the human reasoning is not Boolean, but fuzzy. To make this concept clear, we can think, for example, at the temperature in a room. If we ask people in the room to find a common value for which temperature can be considered high, they will never agree on a precise value. Moreover, suppose that we can consider 25°C as high temperature, this does not mean that other values just below that temperature, e.g. 24°C are not high. According to our natural way of reasoning, 24°C has a lower degree of membership to the sets of high temperatures than 25°C.

In a nutshell, fuzzy logic allows to model the vagueness of human thoughts. This fuzziness plays a fundamental role in our ability to ***summarize*** information, i.e. to extract from a massive quantity of data only the relevant information for the task to be performed. [Zadeh, 1973]. The inclusion of this vagueness and the "***subjective***

knowledge" that it implies would positively affect the decision making processes. [Zadeh, 1973]. .

It is worth specifying that fuzzy logic theory is grounded on crisp logic and, starting from its principles, it tries to introduce a degree of uncertainty. Some notions on the crisp sets and logic need to be introduced and clarified before exploring more in detail the fuzzy counterparts and the main components of a fuzzy logic system.

2.2 Crisp sets

A crisp set can be defined as a collection of elements in a universe of discourse. According to this definition we can identify a crisp set by listing all the elements belonging to it or by specifying the condition satisfied by all its elements. Thus, the two following definitions of a crisp set A can be used:

$$\mathrm{A} = \{\, 1, 2, 3,4,5 \,\} \tag{2.1}$$

$$\mathrm{A} = \{\, \mathrm{x} \mid \mathrm{x} > 2 \,\} \tag{2.2}$$

We can also define a characteristic function (or membership function) $\mu_A(x)$ which indicates whether an element x belongs to A. More in detail the membership function can assume only the values specified in (2.3) with the meaning in (2.4).

$$\mu_A(x) \in \{0,1\} \tag{2.3}$$

$$\mu_A(x) = \begin{cases} 1 \ if \ x \in A \\ 0 \ if \ x \notin A \end{cases} \tag{2.4}$$

The basic operations that can be performed on crisp sets are union, intersection and complement. Let A and B be two crisp sets. We can define these operations as in the following:

- Union:

$$A \cup B = \{x | x \in A \boldsymbol{OR} \; x \in B\} \tag{2.5}$$

- Intersection:

$$A \cap B = \{x | x \in A \boldsymbol{AND} \; x \in B\} \tag{2.6}$$

- Complement:

$$\overline{A} = \{x | x \notin A\} \tag{2.7}$$

Finally, we can define the operations on crisp sets in terms of their membership functions as in (2.8), (2.9), (2.10).

- Union:

$$\mu_{A \cup B}(x) = \mathbf{max}[\mu_A(x), \mu_B(x)] \tag{2.8}$$

- Intersection:

$$\mu_{A \cap B}(x) = \boldsymbol{min}[\mu_A(x), \mu_B(x)] \tag{2.9}$$

- Complement:

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x) \qquad (2.10)$$

These formulas have been introduced to frame and support the following discussion on fuzzy sets.

2.3 Fuzzy sets

Fuzzy set can be considered a generalization of the crisp set. A fuzzy set F is characterized by a membership function which, unlike crisp sets, can take continuous values in the interval [0,1]. This values specify the degree of membership of the element to the fuzzy set. Thus, a fuzzy set F in the universe U can be represented as a set of ordered pairs of a generic element x and its degree of membership (2.11).

$$\mathrm{F} = \{\mathrm{x}, \mu_\mathrm{F}(\mathrm{x}) \mid \mathrm{x} \in \mathrm{U}\} \qquad (2.11)$$

The basic elements of the fuzzy set theory are linguistic variables and membership functions.

2.3.1 Linguistic variables

As stated before, humans don't think in terms of numbers but using more vague elements such as words or sentences. For this reason, linguistic variables have been introduced. A linguistic variable is a variable that can assume words as values. It can be decomposed into a set of terms which cover the entire universe of discourse. Formally, a linguistic variable is characterized by a quintuple: the name of the variable x, the universe of discourse U, the set of terms in which the variable is divided $T(x)$, a syntactic rule G for generating the names of values of x and a semantic rule M for associating its meaning to each value. For instance, let consider the linguistic variable *Temperature*, it can be decomposed into the following set of terms: $T(Temperature) = \{low, medium, high\}$. The next step is to associate a membership function with each term.

2.3.2 Membership functions

The main difference between crisp and fuzzy sets, is that the latter use continuous membership functions. Formally, a membership function is defined as a real values function defined in the interval [0,1]. Let be A a linguistic term, the membership function allows to associate with each point in the universe of discourse a real number in the interval [0,1] which represents the degree of membership of that point to A. Different shapes can be used to model this relationship, the most common are triangular, trapezoidal, Gaussian and singleton (see figure 2.1 (a), (b), (c), (d) respectively).

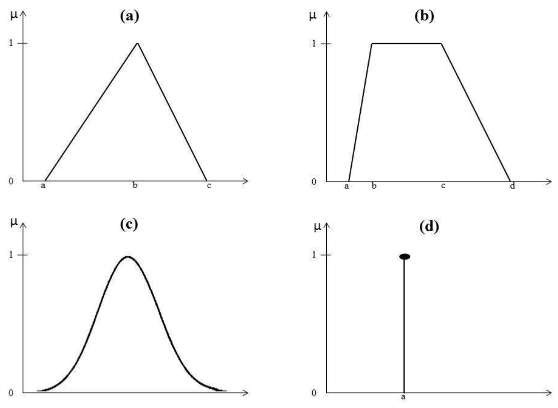


Figure 2.1: Common shapes for membership functions

Going back to the previous example, a linguistic label *Temperature* could be divided into the following terms, each of them having a membership function as reported in the figure below.

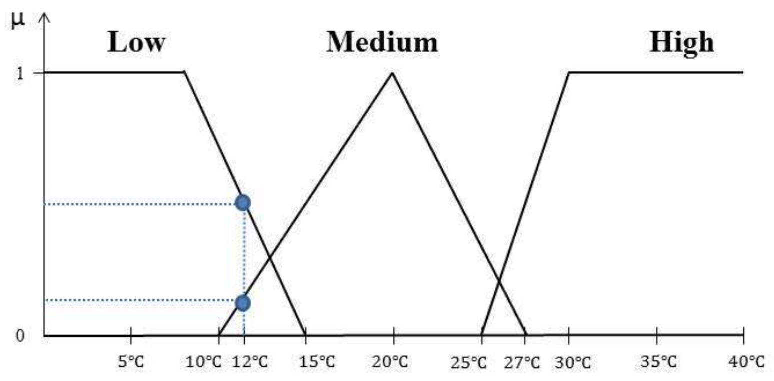


Figure 2.2: Membership functions of the linguistic labels of the variable Temperature

In this case, we interpret a value of temperature as "Low" if it belongs to the interval [0°C, 15°C] , "Medium" if it is in the interval [10°C, 27°C] and "High" if it is in the interval [25°C, 40°C]. Although it is not compulsory, the main strength of this representation is that membership functions can be partially overlapped in order to let some values be part of more than one of the fuzzy sets. For instance, if we consider the value 12°C, it is included both in the "Low" and "Medium" sets, but with different degrees of membership. In this way, the vagueness is represented and incorporated in the model.

2.3.3 Operations

As already done with the crisp sets, we can define union, intersection and complement for fuzzy sets in terms of their membership functions as follows:

- Union:

$$\mu_{A\cup B}(x) = \mathbf{max}[\mu_A(x), \mu_B(x)] \qquad (2.12)$$

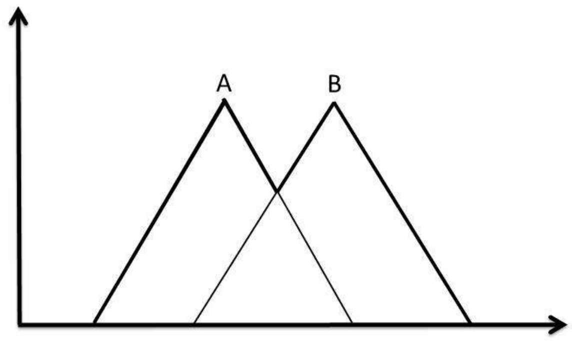


Figure 2.3: Union

- Intersection:

$$\mu_{A\cap B}(x) = \boldsymbol{min}[\mu_A(x), \mu_B(x)] \qquad (2.13)$$

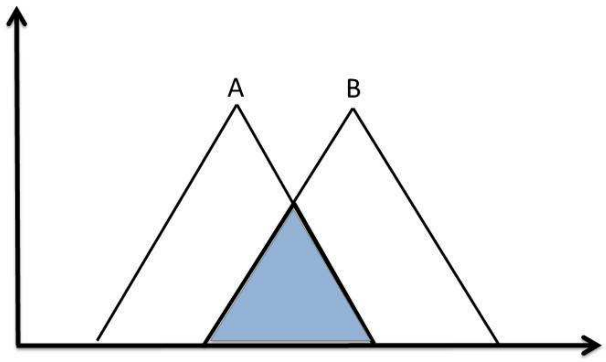


Figure 2.4: Intersection

- Complement:

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \tag{2.14}$$

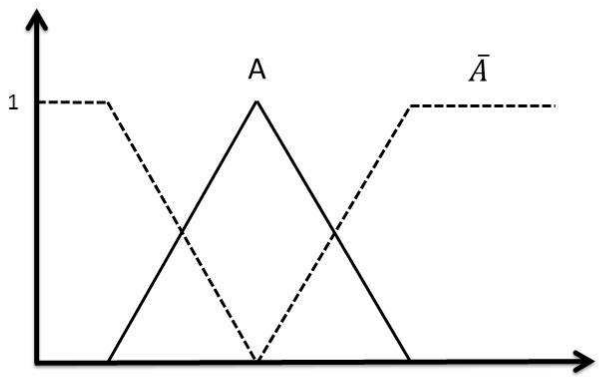


Figure 2.5: Complement

Different kind of operators can be employed, instead of maximum and minimum, to model union and intersection between fuzzy sets. The operators used for the union are called t-conorms, while the ones for the intersection t-norms.

In engineering applications the most common operations are minimum or product as t-norm and maximum as t-conorm.

2.3.4 Fuzzy relations and compositions

The remaining basic elements that need to be addressed to complete the understanding of fuzzy logic systems are fuzzy relations and compositions.

The first represents "a *degree* of presence or absence of association or interaction between the elements of two or more fuzzy sets". [Mendel, 1995]

Let U and V be two universes of discourse, then the fuzzy relation $R(U,V)$ is a fuzzy set in the product space $U \times V$ that can be expressed graphically as in the next figure.

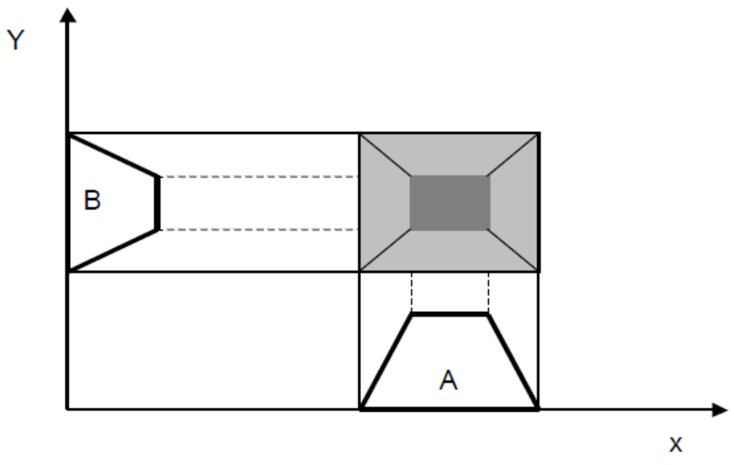


Figure 2.6: Fuzzy relation [Lazzerini, 2009]

If $x \in U$ and $y \in V$, then the membership function of the relationship R can be expressed as:

$$\mu_R(x,y) = \mu_{U\times V}(x,y) = min\,(\mu_U\,(x), \mu_V\,(x)) \qquad (2.15)$$

Since fuzzy relations are fuzzy sets, it is possible to apply the operations of union and intersection previously defined. Let assume $R(x,y)$ and $S(x,y)$ two fuzzy relations in the same product space U x V. The intersection and union of R and S can be defined as:

$$\mu_{R\cap S}(x, y) = \min(\mu_R(x, y), \mu_S(x, y)) \quad (2.16)$$

$$\mu_{R\cup S}(x, y) = \max(\mu_R(x, y), \mu_S(x, y) \quad (2.17)$$

Note that the minimum and maximum operations can be replaced, respectively, by any T-norm or T-conorm operators.

Moreover, it is possible to compose fuzzy relations from different spaces. The definition is the same as the crisp composition, except that in this case the sets are fuzzy sets. Then, let R(X,Y) and S(Y,Z) be two fuzzy relations, the fuzzy composition W(X,Z) is denoted by (2.18) and defined by its membership function $\mu_{R\circ S}(x, z)$.

$$W(X, Z) = R(X, Y) \circ S(Y, Z) \quad (2.18)$$

The membership function of the composition can be calculated as in (2.19) or (2.20):

- Max-Min composition:

$$\mu_{R\circ S}(x, z) = \max_{y\in Y} [min(\mu_R(x, y), \mu_S(y, z))] \quad (2.19)$$

- Max-Product composition:

$$\mu_{R\times S}(x, z) = \max_{y\in Y} [\mu_P(x, y), \mu_Q(y, z)] \quad (2.20)$$

2.4 Crisp logic

Conventional logic is based on propositions. According to [Mendel, 1995], a proposition is “a statement involving terms which have been defined”, i.e. “It is raining”. A proposition must be true or false, i.e. it is always possible to assign a truth value to it. Classical reasoning is based on combinations of propositions. Some of the operators used to combine them are:

- ***Conjunction***:

“It’s raining” AND “the temperature is low”

The whole proposition is true only if both propositions are true.

- ***Disjunction***:

“It’s raining” OR “the temperature is low”

The whole proposition is true if at least one of the proposition is true.

- ***Implication***:

It is usually formulate as IF-THEN rules, such as:

IF “it is raining” THEN “I will take my umbrella”

The IF part is called *antecedent,* while the THEN part is called *consequent*.

For each operation we can define a truth table which is a convenient way to summarize the relationships between several propositions. The table 2.1 shows the truth table of the operations mentioned above.

p	q	$p \wedge q$	$p \vee q$	$p \to q$
T	T	T	T	T
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Table 2.1: Truth table of "and", "or" and "implication" in crisp logic

Finally, crisp logic uses two main types of inference rules. Let P and Q be two propositions:

P : "It's raining"

Q : "I will take my umbrella"

- ***Modus Ponens***:

If P implies Q and P is true, then Q is true, i.e.

"IF it's raining THEN I will take my umbrella" $(P -> Q)$

"It's raining" (P)

Therefore, "I will take my umbrella". (Q)

- ***Modus Tollens:***

If P implies Q and Q is false, then P is false, i.e.

"IF it's raining THEN I will take my umbrella" $(P -> Q)$

"I will not take my umbrella" $(\overline{Q})$

Therefore, "It's not raining" $(\overline{P})$

2.5 Fuzzy logic

Classical logic has a direct correspondence with the Boolean algebra. [Mendel, 1995] The uncertainty in the reasoning process is introduced by expanding the concepts previously used to describe the crisp logic. It is worth noting that, since fuzzy sets are described by continuous membership functions, the truth values of a proposition will assume real values in the interval [0,1].

As an example, consider the variable *Temperature* defined by the linguistic labels in figure 2.2 and the proposition "The temperature is low". Given a value in the universe of discourse, suppose 12°C, the truth value of the proposition is not 0 or 1, but for example 0.5.

More in general, let now consider two propositions P and Q defined as follows:

P : "x is A"

$$Q : \text{“}y \text{ is } B\text{”}$$

where A and B are two fuzzy sets and $\mu_A(x)$ and $\mu_B(y)$ are respectively the truth values of P and Q obtained as in the previous example. Like in the crisp case, we can combine propositions using the operators previously declared. The result of the operations is calculated based on the truth values of the propositions.

- ***Conjunction:***

$$\mu_{P\wedge Q} = \mathbf{min}[\mu_A(x), \mu_B(y)] \tag{2.21}$$

- ***Disjunction:***

$$\mu_{P\vee Q} = \mathbf{max}[\mu_A(x), \mu_B(y)] \tag{2.22}$$

- ***Implication:***

$$\mu_{P\to Q} = \boldsymbol{I}[\mu_A(x), \mu_B(y)] \tag{2.23}$$

where ***I*** represents the implication operator. It can be seen as a relation between the fuzzy sets A and B, whose result is a fuzzy set characterized by a membership function indicated by (2.23). It can be expressed in many different ways. The most used are the “Mamdani implication” and the “product implication”, that can be expressed as in (2.24) and (2.25) respectively.

$$\mu_{P\to Q} = \mathbf{min}[\mu_A(x), \mu_B(y)] \tag{2.24}$$

$$\mu_{P\to Q} = \mu_A(x)\mu_B(y) \tag{2.25}$$

Finally, we can expand the classical Modus Ponens to the fuzzy case by defining the ***Generalized Modus Ponens***. Let consider two fuzzy sets A and B, a conditional proposition can be expressed in the form:

“IF x is A, THEN y is B”

where $x \in A$ and $y \in B$.

If we consider the propositions:

P: “x is A*”

Q: “IF x is A, THEN y is B”

The Generalized Modus Ponens structure is:

“x is A*” (P')

“IF x is A, THEN y is B” (P −> Q)

Therefore, “y is B*” (Q')

Note that A* can be different from A, then, giving a new antecedent A*, it is possible to get a new consequent B*. To make this clear, let consider the following example from

[Mendel, 1995]. Let A be the set of short men, B the set of not professional basketball players and the following rule:

"IF a man is short THEN he will not be a professional basketball player"

The Generalized Modus Ponens allows to use as first premise:

"This man is under 170 cm tall" ($A*$)

And obtain the following consequence:

"he is not a good professional player" ($B*$)

Then, $A*$ is the set of men under 170cm tall which is similar to A, but clearly not the same. To sum up, starting from a premise that is similar to the rule's antecedents, it is possible to get a consequent which is similar, although not exactly the same, to the rule's consequent. This result can be obtained mathematically by considering that the implication is a relation, therefore it is possible to combine it with a fuzzy set A* other than the antecedent of the rule and obtain a different output set, B* in the example. Then, we can write:

$$B* = A* \circ R$$

$$B*(y) = \sup T[A*(x), I(A(x), B(y))]$$

where I is one of the implication operators (minimum or product) and T is a t-conorm (usually maximum).

2.6 Fuzzy logic systems

A fuzzy logic system (FLS) is a “nonlinear system that maps an input data (feature) vector into a scalar output” [Mendel, 1995]. Like artificial neural networks, fuzzy systems can be considered universal function approximators. As such, they have been successfully employed in a wide variety of applications like nonlinear time series forecasting, controllers and so on so forth.

As shown in figure 2.7 fuzzy logic systems consist of four components:

- Fuzzifier
- Knowledge base
- Inference engine
- Defuzzifier

The functioning of a FLS will be explained by providing an overview of the general. architecture. The focus will then shift to the building blocks of a FLS for a more fine-grained analysis.

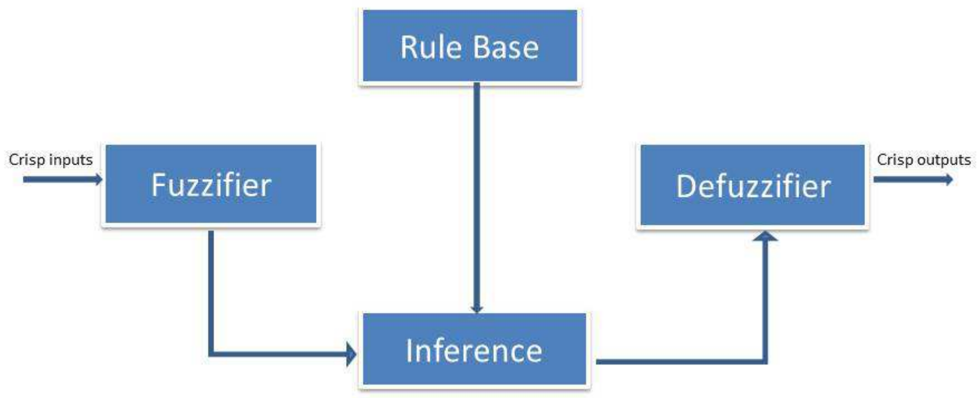


Figure 2.7: Architecture of a FLS

The input of a FLS is a crisp value, which is converted through the fuzzifier into a fuzzy set. Then, a fuzzy reasoning is performed using the set of rules included into the knowledge base. Finally, the resulting fuzzy output is transformed again into a crisp value during the defuzzification step.

2.6.1 Fuzzification

The crisp nature of the observed data in many applications has implications when it comes to the introduction of uncertainty in the model. This will in fact require the conversion of a crisp number into a fuzzy set.

The step just described is called fuzzification. The most widely used fuzzifier is the so called *singleton* fuzzifier, which maps a crisp number into a fuzzy set whose support is a single point.

When data are corrupted by measurement noise, the application of nonsingleton fuzzification can be more appropriate. In this case, a crisp value is mapped into a fuzzy membership function, whose shape is usually triangular or Gaussian.

2.6.2 Knowledge base

The knowledge base is made of two different components, the database and the rule base. The former contains the membership functions, while the latter includes a set of fuzzy rules. The choice of the membership functions, both in terms of shape and number, is subjective and context-dependent and can heavily influence the success of the model so created. This is why, many different approaches have been developed in literature in order to learn the best configuration for the problem at stake. Some of them include the use of evolutionary algorithms and will be discussed in chapter 4 .

With reference to the rules, they are the heart of a FLS. The fuzzy rule base consists of a collection of IF-THEN rules which can be implemented by fuzzy conditional statements. A fuzzy rule can be expressed as:

$$R^{(l)}: IF\ u_1\ is\ F_1^l\ AND\ u_2\ is\ F_2^l\ \ AND \dots\ u_p\ is\ F_p^l\ \ THEN\ v\ is\ G^l \qquad (2.26)$$

where $l = 1, 2, \dots, M,$ with M number of rules. F_i^l and G^l are fuzzy sets in $U_i \subset \mathbb{R}$ and $V \subset \mathbb{R}$ respectively. Finally $u = col(u_1, \dots, u_p) \in U_1, \dots, U_p\ and\ v \in V$, are linguistic variables.

Since the rules represent the knowledge, they are crucial for the success of the approximate reasoning process. Many different approaches to learn rules have been proposed in the literature. See for instance [Wang, 1992(b)] an algorithm for the extraction of the rules, starting directly from the data, is proposed. Other techniques include the use of rules provided by domain experts or their generation using evolutionary algorithms. The latter approach will be discussed in detail in chapter 4.

2.6.3 Inference engine

The inference engine implements the approximate reasoning by using fuzzy logic principles discussed in the previous sections. It employs fuzzy rules included in the rule base to determine the output of the system. As mentioned before, an IF-THEN rule can be interpreted as a fuzzy implication, then the result is a fuzzy set whose membership function is usually computed using the minimum or the product operator. Since the rules can have multiple antecedents, in this case, the input is not a single value but a vector.

Note that, since a rule is fired when the degree of membership of the premise to the antecedents of the rule is not zero, more than one rule is usually fired by the same input vector. This is why, a strategy to combine the output of the fired rules has to be decided.

In order to perform an approximate reasoning, some parameters have to be decided. First of all, the implication operator (a T-norm), then how model the logical connective "and" (a T-norm) and finally, how to aggregate the output of all the fired rules (a T-conorm).

Let assume to have a system with two inputs and a single output. It can be described by the two following rules:

$$R_1: IF\ x_1\ is\ A_{11}\ AND\ x_2\ is\ A_{12}\ THEN\ y\ is\ B_1$$

$$\mathrm{R}_2: \mathrm{IF\ x}_1\ \mathrm{is\ A}_{21}\ \mathrm{AND\ x}_2\ \mathrm{is\ A}_{22}\ \mathrm{THEN\ y\ is\ B}_2$$

where x_1, x_2 are the crisps inputs.

The figure 2.8 shows a graphical representation of the inference procedure. The steps to perform the approximate reasoning are:

1. Fuzzify the crisp inputs, i.e. compute the degree of membership of each input to the fuzzy sets which represent the antecedents of the rule. Typically fuzzy singleton are used to this purpose.

2. Apply the t-norm operator chosen to implement the “and” connective to all the antecedents of the rule.

3. Apply the t-norm operator chosen to implement the implication. If the minimum is used both in step 2 and 3, this means to cut the membership function of the fuzzy set specified by the consequent of the rule at the height corresponding to the minimum value between the degrees of membership of the inputs to the antecedents of the rule.

4. Aggregate the consequent fuzzy sets of the fired rules obtained in the previous step using a t-conorm.

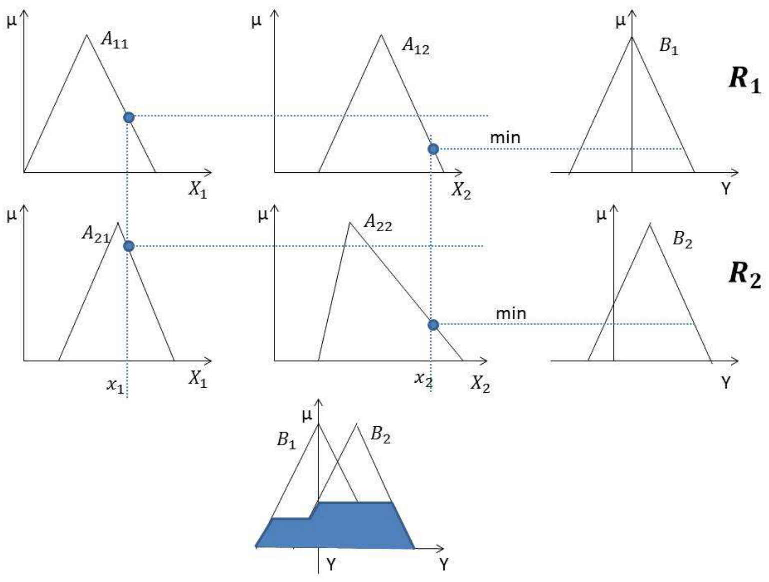


Figure 2.8: Fuzzy inference using minimum for "and" connective and for the implication and maximum for the aggregation.

2.6.4 Defuzzifier

In order to obtain a crisp output value from the FLS, the last step is the defuzzification. Many defuzzifiers have been proposed in the literature, but there are no scientific basis for any of them, consequently the defuzzification can be considered an art rather than a science. [Mendel, 1995].

Usually the computational simplicity is the criterion used to choose a defuzzifier. The most used defuzzifiers are:

- Maximum defuzzifier:

It chooses as output value the point y for which $\mu_B(y)$ is a maximum. This method can lead to interpretability problems when more than a single point reach the maximum value.

- Mean of maxima defuzzifier:

To overcome the problems of the previous method, all the points in which the output set assumes the maximum value are considered and the mean is returned as result of the defuzzification. It can lead to some interpretation problems, for instance in the situation shown in figure 2.9.

- Centroid defuzzifier:

It determines the centre of gravity of the output fuzzy set and returns this value as output of the FLS. The main drawback of this method is that the centroid can be difficult to compute.

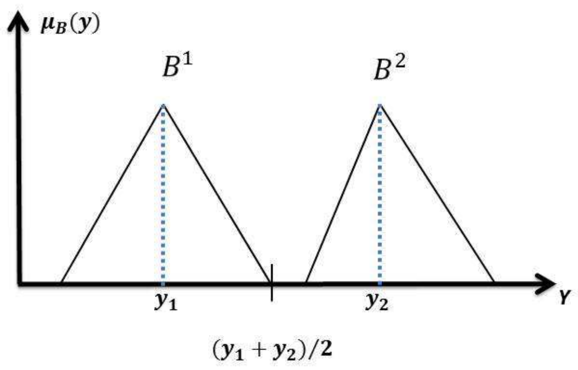


Figure 2.9: Interpretation issue in the mean of maxima defuzzifier

- Height defuzzifier:

It determines the centre of gravity of the output fuzzy set of each rule and then compose the results using the following:

$$y_h = \left[\sum_{l=1}^{M} \bar{y}^l \mu_{B^l}(\bar{y}^l)\right] \Bigg/ \left[\sum_{l=1}^{M} \mu_{B^l}(\bar{y}^l)\right] \qquad (2.27)$$

The main advantage with respect to the previous method is that the centre of gravity of the typical shapes used to represent the membership functions is well known.

2.7 Discussion

Fuzzy logic systems are universal function approximators [Wang, 1992(a)]. They share this property with artificial neural networks, introduced in the first chapter. The aim of this brief discussion is to stress the reasons that make FLSs a better solution in order to create an interpretable but also accurate model for the analysis of financial data. Both models can be applied to perform this task, then a brief comparison of the main features of each of them, may make clearer what are the advantages of using FLSs.

First of all, both models share the necessity of setting many configuration parameters. As for neural networks, their design needs to specify the number of layers, the number of neurons for each layer and the type of transfer functions to be used. On the other hand, also FLSs need to set, for instance, the type of fuzzification, the shape and parameters of membership functions, the operators for the implementation of inference and composition and the defuzzifier method. In both cases, the choice of these parameters can heavily influence the performance of the model. What makes FLSs a better option than a neural network is the possibility to choose the parameters in a smarter way. [Mendel, 1995] In fact, the former, being considered as white boxes, give the possibility to investigate how the choice of the parameters can influence the model. Consequently, the tuning of these systems is faster than for neural networks, for which the parameters are randomly initialized.

Finally, but most importantly, FLSs can handle linguistic knowledge and include uncertainty in the model in a very spontaneous way. [Mendel, 1995]

In light of the above, all the limitations of neural networks systems discussed in chapter one are overcome by the use of fuzzy logic systems. In addition, FLSs increase the interpretability of the models created thanks to the linguistic knowledge representation, and are able to embed uncertainty through the approximate reasoning process.

3 Fuzzy Rule Based Classifiers

3.1 Introduction

FLSs discussed in the previous chapter have been used in many application areas, such as control, clustering, regression and classification. This chapter will explore, more specifically, their application to solve classification problems and support decision making process. The aim of classification is to extract a model of the data describing important data classes [Han, 2001]. In this way, through the model created it is possible to “classify”, i.e. assign a class label to, previously unseen data objects. As concern financial data analysis, a classification model can be built, for instance, to decide if a bank loan is safe or risky, or if a credit card transaction is legitimate or fraudulent. However, in order to really support decision makers in their analysis, the model should be as simple as possible and, at the same time, have a significant explanation ability. Fuzzy rule based classifiers (FRBC) are particularly suitable for this purpose because they are able to mix the simplicity of a rule-based system with the interpretability of linguistic labels that only fuzzy systems can provide.

This chapter attempts to concisely summarize some of the approaches proposed in the literature to generate fuzzy classifiers. First, a general description of the classification process and the structure of fuzzy rule based classifiers will be presented. Then, some of the approaches to learn rules and membership functions will be discussed and, finally, major classification issues will be analysed, with particular attention to the interpretability-accuracy tradeoff.

3.2 Classification as a process

The aim of the classification process is to create a model which describes the data to be analysed and predict the class, i.e. the categorical label, of previously unseen data points. [Han, 2001] This process comprises two main steps:

1) building a classifier starting from a predetermined training set, consisting in a set of data points with an associated label. Each example is represented by a n-dimensional attribute vector, also called feature vector, which can be seen as the description of the data point and a label that is a categorical attribute indicating the class whose the point is assumed to belong to. Briefly, in the first step of the classification process a mapping or a function that separates the data classes is learnt. Although many different forms to express the mapping function can be used, this study will focus on the fuzzy rule-based approach, in which the mapping is represented through fuzzy IF-THEN rules.

2) Once the mapping is learnt, the model is complete and the second step of the classification process can be performed. In this phase the model built in the previous step is used to predict the associated class label of a given data point, which is not included in the training set. The set of tuples used to evaluate the generalization ability of the model is called test set. The reason why a set of unseen examples is used to estimate the accuracy of the classifier, is that a phenomenon called overtraining could occur. It means that the classifier over-fits the training data, including in the model also some anomalies that are not present in the general dataset. The structure of the test set is exactly the same as

the training set one and the accuracy of the model is calculated by submitting the test tuples to the classifier and comparing the label predicted with the actual one. The accuracy is then calculated as the percentage of test tuples correctly classified. More details about the evaluation metrics used to evaluate classifiers will be discussed further below.

3.3 Structure of fuzzy rule based classifiers

A fuzzy rule-based classification system (FRBCS) consists of two main components: the inference system and the knowledge base. The latter, in turn, is comprised of a rule base that contains the set of fuzzy rules and the database including the parameters that define the membership functions associated to the input variables. The overall structure of a FRBCS is shown is fig, 3.1

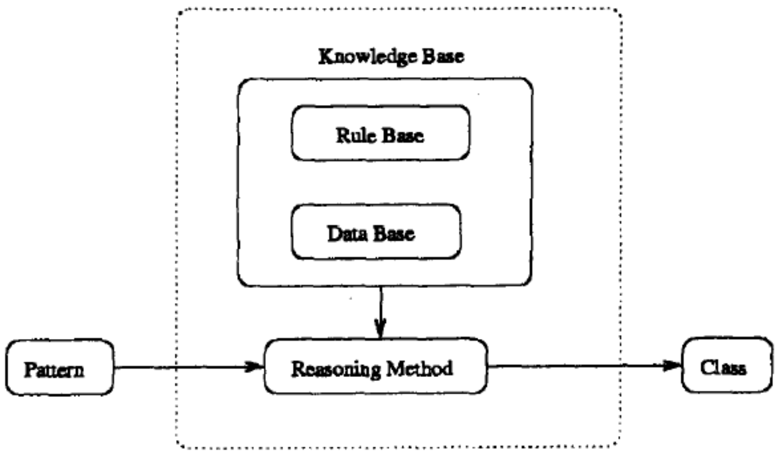


Figure 3.1: Structure of a FRBCS [Cordòn, 1999]

Thus, the design of a FRBC implies the choice of the rules' structure in the rule base, the specification of the membership functions, and the definition of a reasoning method.

The literature provides several approaches for the creation of FRBCs, but before digging into more details, it is important to frame the classification problem starting with a formal definition.

Formally, the aim of a k-class pattern classification problem is to assign a class C_k from the set of all possible classes $C = \{C_1, \dots, C_K\}$ to an object represented by a n-dimensional point in the feature space $\mathfrak{R}^n$. Let $X = \{x_1, \dots, x_n\}$ be the set of the input variables and U_n the universe of the nth variable and suppose it is partitioned, for each variable, into $i = 1, \dots, f$ fuzzy partitions. A generic rule of the FRBC can be expressed as:

$$IF\ x_1\ is\ A_{1i}\ and \dots and\ x_n\ is\ A_{ni}\ THEN\ Y\ is\ C_k$$

where Y is the output of the classifier, C_k is the label associated with the rule and A_{1i} represents the ith fuzzy partition of the first variable.

A weight is often added to each rule and it represents the certainty degree of the classification in the class C_k for a pattern belonging to the fuzzy subspace delimited by the antecedents of the rule. Many alternatives have been proposed in the literature to compute rule weights, the most used are the following:

- ***Certainty factor*** [Cordon, 1999]

$$CF_m = \frac{\sum_{x_t \in C_k} w_m(x_t)}{\sum_{t=1}^{N} w_m(x_t)} \tag{3.1}$$

- ***Penalized certainty factor*** [Ishibuchi, 2005]

$$P\_CF_m = CF_m - \frac{\sum_{x_t \notin C_k} w_m(x_t)}{\sum_{t=1}^{N} w_m(x_t)} \qquad (3.2)$$

where $w_m(x_i) = \prod_{j=1}^{f} A_{ji}(x_t)$ is the matching degree of the rule with the input.

Then, the parameters of a FRBC are: the rules to be used by the reasoning method to perform the classification task; the reasoning method; the shapes and the number of fuzzy partitions for each input variable.

In the following, some of the approaches proposed in the literature for each of the above mentioned parameters, will be analysed.

3.3.1 Knowledge base learning approaches

In order to achieve its goal, a FRBC needs to learn the rules and the fuzzy partitions of each variable. In some applications, especially in control field, they are often provided by human experts. However, specialized literature has recently proposed new approaches that allow to learn the knowledge base automatically and directly from the numerical data thereby enabling the creation of classifiers also when there is not enough expert knowledge.

Usually, these approaches assume a fixed number of linguistic terms for each input variable and a uniform distribution of the linguistic terms into the universe of discourse. The number of membership functions per variable is usually between three and nine and can be different from a variable to another. [Cordon, 2001] In this way, these approaches are only focused on the rule base learning.

Based on the demonstrated influence of the shape and number of linguistic labels per variable on the performance of classifiers, other approaches include a tuning process of the database, which modifies only the definitions of membership function.

Finally, additional approaches to learn database and rule base simultaneously have also emerged. One of them will be used as a case study in chapter 5.

To sum up, the different approaches to knowledge base learning can be divided into three categories:

1. RB learning using a predefined DB
2. Tuning of membership functions
3. Simultaneously learning of the KB components

3.3.1.1 Rule base learning using predefined membership functions

One of the most popular approaches to the rule base learning was proposed by Ishibuchi in his work "Distributed representation of fuzzy rules and its application to pattern classification" [Ishibuchi, 1992]. It describes the extraction of rules directly from numerical data as a two-phases process:

- Fuzzy partitioning of the pattern space

- Determination of a fuzzy rule for each fuzzy subspace

With regard to the first phase, a simple grid partition is employed to divide the input space. For the sake of clarity, let us assume a two-dimensional pattern space and suppose to partition each variable into five fuzzy sets. Using the grid approach, we will generate $5^2 = 25$ subspaces and then 25 potential rules (see fig. 3.2)

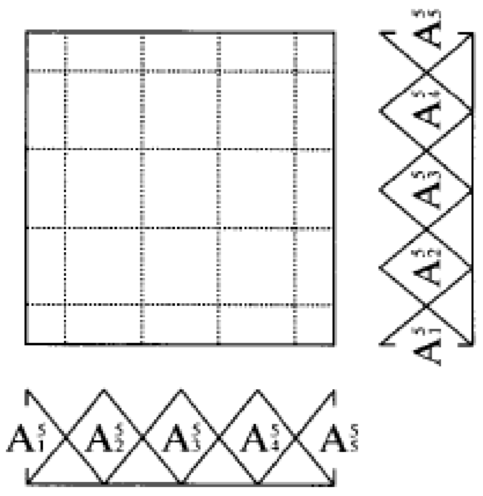


Figure 3.2: Fuzzy grid partitioning [Ishibuchi, 1995]

The main drawback of this approach is that the number of partitions chosen can significantly influence the performance of the system, because if the partitioning is too coarse then many patterns will be misclassified; on the contrary, a too fine partition will generate problems in those subspaces with no training patterns, due to the impossibility of generating rules for them.

It is important to note that the distribution of data should be taken into account when the number of partitions is chosen. This explains why, in this paper, the concept of distributed fuzzy rules has been introduced. The main idea is to use the if-then rules generated through different numbers of partitions (see fig 3.3) simultaneously.

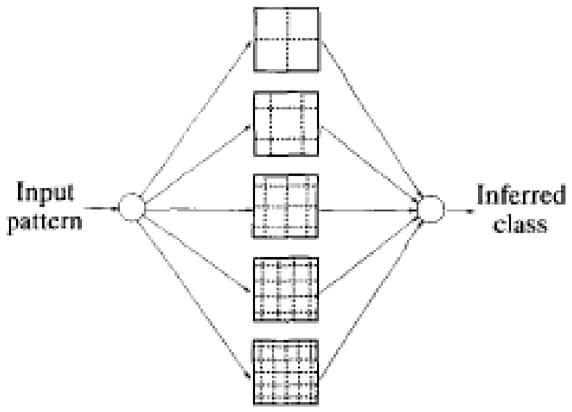


Figure 3.3: Classification system with multiple fuzzy rule tables [Ishibuchi, 1995]

Let $L = 3$ be the maximum level of partitioning, i.e. each variable is partitioned at most using three linguistic labels. In the two-dimensional case the total number of rules generated is: $2^2 + 3^2 = 13$. A generic rule can be expressed as:

$$IF\ x_1\ is\ A_i^L\ and\ x_2\ is\ A_j^L\ \text{THEN} \ldots$$

Once all the possible combinations are generated, the class label is determined using the training data. More in detail, the algorithm assigns each rule to the label of the class with the larger sum of compatibilities to the premise. The procedure to assign the class and the weight to the rule is the following:

- For each class $T = 1, \ldots M$, β_{CT}, i.e. the sum of the compatibility of x with the class T, is calculated as:

$$\beta_{CT} = \sum_{x_p \in CT} \mu_i^L(x_1) * \mu_j^L(x_2) \tag{3.3}$$

- The label of the class for which this coefficient is the maximum is assigned as consequent of the rule. If more than one β_{CT} take the maximum value or if all the β_{CT} are zero, then the class label cannot be determined and the rule is defined as dummy rule, which means that it is not used during the classification process.
- If only a single class takes the maximum value, then the weight is calculated as:

$$CF_{ij}^K = (\beta_{CT} - \beta) \Big/ \sum_{T=1}^{M} \beta_{CT} \tag{3.4}$$

The main drawback of this procedure resides in the incredible number of rules that can be generated, especially in the case of high dimensional datasets. In fact, the number of rules increases exponentially with the number of features. For the sake of clarity, let us assume that six different linguistic values are used for each variable. Then, 6^n rules will be generated in a n-dimensional space. A copying strategy was proposed by [Ishibuchi, 1997] in a later work, based on the intuition of generating only rules with a pre-specified number of antecedents, instead of generating all possible rules. In doing so, [Ishibuchi, 1997] introduced a new linguistic value for the antecedents, the so called "*don't care*" condition. It is represented by a membership function with value 1 in the whole domain of the attribute. Consequently, the antecedents presenting this value can be omitted from the rule. The effect generated in a two-dimensional space is shown in fig. 3.4.

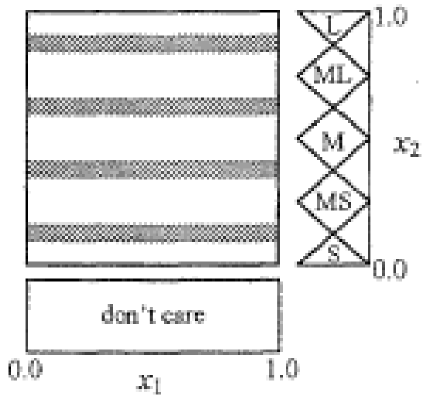


Figure 3.4: Effect of "don't care" condition in a two-dimensional space [Ishibuchi, 1997]

In this way only a small part of the all candidates rule is generated. They also apply genetic algorithms for the selection of the most relevant rules.

Another approach allowing the extraction of fuzzy rules directly from the data has been proposed by Abe in [Abe, 1995]. This study tries to overcome the limitations derived from the necessity of dividing the input space in advance, by introducing a technique to extract fuzzy rules with variable fuzzy regions. The idea is to divide the input region recursively using hyperboxes. There can be two types of hyperbox: activation and inhibition. The former defines the existence region of a class while the latter inhibits the existence of data in that activation hyperbox. A membership function is associated with each hyperbox and the degree of membership of a point x is 1 if x is inside the activation hyperbox, while it decreases as x moves away from it. The process of partitioning can be divided in the following steps:

- Determine activation hyperboxes by including the maximum number of data for each class.
- If the activation hyperboxes of two different classes i and j are overlapped, then the overlapping region is defined as an inhibition hyperbox.
- The inhibition hyperbox is analysed and if there exist data from both classes i and j, an additional activation hyperbox is defined for each of these classes.
- Then, recursively the process is repeated by investigating the overlap region of these new activation hyperboxes and defining inhibition hyperboxes if necessary.

The recursive process to resolve the overlap of activation hyperboxes is summarized in fig. 3.5.

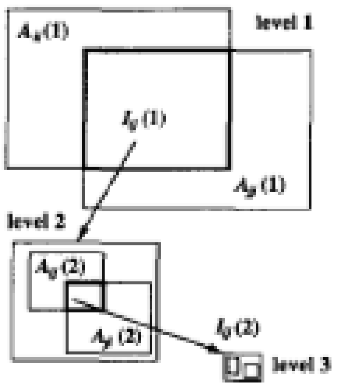


Figure 3.5: Recursive definition of activation and inhibition hyperboxes [Abe, 1995]

Once the partitions are determined, the fuzzy rules have to be generated. The process to obtain fuzzy rules starting from the data can be summarized as follows:

- Determine the activation hyperboxes of level 1, $A_{ii}(1)$, using the set of input data X_i
- If there is no overlap between activation hyperbox of class i and the others hyperboxes of level 1, then a fuzzy rule for class i can be generated:

$$IF\ x\ is\ A_{ii}(1)\ THEN\ class\ i$$

- Otherwise, assuming that $A_{ii}(1)$ overlaps $A_{jj}(1)$, it is resolved by defining the overlap region as an inhibition hyperbox of level 1, denoted as $I_{ij}(1)$
- Then it is possible to define a new fuzzy rule of level 1 using the inhibition hyperbox:

$$IF\ x\ is\ A_{ii}(1) AND\ x\ is\ not\ I_{ij}(1)\ THEN\ class\ i$$

- If the activation box of a class coincides with the overlap region, the rule for the inhibition hyperbox is not created, because it is a void rule.
- Finally, if some data of class i belong to the inhibition hyperbox $I_{ij}(1)$, then an activation hyperbox of level 2 is defined inside $I_{ij}(1)$, and the process recursively continues until a termination condition is reached.
- The process is ended when there is no overlap between the activation hyperboxes of level l.

Finally, a new branch of research is focusing on applying genetic algorithms in order to create the fuzzy rule base. This kind of approaches can be divided in two main categories: rule selection and rule generation approach. The former is focused on selecting the best set of rules from a large amount of candidate rules in order to obtain compact rule base; the latter tries to directly generate a set of rules, without any previous knowledge. Both approaches will be discussed in detail in the chapter 5.

3.3.1.2 Tuning of membership functions

Regarding the definition of the membership functions, in the presence of enough information about the domain, a default partition is usually provided by experts. Otherwise, equally spaced partitions, i.e. uniformly distributed in the domain of the attribute, are often used in order to maintain an important property of the fuzzy systems which is the interpretability. In fig. 3.6 an example of equally spaced (a) and non-uniform partitioning (b) is provided. Uniform partitions can be universally considered the most interpretable, while some problems can arise when the membership functions are excessively overlapped.

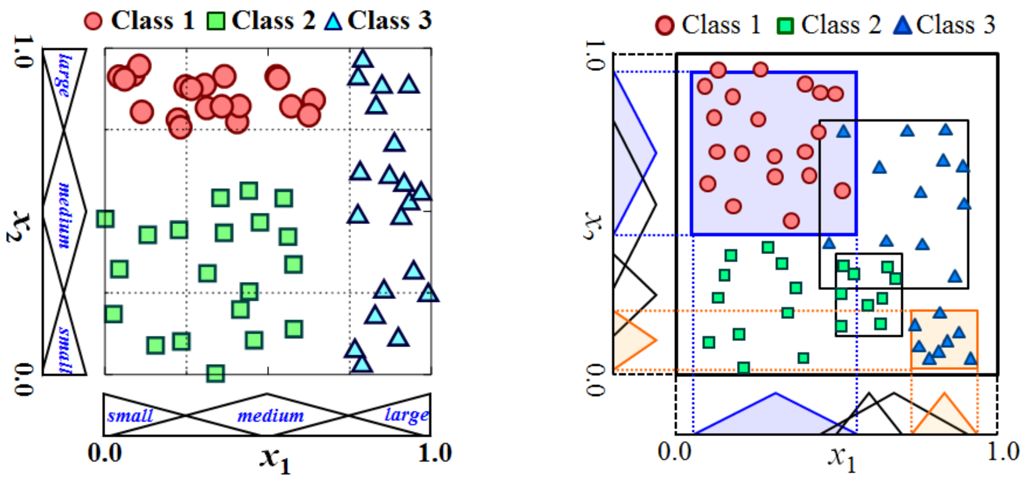


Figure 3.6: equally spaced partitioning (a), non-uniform partitioning (b) [Ishibuchi, 2009]

Convenience would suggest using equally spaced partitions, but there is a caveat that goes along this solution: as mentioned above, the shape of membership functions can strongly influence the accuracy of the classifier. This is why, some techniques to learn also the DB component of an FRBC have been proposed in the literature. The process of adjusting the definition points of the membership functions is called "*tuning*". In light of the above, this tuning should be performed using constraints which avoid the generation of excessively overlapped partitions. In order to achieve this goal, some partition integrity indexes used during the learning process have been proposed in the literature. One of the most common approaches to perform the tuning of membership functions include the use of genetic algorithms [Antonelli, 2011] and will be discussed further below.

3.3.1.3 Learning the KB components simultaneously

The study mentioned before, suggests a way to concurrently learn the rule base and the parameters of the membership functions associated with the linguistic labels. In this work, a multi-objective genetic algorithm is used for optimizing the accuracy of the classifier and the interpretability of the fuzzy partitions. It will be discussed in detail in chapter 5.

3.3.2 Reasoning methods

The reasoning method is the inference procedure through which, starting from an input pattern, it is possible to derive the belonging class. Recalling what explained in the chapter above, the main characteristic of fuzzy reasoning is that it is possible to classify that pattern even in the absence of a perfect match of the new pattern with the antecedents of a rule. In this way a great generalization capability is achieved. Obviously, many different rules can be fired by an input pattern, then a way to univocally determine its class label should be decided.

Several reasoning methods have been proposed in the literature. The classical one is called maximum matching method. In order to classify a new pattern it considers the rule with the maximum matching degree between the pattern and the antecedent part of the rule, as the winner rule and assigns the pattern to the class specified by the consequent part of the rule. More in detail, the process to classify a new pattern can be described as follows:

- Calculate the matching degree of the pattern with the antecedents of the rule. It expresses the strength of activation of the if-part of the rule.

  $$R^l(x) = T\left[\mu_{A_1^L}(x_1), \dots, \mu_{A_N^L}(x_N)\right] \qquad (3.5)$$

  where T is a t-norm operator (typically product or minimum) and $A_N^L$is the nth antecedent of the rule.

- Calculate the degree of association $h(R^l(x), CF_l)$ of the pattern with the class specified in the consequent of the rule, by applying a combination operator (typically product) to the matching degree and the certainty factor of the rule.

  $$h(R^l(x), CF_l) = R^l(x) * CF_l \qquad (3.6)$$

  where CF_l is the certainty factor of the lth rule.

- Finally, the winning rule is the one with the highest value of association degree.

This reasoning method considers only the winner rule to determine the class of the new pattern, while ignoring the others. Yet, other reasoning methods proposed in the literature try to combine the various rules fired by the input pattern in order to obtain a result which takes into consideration the output of all the rules.

3.4 Evaluation metrics

Once the classifier is built, we need to define some measures to evaluate its performance and assess the quality of the model. A set of labeled tuples is given to the classifier and, for each of them, the actual class is compared with the predicted one. Before moving forward with the evaluation of the classifier, it is worth clarifying some of the central and recurring terms and concepts:

Positive tuples: patterns belonging to the main class of interest

Negative tuples: patterns belonging to all the other classes

With respect to the definitions above, when the classifier prediction is compared with the known class of the input pattern, four possible situations can be distinguished:

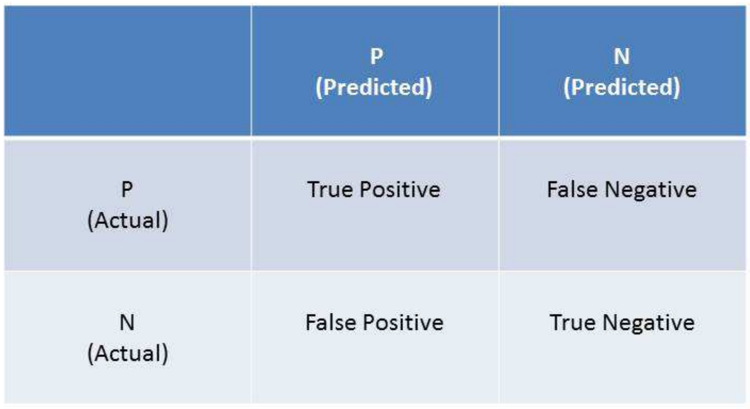
True positive (TP): positive patterns correctly labeled by the classifier

True negative (TN): negative patterns correctly labeled by the classifier

False positive (FP): negative patterns incorrectly labeled as positive by the classifier

False negative (FN): positive patterns incorrectly labeled as negative by the classifier

These elements are usually summarized in the confusion matrix, which is a convenient way to visualize how the errors are distributed with respect to the various classes. A confusion matrix for a binary class problem, i.e. a classification task with only two possible classes, is shown in figure 3.7.



	P (Predicted)	N (Predicted)
P (Actual)	True Positive	False Negative
N (Actual)	False Positive	True Negative

Figure 3.7: Confusion matrix

Each row of the matrix represents the actual patterns belonging to a class, let P and N be the total number of instances belonging to the positive and negative classes, respectively. Each column shows the total number of predicted positive and negative patterns. Finally, a generic element CM_{ij} indicates the number of tuples belonging to class i that the classifier assigned to class j.

In light of the above, several measures have been proposed to assess the performance of a classifier. The most common one is the accuracy. It is defined as the percentage of tuples correctly classified and can be expressed as:

$$Acc(C) = \frac{TP + TN}{P + N} \tag{3.7}$$

This measure can be seen from another perspective and be expressed in terms of the misclassified tuples. In this case it is called error rate and is expressed by:

$$\mathrm{Err(C)} = 1 - \mathrm{Acc(C)} = \frac{\mathrm{FP + FN}}{\mathrm{P + N}} \tag{3.8}$$

Then, a classifier with a good accuracy should have a confusion matrix where most of the elements are distributed along the principal diagonal, while the rest of the entries would be close to zero.

However, the main drawback of the accuracy is that it is not representative of the true performance of a classifier when the dataset is unbalanced, i.e. when the number of patterns of the class of interest is significantly greater than the number of patterns belonging to the other classes. For the sake of clarity, assume that our task is to construct a classifier for the detection of credit card frauds. The related dataset are to be highly unbalanced because most of the transactions are legitimate while the fraudulent ones, which are our class of interest, are sporadic. In such circumstances, suppose that our dataset is composed of 990 negative patterns (legitimate transactions) and only 10 positive patterns (fraudulent transactions). If the classifier under assessment predicts the majority class for all the patterns in the dataset, its accuracy will be 99%. It goes alone that the result obtained is not significant, since the error rate on the class of interest is 100%. In other words, the accuracy is highly biased by the distribution of the class tuples in the dataset.

To tackle this problem, a variety of alternative measures to assess the performance of a classifier have been suggested, among which the most important are: sensitivity, specificity, precision, recall and F-measures.

Sensitivity: It is also referred to as *true positive rate* and expresses the proportion of positive tuples correctly classified. It can be computed as:

$$sensitivity = \frac{TP}{P} \tag{3.9}$$

Specificity: It is also referred as the *true negative rate* and expresses the proportion of negative tuples correctly classified. It can be computed as:

$$specificity = \frac{TN}{N} \tag{3.10}$$

Precision: It can be considered a measure of the *exactness* of the classifier and expresses what percentage of predicted positive tuples is actually positive. It can be computed as:

$$precision = \frac{TP}{TP + FP} \tag{3.11}$$

Recall: It can be considered a measure of *completeness* and expresses the percentage of actual positive tuples which are recognised as positive by the classifier. It can be computed as:

$$recall = \frac{TP}{TP + FN} = \frac{TP}{P} \tag{3.12}$$

F-measures: They allow to combine the precision and recall measures. It is possible to give prominence to precision or recall by fixing the value of the parameter β which can be considered a weight. A general expression of this measure is the following:

$$F_\beta = \frac{(1+\beta^2) * precision * recall}{\beta^2 * precision * recall} \qquad (3.13)$$

The most common values of β are 0.5 and 2.

To sum up, the accuracy is the most common measure used for the evaluation of classifiers, however its reliability depends on the evenly distribution of data classes, while the other measures perform well with unbalanced datasets.

The next chapter will discuss some of the most compelling issues related to unbalanced datasets. Approaches to tackle such issues will be also flagged and examined as suggested by the mainstream literature.

3.5 Main issues in classification problems

The main issues encountered when performing the classification task are mostly related to the dataset structure, in particular to high dimensional and unbalanced datasets. While the problems emerging in high dimensional spaces have been already discussed in the first chapter, the focus of our analysis in the following section will be on unbalanced datasets.

Another issue is related to the so called “accuracy-interpretability tradeoff”. It has been demonstrated that an increase in the interpretability of an FRBC will proportionally decrease the level of accuracy and vice versa. In the following section this problem will be explained.

3.5.1 Unbalanced datasets

As previously explained, an unbalanced dataset is characterized by an uneven distribution of the class patterns, i.e. considering a two-class problem it is possible to identify a majority and a minority class. Moreover, it is important note that the minority class is usually pivotal to the analysis task. Dealing with unbalanced datasets gives rise to some difficulties both in the construction and evaluation phases of the model.

[Lopez, 2012] points out the following issues with regard to the construction phase:

- Learning algorithms are usually biased towards the majority class, because they often tend to attribute higher weights to those rules that predict larger number of samples.
- Learning algorithms do not easily detect small areas with examples of the minority class which are included in a larger area of examples of the other class (see fig. 3.8(a)).
- Overlapping between the samples of the positive and negative class (see fig. 3.8(b)).

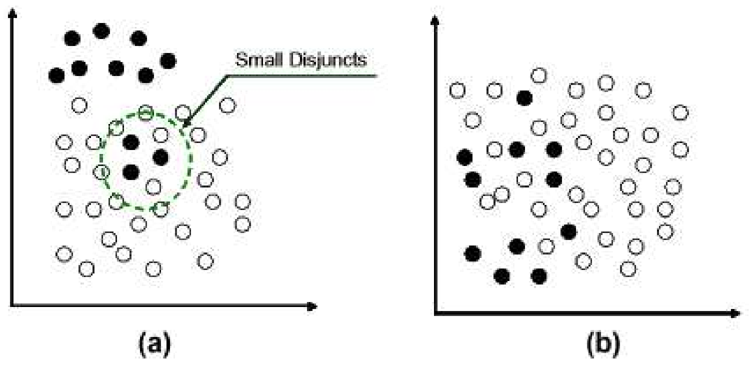


Figure 3.8: Problems with unbalanced datasets: (a) small disjucts and (b) overlapping between classes [Lopez, 2012]

As far as the evaluation phase is concerned, we have already introduced some metrics based on the values contained in the confusion matrix. Another good metric that can be used to assess the performance of classifiers over unbalanced datasets is the Receiver Operating Characteristic (ROC) curve. This shows the trade-off between the *true positive rate*, i.e. the sensitivity, and the *false positive rate*.

The latter is defined as:

$$FPR = \frac{FP}{N} = 1 - specificity \tag{3.14}$$

The ROC curve offers a visual representation of this trade-off. The point A in figure 3.9 indicates the ideal point where the optimal classifier should be placed. In fact, all positive examples in correspondence to it, are classified as correct, while no negative example is misclassified as positive. The line $y = x$ (C) represents a random guessing. The closer the ROC curve is to the diagonal line, the less accurate the model.

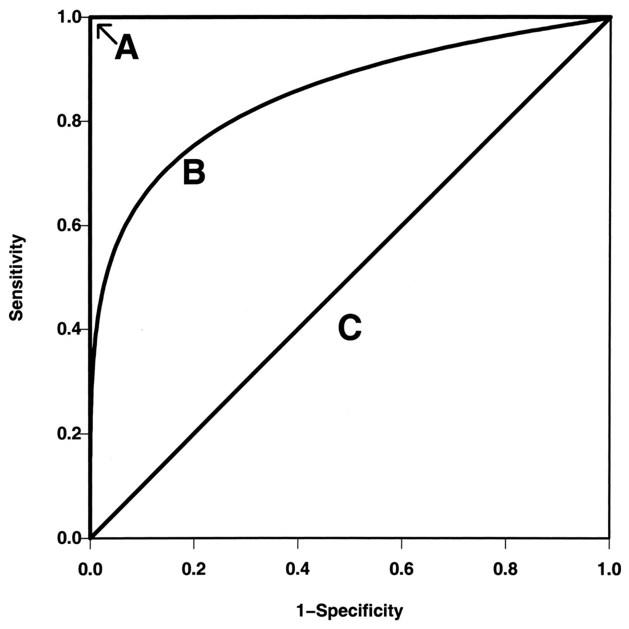


Figure 3.9: ROC curve.

Source: <http://circ.ahajournals.org/content/115/5/654/F2.expansion.html>

The ROC curve is a very effective way to compare different classifiers. It is usually obtained by varying the parameters which represent the classifier and plotting a point in the ROC space for each of them.

In light of the above, the Area Under the ROC Curve (AUC) can be used to evaluate the performance of a classifier and it can be computed as:

$$AUC = \frac{1 + TP_{rate} - FP_{rate}}{2} \tag{3.15}$$

The closer this area is to 0.5, the less accurate the model is. A classifier with a perfect accuracy will have an AUC of 1.0.

3.5.2 Accuracy versus interpretability

The design of accurate and interpretable FRBC is not a trivial task. In fact, the attempt of improving the accuracy generally results in a degradation of its interpretability and vice versa. Since the strength of FRBCs is their explanation ability, it is important to preserve this characteristic. Thus, when a model is generated, these two factors has to be taken into account and a compromise between accuracy and interpretability needs to be found (see fig 3.10).

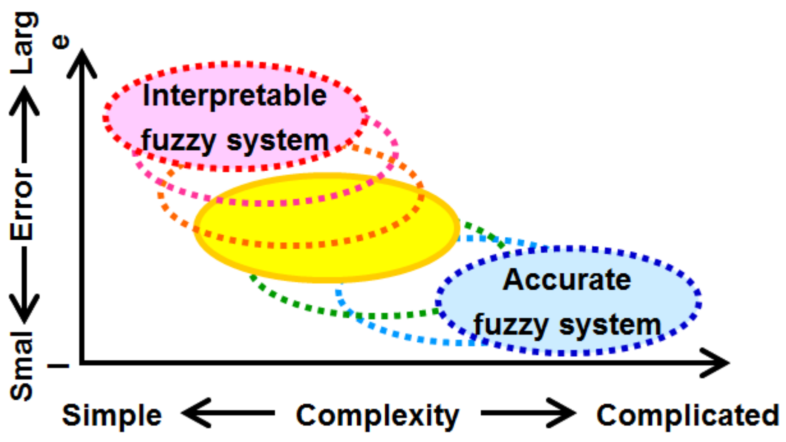


Figure 3.10: Accuracy-Interpretability tradeoff [Ishibuchi, 2009]

In order to study the different methodologies identified in the literature to find the best trade-off, we have to look first at the factors that can influence the interpretability of a classifier. Given the subjectivity intrinsic to the concept of interpretability, this can be influenced by several factors to name a few the shape and number of membership functions, and the structure and amount of rules in the rule base. [Guillaume, 2001] points out the following factors:

- Rule base structure:

  - The number of rules in the rule base should be as limited as possible in order to avoid the overtraining and favour the generalization process, while at the same time facilitating the extrapolation of useful knowledge, from the rules, by the user. The rules in the rule base should be incomplete, i.e. not all the variables should be involved in the rules.

- Membership function definition:

  - Increase the number of membership functions for each variable can produce positive effects on the accuracy of a classifier, since this will enhance the creation of more complexes decision boundaries (see fig. 3.11) However, the number of fuzzy partitions should be carefully considered, because, as already mentioned, too many partitions may results in overfitting of the training data.

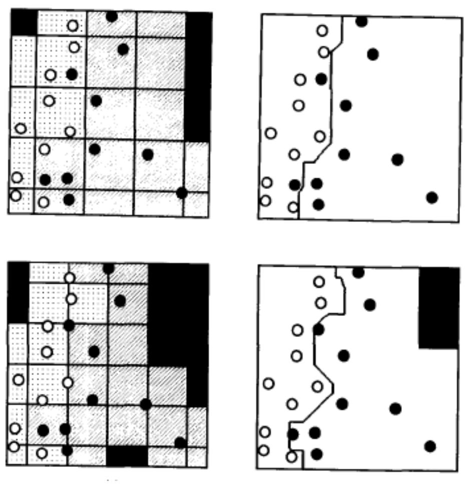


Figure 3.11: Decision boundaries with five fuzzy sets per variable and with six fuzzy sets per variable [Ishibuchi, 1992]

- Fuzzy partitions must be as readable as possible. It is clear that the equally spaced partitioning is the most interpretable, but it has been demonstrated that in order to increase the accuracy of the classifier, the tuning of the membership function can be very useful. The main concern is that by modifying the definition points of the membership functions, the interpretability of the model is downgraded (see fig. 3.12). To preserve the interpretability some constraints should be defined.

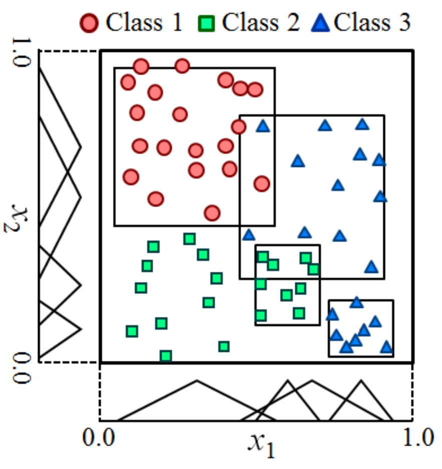


Figure 3.12: Tuning of membership functions [Ishibuchi, 2009]

The literature contains many examples of how to obtain a good trade-off between interpretability and accuracy. Some ideas include: the use of weighted objective functions that take into consideration both the accuracy and the interpretability; the design of two-step fuzzy systems, which search for an accurate solution and then try to simplify it; the use of multi-objective genetic algorithms.

In recent years, most of the studies in this field have focused on the application of genetic algorithms to solve the accuracy-interpretability trade off.

[Ishibuchi, 1995] proposes an approach based on genetic algorithms for selecting a small number of significant fuzzy rules to construct a compact classifier with high accuracy and good interpretability. In order to maximize the accuracy and minimize the number of rules selected, an aggregated objective function is chosen as fitness function.

$$f(S) = w_1 * f_{Error}(S) + w_2 * f_{Complexity}(S) \qquad (3.16)$$

where S is the subset of rules selected by the algorithm and w_1 and w_2 are positive numbers.

This approach presents a main drawback, namely it does not ease the identification and attribution of the best values to the weights, which the system strongly depends on. As shown in fig 3.13 if the weight for the complexity minimization is too high (a), a very simple but not accurate model is obtained; on the contrary, if the weight for the error minimization is too high (b), a very accurate, but also very complex system is obtained.

Moreover, even when the weights are aptly specified, the best value S^* is not always found (see fig. 3.13 (c)).

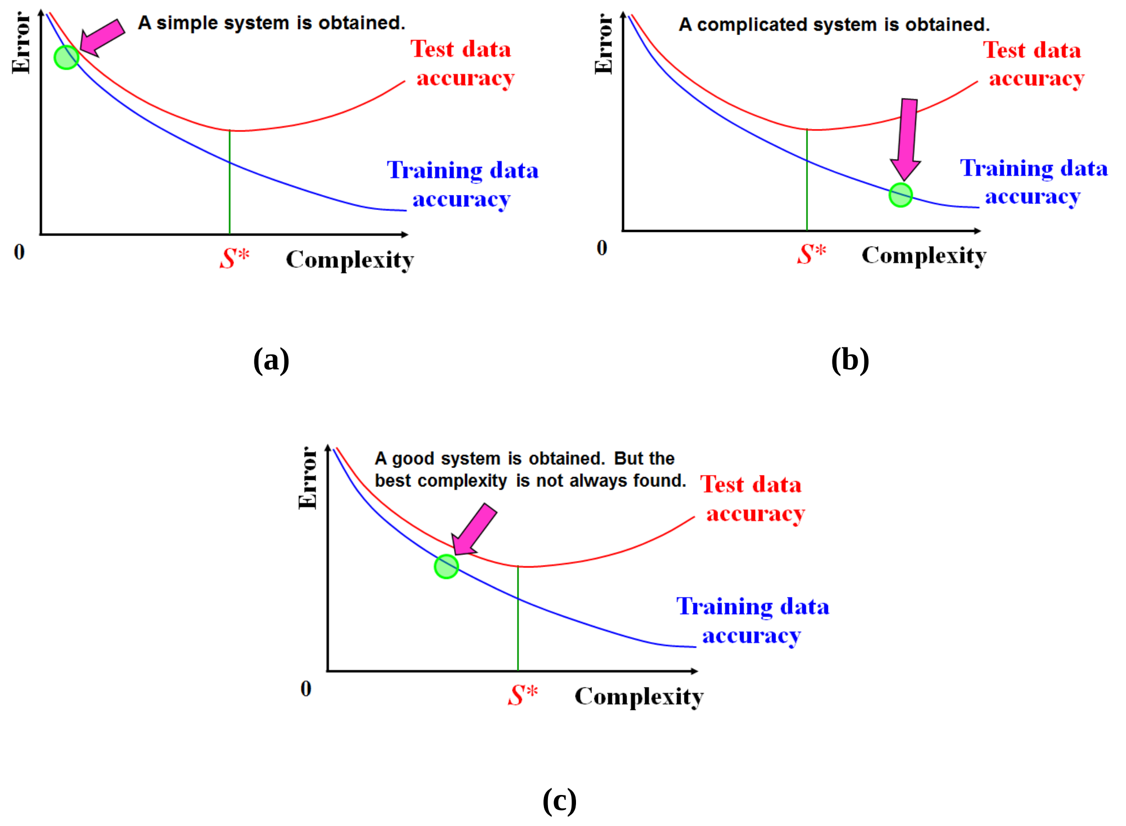


Figure 3.13: Different trade-off depending on the values of the weights [Ishibuchi, 2009]

This explains why the last studies are focused on the application of multi-objective evolutionary algorithms (MOEA). They allow to take into account each objective individually and generate a set of valid solutions. Each solution is characterized by a different tradeoff between accuracy and interpretability (see fig 3.14)

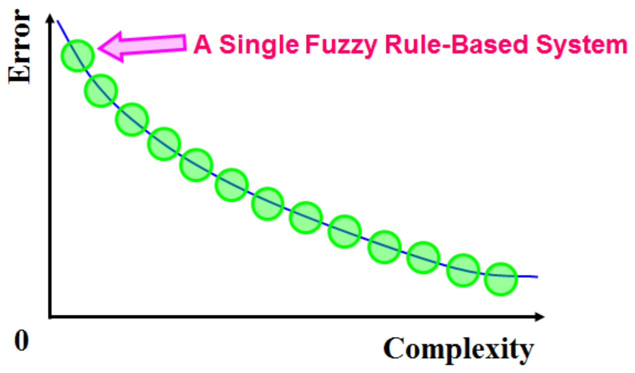


Figure 3.14: Results of a multi-objective evolutionary algorithm [Ishibuchi, 2009]

3.6 Discussion

Once the model is built, its **interpretability** is of great value. The final goal of the various techniques of data analysis is to extract useful knowledge in order to ease the decision making process and make it more efficient. In order to do so, the process leading to the generation of the results needs to be interpreted and understood, which in turn contributes to increase the reliability of the model created. Given the importance of interpretability, fuzzy rule based classifiers are one of the most common approaches used for the classification task. Their strength lies in the possibility to combine the high interpretability of the linguistic representation with the accuracy and simplicity of a rule-based structure. The main drawback of these systems is that it is not easy to find a good compromise between interpretability and accuracy. As proved by several studies, accuracy and interpretability are conflicting objectives and the most effective way to create a system with a good-enough trade-off is through genetic algorithms. In light of these assumptions, the so-called genetic fuzzy rule based classifiers (GFRBC) will be introduced and analysed in the next chapter.

4 Genetic Fuzzy Rule Based Classifiers

4.1 Introduction

In the previous chapters we suggested that the design of an FRBC requires the learning of the two components of the knowledge base (KB): rules (RB) and membership functions (DB). The possibility of defining these two components automatically would be of great help. It has been demonstrated that the process of automatic definition of a fuzzy system can be considered an optimization task [Cordon, 2004]. Given their capacity to find near optimal solutions in large and complex search spaces, genetic algorithms (GAs) are proposed as possible solution to the issue at stake. This said, the combination of FS and GA is expected to be a convenient solution for the attainment of a well-designed classifier. The hybridization of FS and GS is known as Genetic Fuzzy System (GFS).

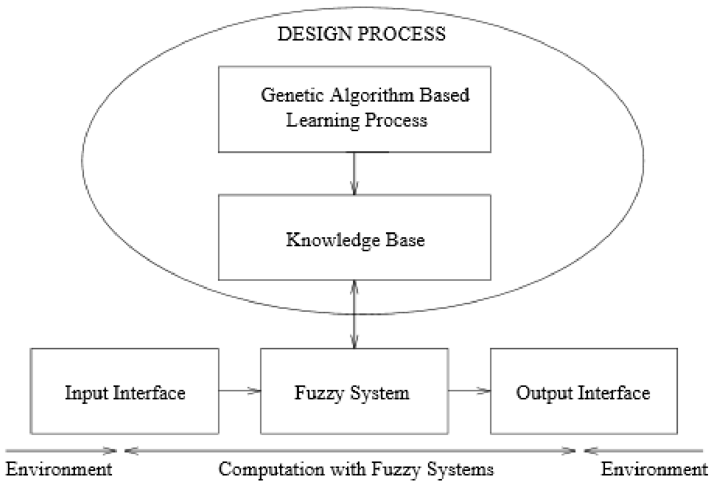


Figure 4.1: Structure of a genetic fuzzy system (GFS) [Herrera, 1997]

Moreover, genetic algorithms can be useful in order to find a good tradeoff between accuracy and interpretability. In fact, generating rule bases through GA makes it possible to create a series of classifiers, each with different tradeoff values. Among the solutions so created, none of them is better than the others, this means that the user's choice is based on the application context that he/she considers the most appropriate.

In light of the above, this chapter will first introduce single and multi-objective genetic algorithms, and then will explore learning opportunities by means of GAs (both for RB and DB components). The final part of the chapter will focus on the multi-objective GAs (MOGA) application for the solution of the tradeoff between accuracy and interpretability.

4.2 Genetic algorithms

Genetic algorithms are general purpose search algorithms that mimic the processes observed in natural evolution [Herrera, 1997]. The main idea is to reproduce the natural evolution processes to solve optimization problems. The basic element is the chromosome, which represents one of the possible solutions to the problem at hand, i.e. a point in the search space. Each chromosome has an associated fitness, i.e. a value through which the survival of the fittest among all the individuals is simulated. At a certain time t, the set of solutions $P(t)$ is called a population.

The analogy with the living organism evolution process is clear. Then, in order to allow the evolution, the individuals of the population must compete one each other for mating. Only some of the individuals are selected for the reproduction, particularly, the ones with a greater fitness value are more likely to be selected. In fact, the fitness value is proportional to the utility or adaptation of the solution represented by the chromosome. The operation that simulates the reproduction is called crossover and it allows the combination of the genes of two good chromosomes to generate new individuals that, in some circumstances, are even better than their parents.

Moreover, an important role in the living organism evolution process is played by the mutation. It consists in a small change into the genetic material of an individual and it allows to maintain a certain structural variability within a population.

The main reason of the success of genetic algorithms is that they are able, starting from a completely unknown search space, to bias the research towards the more useful subspaces. This is reason why they perform particularly well in large and complex spaces. However, it is not guaranteed that the optimal global solution is found, but surely a local optimal one is found in a very short time considering the search space complexity [Herrera, 1997]. The outline of a simple genetic algorithm is shown in the figure 4.2.

From the above, it is clear that the implementation of a genetic algorithm requires the definition of the following elements:

- Genetic representation of solutions
- A method to generate the initial population
- Evaluation function to assign the fitness value
- Crossover and mutation operators

- Values of the parameters of the genetic algorithm, such as population size, probabilities of mutation and crossover, etc.

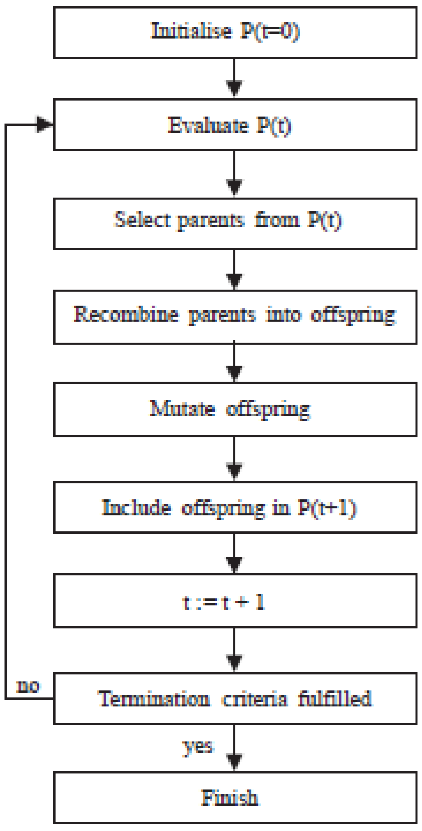


Figure 4.2: Outline of a simple genetic algorithm [Cordon, 2004]

The steps needed to design a genetic algorithm will be explained more in detail in the following section.

4.2.1 Representation and initialization

The representation of solutions plays a key role in the design of a genetic algorithm. It is called encoding scheme and the binary representation is the most common one, although not the only one. In the binary representation a chromosome is composed of several genes, each of them encoded by a bits string. Other types of encoding schemes include, for instance, the use of real numbers to represent the solutions. As stated by [Herrera, 1997], the choice of the encoding scheme is crucial because it can limit the GA capacity of exploring the search space.

Moreover, the initialization of the solutions should be performed thoroughly, because it represents the inception of the research. It can influence the whole evolution of the algorithm, causing, for instance, a premature convergence to suboptimal or undesired points, or also a slow convergence speed, if the similarity among the initial chromosomes is too high or their fitness values are too low. [Chou, 2000]

4.2.2 Fitness function

Another important element in the design of a genetic algorithm is the definition of the fitness function. Given its leading role in the search process, a fitness function must be chosen for each problem, so as to make sure that the returned values are proportional to the goodness of the solution. In this way, the research is biased through the most interesting areas of the search space.

4.2.3 Main operators

The main operators performing the evolution are: selection, crossover and mutation.

- ***Selection***

This operator allows to select the chromosomes for the mating process from a population P by preferring the ones with a high fitness value over the ones with a low fitness value. After its execution, a new population P' is generated which generally contains copies of chromosome with higher fitness values, in order to give them a greater participation in the generation of the offspring. There are many different ways to perform the selection, the most common one is the so called "*roulette wheel*". As shown in fig. 4.3, it assigns a section of the wheel to each chromosome and the size of the space associated is proportional to its fitness. The selection is then performed by repeatedly spinning the wheel and adding the correspondent chromosome to the population P'.

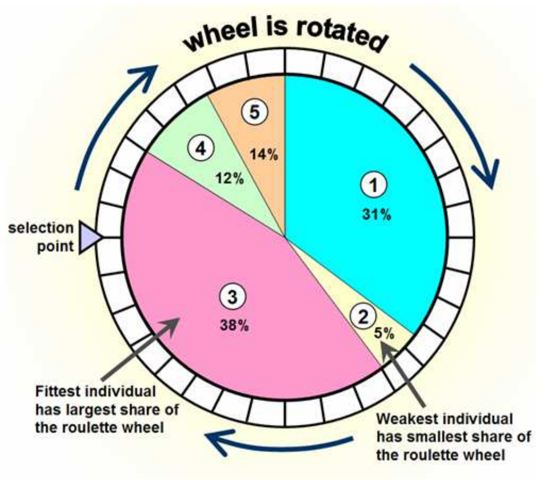

wheel is rotated
5
14%
4
12%
1
31%
selection point
3
38%
2
5%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

Figure 4.3: Roulette Wheel. Source: New Castle University, Engineering Design Centre. <http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php/>

- ***Crossover***

The crossover operator simulates the mating by combining the genes of two parent chromosomes to create two new chromosomes that can have a better fitness value than their parents. However, not all the individuals in the population P' will generate offspring, usually a crossover probability which represents the likelihood of the application of the crossover is fixed as a parameter of the algorithm. Many different types of crossover have been proposed in the literature, such as one point, two points and uniform crossover. They all consist in choosing one or more cut points into the parents genetic material and recombining it such that each child inherits something from both parents. An example of one-point crossover is shown in figure 4.4.

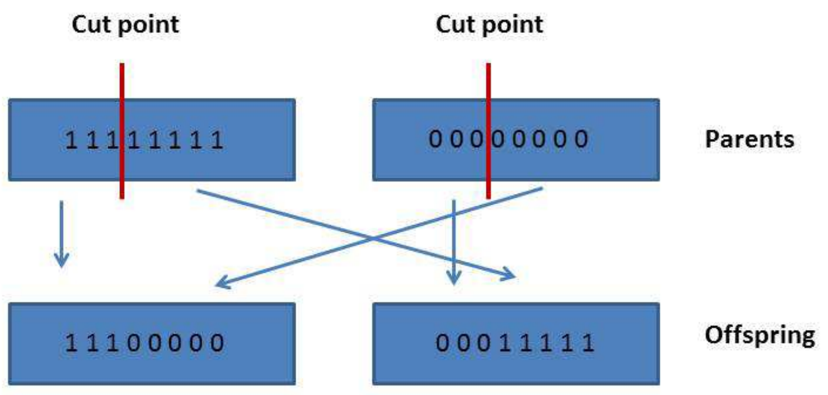


Figure 4.4: One-point crossover

- ***Mutation***

This operator simulates the mutation that happens in the reproduction process. It consists in a small change of some genes and it usually happens with an equal probability for all the genes in a chromosome. Its implementation consists in a random alteration of the genes, each one with a certain probability, whose value is fixed as a parameter of the algorithm. This probability should be not too high in order to avoid a chaotic search.

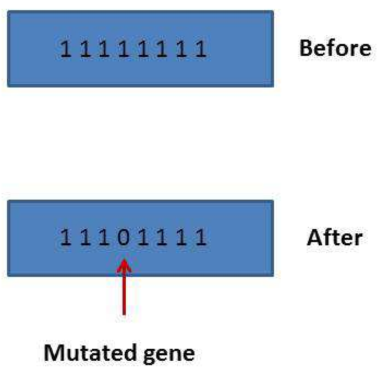


Figure 4.5: Mutation

Finally, after the application of mutation and crossover, the best chromosomes of the population P can disappear, then an **elitist strategy** is usually implemented. This helps to make faster the convergence of the algorithm, but the number of chromosomes to be preserved should be chosen carefully because if it is too high it could cause a premature convergence of the algorithm.

4.3 Multi-Objective genetic algorithms

Most practical problems need the optimization of multiple and often conflicting objectives. These kind of problems are known as "*multi-objective optimization problems*" (MOP). In order to solve these problems, decision makers need not just a single, fix-all-solution, but a set of possible solutions, among which they can choose the one they deem the most appropriate to the application context. The solutions of a MOP are points in the search space that minimizes (or in some cases maximizes) the values of a set of objective functions. [Tamaki, 1996]

Thus, a MOP can be formulated as:

$$\text{Minimize } f(x) = \{f_1(x), f_2(x), \dots, f_n(x)\}$$

where $f_1(x) \dots f_n(x)$ are a set of objective functions.

When dealing at the same time with many objective functions, the notion of "Pareto dominance" has to be introduced in order to compare the different solutions generated. A solution x dominates a solution y if and only if the following conditions hold:

- x is not worse than y for any of the objectives
- x performs better than y at least for one of the objectives

If one of these conditions does not hold, then x does not dominate y.

Starting from this definition, we can define a solution as Pareto-optimal if it is not dominated by any other solution in the search space. The set of Pareto-optimal solutions is denoted as Pareto front. For the sake of clarity, the Pareto front for a problem with two-fold objectives is shown in the following figure:

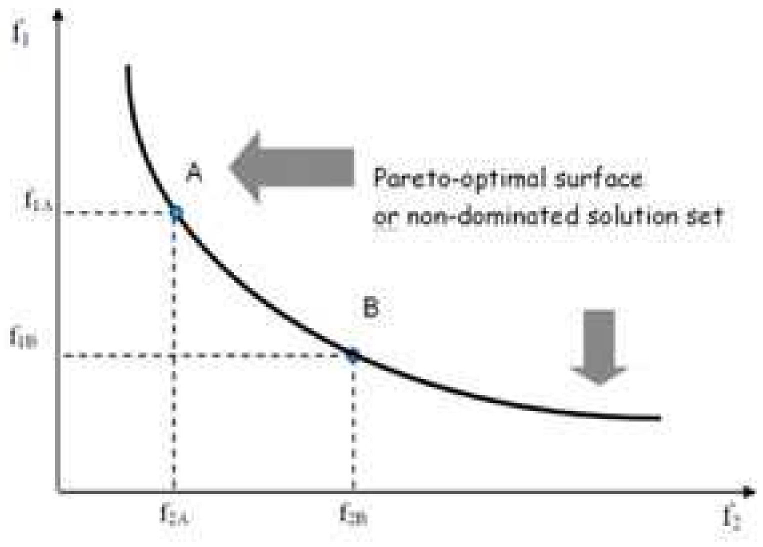


Figure 4.6: Pareto front representation for a two-objectives problem.

Source: University of Sheffield, Automatic Control and Systems Engineering

< http://www.shef.ac.uk/acse/staff/peter_fleming/intromo>

Multi-objective genetic algorithms (MOGA) can be considered a very effective way to search for Pareto-optimal solutions. Their goal is to find a family of solutions that are a good approximation of the Pareto front [Cococcioni, 2007].

In our study, MOGAs are applied for the generation of FRBSs with high interpretability and accuracy, so as to overcome the existing tradeoff. Several algorithms have been proposed in the literature, such as PAES [Knowles, 1999], NSGA [Srinivas, 1994] and its variation NSGA-II [Deb, 2002]. In the following paragraph we will focus our discussion on the PAES algorithm.

4.3.1 Pareto Archived Evolution Strategy (PAES)

The Pareto Archived Evolution Strategy (PAES) was proposed by [Srinivas, 1994]. The aim of the authors was to implement a simple but non-trivial algorithm for generating a good approximation of the Pareto front, by performing a local search in the space of the solutions. The outline of the algorithm is shown in fig. 4.7.

The algorithm is comprised of three main phases:

1. Generation of candidate solution
2. Acceptance of the candidate solution
3. Management of the non-dominated solutions archive

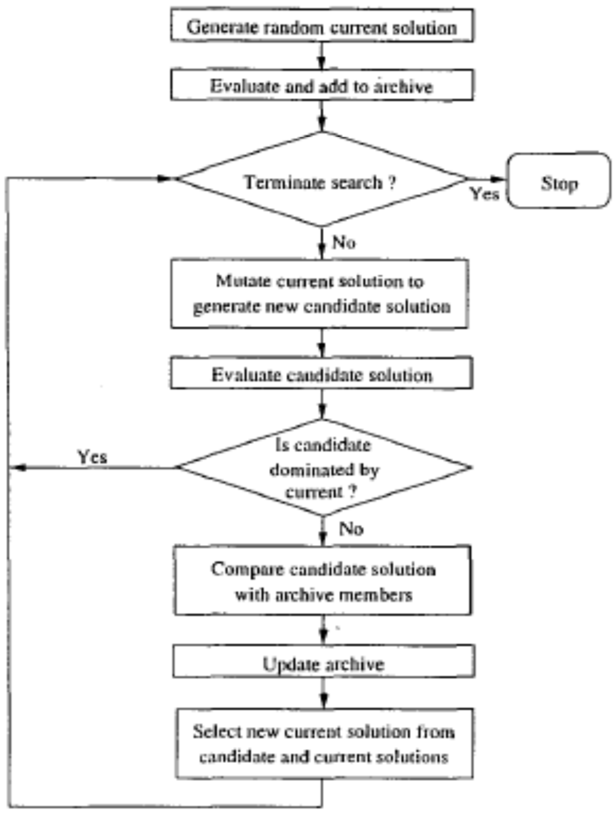

Generate random current solution
Evaluate and add to archive
Terminate search ?
Yes
Stop
No
Mutate current solution to generate new candidate solution
Evaluate candidate solution
Yes
Is candidate dominated by current ?
No
Compare candidate solution with archive members
Update archive
Select new current solution from candidate and current solutions

Figure 4.7: PAES algorithm outline [Srinivas, 1994]

Since the aim of the algorithm is to perform a local search, only the mutation operator is applied to the "current solution" in order to generate the new "candidate solution". The original version considers a single current solution and generates a single candidate solution at each iteration. This explains the appellation "(1+1)PAES". Other variants are proposed in the literature, such as the (1+λ) and the ($\mu + \lambda$) PAES. The former generates λ mutants from the current solution, while the latter maintains a population of μ current solutions and makes λ copies of the fittest ones vis-à-vis the archive. These copies are then mutated to generate the candidate solutions.

Going back to the original version, once the candidate solution is generated and evaluated, it is compared with the current one. If the candidate solution is not dominated by its parent, it is accepted and compared with the solutions within the archive. In such a case, the following conditions might occur:

1. The candidate solution dominates the set of non-dominated solutions. In this case the current solution is always accepted and archived.

2. The candidate solution is dominated by the set of non-dominated solutions. In this case the current solution is always rejected.

3. The candidate solution do not dominate or is not dominated by any solution in the archive. If this is the case, the candidate solution is accepted and/or archived based on the degree of crowding of its grid location.

A complete description of the acceptance and archiving logic is shown in fig. 4.8.

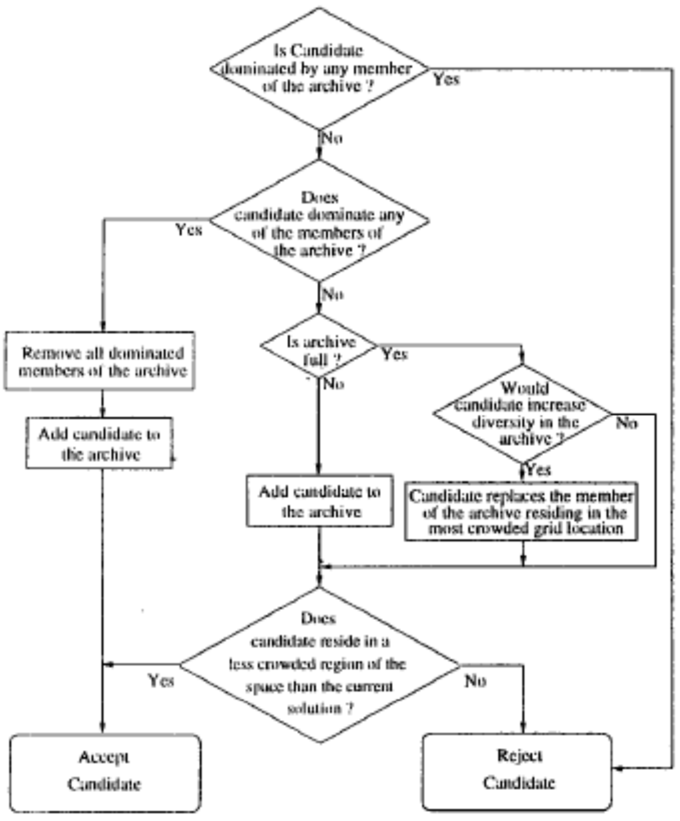

Is Candidate dominated by any member of the archive ?
Yes
No
Does candidate dominate any of the members of the archive ?
Yes
No
Remove all dominated members of the archive
Add candidate to the archive
Is archive full ?
Yes
No
Would candidate increase diversity in the archive ?
No
Yes
Add candidate to the archive
Candidate replaces the member of the archive residing in the most crowded grid location
Does candidate reside in a less crowded region of the space than the current solution ?
Yes
No
Accept Candidate
Reject Candidate

Figure 4.8: Archiving and acceptance logic. Adapted from [Srinivas, 1994]

A grid is used to manage the solutions in the archive, having the latter a maximum size. More in detail, a n-dimensional grid, where n is the number of objectives, allows the partition of the space in a set of n-dimensional cells. Each solution is assigned a cell, depending on the values of the objectives. If a new solution needs to be added to a full archive, the so called "*crowding degree*" is computed by counting the number of solutions belonging to each location. The new solution will replace the previous one, which belongs to the most crowded region of the solution space. This process facilitates the creation of solutions that are uniformly distributed along the Pareto front [Cococcioni, 2007]. A modified version of the PAES, called (2+2)M-PAES [Antonelli, 2009], will be explained in the following chapter.

4.4 Genetic Fuzzy Rule Based Classifier

Genetic fuzzy systems (GFSs) use genetic algorithms to design fuzzy logic systems. Although genetic algorithms do not solve learning tasks, it is possible to model these kind of problems as optimization tasks. Since the main characteristic of GAs is their ability to search in vast and very complex spaces, they can be successfully applied with the aim of designing optimal fuzzy systems.

This study will focus on the application of GFSs for the design of FRBCs. As already analysed in the previous chapter, it is possible to apply GAs to generate and modify automatically the knowledge base. The structure of a GFRBC is shown in fig. 4.9.

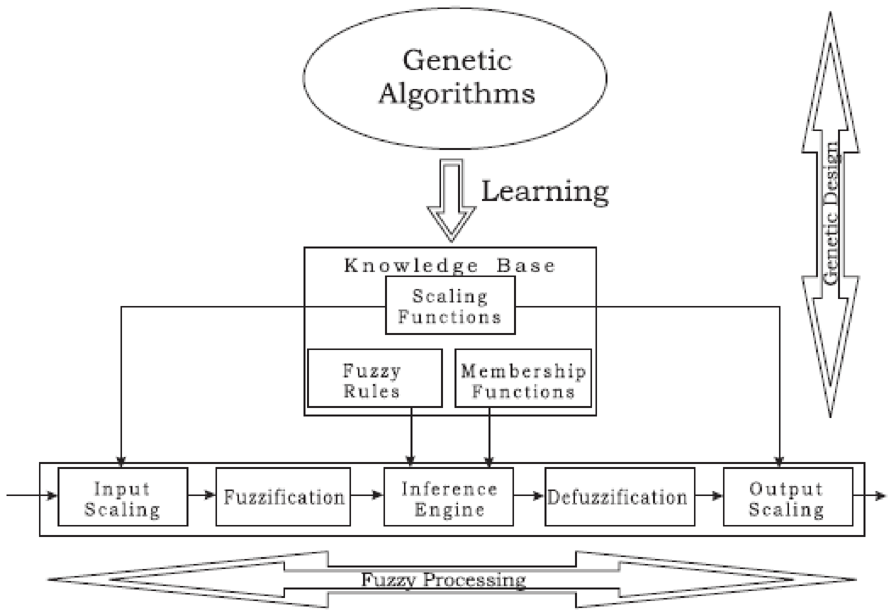


Figure 4.9: Structure of a GFRBC [Cordon, 2004]

It is possible to distinguish three different groups of GFSs according to the KB components that are to be learnt:

1. Genetic learning of the rule base (RB)

2. Genetic tuning of the membership functions (DB)

3. Genetic learning of both components (KB)

The most common approaches to learning process identified in the literature are the Michigan and Pittsburgh approaches. In the former, a chromosome denotes a rule, meaning that a rule set is represented by an entire population; while in the latter, each chromosome is an entire rule base.

The main drawback of the Michigan approach is that it generates a conflict between individual and collective interests: during the evolution, rules compete with each other, in contrast with the final objective that is to identify the best set of rules having, together, the capacity to solve the problem at stake. The Pittsburgh approach, where the competition is between entire rule bases, overcome this limitation, but at the cost of a greater computational complexity. [Herrera, 1997]

4.4.1 Genetic learning of the rule base

The approaches to learn the rule base of an FRBC usually assume a predefined set of membership functions designed with the help of domain experts. Genetic algorithms can be used to automatically generate the rule base component from the data. Two different types of approaches have been proposed by the literature. The rule selection approach assumes that a predefined rule base is already available and the genetic algorithm is applied in order to generate a compact rule base with a good accuracy and interpretability. Then in this perspective, the aim of the genetic algorithm is to search the optimal subset of rules. The first contribution in this field has been provided by

[Ishibuchi, 1995] with a single objective genetic algorithm applied for selecting a small number of linguistic rules from a large amount of candidate rules, previously generated. In this implementation each chromosome represents a rule set and a weighted fitness function is defined in order to take into consideration both the minimization of the number of rules and the maximization of the number of patterns correctly classified. Each chromosome is represented as a string $S = s_1 s_2 \ldots s_N$ where N is the number of candidate rules and the following coding scheme is used:

- $s_j = 1$ the jth rule is included in the rule base
- $s_j = -1$ the jth rule is not included in the rule base
- $s_j = 0$ the jth rule is a dummy rule which means that it has no effect in the classification process.

The main drawback of this approach is that the choice of the weights strongly influence the execution of the algorithm and the quality of the solution generated. This is why, in later works, they proposed two multi objective versions of the algorithm, a two-objective one that considers the same objectives, but separately and a three-objective one that also includes the minimization of the total number of antecedent conditions in order to obtain a better interpretability.

However, the main limitation of rule selection approaches is the necessity of a two-step learning process, in fact, first, the set of all candidate rules has to be generated and then it is possible to apply the genetic algorithm to perform the selection. Moreover the generation of all candidate rules can be computationally heavy in an high dimensional problem.

Then, other alternative approaches to learn the rule base have been investigated in the literature. Another way to address this task is to apply the genetic algorithm to generate the fuzzy rules. In this case the aim of the genetic algorithm is to choose the whole configuration of the rules, i.e. the value of the antecedents of each rule. In this way, it is not necessary to generate all the possible rules, but it is the algorithm that explore all of them by finding the best configuration.

In the next chapters a rule selection and a rule generation algorithm will be comparatively analysed, in terms of computational burden, number of rules and accuracy.

4.4.2 Genetic tuning of the membership functions

This paragraph will focus on the application of GAs for the so called "*tuning*" of the membership functions, which are an important element of the knowledge base. A "tuning" is the process of adjusting the definition points of the membership functions to increase the accuracy of a classifier. As shown in figure 4.10, the decision areas formed through the tuning process effectively reflect the underlying data distribution.

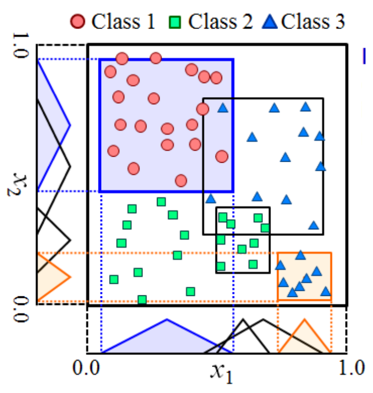


Figure 4.10: Decision areas obtained through the tuning of the membership functions

The genetic tuning process usually assumes a predefined rule base. The logic behind this is to parameterize the membership functions so to obtain a representation of them, then to encode it into a chromosome and let the GA find the best configuration according to a defined fitness function. The most common representation of membership functions is the "*piecewise linear transformation*". It allows to parameterize the membership functions by using a limited number of parameters to finally obtain an efficient representation in terms of memory occupation. Its implementation for triangular membership functions will be discussed more in detail in paragraph 5.4.2.

It is worth noting that the increased accuracy of the model runs in parallel with a degradation of its interpretability. In the final stage of the tuning process, the membership functions might be heavily overlapping and, consequently, it could be very difficult for the decision maker to understand the rules that have been generated. This is why, MOGAs are usually applied to perform the genetic tuning of the membership functions. The two objectives taken into consideration are the maximization of both the accuracy and the interpretability. As already discussed, the interpretability is a subjective concept and researchers do not agree on the application of a single measure. Several partition integrity indexes have been consequently proposed in the literature with a view to assessing the extent of the overlapping among the membership functions, and ultimately the interpretability of the model itself. During the roll-out of the tuning such measures can function as constraint to avoid the generation of overly overlapping partitions.

4.4.3 Genetic learning of the knowledge base

In the previous two paragraphs, different approaches for the genetic learning of the RB component have been discussed, by considering a pre-defined DB and the genetic tuning of DB using a pre-defined rule base. As a complement to this discussion, the benefits of using multi-objective approaches have been also presented.

As reflected in the literature, the two components of the knowledge base are strictly correlated hence the focus on the concurrent learning of both the DB and RB components. This two-pronged approach generally involves a two-parts coding for the chromosomes. A chromosome is composed by two sub-chromosomes that encode separately the two components to be learnt: the first sub-chromosome is composed by the genes representing the rule base, while the other contains an encoding scheme for the configuration of the membership functions. The sub-chromosomes are considered independently as per the crossover and mutation operators application concerned, but are treated as a single entity during the evolutionary process. [Herrera, 1997]

The main drawback of the above mentioned approach is related to the resulting search space, way too large to be handled even by GAs.

The following chapter is dedicated to the MOGAs for the concurrent learning of RB and DB components developed by the researchers of the University of Pisa.

4.5 Beyond the accuracy-interpretability tradeoff

As discussed in chapter 3, the design of a highly accurate and interpretable FRBS is not an easy task, mainly because of the "conflictivity" between accuracy and interpretability. This means that any attempt to improve the accuracy results in a degradation of the interpretability and vice versa. The main advantage of multi-objective approaches over the single objective ones is their ability to generate a family of solutions along the accuracy-interpretability tradeoff surface. [Ishibuchi, 2011] To obtain the same result with a single-objective approach, it is necessary to repeatedly execute the algorithm with different values for the weights of the fitness function. On the contrary, multi-objective approaches allow to achieve the same result with a single execution. Moreover, since in the Pareto front none of the solutions is better than the other, the decision maker can select the final solution according to his/her preferences in the given context application.

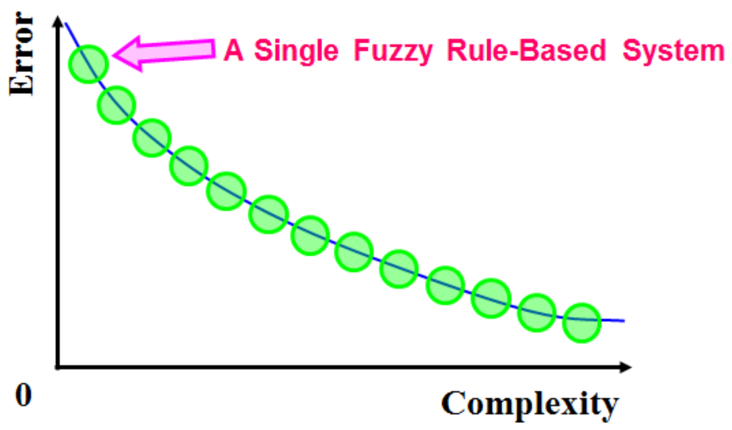


Figure 4.11: Solutions generated by a MOGA [Ishibuchi, 2009]

Since the result of MOGAs is not a single solution but a set of non-dominated solutions, the main drawback of these approaches is to find a way to evaluate and compare their performances. Researchers agreed upon two criteria:

- Minimize the distance between the generated Pareto front and the true one
- Find solutions that are uniformly distributed along the entire tradeoff surface (see fig. 4.11)

Figure 4.12 shows some of the possible distributions of the results generated by an MOGA. It can happen that only a small part of the surface is covered as in fig. 4.12(c) or that the solutions are far from the surface (see fig. 4.12(d)). The solutions to these problems could be, for the situation in 4.12(c), the increase of the diversity among the generated solutions, while for the situation in 4.12(d), the increase of the pressure and/or the computational load in order to approach the solutions to the surface.

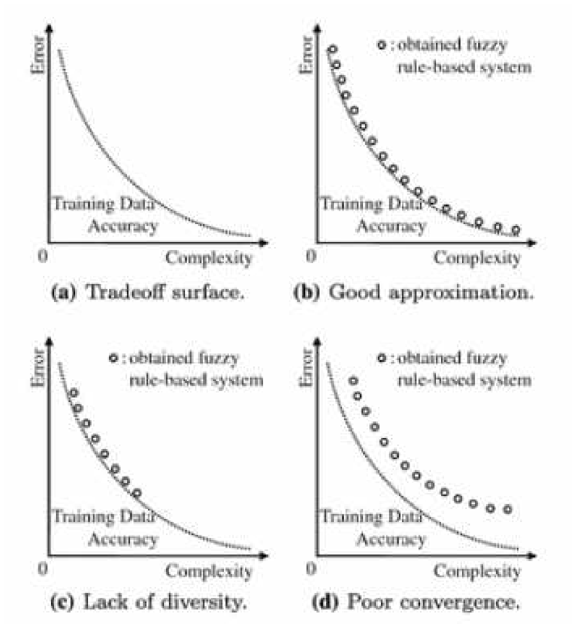


Figure 4.12: Distributions of non-dominated solutions along the tradeoff surface [Ishibuchi, 2001]

However, as pointed out by [Ishibuchi, 2001] the surface is generated considering the accuracy on the training set, whereas the performance of an FRBS should be evaluated with respect to the test set, in order to assess its generalization ability. This means that the ***tradeoff*** between the i***nterpretability*** and the ***test data*** accuracy should be investigated. With respect to the test data accuracy the following situations can occur.

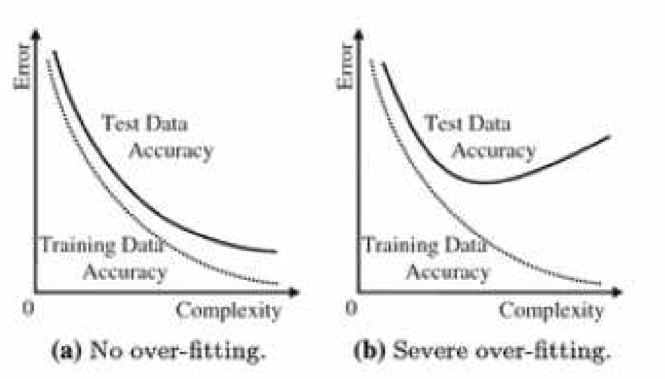


Figure 4.13: Tradeoff between complexity and test data accuracy [Ishibuchi, 2001]

The figure 4.13(a) shows a non-over-fitting phenomenon, which means that an increased accuracy on training set results in an increased accuracy on the test set. If this is the case, solutions should cover all the surface, as in fig. 4.12(b). On the contrary, in fig. 4.13(b) a severe overfitting occurs, meaning that some solutions with high accuracy on training set poorly perform on test set, hence frustrating the need for their generation. If this is the case, the distribution of solutions among the tradeoff surface shown in fig. 4.12(c) might be the most appropriate. Finally, sometimes the solutions in fig. 4.12(d) can have a higher accuracy on test set with respect to the ones in fig. 4.12(b), and therefore a better generalization ability. If this is the case, the approximation of the Pareto front in fig. 4.12(d) is better than the one in fig. 4.12(b).

Finding a reliable and realistic method for assessing the performance of MOGAs has become one of the hot issues and main challenges upon which studies and research have focused and invested in recent years.

4.6 Discussion

In this chapter the GAs were introduced as a tool for the design of accurate and interpretable FRBC. Since the process of automatic definition of a fuzzy system can be considered an optimization task, it is possible to use GAs for this purpose.

GAs are search algorithms capable of solving optimization problems even when the search space is very large and complex. They also display "temporal" benefits, by facilitating the timely detection of sub-optimal solutions. Finally, in order to overcome the accuracy-interpretability tradeoff, the MOGAs have been presented. The advantage of using MOGAs to design FRBS's components is two-fold: they allow to consider both the accuracy and the interpretability during the evolutionary process; and generate a family of solutions which are an approximation of the Pareto front. The final solutions are all equally valid, but each of them represents a different compromise between accuracy and interpretability. Therefore, the decision maker can pick the solution that he/she deems to be the most appropriate vis-à-vis the given application context.

Consequently, all the objectives set at the beginning of this study have been achieved.

More specifically:

1) Automatically design, from the data, the components of an FRBS through the introduction of GAs, resulting in an accurate and easily interpretable model.
2) Increased capacity of the decision maker to understand how the system generates the results and to acquire useful knowledge by means of the linguistic representation and the rule-based structure.

This said, one major drawback affecting the approach proposed, resides in the evaluation and comparison of different multi-objective approaches. In this regard, a new branch of research is currently focusing on the identification of good measures for this purpose.

In the following chapter we will introduce two single-objective and two multi-objective GAs for the design of the rule base of an FRBC. The single-objective approaches are accuracy driven, while the multi-objective approaches also take the interpretability of the final system into consideration.

5 Proposed genetic algorithms

5.1 Introduction

This study is focused on the application of genetic algorithms for the automatic generation of the rule base of a fuzzy rule-based classifier. In this chapter two genetic algorithms, a single objective and a multi objective one are proposed. They both generate directly the rules, without any previous knowledge except from the number and shape of the membership functions which are predefined.

The aim of this chapter is to clarify the implementation of these algorithms and to compare them with two other systems, namely a single objective algorithm based on [Ishibuchi, 1995], and a multi objective algorithm developed by researchers of the Department of Information Engineering at the University of Pisa.

The former is based on a rule selection approach and is accuracy driven, while the latter is able to generate the rules and simultaneously provide a learning of the definition points of the membership functions. In seeking a good tradeoff, the multi objective algorithm takes into account both the accuracy and the interpretability.

5.2 A single objective algorithm for the generation of accurate rule bases

The aim of this algorithm is to go beyond the limitations of the rule selection approach which assumes that a set of candidate rules is defined a priori. The main drawback of this approach is the "*curse of dimensionality*" problem. In fact, it is necessary to generate an amount of rules that grows exponentially with the number of features of the input data. For the sake of clarity, let assume a dataset with 10 features and suppose to use six membership functions for each variable, the number of all possible rules is $6^{10} \cong 6 * 10^7$ which is a huge number. Then [Ishibuchi, 1997] proposed to limit the number of candidate rules by fixing a maximum length, i.e. a maximum number of antecedents for each rule. Even if the length of the rules is less than or equal to three, the problem is not completely solved because the possible combinations in a high dimensional space are computed using the following formula:

$$\binom{n}{k} = \left(\frac{n!}{k!\,(n-k)!}\right) m^k \tag{5.1}$$

where n is the number of features, k the length of the rules and m the number of membership functions for each variable, assuming it is constant for each variable. It is clear that the increase of the dimensions (n) or the length of the rules (k) has a great effect on the number produced by the (5.1). The curse of dimensionality problem, obviously leads to scalability problems, in terms of both computational and memory burden.

The use of genetic algorithms to generate directly the rules avoids this problem and takes full advantage of the strength of the genetic algorithms i.e. their capability to wisely drive the research towards the most interesting areas of the search space even if it is very large and complex.

In the proposed approach a genetic algorithm for the generation of the rules is implemented with the aim of creating an accurate rule base for a fuzzy rule-based classifier.

5.2.1 Encoding scheme

The Pittsburgh approach is used for the learning task, thereby an entire rule base is codified in a chromosome. More in detail, a chromosome is composed by a fixed number of genes, each of them represents a rule. The maximum number of rules per chromosome can be set through a configuration parameter of the genetic algorithm.

A rule is implemented as an object containing the values of the antecedents. The number of antecedents for each rule is equal to the number of features of the input dataset.

Each antecedent can assume the following values:

- **-1** which means a "*don't care*" condition, i.e. the antecedent is removable from the rule.
- An **integer number**, which indicates the selected membership function for that variable, starting from 0.

The “*don’t care*” condition allows to generate rules with only few antecedents, so as to simplify the structure of a rule by making it more generic. This also entails the possibility to reduce the final number of rules, given that the specific rules are embedded into the more generic ones and can therefore be removed from the final rule base.

Finally, the “*don’t care*” condition also increases the interpretability of the rule base as demonstrated by the fact that for human users it is difficult to understand long fuzzy rules, i.e. rules with many antecedents.

Thus, given a dataset composed by seven features and three linguistic labels for each variable, “Low”, “Medium” and ”High”, a gene can be represented as follows:

-1	-1	2	-1	0	-1	1

Figure 5.1: Representation of a gene

The correspondent rule will be:

$$IF\ x_3\ is\ High\ and\ x_5\ is\ Low\ and\ x_7\ is\ Medium\ \ THEN\ \ldots$$

The number of membership functions per variable is fixed by the classifier and can be different for each variable.

The class label is not considered part of the rule during the evolution of the genetic algorithm, but the classifier is used to determine the correct value of the consequent, given the antecedents. In order to assign the class value, the classifier takes confidence and support measures into account [Ishibuchi, 2005].

Finally, it is possible to set the maximum length of the rules as a parameter of the algorithm, i.e. how many antecedents different from the "*don't care*" condition are allowed. Then, rules of different length can be generated during the execution.

5.2.2 Initialization

The initial population is generated by randomly choosing for each rule the position of valid antecedents and also the values they assume. The number of valid antecedents per rule can be set as a configuration parameter of the genetic algorithm. Moreover, each chromosome is generated so that it does not include equal rules.

5.2.3 Operations

The implementation of the main operations for the design of a genetic algorithm are explained in the following sections. For each of them different implementations have been realized.

5.2.3.1 Selection

The selection strategy allows to select the best individuals for the generation of the offspring. Although it is more likely that good individuals, combining their good genetic materials, generate good solutions, this is not always true. On the contrary, the selection pressure should not be too high, so as to give some probability to be selected for the reproduction also to worse individuals. In this way, the diversity among the individuals that are going to produce the offspring is increased resulting in a better exploration of the search space and consequently avoiding the premature convergence of the algorithm to the nearest local optimum. [Al Jaddan, 2005]

In light of the above two different selection strategies have been implemented: the ***tournament selection*** and the ***probabilistic binary tournament selection***. The former includes the choice of the so called “tournament size”, which is the number of random individuals to be picked from the current population. Among them, the chromosome with the best fitness value is chosen and is moved to the mating pool, i.e. the set of individuals selected for crossover. The selection pressure can be adjusted by changing the tournament size, particularly, the increase of the size reduces the possibilities of the weaker individuals to be selected. In the probabilistic binary selection, each “tournament” selects only two individuals and the winner is the one with the greater fitness with a certain probability, otherwise the weaker individual is moved to the mating pool. The tournament probability impacts the selection pressure and can be chosen by setting a configuration parameter for the algorithm.

5.2.3.2 Crossover

Once the selection is performed, the crossover operation is applied to the individuals in the mating pool with a probability fixed as configuration parameter of the algorithm. The most famous types of crossover have been implemented: the one point, the two points and the uniform crossover. All of them are implemented so as the new individuals inherit part of the genes, meaning the rules, from both parents. This implies that the rules are not modified, but only combined differently in the offspring in order to create new rule bases.

The **one point crossover** is implemented by choosing a cut point which is equal for both parents. Let assume the parents and the cut point in the figure:

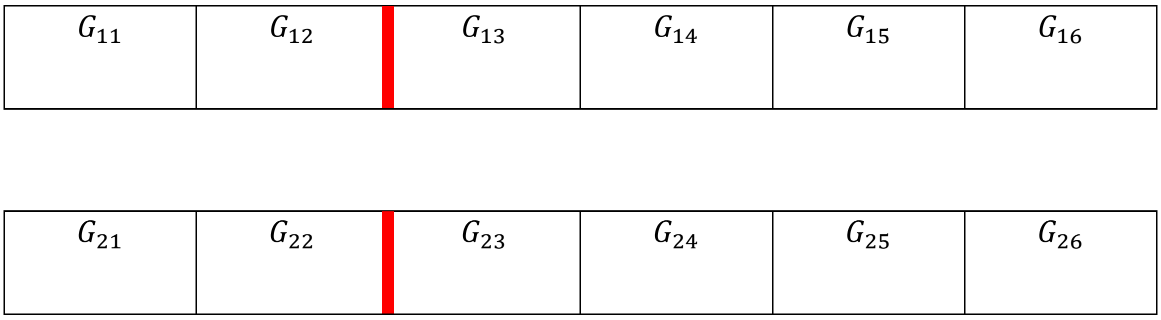

G_{11} G_{12} | G_{13} G_{14} G_{15} G_{16}
G_{21} G_{22} | G_{23} G_{24} G_{25} G_{26}

Figure 5.2: Parents configuration

The offspring will have the following configuration:

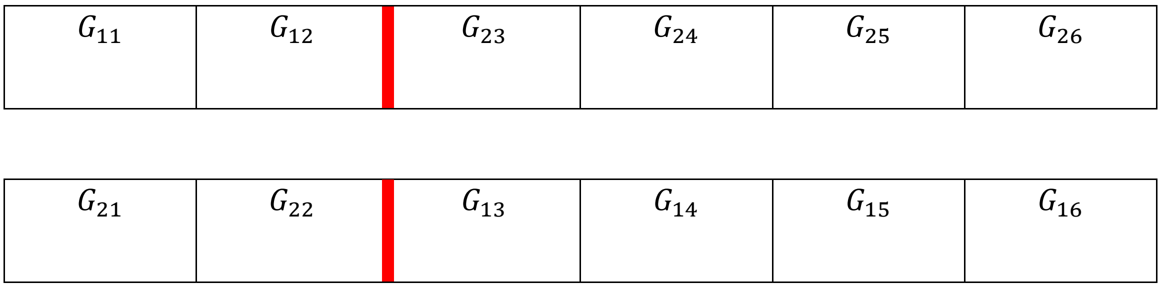

G_{11} G_{12} | G_{23} G_{24} G_{25} G_{26}
G_{21} G_{22} | G_{13} G_{14} G_{15} G_{16}

Figure 5.3: Offspring configuration

In the **two-points crossover**, two different cut points are randomly generated, but some constraints have to be satisfied. If the chromosome is composed of n genes, then the first cut point is randomly chosen in the interval $[1\,, n-2]$, while the second one is chosen in the remaining part of the chromosome. Let assume to have randomly generated the cut points illustrated in the next figure:

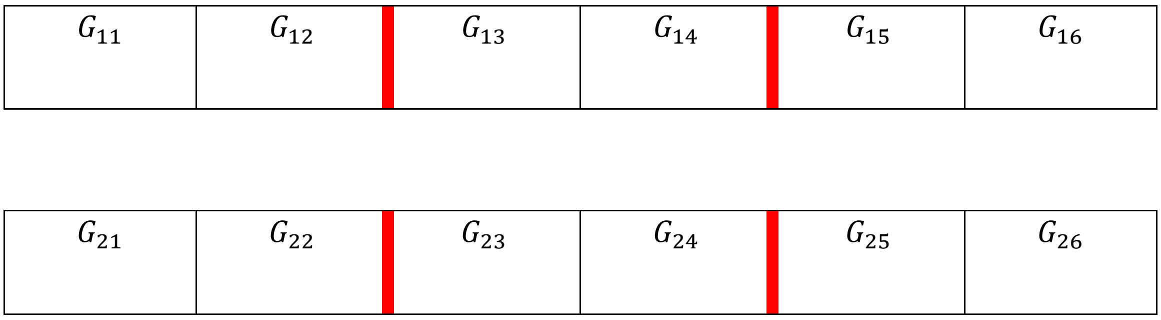


Figure 5.4: Parents configuration

The offspring will be the following:

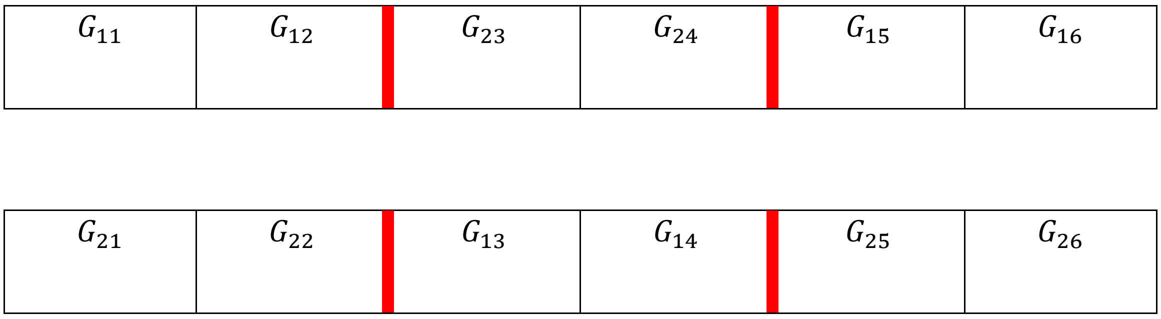


Figure 5.5: Offspring configuration

Finally the **uniform crossover** has been implemented by randomly generating a binary mask whose length is the same as the number of genes in the chromosome. The offspring is generated by choosing, for the first child the ith gene of the first parent if the ith bit of the mask is 1, otherwise the ith gene of the second parent is inherited by the child. For the second child the opposite choice is assumed, i.e. if the ith bit is 1 the ith gene of the second parent is inherited and vice versa.

It is possible to choose which type of crossover operator has to be used by the genetic algorithm by specifying it as a configuration parameter.

5.2.3.3 Mutation

Since the crossover does not change the structure of the rules, the only operator responsible for the creation of new combinations of antecedents is the mutation. Two different types of mutation are used during the evolution process and each of them has a different probability to be executed. The probabilities are fixed as a configuration parameter of the algorithm. In order to change the antecedents configuration, one of the mutation operators replace a valid antecedent with a "*don't care*" condition, while the other one replace any antecedent, i.e. a "*don't care*" or a valid one, with a different linguistic label. More in detail, for each gene in the chromosome the mutation is applied with a certain probability and if it is applied, then one of the two operators is executed. Some constraints need to be satisfied, for instance, the operator that adds "*don't care*" conditions cannot be applied if, as a result, a rule without any valid antecedent would be generated. On the other hand, the operator that adds a valid antecedent cannot change a "*don't care*" condition if the rule has already reached the maximum number of valid antecedents that are allowed.

5.2.3.4 Elitist strategy

As previously mentioned, the elitist strategy avoids that after the application of crossover and mutation the best chromosomes of the previous population disappear. A configuration parameter is available to fix the number of solutions that have to be shifted from the old population to the new one. Since the size of the population is fixed, a replacement becomes necessary, i.e. the chromosomes to be added have to replace some of the individuals in the new population. The chromosomes of the new population that has to be replaced are randomly selected, but they cannot be replaced in order to preserve the best solutions of the new population. The chromosomes of the new population, which are to be replaced, are selected randomly. Nevertheless the best chromosomes of the new population cannot be replaced.

5.2.4 Fitness function

Each chromosome has an associated fitness value which is calculated by the classifier. This is a real number in the interval [0,1] and the best value is 1. The first step to evaluate a chromosome is the translation of the encoding scheme into a rule base. Then, the classifier assigns the class label to each rule, by using the training set and the support and confidence measures. Finally, the goodness of the generated classifier is computed as:

$$f(C) = \frac{TPR + 1 - FPR}{2} \tag{5.2}$$

where TPR is the *true positive rate* and $(1 - FPR) = TNR$ is the *true negative rate*, i.e. the final fitness value is the average recall on the two classes.

5.2.5 Stopping criteria

The genetic algorithm evolves for a predefined number of generations which can be defined by setting the corresponding configuration parameter. Another stopping criteria has been implemented, i.e. if the accuracy of the best solution calculated on the validation set does not improve for a certain number of iterations (fixed by a parameter) then the algorithm is stopped and the best solution on the validation set is selected as final result. This strategy is also useful to avoid that the rule base generated has a good performance on the training set, but a poor generalization ability, i.e. the phenomenon called overtraining.

5.2.6 Additional operations

Some additional operations are necessary to make the algorithm evolve properly.

First of all, since the antecedents of the rules are generated randomly, it can happen that for some of them the classifier is not able to determine the value of the consequent because no training pattern belongs to the part of space identified by the antecedents. In this case these rules are not considered as part of the rule base generated and are ignored when the fitness of the chromosome is computed. Moreover, especially if the maximum number of genes is low, a solution can be composed only by non-valid rules. If this is the case, in order to penalize this individual a fitness value of 0.5 is assigned to it.

It is also important that a rule base has at least one rule for each class. Given that the decision regarding the class label of the rule is taken by the classifier, the genetic algorithm only checks this constraint. If the latter is not satisfied, it assigns automatically a fitness value of 0.5 to the solution as a penalization.

Finally, in order to generate more compact rule bases, the classifier deletes the rules covered by a more general rule. For instance, suppose having these two rules:

$$R_1: IF\ x_1 is\ Low\ AND\ x_3\ is\ High\ AND\ x_4 is\ Low\ THEN\ C_1$$

$$R_2: IF\ x_1 is\ Low\ AND\ \ x_4 is\ Low\ THEN\ C_1$$

Between the two, R_1 is more specific than R_2, the latter containing all the information needed for the classification. Such a case, the classifier will retain only the second rule, while discarding the other one.

5.2.7 Configuration parameters

To sum up, the most important parameters that can be set to configure the genetic algorithm are:

- Population size
- Number of rules per chromosome
- Maximum number of antecedents per rule

- Maximum number of generations
- Stop condition on validation
- Tournament size
- Tournament probability
- Type of crossover
- Crossover probability
- Mutation probability
- Mutation adding a “*don’t care*” probability
- Number of elite solutions

5.3 A multi objective genetic algorithm for designing accurate and interpretable rule bases

The aim of this algorithm is to transform the single-objective approach proposed in the previous paragraph into a multi-objective approach in order to bypass the accuracy-interpretability tradeoff. The main drawback of single-objective algorithms is that, given their accuracy-driven nature, they usually tend to generate complex systems, i.e. rule bases with a high number of rules that are not easily comprehensible to the user. As such, the most important advantage of using a linguistic representation and a rule-based structure, namely the interpretability, is lost. This is why the following multi-objective approach has been implemented. Here the (2+2) M-PAES [Antonelli, 2009] has been implemented to facilitate the comparison with the multi-objective algorithm developed by the University of Pisa.

5.3.1 Encoding scheme

The same encoding scheme used in the single-objective algorithm is implemented. The Pittsburgh approach is also used in this case, implying a chromosome, composed by a pre-defined number of genes, which represent the rules. The maximum number of rules per chromosome is set through a configuration parameter.

The only difference between the two approaches is that in the multi-objective implementation the maximum length of the rules does not represent a constraint. The logic behind this choice is that the minimization of the complexity, expressed as the total number of valid antecedents in the rule base, is one of the objectives of the algorithm. Hence the minimization of the total number of antecedents, during the evolutionary process, is performed by the algorithm that tries to find the best compromise between accuracy and complexity of the rule base.

5.3.2 Objectives and fitness function

The system implemented is a three-objective GA striving for the identification of a good compromise between accuracy and interpretability in the generated FRBC. To achieve this goal, three objectives are taken into consideration: 1) the maximization of the *true* *positive rate* (TPR); 2) the minimization of the *false positive rate* (FPR); and 3) the minimization of the rule base *complexity*. The fitness value associated to each chromosome therefore consists of three different values, stored in the objective vector.

5.3.3 Operations and initialisation

The crossover and mutation operators are the same as in the single-objective approach. The only difference is that if the crossover is not performed, then the mutation is always applied so as to generate different solutions.

As for the initialisation of the solutions, the same strategy used in the single-objective is implemented.

5.3.4 (2+2) M-PAES

A modified version of the well-known (2+2) PAES, denoted as (2+2) M-PAES, has been implemented. The following elements differentiate it from the original version:

- The mutation operator is applied to the current solutions along with the crossover operator;

- The current solutions are changed at each iteration by randomly picking in the archive of non-dominated solutions.

More in detail, the first two solutions are randomly generated. At each iteration, the crossover operator is first applied to the current solutions with a certain probability. Two new solutions, namely candidate solutions, are generated and the mutation operator is applied to those with a certain probability. These solutions are compared with the solutions included in the archive and added to it only if they are dominated by no solution within the archive. If this is the case, all the solutions included in the archive that are dominated by the new solution are removed. Otherwise, if the new solution is dominated by some solution in the archive, it is simply discarded and a new iteration starts by randomly selecting the new current solutions from the archive.

To be noted that since the size of the archive is determined by a configuration parameter of the algorithm, said archive may be filled in some cases. If this is the case and a new solution needs to be added to the archive, this solution will replace the one belonging to the most crowded region of the archive.

5.3.5 Additional operations

Some additional operations are necessary to make the algorithm evolve properly. The solutions composed by only non-valid rules, i.e. those which the classifier is not able to assign the class label to, are penalized by setting the first objective (1-TPR) to 1, the second (FPR) to 0 and the third one (complexity) to a number obtained by considering the maximum number of antecedents for each rule. The same penalization is assigned to those solutions that do not have a rule as a minimum for each class.

5.4 Other approaches to learn the rule base using genetic algorithms

In this section two different approaches to learn a rule base starting directly from the data and using genetic algorithms will be discussed. The first one is a single objective genetic algorithm whose aim is to select a compact subset of rules from a set of all possible rules, in order to find the combination that gives the best accuracy. The second one is a multi-objective genetic algorithm that learns an accurate but also interpretable rule base by randomly generate the set of rules. These algorithms and the proposed ones will be tested on some datasets and compared. The experimental results will be presented in the next chapter.

5.4.1 Rule selection approach

The main idea of the rule selection approach is to generate all the possible rules of a defined length and then select a subset of rules in order to find the combination that maximize the number of patterns correctly classified.

It has to be noted that the number of all possible rules increases exponentially with the number of dimensions. For example, let assume m the number of membership functions of each input and n the number of inputs, the total number of rules N is defined by the following formula:

$$N = m^n \tag{5.3}$$

In order to reduce the number of rules and the amount of memory required to store them, the maximum length of the rules is defined. In [Ishibuchi, 1997] it has been demonstrated that the number of candidate rules is not large if we generate rules whose length is less or equal to three. The length of a rule is given by the number of antecedents that are different from "*don't care*". Let us consider a 10-dimensional pattern problem and suppose to define five membership functions for each variable, the following table shows the total number of rules according to their length:

Length of fuzzy rules	**Number of fuzzy rules**
0	1
1	50
2	1.125
3	15.000
4	131.250
5	787.500
6	3.281.250
7	9.375.000
8	17.578.125
9	19.531.250
10	9.765.625

Table 5.1: Number of fuzzy rules [Ishibuchi, 1997]

The rules generated using this assumption are stored in memory or in a database if the total number is too large to fit in memory. A unique identifier (ID) is also assigned to each rule.

The genetic algorithm follows the Pittsburgh approach, then a chromosome represents a rule base. The number of rules in each solution is defined by a configuration parameter of the genetic algorithm. Each gene contains an integer number representing the ID of the rule. The initial population is created by initialising the genes with random values in the range of the IDs.. The evaluation of the goodness of a rule base is computed by using the same function described in the proposed approach. The crossover operator allows to create new individuals by combining blocks of rules from two parent chromosomes. In order to perform the crossover operation, the population is divided into two halves, which contain the chromosomes with the higher and lower fitness values, respectively. The crossover is implemented by piking one individual from the first half and one from the other. In this way a multithreading strategy can be performed, reducing the execution time. The mutation operator substitutes one of the rules with a random one chosen from the set of all candidate rules. A random number in the range of the ID's is generated and the corresponding rule is added to the chromosome. Finally an elitist strategy is also implemented in order to preserve the best individuals of each generation. The number of elite solutions can be specified by a configuration parameter of the algorithm. Finally, the same stopping criteria described in the proposed single-objective approach are used.

5.4.2 A multi objective algorithm to generate accurate and interpretable rule bases

The multi-objective algorithm developed by the University of Pisa is an implementation of the (2+2) Pareto Archived Evolutionary Strategy (PAES), namely (2+2)M-PAES.

Its overall objective is to generate, through evolutionary algorithms, FRBCs with a good tradeoff between accuracy and interpretability from data. The main characteristic of this approach is that the rules and the membership function parameters are learnt concurrently during the evolutionary process. To do this, a special encoding scheme for the chromosomes is used.

5.4.2.1 Encoding scheme

The Pittsburgh approach is used for the learning task, thereby a chromosome represents an entire rule base. More in detail, a chromosome is composed of two parts: one for the encoding of the rules (C_1) and the other (C_2) for the membership function parameters.

The structure of a chromosome is shown in the following figure.

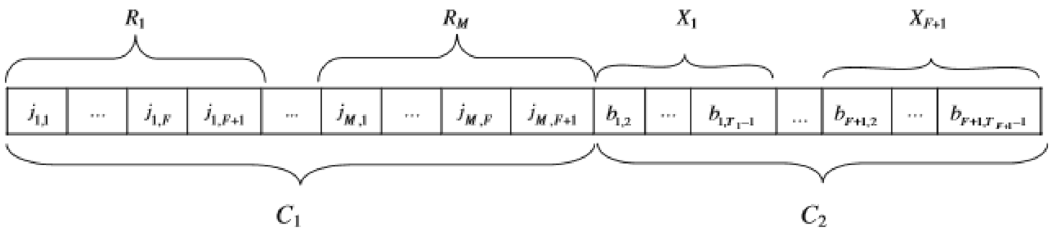


Figure 5.6: Chromosome encoding scheme [Antonelli, 2011]

The encoding scheme assumes that each variable is partitioned by using a certain number (T_f) of triangular membership functions. This number is specified through a configuration parameter of the algorithm. Each rule is coded by using an integer string where each integer represents one of the fuzzy sets of that variable. The "*don't care*" condition is represented by the integer value 0. It is important to note that in this approach also the class label is included in the rule representation and is manipulated by the GA. A chromosome, i.e. a rule base, is obtained by concatenating a certain number of rules. A maximum and a minimum value is defined through configuration parameters.

With reference to the part C_2 of a chromosome, the piecewise linear transformation (PWLT) described in [Klawonn, 2006] and [Pedrycz, 2007] is used in order to obtain a good representation of the membership functions with a limited number of parameters.

More in detail, triangular membership functions are adopted, therefore each fuzzy set can be described by three parameters: (a_f, b_f, c_f). During the tuning process, only the core parameters (b_f) can change their position. Furthermore, in order to preserve the interpretability of the model, adjacent fuzzy sets cannot be overly overlapped. This explains why the position of the b_f parameter of a generic fuzzy set is bounded by the cores of the fuzzy sets next to it.

Given a generic variable X_f, the equally spaced initial partition $\tilde{P}_f = \left\{\tilde{A}_{f,1}, \ldots, \tilde{A}_{f,T_f}\right\}$ and the transformed partition $P_f = \left\{A_{f,1}, \ldots, A_{f,T_f}\right\}$, the piecewise linear transformation can be described as shown in fig. 5.7.

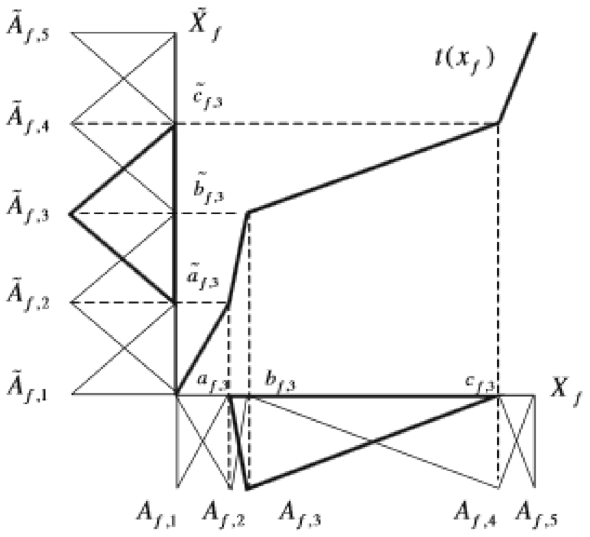


Figure 5.7: Piecewise linear transformation [Antonelli, 2011]

Since only the cores can change their position, they are the only to be included in the representation of the chromosome. In addition, $b_{f,1}$ and b_{f,T_f} are defined and match with the extremes of the variable's universe, so they are not included in the chromosome.

To sum up, the vector $b_f = \left[b_{f,2}, \ldots, b_{f,T_f-1}\right]$ defines the piecewise linear transformation for each variable. The C_2 part of the chromosome is represented by concatenating the b_f vector of all the variables.

5.4.2.2 Objectives and fitness function

As mentioned above, the aim of this algorithm is to generate solutions that are both accurate and interpretable. In order to achieve this goal, three objectives have been identified. The first objective is the minimization of the complexity of the solution. It is measured by summing, the number of valid antecedents for each rule, i.e. the antecedents different from a "*don't care*" condition. The second and third objectives are the maximization of the TPR, *true positive rate*, and the minimization of the FPR, *false positive rate*, respectively. In order to evaluate a solution, a weight is assigned to each rule by using the certainty factor defined by [Cordon, 1999]. Then, the maximum matching method is used as reasoning method to classify the training patterns and compute the TPR and FPR.

Concluding, the fitness value associated to a chromosome consists of three different values, one for each objective. These are stored in a three dimensions vector associated with the chromosome.

5.4.2.3 Operations

The operators are the most important part of the algorithm, because they make it possible the identification of new solutions and the exploration of the search space. Since a chromosome consists of two main parts, different types of crossover and mutation are implemented for each of them.

With regard to the ***crossover*** operator, one-point crossover is applied to C_1, while a BLX-α crossover, with α = 0.5, is applied to C_2. The crossover point is in the same position for both chromosome and it is selected by randomly extracting a number in the interval $[1, \rho_{min} - 1]$, with ρ_{min} as the minimum number of rules in the parent solutions. Moreover the crossover point is chosen between two rules and never within a rule.

As far as the ***mutation*** operator is concerned, three different types of mutation operator have been implemented for C_1. The first one ***adds*** n rules to the rule base. The number of rules to be added is randomly generated, but in such a way that the constraint on the maximum number of rules per solution, established through a configuration parameter, is satisfied.

So, let be M the maximum number of rules and M_{actual} the actual number of rules in the solution, the number of rules to be added is randomly generated in the interval $[1, \gamma_m]$, with $\gamma_m = M - M_{actual}$. Also the configuration of the rules is randomly generated both in terms of number and values of the valid antecedents.

The second mutation operator ***removes*** n rules from the rule base. The number of rules to be removed is randomly chosen, but so that the resulting chromosome has at least a minimum number of rules established through a configuration parameter by the user.

Finally, the third mutation operator ***changes*** n antecedents in the rule base. Both the number, the position and the new values of the antecedents are randomly generated.

As for the mutation operator of C_2, first a variable is randomly selected, then one of the fuzzy sets is randomly chosen, and the related core value (b_{fj}) is changed to a random value in the interval $[b_{f,j-1}, b_{f,j+1}]$.

For all the above mentioned operators, the probability to be executed is defined through configuration parameters.

5.5 Main differences between the approaches described

Before presenting the experimental results, it is worth specifying the main differences between the approaches described in the last paragraphs.

The main characteristics of the algorithms that are going to be compared are shown in the following tables.

	Rule selection approach	**Rule generation approach**
Type of approach	Two-step process: generation of all the possible rules and selection of a subset of them	Random generation of the rules
Encoding scheme	Each gene is an integer number containing the ID of the rule	Each gene is a rule represented by an object containing the value of the antecedents
Class label decision	Decided only once during the generation of the rules using confidence and support measures	Confidence and support measures are computed every time a rule is generated
Number of rules per solution	Fixed during the evolution and reduced only after a post-processing phase	Fixed, but during the evolution non valid rules can be generated so that the solution length can actually change
Mutation operators	Randomly substitute a rule in the chromosome with one of the rules in the set of the candidate rules	Turn a valid antecedent into a "*don't care*" or vice versa

Table 5.2: Comparison between the single-objective algorithms discussed

	Multi-objective algorithm developed in Pisa	**Multi-objective algorithm proposed**
Encoding scheme	Each chromosome is composed of two parts: rules and parameters of the membership functions	Each gene is a rule represented by an object containing the value of the antecedents
Number of rules per solution	Variable and influenced by the mutation operators	Fixed, but during the evolution non valid rules can be generated so that the solution length can actually change
Class label decision	The class label is included into the encoding scheme	Confidence and support measures are computed every time a rule is generated
Shape of membership functions	Triangular but the "*tuning*" process is performed	Triangular and equally spaced
Mutation operators	Can add, change or remove a random number of rules in a solution	Turn a valid antecedent into a "*don't care*" or vice versa

Table 5.3: Comparison between the multi-objective algorithms discussed

6 Experimental results

6.1 Introduction

In this section the approaches to learn the rule base of FRBCs using genetic algorithms described in the previous chapter will be tested on three financial datasets and the results will be compared.

First of all, the characteristics of the datasets will be analysed, then some well-known classifiers will be tested on these datasets in order to obtain some benchmark results with which compare the performance of the described algorithms. Finally, the results of the single objective rule generator approach will be compared with the rule selection one and the proposed multi-objective rule generator approach will be compared with the multi-objective algorithm developed at the University of Pisa.

6.2 Dataset characteristics

The proposed approaches have been tested on three unbalanced datasets representing two-class problems. The characteristics of each of them in terms of distribution of the patterns among the classes and number of features are summarized in the table below. As shown in the table below, the datasets are characterized by different numbers of features and patterns. In this way, the algorithms will be evaluated both on simple and complex datasets.

	Patterns of the majority class	**Patterns of the minority class**	**Total number of patterns**	**Number of features**
Dataset1_trainingSet	764 **(75.94%)**	242 **(24.06%)**	1006	7
Dataset1_testSet	476 **(74.96%)**	159 **(25.04%)**	635	
Dataset1_validationSet	268 **(76.57%)**	82 **(23.43%)**	350	
Dataset2_trainingSet	54606 **(98.50%)**	832 **(1.50%)**	55438	10
Dataset2_testSet	30271 **(98.54%)**	448 **(1.46 %)**	30719	
Dataset2_validationSet	36445 **(98.61%)**	513 **(1.39%)**	36958	
Dataset3_trainingSet	10492 **(78.62%)**	2854**(21.38%)**	13346	20
Dataset3_testSet	3556 **(79.93%)**	893 **(20.07%)**	4449	
Dataset3_validationSet	3161 **(78.97%)**	842 **(21.03%)**	4003	

Table 6.1: Dataset characteristics

6.3 Results of the state-of-the-art algorithms

In this section, the well-known classifiers C4.5, Naïve Bayes and k-nearest neighbours will be tested on the datasets. The training set have been firstly rebalanced by using SMOTE, implemented in WEKA[1]. In order to evaluate the generalization capability of these classifiers the imbalanced test set is used in the evaluation phase.

[1] Weka is an open source software which contains the implementation of the most common machine learning algorithms for data mining tasks. For more information visit http://www.cs.waikato.ac.nz/ml/index.html

The execution options for each algorithm are shown in the table below.

Options	Values
binarySplits	False
collapseTree	True
confidenceFactor	0.25
debug	False
minNumObj	2
numFolds	3
reducedErrorPruning	False
saveInstanceData	False
seed	1
subtreeRaising	True
unpruned	False
useLaplace	False
useMDLcorrection	True

Table 6.2: C4.5 options

Options	Values
KNN	3
crossValidate	False
debug	False
distanceWeighting	No distance weighting
meanSquared	False
nearestNeighbour SearchAlgorithm	*(Default) LinearNNSearch*
windowSize	0

Table 6.3: IB-k options

Options	Values
debug	False
displayModelInOldFormat	False
useKernelEstimator	False
useSupervisedDiscretization	False

Table 6.4: Naive Bayes options

6.3.1 Dataset 1

The characteristics of the rebalanced training set are shown in the table below.

	Patterns of the majority class	Patterns of the minority class	Total number of patterns
Dataset1_BalancedTrainingSet	764	726	1490
Dataset1_testSet	476 **(74.96%)**	159 **(25.04%)**	635

Table 6.5: Distribution of patterns after the execution of SMOTE

The performance values obtained considering the AUC measure are reported in the following table.

	Recall class 0	***Recall class 1***	***AUC***
J48	*96.8%*	*96.9%*	***96.85%***
IB-k	97.9%	98.1%	**98.00%**
Naïve Bayes	60.4%	92.6%	**76.65%**

Table 6.6: Summary of the results

The best result is obtained by the instance based classifier.

6.3.2 Dataset 2

The characteristics of the rebalanced training set are shown in the table below.

	Patterns of the majority class	**Patterns of the minority class**	**Total number of patterns**
Dataset2_BalancedTrainingSet	54606	50752	105358
Dataset2_testSet	30271 **(98.54%)**	448 **(1.46 %)**	30719

Table 6.7: Distribution of patterns after the execution of SMOTE

The performance values obtained considering the AUC measure are reported in the following table.

	Recall class 0	***Recall classe1***	***AUC***
J48	*9.4%*	*98.00%*	***53.7%***
IB-k	24.8%	94.00%	**59.4%**
Naïve Bayes	33.00%	86.8%	**59.9%**

Table 6.8: Summary of the results

The best result is obtained by the probabilistic classifier.

6.3.3 Dataset 3

The characteristics of the rebalanced training set are shown in the table below.

	Patterns of the majority class	***Patterns of the minority class***	***Total number of patterns***
Dataset3_BalancedTrainingSet	10492	9989	20481
Dataset3_testSet	3556 **(79.93%)**	893 **(20.07%)**	4449

Table 6.9: Distribution of patterns after the execution of SMOTE

The performance values obtained considering the AUC measure are reported in the following table.

	Recall class 0	***Recall classe1***	***AUC***
J48	*87.7%*	*19.6%*	*53.65%*
IB-k	70.6%	39.6%	55.1%
Naïve Bayes	51.1%	64.8%	**57.95%**

Table 6.10: Summary of the results

The best result is obtained by the probabilistic classifier.

6.3.4 Summary of the results

The following table shows the best result obtained on each dataset in terms of AUC on test set. These results will be used as benchmarks for the next tests.

	AUC
Dataset1	**98.00%**
Dataset2	**59.9%**
Dataset3	**57.95%**

Table 6.11: Benchmark results

6.4 Comparison between single objective algorithms

The configuration parameters used for the rule selection and the rule generation approaches are shown, respectively, in table 6.12 and 6.13.

Parameters	**Value**
Population size	50
Crossover probability	0.7
Mutation probability	0.07
Max generation number	600
Tournament size	3
Elite solution number	3
Stopping criteria on validation	150

Table 6.12: Configuration parameters Rule Selection approach

Parameters	**Value**
Population size	50
Crossover probability	0.8
Mutation probability	0.2
Add “*don’t care*” probability	0.03
Max generation number	600
Tournament size	3
Elite solution number	3
Stopping criteria on validation	150

Table 6.13: Configuration parameters Rule Generation approach

The following trials have been conducted:

	Number of rules in a rule set	Number of antecedents for each rule
Trial 1	100	3
Trial 2	100	5
Trial 3	300	3
Trial 4	300	5

Table 6.14: Additional configuration parameters

The two single-objective approaches have been compared with respect to the following criteria:

- **Accuracy**, measured in terms of AUC on test set
- **Complexity** of the rule base, measured in terms of number of rules in the final rule base
- **Execution time**

6.4.1 Results

The results obtained in the four trials for each dataset are shown in the following tables.

The best results for each criterion are stressed in boldface.

Dataset 1:

		AUC on Training	AUC on Testing	Number of rules	Execution Time
Trial 1	**Rule selection**	94.47%	**93.91%**	74	121
	Rule generation	94.28%	93.17%	**36**	**104**
Trial 2	**Rule selection**	95.28%	**94.33%**	90	111
	Rule generation	94.28%	93.38%	**26**	**98**
Trial 3	**Rule selection**	92.42%	92.33%	186	205
	Rule generation	93.39%	**92.54%**	**73**	**178**
Trial 4	**Rule selection**	94.78%	**93.70%**	233	165
	Rule generation	93.57%	93.17%	**47**	**123**

Table 6.15: Dataset1 results

Dataset 2:

		AUC on Training	AUC on Testing	Number of rules	Execution Time
Trial 1	**Rule selection**	74.80%	**74.21%**	82	**2208**
	Rule generation	75.65%	74.04%	**35**	3202
Trial 2	**Rule selection**	72.80%	71.80%	98	4155
	Rule generation	74.96%	**73.16%**	**18**	**2215**
Trial 3	**Rule selection**	74.52%	72.99%	125	**4611**
	Rule generation	74.46%	**74.40%**	**40**	5560
Trial 4	**Rule selection**	74.03%	73.42%	262	11226
	Rule generation	75.75%	**74.00%**	**37**	**7876**

Table 6.16: Dataset2 results

Dataset 3:

		AUC on Training	AUC on Testing	Number of rules	Execution Time
Trial 1	Rule selection	63.71%	**61.14%**	93	981
	Rule generation	62.08%	60.79%	**33**	**337**
Trial 2	Rule selection	62.12%	60.89%	100	6240
	Rule generation	62.46%	**61.12%**	**17**	**317**
Trial 3	Rule selection	63.54%	**61.85%**	295	1324
	Rule generation	62.17%	61.70%	**93**	**657**
Trial 4	Rule selection	62.31%	60.20%	300	7022
	Rule generation	61.77%	**61.36%**	**106**	**574**

Table 6.17: Dataset3 results

The following tables show the results taking into consideration a single criterion per time. The results obtained will be discussed in the next section.

Accuracy on test set:

	AUC on Test Set: "Rule Selection" Method			
	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**
Dataset1	93.91%	94.33%	92.33%	93.70%
Dataset2	74.21%	71.80%	72.99%	73.42%
Dataset3	61.14%	60.89%	61.85%	60.20%

	AUC on Test Set: "Rule Generation" Method			
	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**
Dataset1	93.17%	93.38%	92.54%	93.17%
Dataset2	74.04%	73.16%	74.40%	74.00%
Dataset3	60.79%	61.12%	61.70%	61.36%

Table 6.18: Comparison on test set accuracy

<u>Number of rules:</u>

	Number of rules: "Rule Selection" Method			
	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**
Dataset1	74	90	186	233
Dataset2	82	98	125	262
Dataset3	93	100	295	300

	Number of rules: "Rule Generation" Method			
	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**
Dataset1	36	26	73	47
Dataset2	35	18	40	37
Dataset3	33	17	93	106

Table 6.19: Comparison on number of rules

<u>Execution Time:</u>

	Execution Time: "Rule Selection" Method			
	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**
Dataset1	121	111	205	165
Dataset2	2208	4155	4611	11226
Dataset3	981	6240	1324	7022

	Execution Time: "Rule Generation" Method			
	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**
Dataset1	104	98	178	123
Dataset2	3202	2215	5560	7876
Dataset3	337	317	657	574

Table 6.20: Comparison on execution time

6.4.2 Comments

The following conclusions can be drawn from the tables above.

First of all, except for the dataset1, both the algorithms outperform the results obtained by the well-known classifiers in terms of AUC. Consequently they can be considered valid approaches.

Concerning the comparison between the “rule selection” and the “rule generation” approaches, the experimental results suggest the following conclusions.

Regarding the ***accuracy*** on test set, the best value for dataset1 and dataset3 is achieved by the “rule selection” approach. On the contrary, for dataset2, the “rule generation” approach outperforms the other system. Moreover, since the random generation of numbers can influence the evolution of both algorithms and the performance of the final solution, we cannot conclude if one of the systems is better than the other one considering a so small number of trials. It would be interesting to perform a statistical test on a higher number of executions in order to determine if a statistical difference exists in the average results. Anyway, from the experimental results we can conclude that the two approaches are similar in terms of AUC on test set.

Moreover, the ***number of rules*** of the final solution is significantly reduced by the “rule generation” approach. This means that it is able to generate more interpretable rule bases than those generated by the “rule selection” approach. Thus, considering both accuracy and interpretability, we can conclude that the “rule generation” approach manages to find a better compromise than the “rule selection” approach.

Finally, regarding the ***execution time***, the two algorithms show quite different behaviours with respect to the characteristics of the dataset they are tested on. More in detail, both the approaches perform well on datasets with few features and examples, such as dataset1. On the contrary, with large datasets, such as dataset2, the “rule generation” approach shows some difficulties, due to the fact that the “rule generation” approach creates only the antecedents of each rule. The classifier determine the consequent of the rules by computing support and confidence measures that require to take into consideration all the samples of the training set. Therefore, the greater is the number of patterns, the greater is the time necessary to assign the class label to the rules. All these computations are not necessary in the “rule selection” approach, because the rules, including the consequent, are generated in advance and stored in memory or in a database.

On the other hand, the “rule selection approach” performs worse in terms of execution time with high-dimensional datasets, such as dataset3. Even using the constraint on the maximum length of the rules, the number of possible combinations is so large that they do not fit in memory. In order to store them, it is necessary to use a database, but this affect negatively the execution time. The database access is, in fact, significantly slower than the access to the memory.

Concluding, we can summarize the advantages and the drawbacks of the “rule generation” approach with respect to the “rule selection” one, as follows:

Advantages of the “rule generation” approach:

- Since the number of rules generated is always lower using the rule generation approach, we can conclude that it generates ***more interpretable rule bases***.
- Moreover, it can handle ***high dimensional datasets*** better than the “rule selection” approach, in terms of ***execution time.***

Drawbacks of the “rule generation” approach:

- The “rule generation” approach is quite ***slower with large datasets***, cause the choice of the class label of each rule requires to consider the whole training set.
- The ***accuracy*** of the rule base generated by the “rule generation” approach is ***sometimes slightly lower*** than the accuracy produced by the “rule selection” approach. Anyway, this is not always true, as confirmed by the results on dataset2.

6.5 Comparison between multi-objective algorithms

The multi-objective genetic algorithm developed at the University of Pisa and the proposed one have been tested on the datasets whose characteristics are reported in table 6.1. Only training and test sets have been considered, since no stop condition on validation is specified.

The results achieved by the two multi-objective algorithms are compared taking into consideration the following criteria:

- ***Accuracy***, measured in terms of AUC on test set
- ***Number of rules***
- ***Complexity***, i.e. total number of valid antecedents in the final rule base

The configuration parameters of the two genetic algorithms are shown in table 6.21 and 6.22.

Parameters	**Value**
Number of Evaluation	30000
Population length	64
Maximum number of rules	30
Minimum number of rules	2
Maximum number of antecedents	-
Minimum number of antecedents	1
Starting number of rules	30
Prob. RBcross	0.5
Prob. RBmutation	0.01
pmAdd	0.4
pmRemove	0.4
pmModify	0.2
probCrossTuning	0.1
probMutTuning	0.1
TuningPW	1
IndexObj	178
dimVettObj	8
vinc	1
Granularity of digits	100
MAXPARTI	5
MAXPARTO	2

Table 6.21: Configuration parameters multi-objective approach developed in Pisa

Parameters	**Value**
Archive size	64
Crossover probability	0.8
Mutation probability	0.2
Add “don’t care” probability	0.01
Number of Evaluation	30000
Number of rules	30

Table 6.22: Configuration parameters multi-objective approach

It has to be noted that for the algorithm developed in Pisa, the number of fuzzy sets per variable has been reduced to 3 for the dataset3. In fact, using 5 fuzzy sets per variable too many rules with no training pattern belonging to them are generated and the algorithm cannot evolve properly.

On the other hand, as already discussed for the single objective version, the proposed algorithm shows some difficulties with large datasets. In fact, the execution time increases since the classifier needs to do some computations in order to assign the consequent to the rules. This is why, 20000 evaluations, instead of 30000, have been performed for dataset2. Anyway, the reduction of number of evaluations has no effect on the final performance of the algorithm.

6.5.1 Results

In the following paragraphs the results obtained by the multi-objective algorithm developed in Pisa and the proposed one will be presented. For each dataset six trials have been executed, each of them with a different value of the seed for the random function generator. A Pareto front containing the non-dominated solutions is generated from each trial.

6.5.1.1 Multi-objective algorithm developed in Pisa

In the following table the solutions with the highest AUC on the training sets are reported:

	Complexity	# Rules	Total # Features	AUC_TR	AUC_TS
Dataset1	12	5	7	92.59%	92.86%
Dataset2	12	5	8	74.73%	74.46%
Dataset3	7	2	7	60.75%	61.37%

	TPR_TR	TPR_TS	FPR_TR	FPR_TS
Dataset1	97.10%	96.22%	11.91%	10.50%
Dataset2	81.78%	81.25%	32.31%	32.31%
Dataset3	62.82%	64.27%	41.30%	41.53%

Table 6.23: Best solution on training set in terms of accuracy

From the six Pareto fronts generated by the different trials, the non-dominated solutions have been extracted and in the following table the average results achieved by the algorithm are presented.

	AUC_TR	std_dev	AUC_TS	std_dev	TPR_TR	std_dev	TPR _TS	std_dev
Dataset1	88.41%	3.13%	87.7%	3.45%	93.8%	5.13%	92.35%	5.13%
Dataset2	70.92%	5.13%	70.02%	5.46%	77.7%	8.59%	75.74%	8.59%
Dataset3	59.81%	0.79%	59.92%	0.95%	55.23%	7.26%	55.56%	7.26%

	FPR_TR	std_dev	FPR_TS	std_dev	Complexity	std_dev	# Rules	std_dev
Dataset1	16.97%	4.97%	16.95%	4.97%	12.3333	1.2472	5.5	0.5
Dataset2	35.85%	4.4%	35.7%	4.4%	6.8333	2.6087	3	1.1547
Dataset3	35.6%	6.25%	35.72%	6.25%	15	5.9161	3.8333	0.8975

Table 6.24: Average results

6.5.1.2 Proposed multi-objective algorithm

The same trials have been conducted for the multi-objective version of the “rule generation” approach. The solutions with the highest AUC on the training sets are presented in the following table:

	Complexity	# Rules	AUC_TR	AUC_TS
Dataset1	18	9	92.60%	92.34%
Dataset2	25	9	74.97%	74.19%
Dataset3	11	4	60.94%	60.54%

	TPR_TR	TPR_TS	FPR_TR	FPR_TS
Dataset1	93.95%	92.85%	8.75%	8.17%
Dataset2	78.84%	77.00%	28.89%	28.61%
Dataset3	55.58%	55.57%	33.69%	34.49%

Table 6.25: Best solution on training set in terms of accuracy

From the six Pareto fronts generated by the different trials, the non-dominated solutions have been extracted and in the following table the average results achieved by the algorithm are presented.

	AUC_TR	std_dev	AUC_TS	std_dev	TPR_TR	std_dev	TPR_TS	std_dev
Dataset1	91.69%	0.89%	91.71%	0.83%	94.32%	0.58%	93.80%	0.58%
Dataset2	74.63%	0.13%	74.51%	0.25%	80.29%	3.51%	79.87%	3.51%
Dataset3	60.56%	0.46%	60.03%	1.02%	59.43%	6.53%	59.34%	6.53%

	FPR_TR	std_dev	FPR_TS	std_dev	Complexity	std_dev	# Rules	std_dev
Dataset1	10.94%	2.05%	10.38%	2.05%	20.1667	2.7335	8.3333	0.7454
Dataset2	31.04%	3.21%	30.84%	3.21%	19.6667	8.5180	7.0000	2.1602
Dataset3	38.30%	5.83%	39.29%	5.83%	10.8333	2.7335	4.6667	1.1055

Table 6.26: Average results

6.5.2 Comments

As already discussed, it is not easy to compare the results obtained by multi-objective algorithms. In this study, the comparison will be performed, firstly, considering the best solution in terms of AUC on training set generated by the two approaches. Secondly, the average results in terms of AUC, complexity and number of rules, will be compared. Finally, the method proposed by [Alcalà, 2009] is applied to identify the key points of the Pareto fronts. More in detail, since the algorithms have three objectives, first the solutions are projected into the AUC-Complexity plane and then three representative points are extracted. They represents the average of the most, median and least accurate solutions generated in the six trials.

The following tables summarize the best results in terms of AUC on training set obtained by the two approaches.

<u>Dataset1:</u>

	Multi-objective developed in Pisa	Multi-objective proposed
AUC_TR	92.59%	92.60%
AUC_TS	92.86%	92.34%
TPR_TR	97.10%	93.95%
TPR_TS	96.22%	92.85%
FPR_TR	11.91%	8.75%
FPR_TS	10.50%	8.17%
Complexity	12	18
#Rules	5	9

Table 6.27: Best solution on training set for dataset1

Regarding the performances on dataset1, the AUC obtained both on training and test set using the two approaches are similar. However, the solution generated by the proposed approach has more rules and is slightly more complex than the one generated by the other approach.

<u>Dataset2:</u>

	Multi-objective developed in Pisa	Multi-objective proposed
AUC_TR	74.73%	74.97%
AUC_TS	74.46%	74.19%
TPR_TR	81.78%	78.84%
TPR_TS	81.25%	77.00%
FPR_TR	32.31%	28.89%
FPR_TS	32.31%	28.61%
Complexity	12	25
#Rules	5	9

Table 6.28: Best solution on training set for dataset2

Regarding the results on dataset2, the proposed approach generates a solution that is comparable, in terms of AUC, with the solution generated by the algorithm developed in Pisa. Anyway, the best solution of the proposed approach has a higher complexity than the other one.

<u>Dataset3:</u>

	Multi-objective developed in Pisa	**Multi-objective proposed**
AUC_TR	60.75%	60.94%
AUC_TS	61.37%	60.54%
TPR_TR	62.82%	55.58%
TPR_TS	64.27%	55.57%
FPR_TR	41.30%	33.69%
FPR_TS	41.53%	34.49%
Complexity	7	11
#Rules	2	4

Table 6.29: Best solution on training set for dataset3

The results obtained on dataset3 show a similar performance of the two algorithms in terms of AUC on training set. However, on test set the algorithm developed in Pisa outperforms the proposed one. Considering the complexity and the number of rules of the best solution, the results are quite similar.

The following table shows the comparison of the average results in terms of AUC, complexity and number of rules, for both approaches.

		AUC_TR	StdDev	**AUC_TS**	StdDev	**Complexity**	StdDev	**#Rules**	StdDev
Proposed multi-objective	**Dataset1**	**91.69%**	0.89%	**91.71%**	0.83%	20.1667	2.7335	8.3333	0.7454
	Dataset2	**74.63%**	0.13%	**74.51%**	0.25%	19.6667	8.5180	7.0000	2.1602
	Dataset3	**60.56%**	0.46%	**60.03%**	1.02%	**10.8333**	2.7335	4.6667	1.1055
Multi-objective developed in Pisa	**Dataset1**	88.41%	3.13%	87.7%	3.45%	**12.3333**	1.2472	**5.5**	0.5
	Dataset2	70.92%	5.13%	70.02%	5.46%	**6.8333**	2.6087	**3**	1.1547
	Dataset3	59.81%	0.79%	59.92%	0.95%	15	5.9161	**3.8333**	0.8975

Figure 6.1: Comparison between average results

The comparison of the results shown in the table above reveals that the proposed approach tends to create solutions that are more complex, but with an average accuracy greater than the other approach. On the contrary, the algorithm proposed in Pisa manages to find solutions characterized by a lower number of rules and complexity, but to the detriment of the average accuracy of the solutions.

Finally, the method proposed in [Alcalà, 2009] is used to compare the two multi-objective algorithms. In the following figures, three representative points are plotted for each approach. They represents the average of the best, average and worst solutions respectively, in the six Pareto fronts.

Dataset1:

Multi-objective developed in Pisa | **Multi-objective proposed**

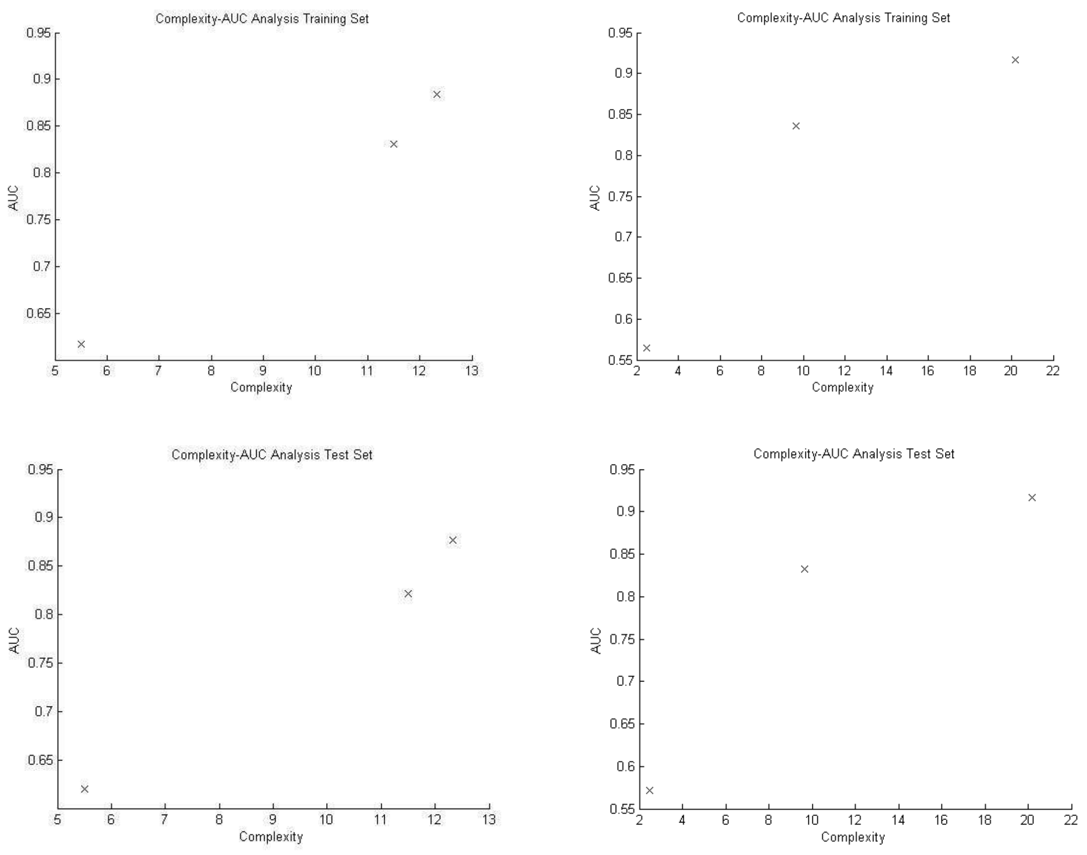


Figure 6.2: Representative points in the complexity-AUC plane for dataset1

<u>Dataset2:</u>

Multi-objective developed in Pisa **Multi-objective proposed**

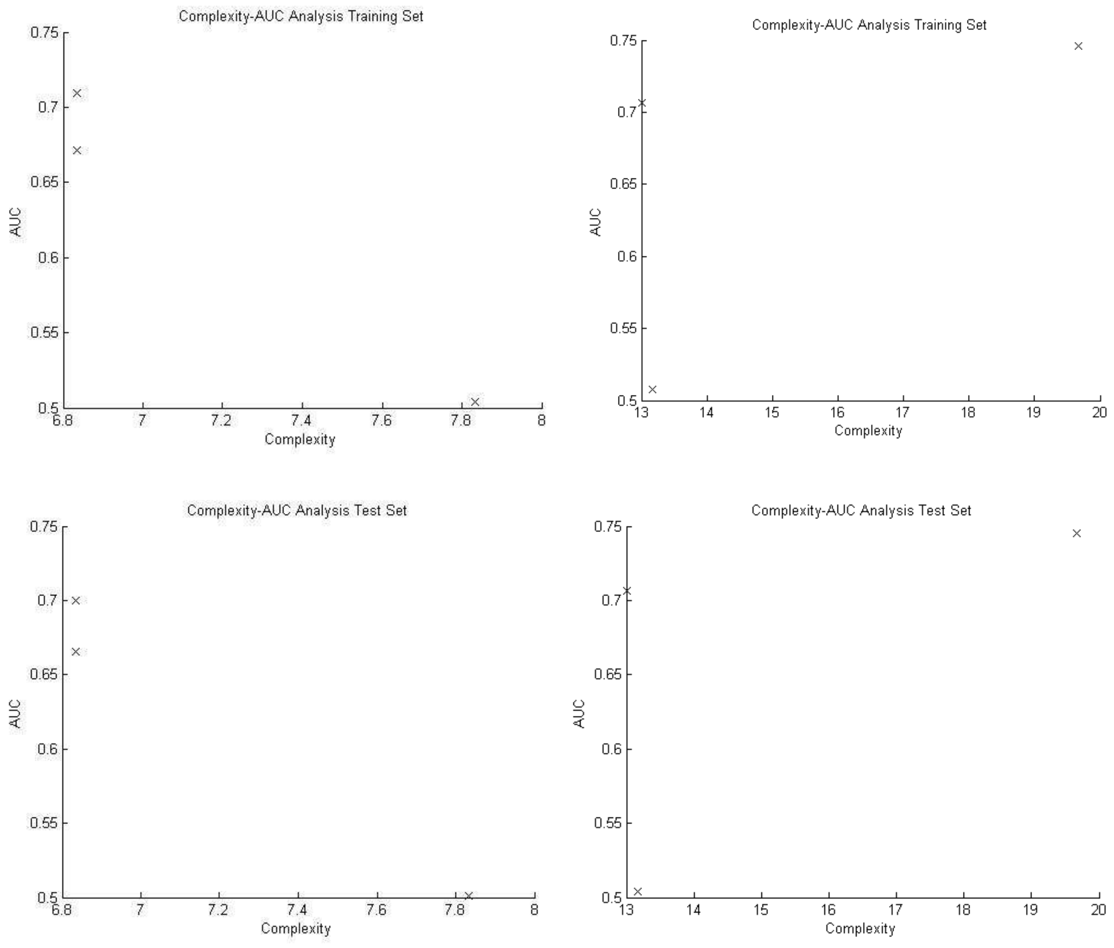


Figure 6.3: Representative points in the complexity-AUC plane for dataset2

Dataset3:

Multi-objective developed in Pisa **Multi-objective proposed**

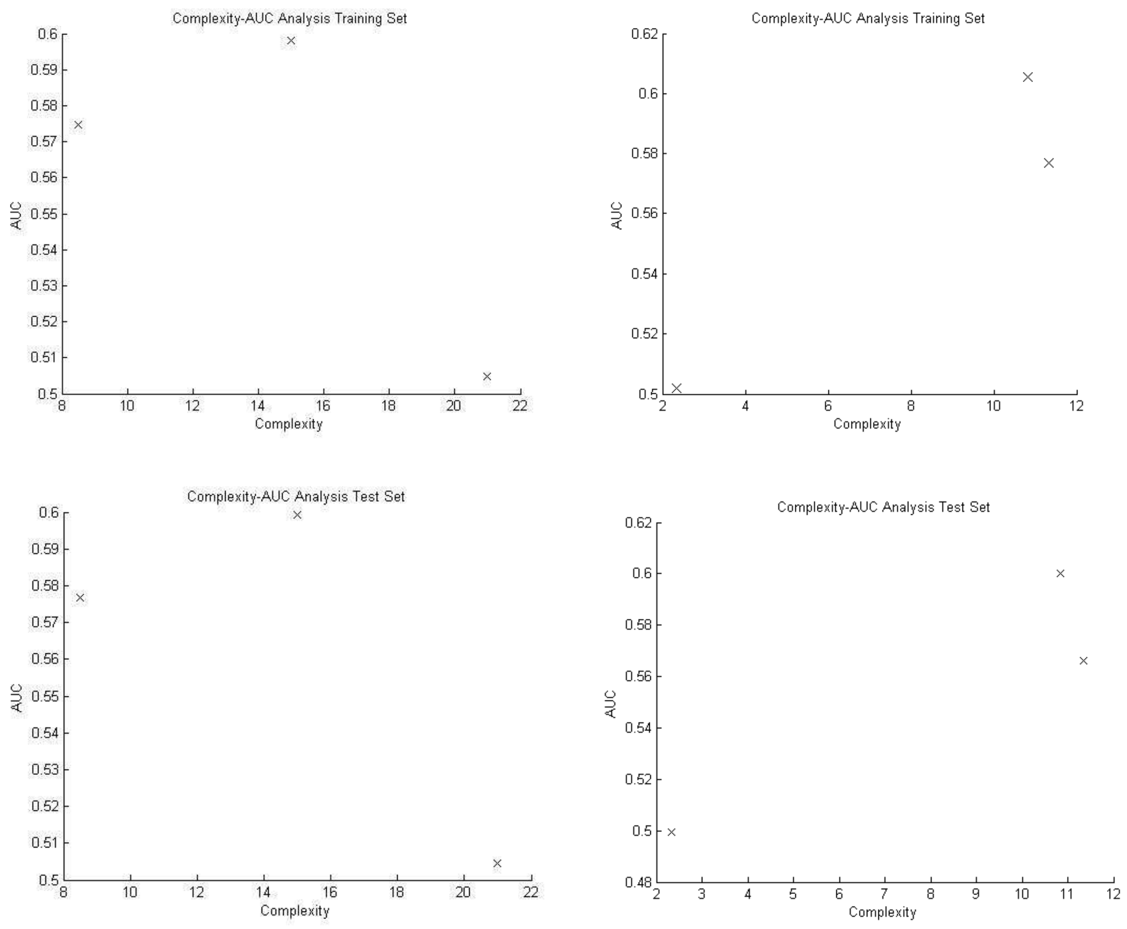


Figure 6.4: Representative points in the complexity-AUC plane for dataset3

From these figures we can see that the proposed approach manage to find average solutions characterized by a greater accuracy, except for the dataset3, but this improvement results in a higher complexity of the solutions.

6.6 Conclusions and future work

In this study we have developed a single objective and a multi-objective genetic algorithm for the automatic learning of the rule base component of a fuzzy rule based classifier.

The single objective algorithm is an accuracy-driven approach that takes into consideration the interpretability of the final model by defining a maximum length for the rules. The application of this algorithm helps overcome the main drawback of “rule selection” approaches, namely the necessity of generating all the possible rules in advance. As such its application is also suitable for the resolution of high dimensional classification problems.

The proposed single objective algorithm and a “rule selection” approach have been applied to three financial datasets. The final solutions generated by the two systems have been compared considering the accuracy, the number of rules and the time of execution of the algorithm. The experimental results show that the models created through the “rule generation” approach are similar to those generated through the “rule selection” approach, in terms of accuracy, but they involve a smaller number of rules and, consequently, a greater interpretability. With regard to the time of execution, the “rule selection” approach performs better than the proposed approach when applied to large datasets, while the opposite is true when it is applied to high dimensional datasets.

Moreover, a multi-objective approach has been introduced to overcome the accuracy-interpretability tradeoff. During the evolutionary process, three objectives are taken into consideration: TPR, FPR and number of valid antecedents in the rule base. The proposed algorithm has been applied to the same datasets used for the single-objective algorithm. The experimental results show that through the multi-objective approach it is possible to obtain very compact rule bases with a lower complexity. This approach has been compared with another three-objective algorithm developed by researchers of the University of Pisa. Said algorithm is able to learn the rule base and the definition points of the membership functions simultaneously. The results of the research show how the approach developed in Pisa prompts rule bases that are slightly more accurate and interpretable than the multi-objective algorithm proposed.

Concluding, the following windows of opportunity for improvement and further investigation are suggested:

- Regarding the single-objective approach, the main drawback identified is related to the time required for the execution of the algorithm on large datasets. This is due to the fact that the class label of each rule has to be decided by the classifier. In order to overcome this limitation it would be interesting to assess the effect that the random choice of the class label produces on the accuracy of the final model.
- With regard to the multi-objective approach, it would be recommended to explore the effects of a representation that includes chromosomes with a variable length, i.e. a number of rules that can vary during the evolutionary process.
- Finally, the effect produced by more sophisticated mutation operators capable of perform a better exploration of the search space both in terms of complexity and accuracy, likewise forms the basis for future research and investigation.

Bibliography

[Abe, 1995] Abe S., "*A method for fuzzy rules extraction directly from numerical data and its application to pattern classification*", IEEE Transactions on fuzzy systems, Vol.3, No.1, February 1995

[Al Jaddan, 2005] Al Jaddan O., Rajamani L. and Rao C.R. 2005. "*Improved Selection Operator for GA*". Journal of Theoretical and Applied Information Technology, 269–277

[Alcalà, 2009] Alcalá R., Ducange, P., Herrera, F., Lazzerini, B., and Marcelloni, F. (2009). "*A multi-objective evolutionary approach to concurrently learn rule and data bases of linguistic fuzzy-rule-based systems*". Fuzzy Systems, IEEE Transactions on, 17(5), 1106-1122.

[Antonelli, 2009] Antonelli M., Ducange P., Lazzerini B. and Marcelloni F. (2009). "*Learning concurrently partition granularities and rule bases of Mamdani fuzzy systems in a multi-objective evolutionary framework*". International Journal of Approximate Reasoning, 50(7), 1066-1080.

[Antonelli, 2011], Antonelli M., Ducange P., Lazzerini B. and Marcelloni F., "*Learning knowledge bases of multi-objective evolutionary fuzzy systems by simultaneously optimizing accuracy, complexity and partition integrity*." Soft Computing 15.12 (2011): 2335-2354.

[Bal, 2011] Bal M., Bal Y., Demirhan A. "*Creating competitive advantage by using data mining technique as an innovative method for decision making process in business*". Annual Conference on Innovations in Business & Management London, UK, 2011

[Bellman, 1961] Bellman R., “*Adaptive Control Processes: A Guided Tour*”. Princeton University Press, 1961.

[Berry, 1997] Berry M. J. A. and Linoff G., “*Data mining techniques for marketing, sales and customer support*”. USA: John Wiley and Sons.

[Chou, 2000] Chou C. H., and Chen J. N., "*Genetic algorithms: initialization schemes and genes extraction*". Fuzzy Systems, 2000. FUZZ IEEE 2000. The Ninth IEEE International Conference on. Vol. 2. IEEE, 2000.

[Cococcioni, 2007] Cococcioni M., Ducange P., Lazzerini B. and Marcelloni F. "*A Pareto-based multi-objective evolutionary approach to the identification of Mamdani fuzzy systems*". Soft Computing 11, no. 11 (2007): 1013-1031.

[Cordòn, 1999] Cordòn O., de1 Jesus M. J., Herrera F., “*A proposal on reasoning methods in fuzzy rule-based classification systems*”, International Journal of Approximate Reasoning 20 (1999) 2145

[Cordon, 2001] Cordón O., Herrera F. and Villar P. “*Generating the Knowledge Base of a Fuzzy Rule-Based System by the Genetic Learning of the Data Base*” IEEE TRANSACTIONS ON FUZZY SYSTEMS, VOL. 9, NO. 4, AUGUST 2001

[Cordon, 2004] Cordon O., Gomide F., Herrera F., Hoffmann F. and Magdalena L. "*Ten years of genetic fuzzy systems: current framework and new trends*". Fuzzy sets and systems 141, no. 1 (2004): 5-31.

[Deb, 2002] Deb K., Pratap A., Agarwal S. and Meyarivan T. A. M. T. "*A fast and elitist multiobjective genetic algorithm: NSGA-II*". Evolutionary Computation, IEEE Transactions on 6, no. 2 (2002): 182-197.

[Donoho, 2000] Donoho D.L., "*High-Dimensional data analysis: the curses and blessings of dimensionality*". Department of Statistics, Stanford University, August 2000.

[Fayyad, 1996] Fayyad U., Shapiro G. P. and Smyth P., "*From data mining to knowledge discovery in databases*". AI Magazine, 17(3):37-54, Fall 1996.

[Guerra, 2012] Guerra M. L. and Sorini L., "*Incorporating Uncertainty in Financial Models*", Applied Mathematical Sciences, Vol. 6, 2012, no. 76, 3785 – 3799

[Guillaume, 2001] Guillaume S., "*Designing fuzzy inference systems from data: an interpretability-oriented review*" IEEE Trans Fuzzy Syst. 9(3): 426-443

[Han, 2001] Han J., Kamber M., "*Data mining: concepts and techniques*", Morgan Kaufmann, San Francisco, CA (2001)

[Herrera, 1997] Herrera F. and Magdalena L., "*Genetic Fuzzy Systems: a tutorial*" Tatra Mountains Mathematical Publications Vol.13, 1997, 93-121. Mesiar, B. Riecan (Eds.) Fuzzy Structures. Current Trends.

[Ishibuchi, 1992] Ishibuchi H., Nozaki K. and Tanaka H., "*Distributed representation of fuzzy rules and its application to pattern classification*", Fuzzy Sets Systems 52 (1992) 21-32

[Ishibuchi, 1995] Ishibuchi H., Nozaki K., Yamamoto N. and Tanaka H., "*Selecting fuzzy if-then rules for classification problems using genetic algorithms*". IEEE Transactions on fuzzy systems, Vol. 3 No. 3 August 1995

[Ishibuchi, 1997] Ishibuchi H. and Murata T., "*Minimizing the fuzzy rule base and maximizing its performance by a multi-objective genetic algorithm*" Fuzzy Systems, 1997., Proceedings of the Sixth IEEE International Conference on. Vol. 1. IEEE, 1997.

[Ishibuchi, 2005] Ishibuchi H., Yamamoto T. (2005) "*Rule weight specification in fuzzy rule-based classification systems*". IEEE Trans Fuzzy Systems 13(4):428–435.

[Ishibuchi, 2009] "*Multi-objective Genetic Fuzzy Systems*" Available: <http://sci2s.ugr.es/gfs/ppt/ISDA-2009_PlenaryTalk_Hisao-Ishibuchi.pdf> Last accessed 26 May 2013

[Ishibuchi, 2011] Ishibuchi H., Nakashima Y. and Nojima Y. "*Performance evaluation of evolutionary multi-objective optimization algorithms for multi-objective fuzzy genetics-based machine learning*". Soft Computing 15.12 (2011): 2415-2434.

[Klawonn, 2006] Klawonn F., "*Reducing the number of parameters of a fuzzy system using scaling functions*". Soft Computing 10(9):749–756 (May 2006)

[Knowles, 1999] Knowles J. and Corne D. "*The Pareto archived evolution strategy: A new baseline algorithm for Pareto multi-objective optimisation*". Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on. Vol. 1. IEEE, 1999.

[Lazzerini, 2009] Lazzerini B., "*Computational Intelligence*", Lecture, University of Pisa. Unpublished

[Lopez, 2012] Lopez V., Fernandez A., Moreno-Torres J. C., Herrera F., "*Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. Open problems on intrinsic data characteristics*". Expert system with applications 39 (2012) 6585-6608

[Mendel, 1995] Mendel J., “*Fuzzy logic systems for engineering: a tutorial*”. Proceedings of the IEEE, 83(3):345-377, March 1995.

[Pedrycz, 2007] Pedrycz W., and Gomide F., “*Fuzzy systems engineering: toward human-centric computing*”. Wiley-IEEE Press, 2007.

[Robles-Granda, 2010] Robles-Granda P. D. and Belik I. V., “*A Comparison of Machine Learning Classifiers Applied to Financial Datasets*”. Proceedings of the World Congress on Engineering and Computer Science 2010 Vol. I WCECS 2010, October 20-22, 2010, San Francisco, USA

[Singh, 2012] Singh H., “*Implementation benefit to Business Intelligence using data mining techniques*”. International Journal of Computing & Business Research ISSN (Online): 2229-6166

[Srinivas, 1994] Srinivas N., and Deb K., "*Multiobjective optimization using non-dominated sorting in genetic algorithms*". Evolutionary computation 2.3 (1994): 221-248.

[Tamaki, 1996] Tamaki H., Kita H. and Kobayashi S., "*Multi-objective optimization by genetic algorithms: A review*". Evolutionary Computation, 1996., Proceedings of IEEE International Conference on. IEEE, 1996.

[Taylor, 1986] Taylor S., “*Modelling Financial Time Series*”. New York: Wiley, 1986.

[Wang, 1992a] Wang L. X. “*Fuzzy systems are universal approximators*”. Proc. IEEE Int. Conf. on Fuzzy Syst., San Diego, CA, 1992

[Wang, 1992b] Wang L. X. and Mendel J. M., "*Generating Fuzzy Rules by Learning from Examples*", IEEE Transactions Systems, Man and Cybernetics, 22(6), 1414-1427,1992

[Zadeh, 1965] Zadeh L.A., "*Fuzzy sets*". Information and Control, Volume 8, Issue 3, June 1965, Pages 338–353

[Zadeh, 1973] Zadeh L.A., "*Outline of a new approach to the analysis of complex systems and decision processes*". IEEE Transactions on systems, man, and cybernetics, vol. smc-3, No. 1, January 1973, Pages 28-44

[Zhang, 2004] Zhang D. and Zhou L. "*Discovering golden nuggets: data mining in financial application*". IEEE Transactions on systems, man, and cybernetics—Part C: Applications and reviews, vol. 34, No. 4, November 2004.