NEW CLASSIFICATION METHODS FOR GATHERING PATTERNS IN THE CONTEXT OF GENETIC PROGRAMMING

By

Alma Lilia Garcia Almanza

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
DEPARTMENT OF COMPUTING AND ELECTRONIC SYSTEMS
UNIVERSITY OF ESSEX
WIVENHOE PARK, COLCHESTER, CO2 3SQ, UK
JULY 2008

Abstract

Machine learning techniques extend the past experiences into the future. However, when the number of examples in the minority class (positive cases) is very small in comparison with the remaining classes, it poses a serious challenge to the machine learning [63],[119],[5],[81]. In this kind of problems, the prediction of the majority class is favoured because it has a high chance of being correct. This characteristic is present in many real-world problems, whose objective is to classify the minority class in imbalanced data sets. However, a prediction that detects more positive cases may be paid for with more false alarms. It is important to determine a balance between the detection of positive cases and false alarms. A range of classifications would give users the option to choose the best tradeoff between detecting positive cases and false alarms according to their requirements. On the other hand, we consider it is important to provide a comprehensive solution, which shows the real variables and conditions in the prediction. Thus, the users could combine their knowledge in order to make a more informed decision.

In this thesis, we present three novel approaches: Repository Method (RM), Evolving Decision Rules (EDR) and Scenario Method (SM). We use Genetic Programming (GP) and supervised learning to build the methods proposed in this thesis. The main objectives of RM and EDR are: to predict the minority class in imbalanced environments, to generate a range of solutions to suit different users' preferences and to provide an comprehensible solution for the user. On the other hand, SM has been designed to improve the precision and accuracy of the prediction. However, such improvement is paid for with a decrease in the recall. But, the users have to make the decision of which of these parameters is more adequate to satisfy their needs.

This work is illustrated predicting future opportunities in financial stock markets. Experiments of our methods were carried out, and these showed promising results for achieving our goals. RM and EDR were compared to a standard Genetic Programming, EDDIE-Arb and C5.0.

The methods presented in this thesis can also be used in other fields of knowledge, these should not be limited to financial forecasting problems.

To my parents: Angeles Almanza and Francisco Garcia
and specially to Angely

Acknowledgements

I would like to thank Edward Tsang, my supervisor, for his patience, suggestions and constant support during this research. I am also thankful for his guidance through the early years of chaos and confusion.

I also want to thank to the Mexican council *Consejo Nacional de Ciencia y Tecnologia* for supporting my PhD studies in the University of Essex and specially to all those Mexicans, whose taxes made possible my scholarship.

I would like to acknowledge all people from *Banco de Mexico* who supported me: Ing. Fernando Castaeda, Dr. Eduardo Jallath, Ing. Octavio Berges and Moises Rivero

I also want to thank to some good friends, who kindly accepted to proof read my thesis: Mick, Marquitos, Edgar Galvan, Juan Pablo, Biliana, Serafin, Edgar Ramirez.

Finally, I wish to thank to some special friends, whose support was crucial during this time: Alan Nixon, Olivia, Lupita Ocampo, Catita and Marco.

Colchester, England — Alma Lilia Garcia Almanza
October, 2007

Related works

Journals

Garcia-Almanza A.L. and Tsang E.P.K, Evolving decision rules to predict investment opportunities, International Journal of Automation and Computing 05(1) January, 2008, pag 22-31

Garcia-Almanza A.L. and Tsang E.P.K, Detection of Stock Price movements Using Chance Discovery and Genetic Programming, Knowledge-Based and Intelligent Information and Engineering Systems Journal, 2006

Book contributions

Garcia-Almanza A.L., Tsang E.P.K and Galvan-Lopez E., Evolving Decision Rules to Discover Patterns in Financial Data Sets in Computational Methods in Financial Engineering, 2007

Conferences

Garcia-Almanza A.L. and Tsang E.P.K, Simplifying Decision Trees Learned by Genetic Algorithms, in proceeding of CCE IEEE World Congress on Computational Intelligence, 2006

Garcia-Almanza A.L. and Tsang E.P.K, The Repository Method for Chance Discovery in Financial Forecasting, in proceedings of KES2006 10th International Conference on Knowledge-Based and Intelligent Information and Engineering Syst, 2006

Garcia-Almanza A.L. and Tsang E.P.K, Forecasting stock prices using Genetic Programming and Chance Discovery, 12th International Conference On Computing In Economics And Finance, 2006

Edward Tsang, Sheri Markose, Hakan Er and Alma Garcia, "EDDIE for Discovering Arbitrage Opportunities", in proceedings in the International Conference on Numerical Methods for Finance, 2006

Garcia-Almanza A.L. and Tsang E.P.K, "Repository method to suit different investment strategies" IEEE Congress in evolutionary Computation, 2007

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine Learning (ML) is part of the Computer Science field. Its objective is to design and develop algorithms and techniques that allow computers to "learn". ML provides a wide range of algorithms, such as decision trees, support-vector machines, neural networks, nearest neighbor algorithms, Naive Bayes and the evolutionary paradigm [83]. In recent years ML techniques have been used extensively to solve a wide variety of real-world problems. In the specific case of the classification problem, ML techniques extend the past experiences into the future. The algorithm is trained using a data set of examples. However, when the number of examples of the minority class is very small in comparison with the remaining classes, it poses a serious challenge to ML [63],[119],[5],[81]. In rare event detection, the prediction of the majority class is favoured because it has a high chance of being correct. This characteristic is present in many real-world problems, whose objective is to classify the minority class in imbalanced data sets, for example: fraud detection [38], detection of oil spills in satellite

radar images [71], medical diagnosis [51], [118] and many other applications. To illustrate the imbalanced class problem, consider a data set that classifies two classes (positive and negative). The total number of examples in the data set is 100 cases, where 3 cases are positive and the remaining are negative, if the classifier predicts all the cases as negative, this is classifying correctly 97% of the total predictions. However, it missed all the positive cases, which is the objective of the classification. Obviously a prediction that detects more positive cases may be paid with more false alarms. It is important to determine a balance between the detection of positive cases and the false alarms. A range of classifications would give users the option to choose the best balance between detecting positive cases and false alarms according to their requirements. This characteristic is very important because a single classification could not be suitable for a specific user. In contrast a range of classifications will be able to satisfy users with different risk-guidelines. Another important advantage is to provide a comprehensive solution that shows the real variables and conditions in the prediction. This allows the users to understand the factors that are involved in the decision, thus they can combine their knowledge in order to make a more informed and reliable decision.

1.1 Motivation

As we have explained above, the detection of the minority class could be a hard task. However, the detection of these events could be crucial, as an instance, consider a dangerous illness. The detection of this illness is crucial, thus the objective is to detect as many cases as possible. However, such detection can trigger the increase of false

alarms. However, it could be more critical not to detect an ill patient than to apply extra tests to a healthy patient, who was wrongly detected as positive. This work is illustrated predicting future opportunities in financial stock markets, in cases where the number of profitable opportunities is scarce. This happens, for example, when an investor is trying to detect very high returns. As the number of opportunities with this characteristics is very small, the system has to cope with imbalanced data sets. On the other hand, different investors could have different risk guidelines according to their current necessities or interests. While some investors are willing to take additional risk for an investment, risk adverse investors may tend to take increased risks only if they are warranted by the potential for higher returns. A range of classifications could suit the requirements of both types of investors. Finally, investors may be interested in understanding the variables and conditions in the prediction, thus the user can analyse the conditions of the event and use additional knowledge to make a more informed decision.

1.2 Objectives and research goals

The general objectives of this thesis are:

1. Classify the minority class in extreme imbalanced environments

2. Produce a range of classifications to suit the user's preferences

3. Generate comprehensible solutions that can be read and understood by the user. The understanding of every condition that is involved in the prediction allows

the users to apply their knowledge to take a more informed decision, more details about comprehensive solutions are provided in section 2.2.

4. Improve the accuracy and precision of decision trees produced by a Genetic Programming (GP) system [69].

To achieve these goals three methods (the three central chapters of this thesis) have been developed. The approaches proposed in this thesis are based on machine learning and supervised learning. The learning technique used in this research is the evolutionary paradigm, which is inspired by the natural evolution theory developed by Charles Darwin [28]. To achieve our goals, GP has been used in this research. The approaches proposed in this thesis are designed to solve a binary classification problem. Now, let us briefly explain the main reasons for selecting GP as a supervised learning tool.

1. GP can evolve dynamic structures in size and shape. In contrast the representation of other ML techniques, such as Genetic Algorithms (GAs) or Neural Networks (NNs), usually have fixed size [4]

2. GP is able to produce decision trees that can be understood. This characteristic is a key factor in this research because the interpretability of the solution allows the user to analyse the variables and conditions which are involved in the solution.

3. GP is able to produce multiple solutions of a single problem, which facilitates the search of patterns.

The approaches proposed in this thesis are designed to reduce the problem of the extra-code generated by the GP, which is a common problem in this technique [2, 88, 104, 74]. The extra code constitutes in between the 40% and 60% of the total code. Given that, our goal is to offer comprehensible solutions, it is important to remove the extra-code in order to show the real variables and conditions involved in the solution. Now, let us briefly describe the approaches proposed in this thesis:

1.2.1 Repository Method

The objective of Repository Method (RM) is to predict the minority class in imbalanced data sets. As the number of examples in the minority class could be very small our aim is to extract and collect different patterns that classify the positive cases (rare instances) in different ways, increasing the probability of identifying similar cases in future data sets. RM starts by analysing a set of solutions (decision trees) provided by a standard GP. Then every rule is delimited, subsequently this is evaluated taking into account the performance and novelty, this method has been designed to:

1. Classify the minority class in imbalanced data sets
2. Produce a set of rules that provide a range of classifications to suit the user's preferences
3. Provide a set of comprehensible rules that show the conditions and variables involved in the decision.

1.2.2 Evolving Decision Rules

Evolving Decision Rules (EDR) is an evolutionary process, which is dynamic. EDR evolves a population of decision trees to form a repository of rules. The resulting rules are used to create a range of classifications that allows the user to choose the best trade-off between the misclassifications and the false alarms costs. This approach shares the same objectives with RM. As can be observed, RM and EDR share the same objectives. However, the main difference of these approaches is the way to generate the solutions. While RM is a deterministic process, whose set of candidate solutions is provided by a GP system. EDR is a dynamic method, which creates new solutions by means of the evolution of rules. A more detailed explanation of the differences of the mentioned methods is provided in Chapter 7.

1.2.3 Scenario Method

Scenario Method (SM) is a pruning procedure for decision trees created by GP. This pruning is based on the analysis of patterns in the decision tree. The characteristics of this method are the following:

1. Analyse every decision tree to detect and remove the rules that do not contribute to the classification task. Given that the pruning is performed by using a threshold, the classification can be slightly tuned as liberal or conservative according to the user preferences.

2. Simplify the decision trees to help to disclose the conditions and variables involved in the solution.

Thus, we have two approaches (Repository method and Evolving decision Rules) focused on the detection of the minority class in imbalanced data sets. These are designed to provide a range of classifications to suit the user's risk-guidelines. According to Ling and Li [78], in some cases a ranking classifier is more reliable than just a classification. Finally, the goal of RM and EDR is to provide a set of comprehensible rules that show the real variables and conditions. On the other hand, SM proposes a pruning to improve the precision and accuracy of the predictions. The methods proposed in this thesis were illustrated using financial data sets from the London Stock market. Obviously a strong supposition in this research is the predictability of financial markets. In technical analysis, it is believed that the investors and speculators react the same way to the same kind of events [66]. It means that the patterns in financial prices are usually followed by similar reactions. However, this has been strongly debated in the financial environments, as this is explained in chapter 4. It is not the intention of this work to enter into debate about the predictability of the financial markets. The approaches proposed in this thesis are general methods and these can be applied to a great variety of classification problems. Specifically, RM and EDR can be used in the detection of the minority class in imbalanced environments, where the detection of the minority class (positive cases) is crucial and the number of undetected positive cases is more expensive than classify false alarms.

1.3 Thesis structure

This thesis has been structured in three parts, the first part provides an overview of the required background literature to support the theory of the contributions in this thesis, the background material has been divided in three chapters. Chapter 2 presents a machine learning literature review. It does not pretend to be either an extensive or complete review of that field. Its objective is to present just the topics closely related to this thesis. It starts with a learning definition and then it focuses on specific topics, such as classification rule induction, metrics to evaluate the performance of classifiers, the receiver operating characteristic (ROC) curve and the imbalanced classification problem. Finally, the context free grammar, which is used to create and maintain the decision trees used in this work, is introduced. Next, chapter 3 focuses specifically on evolutionary computation. It starts with a general description of the evolutionary theory and the evolutionary computation. After that, the evolutionary technique Genetic Programming (GP) is described. Next, some specific GP issues that are related to the contributions of this thesis such as bloat and crossover are explained. Finally, a literature review about evolution of decision rules is provided. Chapter 4 provides a brief explanation of financial issues. The methods presented in this thesis are illustrated with financial data sets and the attributes are formed by indicators derived from technical analysis. First, an explanation of markets efficiency is given. After that, the fundamental and technical analysis is briefly explained. Finally, the financial indicators used in this thesis are described. The second and principal part of this thesis presents the contributions of this research. This part is composed of four

chapters that introduce the three methods proposed in this thesis. Chapter 5 introduces the common procedures that are used in the proposed approaches of this thesis. It was decided to describe the common procedures in a separated chapter in order not to repeat the same information in different chapters and to keep the methods' description short and simple. It is advisable to read carefully this chapter because the justification and aims of those processes are provided. Chapter 6 introduces the first approach of this thesis whose name is *Repository Method* (RM). This is a novel approach, whose aim is to gather patterns from decision trees produced by a GP system. The goals of this method are 1) to predict the minority class in imbalanced environments, 2) to provide a range of classifications to suit the user's necessities and 3) to provide a set of rules that can be interpreted by the user. This approach analyses a set of decision trees produced by a GP system to gather useful patterns to form a repository of rules. This Chapter explains in detail the procedure of this approach, the experiments description, results and finally the conclusions. Chapter 7 describes the *Evolving Decision Rules* (EDR), this approach is an evolutionary method whose objective is to evolve a set of rules (patterns). This chapter provides a literature review about previous works in the context of evolutionary algorithms, the complete explanation of the method as well as the experiments and results to test our approach. Chapter 8 introduces a pruning method for decision trees generated by GP. This approach analyses each decision tree in order to delimit its rules and evaluate the contribution of each rule to the classification task. This approach is called *Scenario Method* (SM). This chapter introduces the complete explanation of this method, the previous works and the experiments that

were carried out to test our approach. Finally, Chapter 9 summaries the contributions of this thesis and provides the conclusions and future work of this research. The third part of this thesis is composed of the appendix, which describes the algorithms of the methods proposed in this thesis.

Part I

Background Literature

Chapter 2

Machine learning

This chapter presents a general description of the machine learning field, it does not pretend to be an exhaustive review of that area, but it provides enough background information to support the contributions proposed in this thesis. The chapter is structured as follows: section 2.1 contains a brief description in general of the Artificial Intelligence and Machine Learning fields. After that, the remaining of this chapter is focused on specific issues that are related to this work. Section 2.2 describes some classification rule induction techniques, while section 2.3 explains the problem of classification in imbalanced environments. Next, section 2.4 describes the metrics that were used to evaluate the performance of the methods proposed in this thesis, thus sections 2.4.1 and 2.4.2 describe the confusion matrix and some evaluation metrics respectively. Section 2.4.3 gives a brief introduction about the Receiver Operating Characteristic (ROC) curve. Finally, section 2.5 provides an introduction of the context free grammars, which are used as a format to create the decision trees used in this research.

2.1 Machine learning

One of the fundamental goals of Artificial Intelligence (AI) is to endow machines with learning capabilities. Turing proposed to mimic intelligence by training machines using "an appropriate course of education" as he literally explained in [117]. Machine learning (ML) is a field of the AI. It is a multidisciplinary area, which embraces other disciplines such as probability and statistics, computational complexity theory, information theory, philosophy, neurobiologist and other fields [83]. ML is focused on techniques that create computer programs capable of producing new knowledge to improve their performance [67]. That knowledge is acquired by means of input information (experience) and the analysis of data. A formal definition of a learning system is provided by Michalski et al.

"Learning denotes changes in the systems that are adaptive in the sense that these enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time"[67]

Mitchell [83] defines the intelligence as the ability to learn, to adapt and to modify behaviour. He asserts that learning involves improving the performance of a task through the experience, and he formally defines this as:

"A computer program is said to learn from experience E with respect to some class of task T and performance measure P, if its performance at task in T, as measured by P, improves with experience E" [83]

At this point let us explain the difference between intelligence and knowledge: knowledge is useful information that has been registered and store by the individual. The intelligence is the capacity of the individual to use this information in order to achieve a specific purpose [41]. According to Mitchell [83], machine learning techniques can be classified by the way that these learn. He categorizes the techniques as 1) supervised learning, 2) unsupervised learning and 3) reinforcement learning.

This dissertation has been developed in the frame of the supervised learning. Thus the other learning techniques' definitions are out of the scope of this dissertation. However the interested reader is referred to [83].

Supervised learning The objective is to create a model from past information which is also called *training data set*, it consists of input values and desired outputs. The main characteristic of the supervised learning is that this kind of learning receives feedback (desired output) to acquire knowledge. After seeing a number of training examples, the task of a supervised learning system is to predict the value of the model for any valid input. To achieve this, the learning system has to generalize from the presented data to unseen situations.

2.1.1 Training data set

The methods presented in this thesis use supervised learning. The experience is provided by a training data set, which contains examples with the *inputs* and the desired *outputs*. Figure 2.1 shows an example of a training data set. The first column represents the desired output of the learning program, in this case it is a classification

of two categories which are represented by 0 and 1. The next columns are *independent variables* or *attributes* which describe the behaviour of the class in that record. In general, every instance in the training data can be the representation of physical objects, images, actions, processes, concepts, etc. A variable is called *ordered* or *numerical* if its values are numbers. A variable is called *categorical* if this takes values from a finite set of options which have not any natural order [15]. In the same vein Flach and Lavrac [40] pointed out the difference between an *example* and an *instance*. They explain that an *instance* is composed of an established collection of attributes: A_i where $i \in \{1, \ldots, n\}$. Every attribute can be a real number (continuous) or a finite set of values (discrete). On the other hand, an example e_j is a vector of attributes which is tagged with a class label e_j=$(v_{1,j}, \ldots, v_{n,j}, c_j)$ where each $v_{i,j}$ is a value for the attribute A_i and $c_j \in \{c_1, \ldots, c_k\}$ is the label of a class from the set of valid classes. It is clear that in supervised learning a set of examples is used to train the learning system. On the other hand, in unsupervised learning a set of instances is used. All the contributions in this thesis have been trained using supervised learning. However, we are going to use the words example and instance as synonymous.

2.1.2 Designing a learning program

Mitchell [83] asserts that the design of a machine learning system requires the following elements: 1) the specification of the type of training experience, 2) the design of the target function that will measure the system performance, 3) the representation of the target function and 4) the definition of the learning process, these are explained below:

Figure 2.1: Example of a training data set, the first column represents the desired output and the remaining columns are independent variables

OUTPUT	INPUT (Attributes or independent variables)						
	Var_1	Var_2	Var_3	Var_4	Var_5	Var_6	$\cdots$
0	.672	.898	.015	.232	.772	.429	$\cdots$
0	.333	.456	.433	.462	.733	.249	
1	.623	.326	.566	.562	.224	.429	
1	.633	.776	.777	.367	.444	.259	
0	.635	.966	.885	.657	.564	.629	
1	.343	.536	.433	.662	.244	.627	
0	.893	.256	.755	.672	.224	.259	
0	.563	.876	.465	.662	.444	.257	
0	$\vdots$						$\ddots$

Training Experience selection The first step to design a learning program is to select the *training data*, which is the experience that is going to provide the knowledge to the learning system. The selection of the experience has an important effect in the success of the system. There are three important characteristics of the information that can have a big impact in the result:

1. The kind of feedback or knowledge that is provided by the training experience, this can be direct or indirect knowledge. In this research the knowledge is supplied directly because the training data provides the desired output (the label of the class) as was explained in the previous section. But in some other cases the only knowledge available is provided in an indirect way.

2. The second characteristic is the way in which the learning system controls the sequence of the training data. The examples in the training data sets

that are provided in this thesis are chronologically ordered.

3. The third characteristic requires that the experience be representative of the situation(s) to learn. Usually learning systems are more accurate when the training and testing data sets have similar distribution. However, sometimes it is necessary to learn from a data set that has different distribution from the final data set.

Target Function selection To design the target function it is important to keep in mind the following factors: 1) the precise type of knowledge that has to be learned by the system and 2) the utility of that knowledge to improve the performance of the learning system. The target function evaluates the performance of the learning program, this evaluation has to measure the acquired knowledge and the way that it is used to solve the problem. It has to assign high scores to the successful actions, so it can be easy to identify the effective ones. At this point it is very important to identify all the features that the target function has to take into account and to evaluate.

Determine the representation of the learning function Once the precise type of knowledge to learn has been determined, the next step is to define the representation that the learning program will use to describe the target function. Some problems that are complex have to use more expressive representations of the target function in order to make a reliable evaluation. The methods proposed in this thesis are based on supervised learning and these will be evaluated using

the metrics described in section 2.4.2. It means that the representation of the target function is determined by an arithmetic formula.

Determine a learning mechanism In this part of the process it is necessary to define the mechanism that is going to learn from the experience. At this point of the thesis it is still too early to fully explain the learning mechanism of our approaches, the descriptions are provided in chapters 6,7 and 8.

2.2 Classification rule induction

This section provides a brief overview of the classification rule induction field, focusing on those aspects that are important for this research. Given a training data a classifier maps objects to Classification rule induction is a field of Machine Learning [20], [67],[83]. According to Fawcett [37], the objective of a classifier system is to create a mapping between examples and predicted classes. The examples are formed by a collection of past observations (*training data set*). Each of the examples is pre-classified using a class label, the goal is to predict unseen examples. In order to measure the performance of the classifier, it is tested in another set of data, which is also called *testing data set*. The result can be a continuous output as a regression, but the result of other models can be a discrete output. For example, a label which indicates the predicted class of the instance [15]. The number of classes could be two (binary classification) or more (multiple classification). The approaches proposed in this thesis are designed to solve a binary classification problem.

Breiman *et al.* [15] pointed out that the main goals of classification are: 1) to

generate an accurate classification model able to predict unseen cases and 2) discover the predictive structure of the problem. It means to provide an understanding of the variables and conditions that control or are involved in the event. The interpretability of the model helps to understand the conditions that set apart one class from another. In many cases both objectives are important. For these reasons a good classifier has to be able to produce accurate classifications, with the limitation of the data, and provide understanding of the predictive structure of the data [15]. This idea is reinforced by other researchers such as Flach and Lavrac [40] who distinguish models designed for prediction from models designed for understanding. They remark that sometimes models, which provide a poor representation of the solution, have better predictive power than methods that provide comprehensible models. As an instance, we can mention the Artificial Neural Networks (ANN)[18].

On the other hand, Henery [55] exposed some characteristics that are required in a classifier, these are the following:

- Accuracy or the reliability of the prediction, it is usually represented by the number of correct predictions. However, in some cases this measure can vary according to the requirements of the user, such as the desire to detect a specific class.

- The speed of the classifier, in some cases where the speed of the classifier is a key factor, the user could prefer to sacrifice accuracy in order to get speed.

- Interpretability of the solution, in many cases the understandability of the solution is needed in order to have a better understanding of the conditions that trigger the event.

- Time to learn, it refers to the size of the training data set and the time that the classifier takes to learn new patterns and to be adjusted to new conditions.

In this research one of the key factors is the interpretability of the solution. Thus, the user will be able to analyse the conditions and relations that control the event. This allows the user to apply his/her professional knowledge in order to make a more reliable decision.

Classification models

In data mining and machine learning a predictive model can be represented by a decision tree, these are also called classification trees. According to Breiman et al. [15], binary trees are a useful way to look at data in classification or regression problems.

In decision tree structures, branches could represent conjunctions and/or disjunctions and the leaves represent classifications and features related to that classification. As was mentioned, one of the classification objectives is to produce a model, it could be a regression when the values are continuous and a label when there the output is discrete. A model is a large structure which summarizes features and relationships that satisfy a large number of cases (e.g. a decision tree), while a pattern is a local structure which meets the requirements of few cases is the search space (e.g a rule) [121].

Classification is about discovering structural patterns in data, in ML there are

different ways for representing patterns. The first step is to understand which is the desired output. Once it has been established, it is easy to design the structure to generate it. The methods described in chapters 6 and 7 are rule discovery tools, whose objective is to collect a set of rules that capture patterns from a specific class.

The model produced by the classifier can be represented in different ways, for example: decision trees [15], [98], [113], finite state automats [109], graphs and networks [18] and binary vectors (in Genetic Algorithms) [26],[64]. Some of these classifiers can be rewritten as a collection of rules, this is the case of C4.5 [98] developed by Quinlan, who argues that 1) a collection of rules is more intelligible than decision trees, especially when these are large and 2) the structure of trees could create sub-concepts to be fragmented. There are many definitions of rule, but in this research we work with the following definition:

"A rule is a substructure of a model which recognizes a specific pattern in the database and takes some actions" [121].

There are many machine learning techniques that have been developed for classification and prediction. We will just provide two illustrative examples of classification works in the field of ML. Given that, the interpretability of the model is a required condition, this section does not mention any classifier that produces hidden models as Artificial Neural Networks. The reader can find a series of classifiers based on evolutionary algorithms in section 3.4. The following paragraph briefly describes two important works in ML, which use decision trees: 1) Classification and Regression Trees by Breiman and 2) Quinlan's classifiers.

The Classification and Regression Trees (CART) algorithm was developed by Breiman and his colleagues [15], this is a statistical algorithm to build binaries trees to solve classification and regression problems. CART uses a set of examples to create a decision tree, the examples hold attributes that provide information to set apart the data. The splitting is performed recursively, in each partition the aim is to minimize the *impurity*, which is an indicator that measures the similarity of a data set, the smaller the number the less impure the sample set is. An stopping criteria is used to finish the algorithm. As the authors described the main elements of CART are: 1) a set of binary questions, 2) a splitting condition that can be evaluated, 3) a stop splitting rule and 4) a policy to assign every terminal node to a class. Each node in the decision tree, holds a binary question about a characteristic of the object to classify. Finally, the leaves of the tree hold the prediction based on the training data set.

The methods to create *top down induction* trees, developed by Ross Quinlan [97], [98]. His first classifier was ID3, subsequently it was refined to produce C4.5 and the last version of this tool is C5. Quinlan's classifiers are based on the idea that each attribute of the data set can make a decision that splits the data into smaller subsets. For that reason, the *Information Gain* is measured for each attribute. The split will be made using the attribute with the highest normalized information gain. This classifier is used in chapters 5 and 6 to compare the performance of the approaches proposed on those chapters.

2.3 Imbalanced classes

Machine learning classifiers, like other forecasting techniques, extend the past experiences into the future. However, the imbalance between positive (minority class) and negative cases (the remaining classes) poses a serious challenge to machine learning techniques [63],[119],[5],[81]. In rare event detection, negative classifications are favoured because these have a high chance of being correct.

A data set is imbalanced, when there are many more examples from one class than the others, it causes that classifiers tend to predict the largest class(es) ignoring the small one(s). Consider a data set that classifies two classes (positive and negative). The total number of examples in the data set is 100 cases, where 3 cases are positive and the remaining are negative. If the classifier predicts all the cases as negative, this is classifying correctly 97% of the total predictions. However, it missed all the positive cases, which could be the objective of the classification.

According to Chawla [21], the imbalanced data set problem has been addressed in two different ways: 1) the data level and 2) the algorithmic level. Let us briefly describe some of the most relevant works of these techniques.

Data level

The data level techniques involve a pre-processing of data, thus many re-sampling methods, such as random over-sampling [1] and random under-sampling [2] have been used. However, more sophisticated forms of re-sampling have been proposed. Next

[1]Random over-sampling - Random replication of the minority class
[2]Random under-sampling - Random removal of the majority class

paragraphs describe some of the most relevant approaches of re-sampling.

Hart [54] proposed the Condensed Nearest Neighbour Rule (CNN); it is an under-sampling method that eliminates the redundant examples of the majority class that are distanced from the decision border. The mentioned algorithm extracts a subset of examples from the training data set using the idea of mutual nearest neighbourhood to select examples near to the decision line.

After that, Tomek [111] improved the CNN method introducing an algorithm to detect borderline and noisy examples. To briefly explain this approach, let e_i and e_k be two examples from different classes, and $d(e_i, e_k)$ be the distance between e_i and e_k. A tuple (e_i,e_k) is a *Tomek link* if there is not an example e_α, such as $d(e_i, e_\alpha) < d(e_i, e_k)$ or $d(e_k, e_\alpha) < d(e_i, e_k)$. If two examples form a Tomek link, then either 1) one of these examples is noise or 2) both examples are borderline. Tomek links can be used as an under-sampling or as a data cleaning method. When Tomek links are used as an under-sampling method, only the examples of the majority class are removed. In contrast, examples of both classes are eliminated when Tomek links are used as a data cleaning method.

Kubat and Matwin [70] proposed an under-sampling technique for removing the negative examples and keeping all the positive instances. First, the authors classified the examples in four categories: 1) Noisy examples, 2) borderline examples, 3) redundant examples and 4) safe examples. The authors proposed to remove the noisy and borderline examples by using the Tomek links technique [111]. The redundant examples from the negative class are removed using CNN [54].

Chawla et al.[87] proposed the Synthetic Minority Over-sampling Technique (SMOTE). The mechanism of this approach is to under-sample the majority class and to over-sample the minority class using synthetic minority class examples. The latest are created by taking each minority class instance and introducing synthetic examples along the line segments joining any/all of the k- minority class nearest neighbours. Depending on the amount of over-sampling required, neighbours from the k-nearest neighbours are randomly chosen. Experimental results using C4.5, Ripper and a Naive Bayes classifier, showed that SMOTE achieved better results that just under-sample the majority class. To measure the result the authors used the ROC curve (see section 2.4.3).

Batista et al. [5] performed an evaluation of ten different sampling methods, the following algorithms were tested:

1. Random over-sampling
2. Random under-sampling
3. Tomek links [111]
4. Condensed Nearest Neighbor (CNN) [54]
5. One side selection - Tomek links [111] + CNN [54]
6. CNN [54] +Tomek links [111]. As can be realized, this method is similar to *one side selection*. However, it first applies CNN and after Tomek links.

7. Neighborhood Cleaning Rule [76]. Let e_i be an example from the majority class, for each example e_i the three nearest neighbours are found. If the classification of the three nearest neighbours of e_i contradicts the class of e_i, then e_i is removed. However, if e_i belongs to the minority class and its three nearest neighbours misclassify e_i, then the nearest neighbours that belong to the majority class are eliminated.

8. Synthetic Minority Over-sampling Technique (SMOTE) [87]

9. Smote [87] + Tomek links [111]

10. Smote [87] + ENN - the latest tends to eliminate more examples than Tomek links. ENN is used to remove examples from the minority and the majority class. Thus, any example that is misclassified by its three nearest neighbours is removed from the training data set.

To measure the performance of the classifiers the authors used the area under the ROC curve (see section 2.4.3). The experimental results showed that the over-sampling methods provide more accurate results than the under-sampling techniques. In addition two of the new approaches in that work (Smote + Tomek and Smote + ENN) performed very well for data sets with a small number of positive examples. Additionally, experimental results disclosed that random over-sampling is very competitive in comparison to more complex over-sampling techniques.

Hall and Joshi [52] proposed to enhance SMOTE. As was mentioned, SMOTE rebalances the data sets by under-sampling the majority class and generating synthetic

examples of the minority class. However, the authors argued that the percentages of under-sampling and over-sampling are not predefined, thus they proposed a method to determine these percentages. The authors proposed a wrapper to evaluate the performance function, which includes a measure of the minority class accuracy. The experiments were performed using Ripper as classifier; the results showed that this approach is effective in optimizing the evaluation function,

However, re-sampling techniques may cause some problems in the learning process. As an instance, the introduction of examples of the minority class can slow down the performance of the classifier when the data set is large. When the over-sampling introduces copies of the minority class, it can produce over-fitting. On the other hand, the under-sampling may remove useful examples from the data set.

Algorithm level

This level comprises techniques that modify the algorithm of the classifier, such as, adjusting the cost of each class and learning from one class. Next paragraphs briefly describe some approaches, at the algorithm level, designed to deal with the imbalanced data set problem.

According to Ling and Sheng [79] in data mining the Cost-Sensitive Learning is a type of learning that takes the misclassification costs into consideration. The aim of this type of learning is to minimize the total misclassification cost. This technique has been applied by Pazzani et al. [92], the authors proposed the *Reduced Cost Ordering algorithm*, they assigned different weights or costs to each example depending on the

class. Li [77] built a binary classifier (buy or not buy) to predict investment opportunities using a GP system. However, when the number of profitable opportunities was small, he proposed to assign costs to each class by using a constrained fitness function.

[99] Raskutti and Kowalczyk proposed to combine under-sampling and over-sampling methods with weighting imbalance compensation techniques. The authors increased the imbalance until extreme situations when the large set of negative examples was ignored and the training was provided from positive examples only. The experimental results suggest that one-class learning from positive examples is a very effective technique for imbalanced data sets and high dimensional noisy feature space.

To solve the problem of imbalanced data, Chen et al. [22] incorporated new features to the Random Forests method (RF) [14]. The authors proposed 1) to balance RF and 2) to weight RF. The Balanced Random Forest (BRF) algorithm is briefly described as follows: 1) For each iteration in RF, draw a bootstrap sample from the minority class and randomly draw the same number of cases with replacement from the majority class. 2) The CART algorithm is used to induce a classification tree, but at each node, instead of searching through all variables for the optimal split, only search through a set of randomly selected variables. 3) The first and second steps are repeated a specific number of times. Aggregate the predictions of the group and make the final classification.

The Weighted Random Forest (WRF) assigned weight to each class; these weights are used in the RF algorithm in two places: 1) in the tree induction procedure the class weights are used to bias the Gini criterion for finding splits and 2) the class prediction

of each terminal node is determined by a weighted majority vote. The class prediction for RF is determined by aggregating the weighted vote from each individual tree, where the weights are average weights in the terminal nodes.

Imbalance levels

According to japkowicz [63], data sets can have different levels of imbalance, in that work a series of experiments was performed using different levels of imbalance and three methods to deal with this problem: over-sample, under-sample and use recognition ignoring one class. The mentioned work concluded that every method works differently for every level of imbalance. The present work is focused on finding opportunities in imbalanced environments. In this work we used elements of chance discovery. It is important to keep in mind that in imbalanced data sets the classifier performance must not be measured only by the accuracy [71],[96]. Apparently, the accuracy is the metric, which is evident to use, to measure the performance of a classifier. However, this measure assumes that the class distribution in the data set is constant and relatively balanced [95]. The accuracy is not necessarily an efficient metric for a classifier's performance [65].

2.4 Metrics to evaluate a classifier performance

This section describes some metrics that have been used in ML to measure the performance of binary classifiers. First, the concept of confusion matrix is explained in section 2.4.1. After, some metrics that have been defined using the components of the confusion matrix have been introduced in section 2.4.2. Next, the Receiver Operating

Characteristic curve is described in section 2.4.3.

2.4.1 Confusion Matrix

A confusion matrix displays the data about actual and predicted classifications done by a classifier [68]. This information is used in supervised learning to determine the performance of classifiers and some learning systems. Given an instance and two classes (positive and negative) there are four possible results: The instance is positive and it is classified as positive (*true positive*). The instance is negative and it is counted as positive (*false positive*). The instance is positive and it is classified as negative (*false negative*). The instance is negative and it is predicted as negative (*true negative*). Table 2.1 shows a confusion matrix for two classes.

Table 2.1: Confusion Matrix to classify two classes

	Actual Positive	Actual Negative
Positive Prediction	*True Positive* (TP)	*False Positive* (FP)
Negative Prediction	*False Negative* (FN)	*True Negative* (TN)
	Total Positive	Total Negative

True Positive	(TP)	Number of correct predictions in positive cases
False Positive	(FP)	Number of incorrect predictions that were classified as positive when the instance is negative
False Negative	(FN)	Number of incorrect predictions that were classified as negative when the instance is positive
True Negative	(TN)	Number of correct negative predictions

2.4.2 Metrics using Confusion Matrix

Many metrics have been defined using the elements in the confusion matrix, Table 2.2 presents just the metrics that are relevant for this work. There are more metrics derived from the confusion matrix information. However, the most relevant measures for the methods proposed in chapters 6, 7 and 8 are the accuracy, precision and recall. The accuracy has been used widely to measure the performance of many classifiers [40]. However, the this may not be an adequate metric when the number of cases in the minority class is very small in comparison with the number of cases in the other class(es) [71]. As an instance, consider a data set of two classes that holds 100 examples, where 98 of them are negative and just 2 of them are positive cases. If the system classifies all of them as negative, the accuracy will be 98% , apparently it is a very high performance. However, the classifier missed all positive cases, this phenomenon is called "imbalanced classes", more information is provided in section 2.3. Unfortunately in the real world, the detection of relevant events relies on the prediction of the minority class. For example: Financial opportunities detection, illness detection, fraud detection, etc. For that reason, the next section introduces Receiver Operating Characteristic, which is a more reliable way to measure a classifier performance.

Table 2.2: Some metrics using the components of the confusion matrix

Accuracy	is the proportion of the total number of predictions that were correctly made, this is determined by the equation: $Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$
Recall	it is also called *sensitivity* or *true positive rate*, the recall is the proportion of positive cases that were correctly identified, it is determined by the formula: $Recall = \frac{TP}{TP+FN}$
Precision	is the proportion of positive cases that were correctly predicted. This is calculated as follows: $Precision = \frac{TP}{TP+FP}$
Specificity	it is also called *true negative rate* and it is the proportion of negative cases that were correctly predicted, it is determined by the equation: $Specificity = \frac{TN}{TN+FP}$
False positive rate	is the proportion of negative cases that were wrongly predicted as positive. It is determined by the formula: $False\ positive\ rate = \frac{FP}{FP+TN}$
False negative rate	is the proportion of positive cases that were wrongly predicted as negative, it is calculated using the equation: $False\ negative\ rate = \frac{FN}{FN+TP}$

2.4.3 Receiver Operating Characteristic

Receiver Operating Characteristic (ROC) is a technique that graphs the performance of a classifier that predicts two classes (positive and negative). It is able to select the best trade-off between successful cases and false alarms based on benefits and costs [37],[94]. It has been used to evaluate the performance of diagnosis tests [91], [118], [51],[80].

The ROC graph is constructed by plotting the *true positive rate* (recall) on the

Y-axis and the *false positive rate* on the X-axis [53],[37]. Figure 2.2 shows the ROC space. As can be seen, ROC graph is plotted in the space of (0,0) and (1,1). The performance of the classifier is plotted in (0,0) when it does not find any positive case and it does not report any false alarm. Thus, this gets all the negative cases right but it gets all the positives wrong. The other point is (1,1) where the totality of the cases are classified as positive, it reports all the positive cases, but it fails to predict the negative instances.

According to Fawcett [37], the performance of a classifier is better than another if it is plotted in a left upper part of the ROC space. The classifiers whose performance is plotted in the left hand side in the ROC space close to the X-axis, are called *conservative*, because these make a positive classification just when these have strong indications or evidence, as a result these have few false alarms. On the other hand, classifiers on the upper right hand side of the ROC space are called *liberal* because these make positive classifications with unsubstantial evidence. Thus, these are able to classify almost all the positive cases. However, these have a high rate of false positive. Finally, the diagonal line between (0,0) and (1,1) describes the performance of a classifier that randomly predicts the instances. The space behind the diagonal usually remains empty but in some cases, where the classifier performs worse than random, it is suggested that the classifier answer is negative correlated with the actual data.

ROC curve

The result of a discrete classifier is a confusion matrix, which represents a single tuple of *false positive* and *true negative* rate. This produces a single point in the ROC graph.

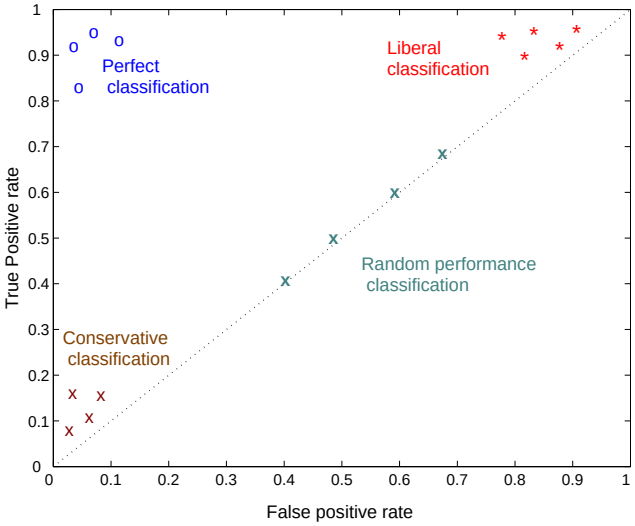


Figure 2.2: ROC space

There are some classifiers that manage a threshold in order to tune the precision of the classification. Since every threshold value produces a point in the ROC space, a ROC curve can be formed by moving the threshold.

Area under the ROC curve (AUC)

One of the most used ROC curve metrics is the Area Under the Curve (AUC). The AUC has been used widely in fields such as signal detection [106], in medical diagnosis [50] and many other fields to indicate the quality of the classifier [118]. Huang and Ling [60] showed theoretically and empirically that AUC is a better measure than accuracy. When AUC =1 it means that the classifier is perfectly accurate. The closer AUC is to 1, the better the classifier performance is. When AUC is close to .50 it represents the performance of a random classifier.

Choosing the best operating point

ROC can be used to estimate the best threshold of a classifier, calculating the best balance between the cost of missclassifying positive and negative cases. In order to calculate the best trade-off:

Let μ be the cost of false positive or false alarm
β be the cost of false negative
ρ be the percentage of positive cases

The following equation defines the slope of the line that describes the best trade-off between misclassifying positive and negative cases.

$$Slope = \mu \cdot (1 - \rho)/(\beta \cdot \rho) \tag{2.4.1}$$

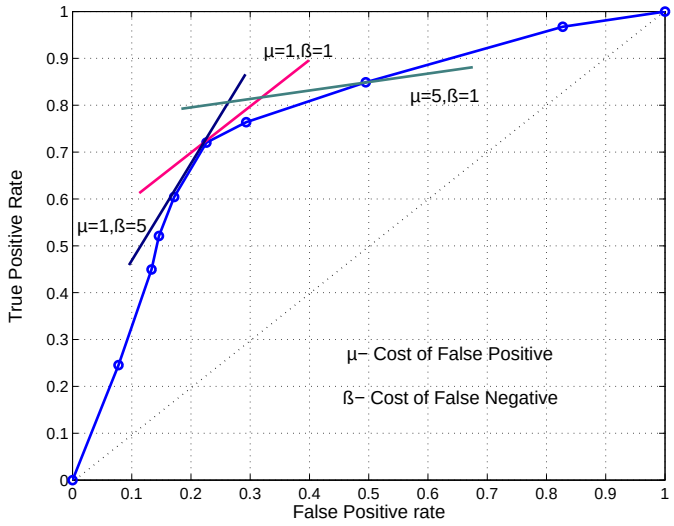


Figure 2.3: shows a ROC curve and the tangent lines which indicate the best trade-off between the false alarms and misclassification

The optimal classification point of the classifier (according to μ and β) is the point, which the line is tangent to the ROC curve. Figure 2.3 shows a ROC curve and the tangent lines which indicate the best balance between the false alarms and misclassification cases using different costs. We have selected ROC to measure the performance

of the approaches in this thesis for the following reasons:.

1) ROC is able to deal with imbalanced data sets. One of the main goals of this research is to detect the minority class in imbalanced environments. Chapters 6 and 7 are focused on extreme imbalanced environments.

2) ROC is capable of measuring the performance of classifiers that rely on thresholds. The methods proposed in this thesis are tuned by thresholds (see Chapters 6 and 7).

3) ROC is able to tune the behaviour of the classifier. The methods proposed in this thesis are desingned to provide a range of classifications, which can be plotted in the ROC space. The objective is to allow the user to choose the best trade-off between miss-classification and false alarms costs.

2.5 Context-Free Grammar

This section introduces the concept of Context-Free Grammar (CFG), this technique is used to describe the structure of the decision trees used in this research. The use of CFG offers the following advantages:

1. it determines the initial structure of the decision trees
2. it establishes a format that helps to maintain the syntax of decision trees during the evolutionary process
3. it makes the tree structure comprehensive to the user

In the field of computer science, as well as in the linguistics, a CFG is a formal system that describes a language, it specifies the valid texts that can be conceived from each symbol [101]. The concept of CFG was introduced by Chomsky [25] as a form to describe natural languages, he asserts that each language can be represented by different grammars, all of them constructed by the same method. Subsequently Backus Normal Form (BNF) was presented as a formal notation to represent the syntax of ALGOL programs [35]. Nowadays BNF is the most common notation used to express context-free grammars. Let us introduce two formal definitions of grammar and CFG.

Definition 1. *"A context free grammar* G *is a four-tuple* $G = \{N, \Sigma, P, S\}$ *where* S *is the start symbol,* N *and* Σ *are* alphabets[3]*.* N *is the set of elements called nonterminals and* Σ *contains the terminal symbols.* P *is a finite set of ordered pairs* (A, α) *where* $A \in N$ *and* $\alpha \in (N \cup \Sigma)$*. The elements* $(A, \alpha) \in P$ *are named* productions *and they are written as:* $A \rightarrow \alpha$*"* [86]

In other words, CFG is a set of productions and this starts with a non-terminal symbol, every symbol can be replaced by a given sequence of terminal or non-terminal symbols [101], as an instance :

<S>	→	<Condition>
<Condition>	→	<Operation>, <Variable>, <Threshold>
<Operation>	→	"<" | ">"
<Variable>	→	var_1 | var_2 | ... var_n
<Threshold>	→	Real number

The previous grammar is composed of five productions, <S> is the start symbol and <Condition> is composed of the elements <Operation>, <Variable> and <Threshold>. All of them are called nonterminal because these can be replaced by

[3] *An* alphabet *is a finite set of symbols [86]*

other values. The first production asserts that the symbol <Operation> can be replaced by the sequence consisting of the two symbols "<" or ">", a sequence separated by the vertical bar '|' indicates a choice. These elements are called terminal symbols because these cannot be replaced. In other words, the symbols that never appear on the left side of the production are called terminals. The following statement represents an instance of <Condition>, using the inverse Polish notation, this can be translated as the equation: $var_3 > 9.87$. Figure 2.4 shows the graphic representation of that production.

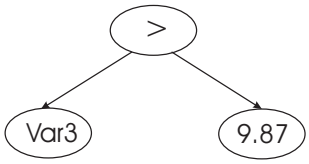


Figure 2.4: The graphical representation of <Condition> → ">", var_3, 9.87

Figure 2.5 shows Grammar 1, which is an example of a grammar to generate decision trees that classify two classes. Figure 2.6 shows an example of a decision tree that meets the syntax of Grammar 1. Tracking the logic of the decision tree the following rules can be identified:

$R_1 = \{var_3 > 0.5 \text{ and } var_4 > 0.7\}$
$R_2 = \{var_3 > 0.5 \text{ and } var_1 < 0.1\}$

2.6 Summary of the chapter

This chapter provided an overview of the Machine Learning field and the specific topics that are related to this thesis. A brief description about classification rule induction was described in section 2.2. An explanation about the problem of imbalanced classes

Figure 2.5: Grammar 1

G	→	<Root>
<Root>	→	"If-then-else",<Conjunction> | <Condition>,"Class","No Class"
<Condition>	→	<Operation>, <Variable>, <Threshold>
<Conjunction>	→	"AND"|"OR",<Conjunction>|<Conditional>, <Conjunction>|<Conditional>
<Operation>	→	"<" | ">"
<Variable>	→	var_1 | var_2 | ... var_n
<Threshold>	→	Real number

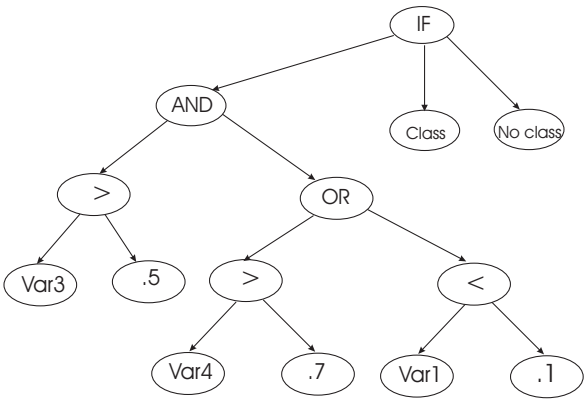


Figure 2.6: A decision tree that meets the syntax of Grammar 1.

was introduced in section 2.3, additionally some approaches to deal with this problem were briefly described. Section 2.4 described the metrics that were used to evaluate the performance of the approaches in this thesis. Section 2.4.3 provided a brief introduction about the ROC curve. Finally, section 2.5 gave an introduction of the context free grammars.

Chapter 3

Evolutionary Computing and Genetic Programming

Last chapter provided an overview of the machine learning area, now the present chapter focuses on the specific machine learning paradigm that is used in the contributions of this thesis, which is Evolutionary Computing. First a brief introduction of the biological processes that have inspired the field of evolutionary computing is given in section 3.1. Subsequently a resume about the main aspects of evolutionary computing is provided. Next, section 3.3 introduces Genetic Programming (GP), which is the specific evolutionary technique used to build the contributions of this thesis. In this section there are some specific topics whose theory supports the approaches proposed in this thesis, such as the GP properties (section 3.3.1), introns and bloat (section 3.3.3), convergence and diversity (section 3.3.4). Finally, a survey literature review about evolution of rules is given in section 3.4.

3.1 Evolutionary theory

The theory of the evolution was developed by Charles Darwin in 1859 [28], he explained that in a population of individuals, natural selection plays an important role and it favours those individuals that are better adapted or fitted to the environment. If an individual behaves positively in the environment it will be propagated by means of the offspring, otherwise this will die without producing any descendant. Darwin identified small random variations between generations, whose objective is to introduce variety in the population, these alterations are called *mutations*. He pointed out that the success of an individual depends on how well it is adapted to the environment. The remaining of this section is dedicated to relevant issues in the understanding of the evolutionary process. Thus, the next paragraph introduces the concept of Deoxyribonucleic Nucleic Acid, which has inspired the representation of individuals in evolutionary computing.

Deoxyribonucleic Nucleic Acid

The discovery of Deoxyribonucleic nucleic acid (DNA) started in 1869 when Friedrich Miescher obtained the first crude purification of DNA. He analysed the properties and composition of the DNA substance. Later in 1953 Watson and Crick deciphered the DNA structure. It was not until 1944 that Avery, MacLeod, and McCarty demonstrated that DNA is the molecular material that transmits the hereditary information [27]. DNA comprises the genetic instructions used in the process of development and functioning of all living organisms. DNA contains the code needed to create other components of cells, such as proteins and molecules. The DNA segments that carry

the genetic information are called *genes*. DNA is organised into structures called *chromosomes* and the set of chromosomes within a cell forms a genome. A chromosome constitutes a physically organised structure of DNA. Evolutionary computing has taken these concepts as inspiration to represent individuals. For more detailed information about this topic the interested reader is referred to [7], [4].

Genotype and phenotype

In 1911 Johannsen pointed out the difference between *phenotype* and *genotype*. *Genotype* is the specific DNA code of a living organism or individual, this information is written in a genetic code. This is copied at the time of cell division or reproduction and is passed from one generation to the next [4]. These control the formation of protein macromolecules, as well as the regulation of the metabolism and synthesis. *Phenotype* is the physical manifestation of a living organism, this is composed of the set of observable features of the individual, such as the physical appearance and constitution or a specific manifestation such as size, colour, or behaviour. These concepts will be used in chapters 6 and 7 where a collection of rules is formed. The selection of these rules is based on the performance (phenotype) and the novelty, which is determined by the genotype.

3.2 Evolutionary Computing

Evolutionary Computing (EC) is an area of computer science, it involves the study of problem solving, optimization and ML techniques. Evolutionary Computing is inspired

by biological processes such as population-based evolution and natural selection. EC mimics the principles of evolution and hereditary [34]. One of the first works in this field was made in 1965 by Fogel [42] and it was called *evolutionary programming*. In 1975 Holland introduced the genetic algorithms [58]. Michalewicz [82] uses the term Evolution Program (EP) for all the systems that are based on the evolutionary theory. He defined an EP as a probabilistic algorithm that embraces a *population* of *individuals* $P(t) = \{x_1^t, \ldots, x_n^t\}$ for *generation* t. Where each individual represents a possible solution to the problem. A new population is created (generation $t + 1$) by means of selecting the fittest individuals. In order to create new solutions, part of the population is renovated by means of the *genetic operators*. The most common operators are *mutation* and *crossover*. Mutation is a small alteration in the individual, while the crossover is the combination of two (or more) individuals to produce new descendants, Figure 3.1 describes the steps of a general EP algorithm. Michalewicz [82] considers that every EP must have the following components:

1. A genetic representation to express the potential solutions of the problem
2. A mechanism to create an initial population of potential solutions
3. A function to evaluate the performances of the individuals.
4. Genetic operators that create new individuals.
5. Values for the parameters that control the EP.

Now, let us describe the general and common aspects of the mentioned elements. However, the specific characteristics that only apply for Genetic Programming, are

```
Procedure evolution program
Begin
  t ⟵ 0
  initialize P(t)
  evaluate P(t)
  while (not termination-condition) do
  Begin
    t ⟵ t + 1
     select P(t) from P(t − 1)
     alter P(t)
     evaluate P(t)
   end
end
```

Figure 3.1: Structure of an evolution program [82]

given in section 3.3.

The representation of the individuals

Representation is the way to symbolize the genotype of the individuals. The representation must be able to express the solution to the problem. The individual could be represented by a binary string as in Genetic Algorithms [58], decision trees as in Genetic Programming [69], graphs [110] or grammar [100].

A mechanism to create an initial population of potential solutions

The objective of the population is to hold a set of possible solutions of the problem. Usually the initial population is generated at random, but some heuristics can be used to guide the initial individuals toward the optimal solution.

Fitness function

The fitness function measures the ability of the individual to solve the problem, in other words, it measures how the individual is adapted to the environment (μ). The fitness can be evaluated in different ways, it depends on the nature of the problem, the most common approach to measure fitness is to create a metric, which assigns a scalar fitness value to each individual in the population.

Genetic Operators

The objective of the genetic operators is to create new individuals derived from the existing ones. The most popular operators are *crossover* and *mutation*, let us explain each of them in detail:

Crossover The objective of the recombination or crossover is to create new individuals by merging the information of two (or more) individuals which are called parents. Crossover is a stochastic process because this is applied based on a probability, where the fittest individuals have a high chance to be selected to have offspring. However, the population may converge to a global suboptimal, when a mediocre individual (suboptimal) has extraordinary good fitness in comparison with the remaining population. In that cases there is a genetic operator that can reintroduce new variety in the population, it is called *mutation*.

Mutation A mutation is a small variation in a random place of the genotype of the individual. This is used mainly to introduce new variations in the population. This operator reconstitutes the diversity that has been lost during the exploration

phase. This is a stochastic process because it is applied at random based on a probability that is determined by the user.

Values for the parameters that control the EP

All the evolutionary techniques hold a set of parameters that control the execution, the commonest parameters are: the number of generations, population size or the probability to apply the genetic operators. The list of parameters varies according to the type of technique used and the specific characteristics of the EP.

Finally, another important aspect of the EP is the stopping condition. The population could be evolved until a performance goal is achieved. However, EPs are stochastic processes and there is not the certainty that a specific optima is going to be reached. For that reason, there are more stopping criteria, such as to reach a specific number of generations or to evolve the process for a specific period of time.

3.3 Genetic programming

This section introduces the specific evolutionary technique that is used in the approaches proposed in the contributions of this thesis. Genetic Programming (GP) developed by Koza [69] is an evolutionary technique whose individuals are computer programs. These are hierarchically structured and undergo adaptation by means of changes in size and shape through the evolutionary process [69]. The most popular representation of the GP systems are decision trees. However, other structures, such as graphs and finite state machines [108], are also used.

Reasons to choose GP as ML tool

To achieve our objectives GP has been selected as a supervised learning tool for the following reasons:

1. GP is able to evolve dynamic structures in size and shape to represent the solutions.

2. GP is capable of producing multiple solutions to a single problem. This characteristic is very important in RM and EDR (chapters 6 and 7 respectively). The variety of solutions helps to form a bigger set of rules to identify rare cases in imbalanced environments.

3. GP is able to produce decision trees that can be understood by the user. This characteristic is a key factor in this research, because the interpretability of the solution allows the user to analyse the variables and conditions which are involved in the decision. This allows the user to combine his/her knowledge to make a more informed decision.

The elementary components to build an individual (program) in GP are the *terminals* and *functions*, these are the alphabet of the programs. The terminal set is composed of the variables and constants of the programs. In a typical tree-based GP the terminals are located on the end of every branch. On the other hand, the functions are responsible for processing the values of the system, these can be terminals or other functions' output. The function set is composed of the functions and operators that help to express the solution of the problem. The function set can be constituted by a

great variety of operators, such as:

- Arithmetic operations (+,-,*,etc.)
- Mathematical functions (sin cos, exp,log)
- Boolean operations (AND, OR, NOT)
- Conditional operator (if-Then-Else)
- Functions causing iteration (Do-Until)
- Comparison operators ($>$, $<$, $\neq$, $=$)
- Other domain-specific functions may be defined

Koza [69] has summarized the main three steps for evolving a population of computer programs as follows:

1. **Generate an initial population** - GP starts with an initial population of computer programs, these must be suitable programs to solve or approximate a solution of the problem. These were created using the pertinent terminals and functions to deal with the enquired problem. Usually these programs are generated at random, but some initial heuristics can be used.
2. **Perform the following action until a termination condition has been reached**

2.1 Execute every program in the population to measure its fitness, this is associated with the quality of the solution proposed for the problem.

2.2 Create a new population of computer programs by means of the actions described below, these are applied using a probability based on fitness.

(a) Copy existing computer programs to the new population, it is also called *clonation*

(b) Produce new computer programs by means of the recombination or *crossover*.

(c) Introduce random changes in the structure of the individuals by means of the *mutation*.

3. The best computer program, according to the fitness function, is designated the best approximated solution to the problem.

Initialization of the population

The initialization is the creation of the first population, in the majority of the cases it is generated at random. However, in some cases it is created using heuristics. Given that the representation of the GP individuals has variable size, the initialization of every individual can use the full and/or the growth method. The graphic representation of the initialization methods is illustrated in Figure 3.2

- **Full method** each branch of the tree has a predefined number of nodes. Thus, the decision trees created by this method have regular shape, as Figure 3.2 shows.

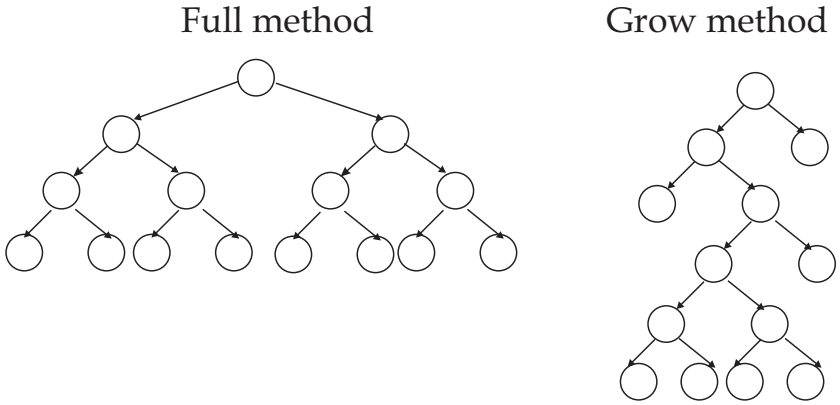


Figure 3.2: The graph representation of the *Full* and *Grow* methods to create a new population of programs

- **Grow method** every branch may have a different number of nodes, the tree is constructed stochastically with a *maximum program size*. For a tree-based GP the maximum program size can be seen in two different ways: 1) maximum number of nodes in the whole decision tree or 2) the maximum depth of the tree. The depth of a node n_i is measured by the minimal number of nodes from the root to the node n_i. Thus, the maximum depth of the tree T is the maximum depth of the nodes in T.

Fitness and Selection

The fitness is a metric that indicates how well a program is performing to solve a problem. This measure is used during the evolution to determine the probability to be selected for crossover or reproduction. There are different selection techniques based on the fitness of the individual. However, the most popular methods are the following:

fitness-proportionate selection This method was proposed by Holland [58]. Let $f(s_i(t))$ be the fitness of the individual s_i in generation t. Under this method the

probability that individual s_i will be copied in the next generation is given by the following formula:

$$\frac{f(s_i(t))}{\sum_{j=1}^{n} f(s_j(t))}$$

where n is the number of individuals in the population

Tournament In the tournament selection a set of individuals is picked out from the population at random, the number of individuals involved in the tournament is determined by the user. Then the individual with the best fitness is finally selected.

The main genetic operators are: crossover, mutation and reproduction [4]. The reproduction operator is very simple, it consists in making an identical copy of the individual. However, the other operators need a more detailed explanation:

Crossover (sexual recombination)

The crossover creates new offspring by taking parts of each parent. As GP is usually represented by decision trees, the crossover works as follows: Once the parents are chosen from the population, a node in each parent is selected randomly, it is called cross-point. Notice that, the size of the parent in the majority of the cases is different. The crossover cuts parents in two parts: 1) the subtree from the root to the cross-point and 2) the subtree lying from the cross-point. The first child is built using the rooted part of the first parent and the subtree lying of the second parent. The second child is formed in similar way, swapping the roles of the parents. In other words, the second parent contributes to the rooted part of the subtree. Figure 3.4 shows a recombination

of two individuals to create new offspring. The idea behind the crossover is that if both parents are good enough to deal with a problem, these new individuals that combine parts of those parents can outperform the parents fitness.

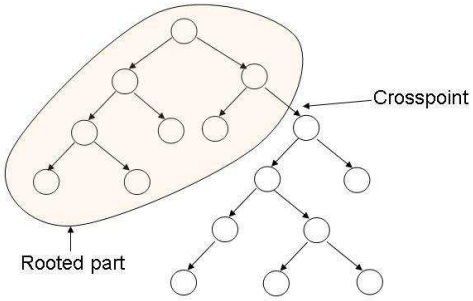


Figure 3.3: The cross-point and the division of the decision tree made by the cut

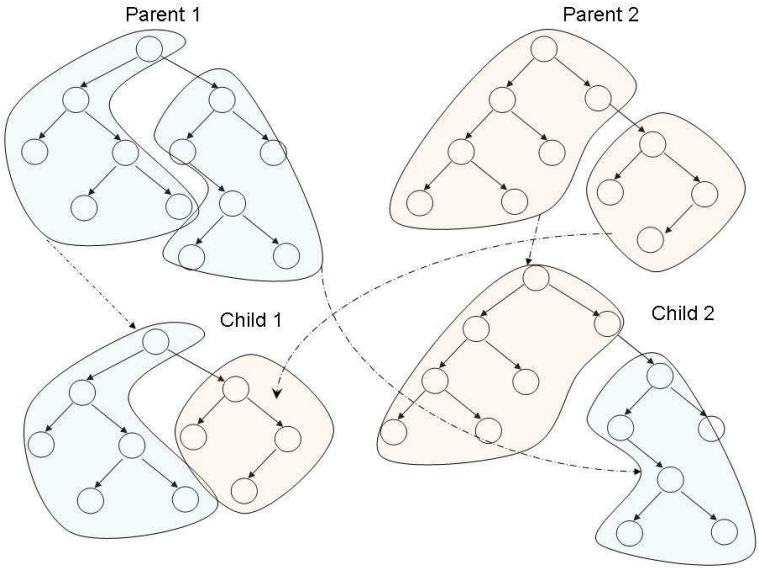


Figure 3.4: A recombination of two parents in order to create a new individual

Mutation

The mutation in an asexual operator, it introduces random changes in the structure of the individuals. The mutation starts selecting a node *mutation-point* in the individual,

the subtree lying in this mutation-point is removed and this is replaced by a subtree generated at random. Figure 3.5 shows an example of a decision tree that was mutated.

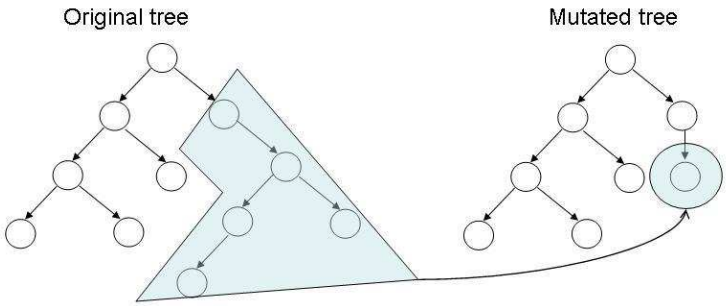


Figure 3.5: Figure shows a mutated decision tree

3.3.1 GP closure and Sufficiency properties

As we have previously mentioned, GP is an evolutionary technique that uses dynamic structures that change in shape and size. For that reason, there are some properties that are exclusively for GP and these are not competence of other evolutionary algorithms, such as the *closure* and the *sufficiency* properties. These properties are relevant for this research, for that reason this section provides a brief introduction about the mentioned properties.

Closure of the function and terminal sets

Given that GP deals with variable structures that have to be combined, it is important that the function and terminal sets have the *closure property*. This property states that the each function should be able to deal with all the values that it may receive as input [4]. To understand the closure property, let F be the set of functions and T be the

set of the possible terminals in the GP definition. The closure property requires that functions in F accept any possible value that may be output from any composition of functions in F and terminals in T [69]. This is easy to achieve for some function and terminal sets, such as boolean functions. However, more attention is needed in most domains. The closure property is desirable, but it is not absolutely needed, instead it is possible to discard individuals which does not represent a feasible solution or to assign penalties.

Sufficiency of the function and the terminal sets

The *sufficient property* requires that the set of functions and terminals of the GP will be able to express the solution of the problem. When a GP is designed it is important to analyse the problem carefully in order to select the variables with explanatory power and the operators that can represent potential solutions to the problem.

3.3.2 Building Blocks

The schemata theory presented by Holland [58] gives a robust explanation of why GAs works. This theory was designed for fix-length strings, for a detailed explanation the reader is referred to [58], [82]. However, the analogue theory for GP is more complex because individuals have dynamic structure with variable size and shape. Koza has asserted that the individuals generated by GP hold *building blocks*, this could be any subtree or tree. Individuals that hold good building blocks have better fitness. When individuals have good fitness these improve the likelihood of being selected for reproduction or crossover, it means that good blocks are likely to multiply and spread

over the population. However, the argument of Koza was informally formulated, this principle was extended by O'Reilly and Oppacher [90], who based on the schemata theory from Holland, and the do not care symbol # was introduced to define fragments of tree. After that Poli and Langdon [75] have developed a new schema theorem for GP using a one-point crossover and one-point mutation to evolve the GP.

3.3.3 Bloat and introns

This section introduces the concept of introns and bloat, which will be used in chapters 5,6,7 and 8.

One of the main advantages of GP is that it is able to evolve a population of programs that have dynamic shape and size representation, which lets them adapt to the solution. However, it is difficult to determine the size or the shape of a program in advance. The programs produced by GP tend to grow. However, this growth is not necessarily proportional to the quality of the resultant solution. The code growth is a common problem in GP [2, 88, 104, 74] , the extra code constitute between the 40% and 60% of the total code, this effect is called *bloat*. This phenomenon was discovered by Koza [69]. However, Angeline [2] was the first researcher to associate this junk code with the term *introns*. In biology, introns are assumed as a sequence of junk DNA with no function. However, more recently this definition has been questioned, since it is known that introns contain several short sequences that are important for efficient splicing. In GP the code growth has been studied for Tackett [107], he demonstrated that the "size problem" depends directly on selection pressure. He asserted that under

random selection there is no size problem. On the other hand, Blickle and Thiele [8] asserted that code growth protects individuals against the disruptive effects of the crossover, they asserted that an individual with "redundancy" is more likely to survive the destructive effects of crossover. The same argument has been supported by other research, such as [88], [74]. On the other hand, Soule [104] asserts that crossover and mutation protection is not the only cause of code growth. He argued that the representation of the problem (syntax of the decision tree) also affects the code growth.

Code growth has been controlled introducing a variety of methods. According to Soule [105], the methods to control bloat can be grouped in three main types: *parsimony pressure*, *operator modification* and *code modification*. Parsimony pressure tries to evolve small solutions penalizing large individuals, for instance, to establish a maximum depth allowed [69], tarpeian method [1] [93] or the implementation of the Minimum Description Length principle (MDL) in the fitness function [61]. Operator modification is represented by non-destructive crossover [103]. Finally, code modification involves changing the structure of the code during or after the evolution, e.g. the pruning method implemented by Eggermont *et al.* [33].

According to Soule, code modification methods have not been explored in depth because these involve the use of more computational resources [103]. However, research on other machine learning techniques [15, 98] prefers pruning instead of a stopping criterion, it has been pointed out that pruning is slower but more reliable because this produces more exploration. Unfortunately, the pruning procedures for decision trees

[1] This method assign zero to a proportion of the offspring whose length is above the average of the population size

generated by statistical methods as CART [15] or Quinlan's classifiers [97] are not suitable for decision trees generated by GP systems. The decision trees created by GP are built in different way and the pruning affects are different, as can be noticed in the following cases: 1) pruning a leave from a statistical decision tree causes generalizing the classification. But, the decision trees created by GP tend to accumulate unused code, thus pruning a node from a GP decision tree may or may not generalize the classification. 2) statistical decision trees hold the most general conditions near to the root, thus the information of each condition is more specific depending on the node level[2]. In contrast, the type of information provided by the conditions in the GP decision trees is not necessarily related to the position of the node.

In our understanding the only method to prune decision trees produced by a GP system, before the submission of this thesis, is the approach proposed by Eggermont [33]. He proposed to analyze the decision trees to remove the unused code generated by the GP system. He argues that the resulting trees contain only meaningful information and these are smaller and easier to understand and evaluate.

Eggermont used static analysis techniques to detect introns from decision trees. Thus let T be an individual, thus T has to be evaluated is scanned for introns. These introns are marked and a pruned copy T_0 of T is made in which the introns are removed. If T0, which is semantically the same as T, matches a tree T_c found in the fitness cache the individual T is assigned the fitness of T_c. If T_0 does not match any tree in the cache it is evaluated using the fitness function and stored in the cache. In this case T is

[2]The level of a node n_k is the number of nodes from n_k to the root

assigned the fitness of T_0. The experimental results showed that the understandability and speed of GP classification algorithms can be improved, without compromising the accuracy of the prediction. Additionally, the detection and pruning of introns also lets to identify syntactically different trees that are semantically the same.

3.3.4 Population diversity and convergence

This subsection introduces some material related to the diversity, the objective is to support the contributions that are presented in chapter 6 and 7 of this thesis, which implemented a mechanism to ensure the variety of the population. In order to emphasize the relevance of the diversity, we present the following citations from purely biological material. Charles Darwin wrote in his book *On the Origin of Species*

> *"...unless profitable, variations do occur, natural selection can do nothing"* [28]

Furthermore Murray [84] asserted the following :

> *"Variation is an essential property of biological systems, every level or organisation presents variations in some parameters, in some space or time". "...Probably a very small biochemical change will give a host species a substantial degree of resistance to a highly adapted microorganisms. This has an important evolutionary effect. It means that it is an advantage to the individual to possess a rare biochemical phenotype. Because of its rarity it will be resistant to diseases which attack the majority of its fellows. And it*

means that it is an advantage to the species to be biochemically diverse, and event to be mutable as regards genes concern in disease resistance" [84]

In the computer science field Koza [69] defines variety as the percentage of individuals that have exact duplicate(s) in the population. According to Burke [16], the loss of diversity during the exploration phase may cause a poor global search. Burke and fellows presented an analysis of loss of diversity in GP, from results they concluded that this can cause different behaviour in different problems, in a deceptive problem the increase of diversity helps to tackle the deception.

The *convergence* suggests the lost of diversity in the population and the beginning of the local search phase [16]. According to Koza [69], it is called premature convergence when the population converges to a global suboptimal. Premature convergence can take place when a suboptimal individual has extraordinary good fitness in comparison to the rest of the population, causing that the GP falls in a suboptimal solution.

3.4 Evolution of decision rules

This section introduces the most relevant works about the evolution of decision rules, which is the main objective of the contribution in Chapters 6 and 7. The literature review is divided in three parts, the first introduces some relevant works that follow the *Pittsburgh approach* using Genetic Algorithms (GAs) [58]. The second part describes some systems that have been developed using GAs and the *Michigan approach*. Finally, the last part describes some works that evolve decision rules by using GP.

GAs creates a *population* of *individuals* or solutions using fixed-length binary strings

to represent the individuals in the population. Those works have been grouped by De Jong [30] in two different approaches, the first was developed by Holland [56], it is called *Michigan approach*. The main characteristic of this technique is to evolve rules, where each rule is represented by a chromosome or individual and the complete population constitutes the solution of the problem. On the other hand, there is another perspective to evolve decision rules, which was developed by Kenneth De-Jong and Stephen F. Smith [102]. Each individual represents a complete set of rules, every rule is fixed-length, but every individual is composed of a variable number of rules. This approach has been called *The Pittsburgh approach*. So the fitness function has to be able to measure the effectiveness of the rule set. The interested reader is referred to [30], [62] for more details about the differences in the mentioned approaches. In the beginning both approaches were introduced to classify data sets that hold nominal attributes. The following paragraphs briefly describe some subsequent works that have introduced techniques to deal with continuous values and variable length rules

Pittsburgh approach

De-Jong and colleagues [64] used GA to evolve a set of rules (Pittsburgh approach). The authors called this application GA batch-incremental concept learner GABIL. The GA evolves fixed-length rules, for attributes whose values are nominal. Each individual is composed of a variable number of rules, it means that the individual size is variable too.

Janikow [62] follows the Pittsburgh approach and he proposed GIL (Genetic-based

Inductive learning). He declared three types of operations: 1) the rule set level, 2) the rule level and 3) the condition level. The operations in the rule set level are composed of *rules exchange*, *rules copy*, *new event*, *rule generalization* (it picks two random rules and replaces these by the most specific generalization of both rules) and *rule specialization*, the latest replaces two random rules by the most general specialization of them. In the rule level the operations can introduce or drop conditions, turning conjunctions into disjunctions. Finally, the condition level holds three operators 1) *reference change*, it removes or adds a single domain value to the condition. 2) *Reference extension* whose objective is to extend the domain of the variable in the condition and 3) *reference restriction*, this operator removes some domain values from the condition.

Corcoran and Sen [26] used a GA to evolve a set of rules based on the Pittsburgh approach. The contribution of that work was to evolve rules with continues variables rather than using a binary representation. They reported that using a GA with real number encoding can be used effectively in classification problems. This approach uses a fixed-length chromosome to represent a set of rules. This approach tries to maximizes the number of correct predictions. When rules predict different classes a voting system is used to solve the conflict.

Kwedlo and Kretowski [72] introduced a new approach named Evolutionary Decision Rule Learner with Multivariate Discretization (EDRL-MD). This approach has the ability to search simultaneously for threshold values for all the attributes that hold continuous values (multivariate discretization). The search technique used by this approach is an Evolutionary Algorithm (EA). The individual's chromosome is represented

by a variable-length string, which is formed by a set of fixed-length substrings, each substring encodes a condition related to one attribute. The chromosome encodes a rule set. This approach uses the following genetic operators: 1) changing condition (mutes a single condition of the attribute), 2) positive example insertion, 3) negative example removal and 4) rule drop which are applied to a single chromosome or rule set. Additionally, the crossover and rule copy requires two rule sets.

Bobbin and Yao [9], [10] were interested in evolving rules for nonlinear controllers by using the Pittsburgh approach. They proposed an evolutionary algorithm to evolve sets of rules using a novel degenerated tree rule structure. Every gene represents a single rule and the genes are combined in a complex topology. This allows the coming genes to modify the actions of previously activated genes. Each individual in the population represents a set of rules which finally determines the complete solution of the problem. Given that the algorithm is able to add and remove rules, it means that the algorithm evolves variable-length structures.

Fidelis and co-workers [39] proposed an approach based on GAs in the hope that it could discover comprehensible IF-THEN rules. The novelty of that approach was to have a fixed length at the genotype level but the number of conditions (which is mapped at the phenotype level) is variable. The key ingredient that allows this feature in the genotype-phenotype mapping was an element at the genotype level that the authors called *weight*. As the authors explained in their paper, the results in regard to accuracy were promising in one of the data sets but more importantly, they showed how the GA was able to find consistent and comprehensible rules.

Michigan approach

Let us introduce a brief review about the Holland's classifier systems [58], this approach is the pioneer of the Michigan style. The complete population represents the solution to the problem, every individual in the population represents one single rule, which is composed by a set of conditions. Every rule holds a credit, it determines the fitness of the rule. A key element is a *macht* list, when an instance is classified the system selects all the rules that mach that instance, the "don't care" symbol in the syntax of the conditions is used to allow the generalizations. Given that, the different rules can predict different results, thus a formula is applied to determine the prediction. The rules that match the instance and participate in the decision are rewarded or punished according to the quality of the solution. The evolutionary process creates new solutions by means of the crossover and mutation, these are applied using a probability according to the reward of each individual.

A Learning Classifier Systems (LCS) [57] is a machine learning approach that uses evolutionary techniques, reinforcement learning and other heuristics to create an adaptive system. This is composed of a population of rules and a credit system, which is aided by reinforcement learning and a GA. A distinguished work in this area was developed by Wilson [120], his system (XCS) uses a GA in niches defined by the match sets, instead of applying this to the whole set of examples. According to the author, the population tends to form a complete and accurate mapping from inputs and actions to payoff predictions. The interested reader can find more information about learning classifier systems in [17].

Evolving decision rules using GP

Now, let us introduce some systems to create classification rules based on GP: Bojarczuk et al. [12], [13] proposed a GP system to discover classification rules. Each individual in the population represents a rule set in disjunctive form and classifies just one class of the k classes. Every individual will be evaluated for every possible class. Then the tree is said to classify the class that better suits the tree. The result of the evolution are k individuals, where each of them represents a set of rules for each class. Finally, the performance of every individual is evaluated by its ability to predict the class and the simplicity of the solution. The same authors in a previous approach [11] had proposed a similar method but instead of assigning a different class to each decision tree the GP was run as many times as the number of classes in order to generate a decision tree per class. Obviously the final step is to create an agreement between the predictions of the different k classifiers.

De Falco et al. [29] used a GP to perform an automatic discovery of classification rules. This approach evolves decision trees, that classify just one class, and the GP is run as many times as the number of classes. According to the authors, individuals are formed by rules encoded as a tree structure. In other words, the tree represents a rule which is composed of disjunctions and conjunctions. The contribution of this work is the post-processing work to assign every instance to just one class and eliminate the undetermined cases.

Niimi and Tazaki [85] proposed a rule discovery technique by means of the generation of association rules using an apriori algorithm, those rules are converted in

decision trees, which will be used as the initial population of the GP. After evolving that population, the best individual of the population is converted into classification rules.

Cao et al. [19] introduced a hybrid evolutionary algorithm (HEA) to discover rule sets to predict the concentration of chlorophyll. This application has two main stages: 1) a GP is used to evolve a set of rules and 2) a GA is applied to optimize the random parameters in the rule set. Each chromosome can be represented as a vector of binary trees. Next, a simplification is performed by means of the following procedures: 1) simplification of the arithmetic subtrees and 2) simplification of the comparison subtrees and 3) simplification of the logical subtrees. Finally, each generation, a parameter optimization is performed using a general GA.

It seems that the majority of the research to evolve decision rules has been done in the frame of the GAs, some of the reasons are the following:

- Since a decision tree could contain several rules, a process to separate the decision trees in rules is needed. Especially when rules have to be evaluated individually as the Michigan approach does.

- The GP systems tend to accumulate introns, which may introduce confusion in the interpretation of the rules. Thus, a simplification process is needed to remove the extra-code.

- Some systems that use the Michigan approach, in discrete domains as [17], [120],

have a mechanism to ensure having at least one rule that match every possible instance. The representation of the chromosome of the GAs facilitates the matching conditions. This task is aided by the use of the "don't care" symbol in the syntax of the conditions, which allows generalizations.

As can realize it is easier to implement the Michigan approach using a GA. In contrasts, a Pittsburgh classifier can be implemented using a GP system, because every individual represents a complete set of rules, as a decision tree usually does. For that reason, some approaches to evolve decision rules using GP have been proposed [13], [29], [19] and [85]. However, we have to bear in mind that GP generates extra-code that should be removed in order to avoid confusion in the interpretation of the rules.

Differences and advantages of our contributions and the previous works

As can be seen from previous paragraphs, it is clear how GA and GP have been used to evolve decision rules. Our approaches, as will be explained in Chapters 6 and 7, have been designed to produce a set of rules to classify positive cases in imbalanced environments and to provide a range of classifications. The next paragraphs present a brief discussion about the differences and advantages of our contributions and the previous works.

Learning classifiers that use the Michigan approach, as [120], [57], have implemented heuristics to ensure having at least one rule that matches every potential instance in the data set. Since at least one of the rules has to match every instance, some problems may arise when the number of attribute is high, because the number of possible cases

could be extremely big. Additionally when more that one rule classify the instance and these do not agree about the classification, then a formula or a voting system has to be implemented to decide the class.

One of the objectives of this work is to predict the minority class in imbalanced data sets, for that reason our approaches (chapters 6 and 7) focus on gathering patterns that identify the minority class. However, to decide if a rule is good enough to make the prediction could be a problem. We have taken advantage of the collection of rules and we have proposed to group these by their precision, it allows us to create a range of classifications to suit the risk preferences of the investor. In contrasts, despite the fact that the result of a traditional Michigan LCS is a set of rules, it is not able to provide a range of classifications to suit the user's needs as our approaches do. Because the Michigan approach generates rules that match every possible case, if the rules were grouped by precision or another parameter to create a range of solutions many cases could not be considered in each group. Another mayor difference is the representation of the rules; our approaches are able to represent variable-length rules by evolving decision trees using GP. However, the side effect is the risk of having extra-code (introns), for that reason our approaches have implemented a process to remove the redundant code and the conditions that are not affecting the rule.

On the other hand, standard LCS represent rules using fixed-length structures that have to be carefully designed to achieve a good representation of the solution, specially to represent continuous values. Finally, our approaches are able to tune the conditions' thresholds taking advantage of the complete evolutionary process.

Since an objective of this work is to detect the minority class in high imbalanced data sets, we have proposed to gather as many patterns as possible of the minority class, it means that the same example could be classified using different rules. In the Pittsburgh approach each individual holds a set of rules that represents the complete solution to the problem. In this framework it is not possible to collect different patterns that classify the same examples as our approaches do.

As was mentioned, some researchers have proposed approaches to evolve decision rules by using GP [85], [12], [13]. Every decision tree classifies a single class, when the evolutionary process has been completed the best individual of the evolution is converted into classification rules. The authors claim that their approaches are able to generate comprehensible rules. However, GP systems accumulate extra-code [69],[2],[107], [103] and unless the rules are simplified to remove the extra-code it is not possible to identify the real conditions in the decision rule. In contrast, the methods proposed in this thesis provide a rule simplification process that removes the extra-code helping to identify the real conditions of the rules. Cao et al. have included a simplification process in their approach. However, this is focused on facilitating the execution of the program by removing the redundant conditions. But it does not remove all the conditions that are not affecting the performance of the decision rules such as the *vacuous conditions* (see definition 4 in section 5.6). In contrast our simplification process removes those conditions. On the other hand, our approaches collect as many patterns as possible of the minority class. But the approaches that use GP to evolve a set of rules, as [12], try to minimize the size of the decision tree to reduce the number of

introns, this may limit the productions of patterns.

Finally, the approaches that use GP to get a set of rules obtain the rules from the best individual of the evolution. In contrast we believe that the complete population could contain useful information, for that reason our approaches collect rules and tune the conditions' thresholds using the information in the complete population.

3.5 Summary of the chapter

This chapter provided an overview of the Evolutionary Computing field. A brief introduction of the biological processes that have inspired this field was given. The chapter also introduced the evolutionary technique, which is called Genetic Programming. The latest will be used in the subsequent chapters to build the contributions of this thesis (Chapters 5, 6, 7 and 8). Finally, a survey literature review about evolution of decision rules was provided, this information will be used in chapter 7.

Chapter 4

Financial Analysis

Many machine learning techniques have been applied to financial problems. For example, [23],[24] provide excellent readings from various areas of computational finance. Financial forecasting is one of the most important fields in this area [115]. As was mentioned in previous chapters the methods proposed in this thesis use supervised learning. Our approaches are illustrated to predict opportunities in financial data sets, the examples to train our methods are composed of a signal and a set of attributes or independent variables. Those variables are financial *indicators* derived from *technical analysis*. This chapter explains basic financial elements in order to provide the reader with an overview of financial markets and technical analysis. This chapter is organised as follows: Section 4.1 provides a brief description of financial markets. Next, section 4.2 introduces the Efficient Markets Hypothesis (EMH).

The next sections introduce two different ideologies for the financial markets, thus section 4.2.1 introduces the *fundamentalism*, while section 4.2.2 describes the *technical analysis*. Finally, section 4.2.2 describes the technical analysis indicators used in this research.

4.1 Financial Markets

Financial markets can be defined as an association of institutions whose objective is to act as an intermediary between suppliers and users of money. This mechanism allows to trade financial securities, commodities and other exchangeable items. According to Bain [3], the financial system has five main groups of participants: 1) savers, 2) Investors and borrowers, 3) financial intermediaries, 4) brokers and advisers and 5) regulators. The end-users are the *savers* and *investors*, the savers have money accessible to lend and the investors need to borrow money to buy capital goods or improve their business.

One of the main objectives of financial markets is to exchange funds between different economic units [31]. The financial markets can be divided into different categories according to the traded assets:

- Capital markets
- Bond markets
- Commodity markets
- Derivatives markets
- Futures markets
- Foreign exchange markets

The examples presented in this thesis are focused on the analysis of prices in stock markets, which are part of the capital markets.

4.2 Efficient Market hypothesis

One of the most relevant concepts in finance is efficiency. The Efficient Market hypothesis (EMH) asserts that financial markets are "informatively efficient", or that prices on traded assets, reflect all known information and therefore are unbiased in the sense that they reveal the collective beliefs of all investors about future scenarios [36]. The EMH states that it is not possible to consistently outperform the market by using any information that the market already knows, except by luck. In EMH hypothesis, information is defined as anything that may affect prices that is known in the present and thus appears randomly in the future. A market is efficient with respect to a particular information set A if it is impossible to make abnormal profits by using this set of information to make trading decisions. There are different forms of efficiency [31]:

1. *Weak Form Efficiency.* A market is weak form efficient, if security prices reproduce the information in past price movements

2. *Semi-strong Form Efficiency.* A market is semi-strong form efficient, if the security price fully indicate all publicly available information, such as the company annual reports, earning and dividends announcements and so on.

3. *Strong Form Efficiency.* A market is strong form efficient, if the security prices fully reflect all relevant information publicity available or not.

4.2.1 Fundamental Analysis

The aim of *Fundamental analysis* is to examine the corporation's financial statements and balance sheets in order to prognosticate future trends of their securities. Fundamental analysis studies past records of assets, earnings, sales, products, management and markets, such analysis relies on the idea of the existence of a fundamental value for the financial asset. Fundamental analysts believe that the asset will be priced at its real value. In other words, if the asset is *undervalued* the recommendation is to buy, because the asset will rise to its fundamental value. Such strategy produces a profit due to the fact the asset was bought when it was cheaper. On the other hand, when the asset is *overvalued* the recommendation is to sell.

4.2.2 Technical Analysis

The historical data has been used by financial analysts as a source of information to try to predict future events. It is believed that investors and speculators react on the same way to the same kind of events [66]. This means that the patterns in financial prices are usually followed by similar reactions.

Technical analysis is a technique to evaluate stocks by analysing statistics generated by market activity, such as past prices and trading volume [32]. Technical analysts attempts to identify patterns that can suggest future behaviour, it does not pretend to measure a security's intrinsic value. Technical analysts believe that the historical information about the stocks and markets can indicate the tendency of future performance. The objective of technical analysis is to find patterns in price changes, rates of

change, and changes in volume of trading without taking into account the fundamental market factors. There are investors who make their decisions based on the advice of technical analysis, they are called technicians. Technical analysis supposes that stock prices have trends, and the motion of the stock trend will continue in the same direction. The technical analysis aim is to detect the direction and the strength of the trend. The earlier the trend is detected the more profit can be made by the investor. Two classical concepts in technical analysis are support and resistance. Such concepts refer to price levels at which prices stop going up and down.

4.2.3 Financial indicators

There are many indicators derived from the technical analysis. However, this chapter only describes the financial indicators that were used in this research. Nevertheless, the techniques developed on this work are not limited to such indicators.

Moving average

Nowadays there are many types of moving average. However, the basic is the simple moving average, this is the mean of the previous n data points. Let $P(t)$ be the price in the time t then the moving average of the last n closing prices is determined by the formula:

$$MA(t,n) = \frac{\sum_{k=1}^{n} P(t-k)}{n} \tag{4.2.1}$$

Two important aims of moving averages are to emphasize the direction of a trend

and to smooth the fluctuations on prices and volume, having as a consequence the reduction of noise. Typically, upward momentum is detected when a short-term moving average (e.g.5-day) crosses above a long-term moving average (e.g. 50-day). On the other hand, a *downward momentum* is confirmed when a short-term average crosses below a long-term average.

Technicians need to keep their perspective. Thus, they need to detect changes on minor and major trends. For that reason, technical analysts compute moving averages with different periodicity (usually a short-term and a long-term). For example, it is useful to use a moving average with a short and long period of time, a moving average of 5 days is able to detect a change in a small trend and a moving average of 60 days can detect changes in a bigger trend.

It can be seen that, technical analysis indicators are statistical measures that can easily be applied in a great variety of situations, not just to financial problems.

Filter indicator

This indicates to buy and sell stocks if their price movement reverses direction by a minimal predefined percentage. For example the filter rule indicates that the stock should be bought if this reverses a downtrend and rises by specific percentage from its low price.

The equation of the filter indicator is the following:

$$Filter(t,n) = \frac{P_t - P_{min(1,n)}}{P_{min(1,n)}} \tag{4.2.2}$$

where P_t is the price in the time t and $P_{min(1,n)}$ is the minimum price in the time from 1 to n.

Breakout indicator

At this point let us introduce the *support* and *resistance* definitions. The support is a price level at which the price stops going down. On the other hand, resistance is the point which the price stops going up. These concepts will be useful to understand some indicators. The *breakout* happens when a new trend (upwards or downwards) starts. The point at which a price breaks out either above or below a stable tendency or trend-line. Once the price breaks above its resistance or below its support, its likely to continue in the same direction. The formula to calculate the Trading break-out indicator is described below.

$$TRB(t,n) = \frac{P_t - P_{max(1,n)}}{P_{max(1,n)}} \tag{4.2.3}$$

where $P_{max(1,n)}$ is the maximum price in the time from 1 to n.

Momentum indicator

The Momentum indicator measures the acceleration or speed at which the security's price is changing. It is determined by the formula:

$$Mom(t,n) = P(t) - P(t-n) \tag{4.2.4}$$

The moving average momentum indicator is defined as follows:

$$MomMA(t,n) = \frac{\sum_{k=1}^{n} Mom(t-k,n)}{n} \tag{4.2.5}$$

In addition, some financial indexes were used, such as:

Financial Times-Stock Exchange 100 stock index

The *Financial Times-Stock Exchange 100 stock index* is the index of 100 large capitalization companies stock on the London Stock Exchange, it is also known as "Footsie"

LIBOR

The LIBOR is established on a daily basis by the British Bankers' Association. It is an interest rate at which banks can borrow funds from other banks in the London interbank market. The LIBOR is derived from a filtered average of the world's most creditworthy banks' interbank deposit rates for large loans with maturities from overnight to one full year. The LIBOR is the world's most widely used point of reference for short-term interest rates. This is important since it is the rate, at which the world's most preferred borrowers are able to borrow money. Some of the countries that use the LIBOR as a reference rate are the United States, Canada, Switzerland and England.

4.2.4 Risk

The approaches presented in chapters 6 and 7 present two different approaches, whose objective is to provide a range of classifications. Thus, the user can choose the best trade-off between misclassification and false alarms according to the investor preferences, which usually are related to the user's risk guidelines. This section introduces

some concepts related to the risk.

A fundamental idea in finance is the trade-off between risk and return. The greater the amount of risk that an investor is willing to take on, the greater the potential return is. Thus, lower returns are associated with lower risk investments and higher potential returns are associated with investments of higher risk, because investors need to be compensated for taking on additional risk.

A *risk averse* investor tends to take higher risk only if it is warranted by the potential for higher returns. On the other hand, a *risk lover* investor behaves in opposite direction: i.e. making investments of higher risk with a lower expected return.

Because of the risk-return tradeoff, investors must bear in mind their personal risk tolerance when choosing investments. Taking higher risk is the price for achieving higher returns; however, in order to make money investors can't cut out all risk. The goal instead is to find an appropriate balance that generates profits, at a bearable risk rate to the user.

4.3 Summary

This chapter has presented an overview of the financial topics involved in this thesis. The financial indicators presented in this chapter will be used to form the data sets used to test the contributions of this work.

Part II

Thesis Contributions

Chapter 5

Research overview

The contribution of this thesis is composed of three methods called: Repository Method, Evolving Decision Rules and Scenario Method. These approaches share common procedures such as the *delimitation of rules* or *rule simplification*. This chapter contains the details of those processes that are used in more than one method. It was decided to set apart those procedures in order not to repeat the same information in different chapters and to simplify the description of each method. This chapter also presents the general structure of the data sets used in the experimental sections. The detailed information about specific data sets is usually described in each chapter in the experimental section. This chapter is very important because it describes in detail the objective and the mechanism of each procedure, which will be used in the subsequent three chapters.

This chapter is organised as follows: First, section 5.1 briefly introduces the methods proposed in this thesis, this description is used as a reference to understand the processes described in this chapter. Section 5.2 describes the procedure to assign the output (label) to the training and testing data sets. Next, section 5.3 describes in

general the data sets used in this research. Section 3.3 explains the reasons why GP has been chosen as a machine learning tool to discover patterns. Section 5.4 explains the process to create decision trees capable of detecting an increase/decrease in stock prices. The following sections describe common procedures that were used in the methods proposed in this thesis. Thus, section 5.5 explains the *rule extraction procedure*, while section 5.6 introduces the process of *Rule simplification*. Next, section 5.7 describes the actions taken to detect new rules (*New rule detection*). Finally, section 5.8 summarizes the main aspects of the chapter.

5.1 Approaches proposed in this thesis

We focus our attention on the detection of the minority class (positive cases) using binary classification. This inspired by the desire to predict big movements in stock prices. The occurrence of such movements is infrequent, but these may have a significant impact. On the other hand, we are interested in generating a range of solutions capable of fitting different user's needs. Finally, an important characteristic for the user is the interpretability of the solutions, this allows us to analyse the conditions and variables that are involved in the solution. This understanding allows the user to apply his/her knowledge in order to make a better decision. Repository Method and Evolving Decision Rules, described in chapters 6 and 7 respectively, are focused on situations where positive instances are rare. On the other hand, Scenario Method (Chapter 8) proposes a pruning procedure, whose objective is to improve the precision and accuracy of the predictions. Notice that, the complete explanation of each method is described

in the subsequent chapters, the information provided in the coming sections does not pretend to be a substitute for the full explanation. Therefore, the following sections are useful in providing a context in which to refer to the procedures described in this chapter.

5.1.1 Repository Method

The objective of Repository Method (RM) is to extract and collect different rules that classify the positive cases (rare instances) in different ways, increasing the probability of identifying similar cases in future data sets. RM analyses a set of solutions provided by a standard GP. Then the rules (patterns) in every decision tree are identified. Subsequently, these are evaluated taking into account the performance and novelty. The performance is compared to a predefined precision threshold. A range of classifications can be created by varying the value of the precision threshold. Let us now outline the main steps of RM:

1. *Creation of different solutions:* to generate a set of solutions by using a GP.

2. *Rule extraction:* to analyse every decision tree created by a GP system in order to define their rules. Each rule is evaluated individually to select those patterns that contribute to the classification task.

3. *Rule simplification:* to simplify the selected rules by removing redundant and unused code.

4. *New rule detection:* to detect and store useful and new rules.

5.1.2 Evolving Decision Rules

Evolving Decision Rules (EDR) is an evolutionary system that has been designed to evolve decision rules to form a collection of patterns. The result is a set of rules, these are grouped by precision in order to create a range of classifications. This allows the user to choose the classification that best fit the user's needs looking for the best trade-off between the misclassifications and the false alarms costs. The steps in this approach are describe below.

1. *Initialization of population:* to create a new population of random decision trees.

2. *Rule extraction:* to analyse every decision tree to define its rules.

3. *Rule simplification:* to simplify the representation of each rule by removing unused code.

4. *Adding new rules in the collection:* to examine the simplified rules for detecting new patterns in order to form a collection of rules.

5. *Creation of the next generation:* to create a new generation of decision trees taking as parents the collection of rules originated in the previous steps.

As can be noticed RM and EDR share some procedures, for that reason it is important to focus our attention on the difference between EDR and RM. As was mentioned, the aim of RM is to analyse a set of decision trees (population) produced by a GP system to pick up useful patterns (rules). Thus RM is a deterministic process when a specific precision threshold is applied to a specific population of decision trees. On the

other hand, EDR is an evolutionary process that evolves decision rules. The rules that achieve a specific precision threshold are stored in a repository and the new individuals are created using the rules in the repository as parents. Thus EDR is able to generate different solutions every time this is performed.

5.1.3 Scenario Method

Scenario Method (SM) is a pruning procedure for decision trees that were produced by a GP system. SM analyses every decision tree T to delimit its rules. Every rule is evaluated individually in order to identify the useful patterns in T. The rules that are not contributing to the classification task will be removed from T. We believe that the pruning of non useful rules can help to simplify the decision tree and to improve the accuracy of the prediction. Let us outline the main procedures which are involved in SM.

1. *Rule extraction:* to identify the rules in every decision tree.

2. *Rule evaluation:* to evaluate the performance of each rule.

3. *Rule selection:* to select the rules that contribute to the classification task.

4. *Tree pruning:* to remove from the decision tree, those rules that are not contributing to the classification task.

5.2 Data sets creation

The approaches in this thesis are illustrated by detecting investment opportunities in financial stock markets, using supervised learning (see section 2.1). The experience is provided by a training data set, which is formed by examples, each example provides a series of attributes and the desired outputs. The attributes of each example are indicators derived from the financial technical analysis (see chapter 4). The output is a signal, which is determined looking ahead to a horizon of n units of time, it tries to detect an increase/decrease of at least $r\%$.

Let	P_t	be the price of the stock in time t
	$r\%$	be the minimum percentage of increase/decrease required
	n	be the number of units of time that we are looking ahead
	δ	be the signal, it is 1 when there is an increase/decrese of at least $r\%$ and 0 otherwise.

Thus, the signal is calculated as follows:

$$\delta_{Increase} = \begin{cases} 1 & \text{If } P_t \cdot (1+r) < maximum(P_{t+1}, \ldots, P_{t+n}) \\ 0 & \text{Otherwise} \end{cases}$$

$$\delta_{Decrease} = \begin{cases} 1 & \text{If } P_t \cdot (1-r) > minimum(P_{t+1}, \ldots, P_{t+n}) \\ 0 & \text{Otherwise} \end{cases}$$

The mechanism to find the signal has been previously used in other works [77], [112]. Figure 5.1 illustrates the signal detection. As can be seen, the tendency of the future prices is analysed, looking for a decrease/increase that achieves a predefined threshold.

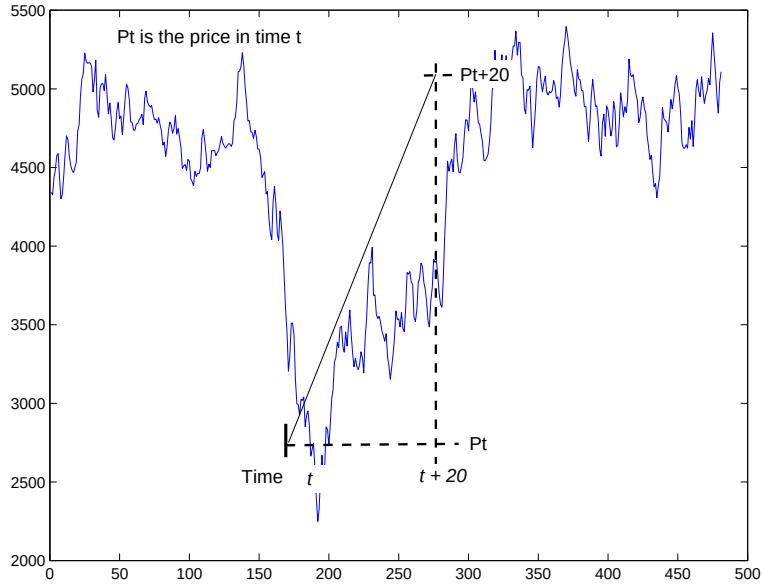


Figure 5.1: How to create the training and testing data sets

5.3 Data sets description

This section introduces the attributes of the training and testing data sets, which were used in the experiments of this thesis. The data sets Barclays, Tesco that were used in chapter 6, 7 and 8, are composed of the independent variables in Table 5.1. As can be seen, every indicator has been calculated using two periods of time, a short and a long one. The detail about the number of examples, the rate of return and the number of days that we are looking ahead for the movement is specified in the experimental section in the corresponding chapters.

According to Bellman [6], the higher the dimensionality, the harder the problem. This is due to the search space becomes bigger and it is more difficult to find the solution to the problem. Some of the data sets used in this research are composed of around twenty attributes, which increases the complexity of the search.

Table 5.1: Financial indicators used in the experiment

Indicator name	Short period (Days)	Long period (Days)
Price moving average	12	50
Price Trading breaking rule	5	50
Filter rule	5	63
Price volatility	12	50
Volume moving average	10	60
Momentum	10	60
Momentum 10 days moving average	10	–
Momentum 60 days moving average	60	–
Generalized Momentum indicator	10	60
FOOTSIE moving average	12	50
LIBOR: 3 months moving average	12	50

In order to compare our approach to EDDIE-Arb [114], we used the data set called *Arbitrage*. According to the authors, the data were obtained from LIFFE and the independent variables used in this data set reflect the fundamental relationship between the spot, options and futures, none of which are technical trading indicators. The independent variables used in the Arbitrage data set are listed in Table 5.2. Given that our objective was to compare our approaches with EDDIE-Arb we used the same data set with the same independent variables.

5.4 Individual representation

According to Koza [69], the set of terminals and primitive functions have to be able to represent the solution of the problem (sufficiency property, section 3.3.1). The Discriminator Grammar (DG) in Table 5.2 is a BNF grammar (see section 2.5) that produces decision tree structures that classify or not a single class (binary classification). DG is proposed in this work because it helps to simplify the delimitation of rules, which

Table 5.2: Explanatory variables used in the data set *Arbitrage* (see [114])

No	Name	Description
1	Moneyness	Strike Price/Underlying Index Level
2	Basis % (x10000)	Futures price minus spot index level, divided by futures price, multiplied by 10,000
3	Und (x10)	Spot index level divided by futures price, multiplied by 10
4	Interest Ask %	The LIBOR ask rate for the maturity closest to the maturity of futures contract, multiplied by 100
5	Futures (T-t)	The nave trigger, profit after transaction costs, divided by futures price, multiplied by 1,000,000
6	C-P % (x100)	The difference between the call and the put prices, divided by futures price
7	Profit after TC (x1,000,000)	The nave trigger, profit after transaction costs, divided by futures price, multiplied by 1,000,000

is a task used in all the approaches proposed in this thesis. The objective of DG is to capture patterns that describe a specific class. Notice that a DG decision tree is true when at least one rule is satisfied, this implies that the decision tree classifies the instance as positive and it is not necessary to execute the remaining parts of the tree.

To explain the execution of DG decision trees, let LHS be the left hand side node of its parent and RHS be the right hand side node of its parent. As it is mentioned in the definition 2, section 5.5, a rule is composed by a set of conditions associated by conjunctions. It means that the operator "AND" indicates that the LHS and RHS conditions are part of the same rule. In contrast, the operator "OR" indicates that the tree has been splitting in different rules. In other words, the subtrees that are lying down the LHS and RHS are creating different rules.

Figure 5.3 illustrates a decision tree that was created using DG. When there is

Figure 5.2: Discriminator Grammar

G	→	<Root>
<Root>	→	"If-then-else",<Conjunction> | <Condition>,"Class","No Class"
<Condition>	→	<Operation>, <Variable>, <Threshold> | <Variable>
<Conjunction>	→	"AND"|"OR",<Conjunction>|<Conditional>, <Conjunction>|<Conditional>
<Operation>	→	"<" | ">"
<Variable>	→	var_1 | var_2 | ... var_n
<Threshold>	→	Real number

The DG grammar has three main functions:

1. Classify just a single class, this allows us to capture useful patterns that distinguish one class from another (binary classification).

2. Establish the format or structure to create the initial population of decision trees in the evolutionary process.

3. Maintain the validity of the decision trees' structure. This is achieved by instructing the genetic operators (crossover and mutation) to attend the DG guidelines, as a consequence the validity of the decision trees is maintained.

As can be seen, DG creates decision rules that hold comparisons of two variables and comparisons of a variable and a threshold. A similar structure was used by [11], [12], [13], [29] to evolve decision rules that classify or not a single class.

maintaining the closure property

This subsection explains how the DG grammar helps to maintain the GP closure property (see section 3.3.1). Figure 5.4 shows an example of how the genetic operator

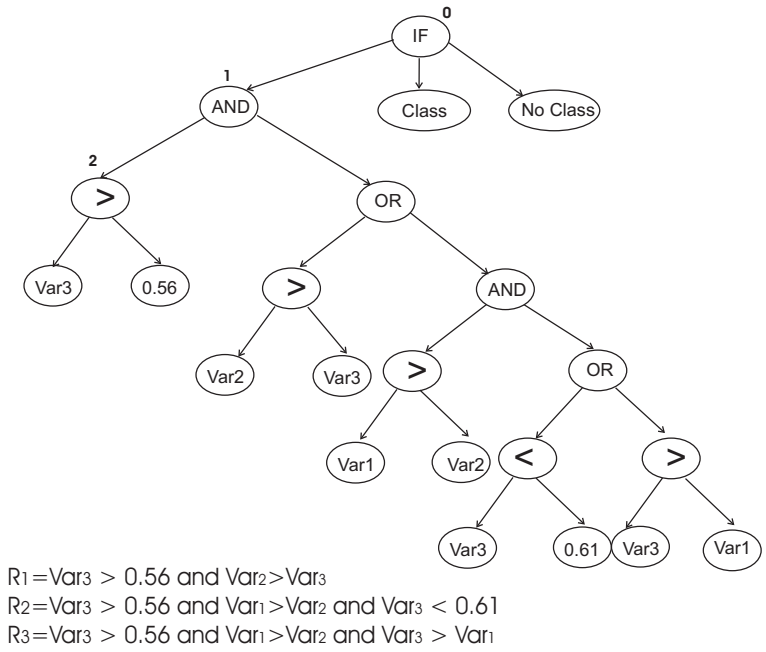


Figure 5.3: An decision tree created by using the Discriminator Grammar

mutation uses the grammar to maintain the strong type GP, satisfying the closure property. The structures are constrained by the grammar producing just valid individuals according to the specifications in the DG. As can be observed in Figure 5.4, the node to mutate is *node* 5. Thus, the parent of *node* 5 is *node* 1 and the syntax of that node is determined by <Conjunction> in DG, analysing the syntax of the second child of <Conjunction>, it can be a <Conjunction> or a <Conditional>. Thus, *node* 5, can be replaced by a subtrees with syntax <Conjunction> or <Conditional>.

In the case of the crossover a cross-point is selected randomly in each parent, if these are compatible (i.e. these can be swapped producing valid trees according to DG) the crossover is performed. Otherwise new cross-points are selected until these are compatible.

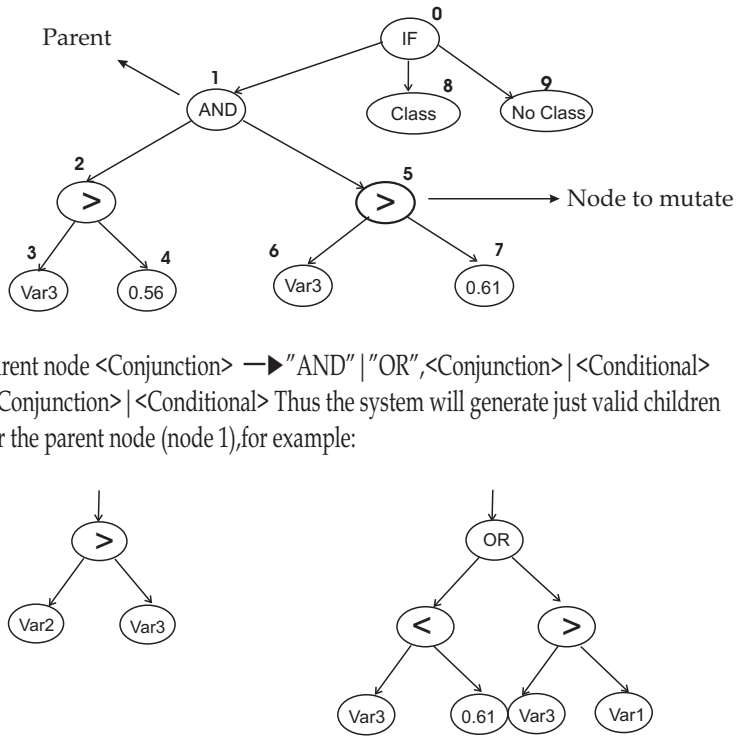


Figure 5.4: Generating valid subtrees to mutate the node 5 in the decision tree

5.5 Rule extraction

Rule extraction is a process that divides decision trees into rules; this allows us to analyse the performance of each component. Rule extraction is used in RM, EDR and SM (chapters 6,7 and 8 respectively). In each method the rule extraction is used for different purposes. RM (chapter 6) extracts rules from a set of decision trees (candidate solutions) in order to form a collection of rules. EDR (Chapter 7) divides decision trees into rules in order to evolve a population of rules. On the other hand, SM (Chapter 8) defines the rules to perform a selective pruning of the decision tree, with the purpose of improving the precision and accuracy of the prediction.

Objective

At this point the following question arises, what is the advantage of converting a decision tree to a set of rules? Researchers of other machine learning techniques, such as, Quinlan [98] (classifier C.45) and Breiman et al [15], have rewritten the decision trees produced by their classifiers in order to obtain a set of rules. Quinlan pointed out two main arguments: firstly, the collection of rules is more comprehensible than decision trees, especially when these are large. Secondly, the structure of the tree could create sub-concepts that can be fragmented and replicated in the same tree. In this work we have identified two main reasons for dividing a decision tree into its rules:

1. Identify the patterns in every decision tree. The approaches proposed in this thesis are focused on patterns rather than models, for that reason the rules that represent valuable patterns need to be identified. As was mentioned in section 2.2, a decision tree represents a *model* of the data set, while a rule represents a *pattern*, it means a local structure which meets the requirements of few cases in the search space.

2. Rules are easier to understand than decision trees, this characteristic allows the user to appreciate the real variables and conditions that are involved in the decision rule.

Procedure

In the context of this work, let us define a *condition* as an association of nodes with syntax <Condition> in DG. For illustrative purposes, the nodes in Figure 5.3 have

been numbered. When we refer to condition c_α, it is the condition formed by nodes α, $\alpha + 1$ and $\alpha + 2$. For example: $c_6 = \{var_2 > var_3\}$ in Figure 5.3.

Definition 2. *A rule $R_i \in T$, is a set of conditions that are associated by conjunctions, "AND". R_i represents a minimal set of conditions that satisfies the tree* T, *$\forall R_i \in T$.*

Rule extraction is a process that analyses a decision tree to delimit its rules by identifying the minimal sets of conditions that satisfy the tree T. As an instance, the minimal sets of conditions that satisfy the tree in Figure 5.3 are R_1=$\{c_2, c_6\}$, R_2=$\{c_2, c_{10}, c_{14}\}$ and R_3=$\{c_2, c_{10}, c_{17}\}$, where R_1,R_2 and R_3 are rules and c_i represents a condition, whose operator is in node i. Thus, the decision tree is satisfied when at least one of its rules has been satisfied. If the rules embedded in the decision tree in Figure 5.3 are represented as independent decision trees, the result is the set of decision trees in Figure 5.5. As can be observed, all the conditions are associated by conjunctions. This means that those decision trees are satisfied when all their conditions are satisfied. Any instance in the data set that satisfies at least one of the decision trees in Figure 5.5 is able to satisfy the tree in Figure 5.3. Expressing a rule as a decision tree, whose conditions are associated just by conjunctions, is a key factor for EDR (chapter 7) because it allows us to create new decision trees from decision rules.

Notice that, if the precision of the rule R_k is bigger than 0, then R_k classifies at least one positive case. This constraint is very important for RM, EDR and SM because it discards rules which are composed of contradictory or unresolved conditions, (e.g. $R_1 = \{var_1 > var_1\}$ or $R_2 = \{var_1 > var2$ and $var1 < var2\}$). This constraint guarantees that rules will be composed of combinations of conditions that are feasible

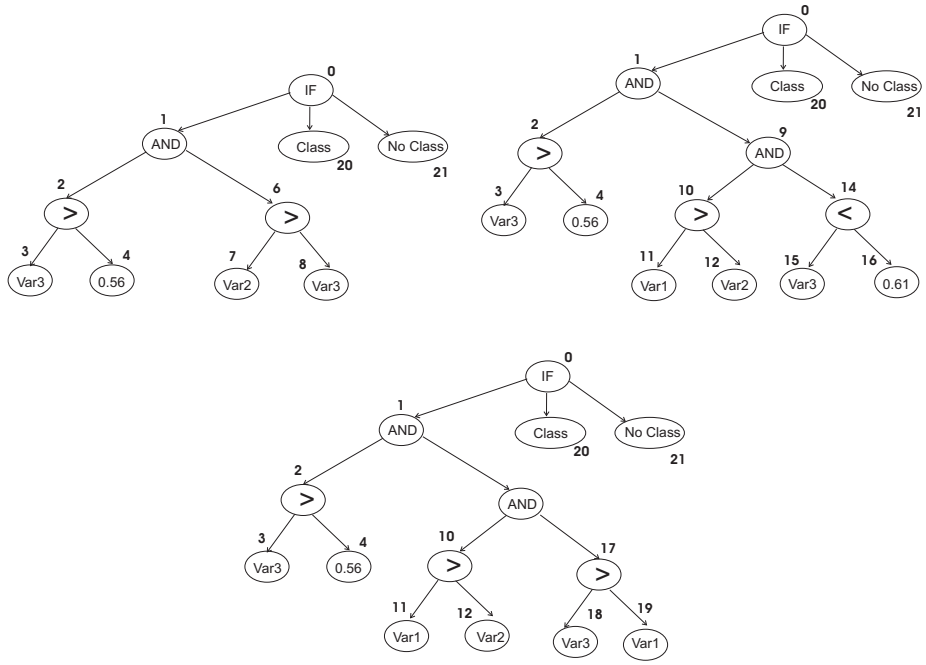


Figure 5.5: Rules from tree in Figure 5.3. Notice thta the node numbers correspond to the original ones in the decision tree in Figure 5.3

to satisfy. The pseudo-code of this procedure is described in the algorithm *ExtractRules* (see appendix)

5.6 Rule Simplification

The main aim of the rule simplification process is to remove the unused code that is produced by the GP [103]. The simplification of rules is used in RM and EDR methods, described in chapters 6 and 7 respectively.

Before to provide a more detailed explanation about the objective of rule simplification, let us define the following terms.

Definition 3. *Redundant condition: A condition c_i in a rule R is redundant if there exist at least another condition $c_k \in R$ such as c_k implies c_i*

Definition 4. *Vacuous condition: A condition c_i is vacuous in a rule R in relations to a data set D if under the conditions in R, c_i does not affect the decision of the rule, such as the $Performance(R) = Performance(R - c_i)$*

Redundant conditions are those which are repeated or report the same event e.g. $R_1 = \{Var_1 > 0.5 \text{ and } Var_1 > 0.7\}$ the first condition is redundant. Given that decision trees in the GP process are initially created at random, thus, there is the risk that trees hold repeated conditions or conditions that does not affect the real performance of the rule. This problem gets worse through generations because of the accumulation of introns [2],[103]

Vacuous conditions do not affect the decision of the rule, for example, in financial markets the trading volume is always bigger or equal to 0. Thus, a condition such as *Trading Volume*>0 is always valid without exception, so this condition does not affect the decision of the rule. Some conditions do not affect the decision of a tree or rule in combinations with other conditions and in a specific data set. However, these may affect the decision in other data sets. For that reason, it is important to remove that code.

Objective

The aim of rule simplification is to remove the extra-code that is not affecting the performance of the rule. The main objective of removing the redundant conditions is to simplify the representation of the rule, whose advantages are:

1. to reduce the computational effort by reducing the number of evaluations

2. to make the rule more comprehensible for the user

On the other hand, the advantages provided by removing the vacuous conditions are the following:

1. to disclose the real variables and conditions of the patterns represented by the rule.

    - it allows the user to understand the real conditions that are involved in the decision.

    - it allows to identify the duplication of patterns in a collection of rules, which is an important element of EDR and RM because it assures to have different rules in terms of genotype, increasing the variety of the solutions. In other words, this helps to detect the real similarities and differences in a set of rules.

2. to remove conditions that are not affecting the performance of the rule in the training data set, this reduces the risk of including conditions whose behaviour could be unpredictable in future data sets.

3. to reduce the computational effort by reducing the number of evaluations

There are some approaches that have been used GP to generate a set of rules, for example: [11], [85], [12], [13], [29]. These approaches claim to generate comprehensible rules. However, the GP tends to grow and accumulate introns or extra-code [69], [2], [107], [8], [104]. We presume that a simplification process has to be performed in order

to recognize the real variables and conditions that are involved in each rule. Only Cao et al [19] included a simplification process in their approach to remove redundant code. However, that simplification does not remove the extra-code that is not affecting the decision. The reader interested in introns and the generation of rule sets using GP is referred to chapter 3.

Procedure

To simply decision rules, the first step is to remove the redundancy, and then remove the vacuous conditions. To remove redundant conditions, we have defined two types of conditions: *hard conditions* and *flexible conditions*.

Definition 5. *A hard condition is the equation that compares two* variables *(e.g.* $var_1 < var_2$*)*

Definition 6. *A flexible condition is the equation between a variable and a threshold (e.g.* $var_1 < 0.8$*)*

Definition 7. *Two conditions are said to be similar if these have the same variable (see DG). For example,* $var_1 < 3$ *and* $var_1 < 2$ *are similar conditions.*

Definition 8. *similar conditions is a term to define a group of conditions that have the same variable and operator(see DG). For example,* $var_1 < 3$ *and* $var_1 < 2$ *are similar conditions.*

Conditions have been divided, in hard and flexible, because the conditions that compare thresholds could be difficult to differentiate (e.g. $var_1 < 0.8912$ and $var_1 < 0.8910$). However, these can be easily simplified (e.g. $Var_1 < 0.8910$). The simplification of rules can be summarized in the following steps:

Let $R_k = \{c_i\}$ be the set of conditions in the rule R_k.

- If $c_1, c_2 \in R_k$ are hard conditions and $c_1 = c_2$ then $R_k = R_k - c_2$
- If $c_1, c_2 \in R_k$ are flexible conditions and c_1 and c_2 are similar conditions then c_1 and c_2 are simplified using the *simplification table* (see Table 5.3).
- If $c_i \in R_k$ and $Performance(R_k) = Performance(R_k - c_i)$ then $R_k = R_k - c_i$

As can be noticed, the first and second steps remove the redundant conditions, while the third step removes the vacuous conditions.

Simplification of flexible conditions

The last paragraph described the general explanation for simplifying rules. In this section the procedure to simplify a set of flexible conditions using the *simplification table* is explained and an example showing this mechanism is given.

Let R_k be a rule whose conditions are related by conjunctions and meet the syntax <Condition> in DG. Thus, the value of every variable var_j can be described by two thresholds *inferior limit* T_i and *superior Limit* T_s, where $T_i \leq Ts$. Those thresholds can be real values or can be $\emptyset$ value. It means that the value of variable var_j is not restricted to that frontier. The following paragraph describes the procedure to simplify a set of similar conditions $c_1, c_2 \ldots, c_n$. It is important to bear in mind that similar conditions are flexible conditions and these compare the same variable var_j.

1.- Initialize the thresholds of the variable var_j, thus $T_i = T_s = \emptyset$. It means that there is not any previous condition that restricts the value of var_j.

2.- For each new condition c_a find the simplification instruction in Table 5.3 by matching the following criteria:

2.1 Match T_i and T_s in column *Present Thresholds* in Table 5.5

2.2 Match the syntax of condition c_a in the column *New condition* in Table 5.3

2.3 The new threshold T_n has to satisfy the equation in column *Previous condition* in Table 5.3

3.- Once the simplification instruction has been found, this has to be applied to T_i and T_s in order to update the *present thresholds*.

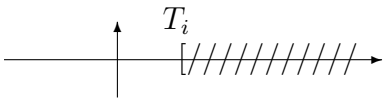
Simplification table

This paragraph explains the elements of the simplification table (see table 5.3). Each row in the table describes conditions that restrict the value of the variable *var* by means of the thresholds T_i, T_s and T_n (inferior, superior and new threshold, respectively). It is important to bear in mind that the set of conditions must be feasible to satisfy and T_i must be less or equal to T_s.

Given the mentioned constraints, the valid cases for T_i and T_s are illustrated in figure 5.6. Each of those cases is combined with conditions that describe the relation between T_i,T_s and T_n. However, not all that cases are presented in the simplification table because some of them do not meet all the constrains. Table 5.4 shows the invalid cases. Cases 1 and 2 are invalid because the conditions can not be satisfied, it means that these conditions does not classify any positive case. It is fear to say that using GP these combinations of conditions may appear. However, the methods proposed in this dissertation use the simplification procedure just when the set of conditions has classified at least one positive case, it means that these can be satisfied. On the other

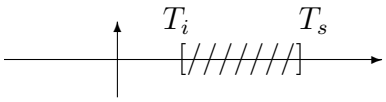
hand, cases 5 and 6 are invalid because the conditions are satisfied just when $T_s < T_i$.

If $T_i, T_s = \emptyset$



If $T_i \neq \emptyset$ and $T_s = \emptyset$

If $T_i = \emptyset$ and $T_s \neq \emptyset$



If $T_i, T_s \neq \emptyset$ and $T_i \leq T_s$

Figure 5.6: Valid cases for the thresholds T_i and T_s (inferior limit and superior limit, respectively) where $T_i \leq Ts$.

To illustrate the function of Table 5.3, let us explain the following example: The objective is to simplify the following set of similar conditions: $R_1 = \{c_1, c_2, c_3, c_4, c_5\}$ where

$$
\begin{aligned}
c_1 &= \{var_j > 0.1\} \\
c_2 &= \{var_j < 6.0\} \quad \textit{Simplification} \\
c_3 &= \{var_j < 4.0\} \qquad \Longrightarrow \qquad 0.1 < var_j < 2 \\
c_4 &= \{var_j > 0.0\} \\
c_5 &= \{var_j < 2.0\}
\end{aligned}
$$

Table 5.3: Simplification table, T_i, T_s and T_n are thresholds (new, inferior and superior respectively)

No. Instruction	Present Thresholds		Conditions			Simplification Instruction	
	Inferior limit	Superior limit	New Inf lim	Previous conditions: Sup lim	Previous conditions: condition	Inferior limit	Superior limit
1	$\emptyset$	$\emptyset$	$var < T_n$			$\emptyset$	T_n
2	$\emptyset$	$\emptyset$	$T_n < var$			T_n	$\emptyset$
3	$\emptyset$	T_s	$var < T_n$		$T_s \leq T_n$	$\emptyset$	T_s
4	$\emptyset$	T_s	$var < T_n$		$T_n < T_s$	$\emptyset$	T_n
5	$\emptyset$	T_s	$T_n < var$		$T_n < T_s$	T_n	T_s
6	T_i	$\emptyset$	$var < T_n$	$T_i < T_n$		T_i	T_n
7	T_i	$\emptyset$	$T_n < var$	$T_i \leq T_n$		T_n	$\emptyset$
8	T_i	$\emptyset$	$T_n < var$	$T_n < T_i$		T_i	$\emptyset$
9	T_i	T_s	$var < T_n$	$T_i \leq T_n$	$T_n < T_s$	T_i	T_n
10	T_i	T_s	$var < T_n$	$T_i \leq T_n$	$T_s < T_n$	T_i	T_s
11	T_i	T_s	$T_n < var$	$T_i < T_n$	$T_s \leq T_n$	T_i	T_s
12	T_i	T_s	$T_n < var$	$T_i \leq T_n$	$T_n < T_s$	T_n	T_s

Notice that, the number of the conditions has no relation to the position of the node in the decision tree. Table 5.5 presents the steps to simplify the set of conditions in R_1, let us explain every row of the table in detail.

Row 1 First, initialize the *inferior limit* T_i and the *superior limit* T_s of var_j using $\emptyset$.

Next, look for the simplification instruction in Table 5.3 following the next steps:

- Since T_i, $T_s = \emptyset$, we look for these values in the *present thresholds* column, in Table 5.3. As can be seen, the instructions that meet this requirement are instructions 1 and 2.
- The new condition to be introduced is c_1=$\{var_j > 0.1\}$. It means that c_1 has the syntax $var > T_n$. We look for the syntax in the *New Condition* column (Table 5.3). The simplification instruction that satisfies this requirement is

Table 5.4: List of the invalid cases which are not included in the Simplification table

No. Instruction	Present Thresholds		Conditions			Simplification Instruction	
	Inferior limit	Superior limit	New Inf lim	Previous conditions Sup lim	condition	Inferior limit	Superior limit
1	$\emptyset$	T_s	$T_n < var$		$T_s \leq T_n$	–	–
2	T_i	$\emptyset$	$var < T_n$	$T_n \leq T_i$		–	–
3	T_i	T_s	$var < T_n$	$T_n \leq T_i$	$T_n \leq T_s$	–	–
4	T_i	T_s	$T_n < var$	$T_n \leq T_i$	$T_s \leq T_n$	–	–
5	T_i	T_s	$var < T_n$	$T_n \leq T_i$	$T_s \leq T_n$	–	–
6	T_i	T_s	$T_n < var$	$T_n \leq T_i$	$T_n \leq T_s$	–	–

the instruction number 2.

- Applying the instruction 2, the new threshold T_n has to replace the *inferior limit* thus $T_i = T_n = 0.1$

Row 2 The *present thresholds* are $T_i = 0.1$ and $T_S = \emptyset$. The simplification instructions that satisfy this specification are 7,8,9 and 10. But the following specifications have to be achieved too:

- The *new condition* is $c_2 = \{var_j < 6\}$, it means that c_2 has syntax $var_j < T_n$. Thus, the simplification instructions 7 and 8 meet the new requirement (see column *New condition* in Table 5.3).

- As $T_i < T_n$ (0.1<6), we look in the *previous conditions* column (Table 5.3) this formula. Thus, the simplification instruction that meet all the requirements is instruction 8.

- Applying the simplification rule 8 the thresholds are modified as follows: $T_i = T_i = 0.1$ and $T_s = T_n = 6$.

- The simplification of the remaining rows follows the same procedure. The simplification of the example is $0.1 < var_j < 2$

The pseudo-code that describes the simplification of flexible conditions is described in the algorithm: *IdentifyRange()* in the appendix

5.7 New rule detection

Objective

This process helps to maintain the variety of the patterns, it is used in RM and EDR (Chapters 6 and 7 respectively). The aim of this procedure is to identify new patterns to form a set of rules. Once a rule R_k has been simplified, we have to determine the novelty of R_k, comparing this against the rules in the collection. To compare rules effectively, these have been divided into *hard* and *flexible*

Definition 9. *R_i is a hard rule, if it is composed exclusively of hard conditions*

Definition 10. *R_i is a flexible rule if it has at least one flexible condition*

Table 5.5: Steps to simplify a set of flexible conditions: using the Table 5.3

No. Step	Inferior limit T_i	Superior limit T_s	New Condition	Simplification instruction Table 5.3	Final Condition
0	$\emptyset$	$\emptyset$	$var_k > .1$	2	$0.1 < var_k$
1	0.1	$\emptyset$	$var_k < 6$	8	$0.1 < var_3 < 6$
3	0.1	6	$var_k < 4$	12	$0.1 < var_k < 4$
4	0.1	4	$var_k > 0$	14	$0.1 < var_k < 4$
5	0.1	4	$var_k < 2$	12	$0.1 < var_k < 2$

Definition 11. *R_k and R_i are similar rules if these have the same hard conditions and similar flexible conditions.*

The comparison of hard rules is straightforward, but the comparison of flexible rules is more complex, because rules contain thresholds, for instance:

R_1= $(Var_1 > Var_2, Var_3 > 0.30)$ and
R_2= $(Var_1 > Var_2, Var_3 > 0.35)$ are similar rules

At this point an important question arises, how to choose between two similar rules?, we propose to pick the rule with the best performance, it helps to tune the thresholds in flexible conditions by taking advantage of the knowledge acquired by the evolutionary process.

Let	Rep	be the repository of rules
	μ	be the maximum allowed number of rules in Rep
	R_k	be a rule such as $R_k \notin Rep$
	$Fitness$	be the fitness function
	R_w	be a rule such as $R_w \in Rep$ and $Fitness(R_w) \leq Fitness(R_i)\ \forall R_i \in Rep$
	$Size$	be the function that measure the number of rules in Rep

1. R_k is a hard rule and $\exists\ R_i \in Rep$ such as $R_i = R_k$ thus the rule is discarded in order not to duplicate R_i

2. R_k is a hard rule and $\nexists\ R_i \in Rep$ such as $R_i = R_k$ and $Size(Rep) < \mu$ thus R_k has to be added to Rep, it means $Rep = Rep \cup R_k$

3. R_k is a hard rule and $\nexists\ R_i \in Rep$ such as $R_i = R_k$ and $Size(Rep) = \mu$ and $Fitness(R_k) > Fitness(R_w)$ thus R_k has to be added to Rep

4. R_k is a hard rule and $\nexists\ R_i \in Rep$ such as $R_i = R_k$ and $Size(Rep) = \mu$ and $Fitness(R_k) \leq Fitness(R_w)$ thus R_k is discarded

5. R_k is a flexible rule and $\nexists\ R_i \in Rep$ such as R_k and R_i are similar rules and $Size(Rep) < \mu$ thus R_k is added to Rep

6. R_k is a flexible rule and $\nexists\ R_i \in Rep$ such as R_k and R_i are similar rules and $Size(Rep) = \mu$ thus R_k is discarded

7. R_k is a flexible rule and $\exists\ R_i \in Rep$ such as R_k and R_i are similar rules and $Fitness(R_k) \leq Fitness(R_i)$ then R_k is discarded because there is a similar rule R_i whose conditions hold thresholds that perform better than the thresholds in R_k.

8. R_k is a flexible rule and $\exists\ R_i \in Rep$ such as R_k and R_i are similar rules and $Fitness(R_k) > Fitness(R_i)$ then R_i is replaced by R_k because the latest performs better.

The instructions to deal with the mentioned cases can be summarized as follows:

- If R_k is a hard rule and $\nexists\ R_i \in Rep$, such as $R_i = R_k$ then $Rep = Rep \cup R_k$, but if $Size(Rep) > \mu$ then $Rep = (Rep - R_w) \cup R_k$

- If R_k is a flexible rule and $\exists\ R_i \in Rep$ such as R_k and R_i are similar rules and $Fitness(R_k) > Fitness(R_i)$ then $Rep = (Rep - R_i) \cup R_k$

- If R_k is a flexible rule and $\nexists\ R_i \in Rep$ such as R_k and R_i are similar rules then $Rep = Rep \cup R_k$, but if $Size(Rep) > \mu$ then $Rep = (Rep - R_w) \cup R_k$

Notice that, the number of rules is limited by the parameter μ and when the size of *Rep* is bigger than μ the worse rule in *Rep* has to be deleted. On the other hand, the replacement of rules is an important part of this process because this is applied to flexible rules (those rules that hold conditions with continuous thresholds). Thus, every time a flexible rule is replaced by a better similar rule, the thresholds are being approximated to the optimal values.

5.8 Summary of the chapter

The chapter focused on describing and justifying common procedures that are used to build up the contributions presented in the next three chapters of this thesis. Basically, this chapter is composed of two parts. The first is a description of the data sets used in this research, whose aim is to describe the process to assign categories to each record in the data sets. The second is a description of the basic procedures used to build the approaches proposed in this thesis.

Chapter 6

Repository Method

This chapter presents a novel approach called Repository Method (RM) [46], [49], [43], [45]. This method is based on machine learning and supervised learning (section 2.1). Machine learning classifiers extend the past experiences into the future. However, the class imbalance in the data sets poses a serious challenge to machine learning techniques [63], [119], [5], [81]. Because the prediction of the majority class is favoured, due to it has a high chance of being correct.

Objective

The main objectives of RM are:

1. to classify the minority class in imbalanced environments
2. to produce a range of solutions to suit different user's preferences
3. to generate a set of comprehensive decision rules that can be understood by the user and present the real variables and conditions in the decision

The remaining of this chapter is organised as follows: Section 6.1 presents the introduction and motivation of this work; Section 6.2 describes the RM procedure, while Section 6.3 describes the experiments to test our method. Finally, Section 6.4 summaries the conclusions.

6.1 Introduction and motivation

RM is inspired by a previous work called EDDIE [112], [77], [116], [114]. EDDIE is a financial forecasting tool that trains a GP using a set of examples. Every instance in the data set is composed of a set of *attributes* (financial indicators). The signal is calculated looking ahead to a future horizon of n units of time, trying to detect an increase/decrease of at least $r\%$ (see section 5.2). However, when the value of r is very high, which implies an important movement in the stock price, the number of elements in the minority class is extremely small and it becomes very difficult to detect those cases.

This work is motivated by the interest of finding high movements in stock prices, which requires detecting the minority class in imbalanced environments. Evolutionary Algorithms have the ability to create multiple solutions to a single problem. To take advantage of this feature, we propose to mine the knowledge acquired by an evolutionary process. The aim is to extract and collect different rules (patterns) that classify the cases in the minority class in different ways, increasing the probability of identifying similar cases in the future.

Previous works

In our understanding, there is no similar application that mines the knowledge acquired by a evolutionary process as RM does. The closest application that we found was PADO by Teller [108]. He used several individuals (programs) from the population in order to classify a set of examples. Then a voting system was used in order to achieve

an agreement between individuals. In contrast, our application is focused on patterns rather than models (see section 2.2). RM collects patterns from different individuals in the population and from different generations of the evolutionary process. Those patterns classify a single class (positive), the absence of patterns indicates that the classification is negative, in other words RM creates a binary classifier. Furthermore, RM performs a simplification of conditions to provide a comprehensible set of rules that other applications based on GP would not able to offer because GP systems generate extra-code [2, 88, 104, 74] that constitute between 40% and 60% of the total code (see section 3.3.3). And unless a simplification of that code is performed it is not possible to offer a solution that shows the real variables and conditions in the decision.

There exist other evolutionary techniques to produce a set of decision rules (see section 3.4). The majority of them use GAs to evolve a population of rules (Michigan approach [58], [120],[17]) or a population of sets of decision rules (Pittsburgh approach [64]). Both approaches are developed in the frame of GAs whose representation (length strings) could limit the search of the optimal solution. Since the Michigan approach (see section 3.4) is concerned about covering all possible instances, this could be problematic when the number of variables or attributes is high. In contrast RM is focused on capturing the patterns of the minority class. On the other hand, the Pittsburgh approach (see section 3.4) evolves sets of rules, it limits the number of possible rules because the size of the individual increases every time a new rule is added

Due to the characteristics of RM, it has been used in the area of *Chance discovery*, which is the discipline that studies the detection of rare, but significant events that

may have an important impact [1],[89].

6.2 Repository Method Procedure

To detect the cases in the minority class, let us call them *positive cases*, we propose to compile several solutions to classify in different ways the positive examples. To generate multiple solutions, we use Genetic Programming (GP), which is an evolutionary technique that is able to produce multiple solutions to a single problem (see section 3.3). The aim of RM is to collect a big collection of rules that holds different patterns (see section 2.2) of the positive cases. In the context of this work, patterns are represented by rules. Let us define a rule $R_i \in T$ as a minimal set of conditions whose intersection satisfies the decision tree T (see sections 2.2, 5.5). We believe that the patterns that hold few positive cases could be eliminated in the evolutionary process. That is, consider the rules R_1 and R_2 in the following example:

Let	R_1, R_2	be two different rules
	E_1	be the set of positive examples correctly classified by R_1
	E_2	be the set of positive examples correctly classified by R_2

Consider $E_1 \subset E_2$ and the $Precision(R_1) \leq Precision(R_2)$. It is more likely that rule R_2 survives during the evolutionary process. Because R_2 is better, however, it is not possible to know if R_1 and R_2 would classify the examples using different criteria that could be useful for detecting positive cases in future data sets. If R_1 and R_2 are different rules maybe in another data set R_1 is able to classify examples that R_2 cannot, for such reason we consider useful to keep R_1. If we are evolving decision trees and the

tree T holds rules R_1 and R_2, thus the part that holds R_1 can be removed from the tree without affecting (or affecting positively because $Precision(R_1) \leq Precision(R_2)$) the overall performance of the tree, an illustrative example is presented in section 7.3.5.

Repository Method overview

To generate many candidate solutions, we use a population or a set of populations created by a GP system. RM analyses the decision trees in order to select and collect patterns capable of identifying the positive cases. Those patterns or rules will be stored in a structure that is called *repository*. The selection is based on the *performance* and *novelty* of the solution. The performance is determined by the precision of the rule (phenotype), while the novelty is determined by the conditions and variables that compose every rule (genotype). Let us introduce an overview of the main steps of our approach:

1. *Creation of different solutions:* to create a large set of candidate solutions.

2. *Rule extraction:* to analyse every decision tree in order to identify the rules in the tree. Every rule R_i is evaluated using the training data set. If R_i achieves a predefined Precision Threshold (PT), then R_i will pass to the next step, otherwise R_i is discarded (see section 5.5).

3. *Rule simplification:* to simplify the decision rule R_i by removing the redundant and vacuous conditions (see section 5.6).

4. *New rule detection:* to compare the rule R_i to the rules stored in the repository.

The objectives are: a) to pick and store new rules and b) to replace an existing rule when the new rule R_i is similar and performs better (see section 5.7).

A general overview of RM has been presented. The next sections explain every procedure, but some detailed explanations are provided in Chapter 5.

6.2.1 Creation of different solutions

To generate multiple solutions, we use a GP system to evolve a population of decision trees. However, in the majority of the evolutionary algorithms, the normal procedure is to choose the best individual of the complete evolution as the optimal solution of the problem. In contrast, we believe that the remaining individuals in the population could contain useful patterns that are not necessarily present in the best individual of the evolution.

To generate a set of solutions we propose to save a complete population of decision trees generated by a GP system. Thus, a population of n individuals is evolved using a standard GP, in a specific generation the entire population is saved. Let P_k be the population in generation k. Since we believe that some useful rules could be lost during the evolutionary process, we propose to gather information from a set of populations: $P_{10}, P_{20} \ldots P_{100}$. Notice that, the fitness function of the GP in the experiments of this chapter is the geometric mean of the product of the precision and recall, which is a common metric to measure the performance of a classifier in imbalanced environments [71].

The decision trees in this work meet the syntax of Discriminator Grammar (DG)

(see Figure 5.2). This grammar helps to simplify the delimitation of rules, producing decision trees that classify or not a single class. A more detailed explanation of the representation of the decision trees is provided in section 5.4.

6.2.2 Rule extraction

The aim of this procedure is to analyse the decision trees in order to delimit their rules, by identifying and separating the patterns in every decision tree. Once a rule $R_k \in T$ has been delimited, it is evaluated using the training data set. If the precision of R_k achieves a predefined Precision Threshold (PT), then R_k is considered for the next step, otherwise the rule is discarded. PT is a parameter set by the user. Section 5.5 provides the complete explanation of the rule extraction process.

6.2.3 Rule Simplification

The objective of rule simplification is to remove redundant and vacuous conditions (see definitions 3 and 4 in section 5.6). Given that, decision trees that are generated by a GP process tend to grow and accumulate extra-code (often known as introns) [2],[88],[104],[74]. The complete explanation of this process is provided in section 5.6. Rule simplification is a relevant task for the following reasons:

1. Disclose the real variables and conditions of the patterns
2. Remove conditions that are not affecting the performance of the rule in the training data set, reducing the risk of unpredictable behaviour in future data sets.
3. Identify the duplication of rules in the repository

6.2.4 New rule detection

The aim of this procedure is to identify new patterns to form a repository of rules. The procedure is to add new rules to the repository of rules and replace old rules with similar rules that perform better. The complete explanation of this procedure is provided in section 5.7. The following procedure determines if the rule R_k is added or not to the rule repository *Rep*.

6.2.5 The repository of rules

After explaining the RM procedure, an important question arises. How to choose the best value of PT to create the repository of rules?. We propose to use several values for the PT in order to produce several repositories. The idea behind this procedure is to provide a range of classifications. Thus, the user can choose the best trade-off between misclassifications and false alarms according to its preferences. The results can be plotted in the ROC space (see section 2.4.3). Thus, when the PT is close to 1 the prediction will be conservative and when PT is close to 0 the prediction will be liberal.

6.3 Experimental Section

To test our approach a series of experiments was performed. The first experiment (section 6.3.1) was designed to test the performance of RM. Furthermore it demonstrated how to select the best trade-off between misclassification and false alarms using the ROC curve. The experiment in section 6.3.2 compares the performance of RM with

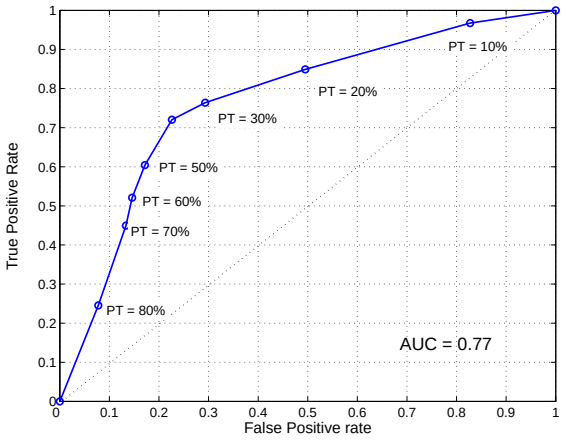
EDDIE-Arb. The experiment in section 6.3.3 compares RM with C5.0. The next experiment was designed to show the impact of the evolutionary process in the performance of RM (see section 6.3.4), this experiment shows the effect of 1) the accumulation of rules between generations and 2) the number of decision trees analysed by RM. Finally, an experiment to investigate the number of rules that each individual in the population provides to the repository of rules is explained in Section 6.3.5. A series of preliminary experiments were performed in order to select the parameters to execute the GP.

6.3.1 Experiment: Comparison of RM performance with a standard GP

Data set:	BARCLAYS
March,1998-January,2005	
Rate of return :	15%
Number of days :	10
Tendency movement :	Increase
Training data set :	
Training examples:	887
Positive examples:	39 (4.3%)
Testing data set	
Testing examples:	831
Positive examples:	24 (2.8%)
Number of runs :	20
AUC :	0.77

Populations

$P_{10}, P_{20}, \ldots, P_{100}$



GP Parameters	
Population size :	1,000
Crossover probability:	.80
Mutation probability :	.05
Hill-climbing prob :	.05
Selection :	tournament (2)
Fitness function :	$\sqrt{Precision \cdot Recall}$

Figure 6.1: RM parameters and ROC curve using Barclays data set

Objective

The aim of this experiment is to test the performance of RM and compare it with a standard GP system. In addition, it shows the use of the ROC curve to look for the best trade-off between misclassifications and false alarms costs.

Procedure

RM was tested using different values for the precision threshold PT ={10%,20%,...,80%}. In this experiment, RM gathered rules from the entire populations $P_{10}, P_{20}, \ldots, P_{100}$ to form a repository of rules. The recall, precision and accuracy of RM and the best

individual of the standard GP are presented in Table 6.1. The table displays averaged results from 20 runs of RM and the GP using the testing data set.

Observations

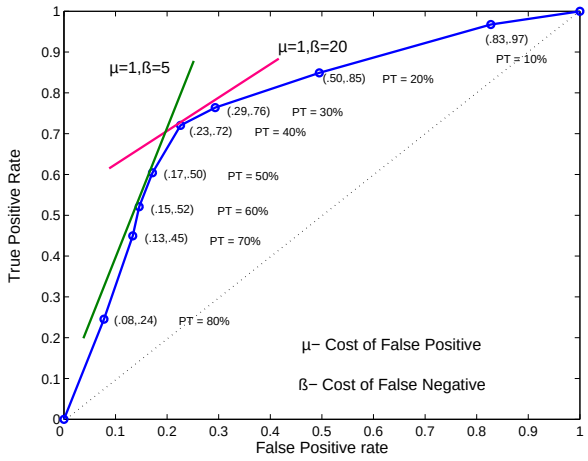
A standard GP produces a single prediction for every data set, in contrast, RM was designed to provide a range of classifications, which allows the investor to tune the prediction according to his/her risk guidelines. If the user's requirement is to detect as many positive cases as possible, the PT has to decrease to move to the liberal part in the ROC space. In contrast, if the user's preference is to reduce the risk, then the PT has to increase to move to the conservative part. Notice that, the recall declines when PT increases. Because the selection of rules becomes stricter and fewer rules are selected decreasing the number and variety of rules in the repository. Note that the results provided by RM have better recall than the recall of the best GP individual. The result of the standard GP is plotted in (0.10, 0.15) in the ROC space, which describes a conservative prediction. The RM predictions have been distributed along the ROC curve, this allows users to choose the most suitable choice for their requirements. According to the experimental results, it was possible to detect from 24% to 76% of the positive cases with an accuracy higher than 71% (see Table 6.1). The GP has better accuracy than the majority of the choices in RM because the GP tends to predict negative classification, which has a high chance of being correct. The experiment shows that RM is able to pick out rules that together classify more positive cases. In addition, most of the extra positive classifications were correctly predicted, which is reflected in both recall and precision. Since more errors were made (not as

Table 6.1: RM Results found when using Barclays.

PT	*Recall*	*Precision*	*Accuracy*
0.8	0.24	0.085	0.90
0.7	0.45	0.091	0.85
0.6	0.52	0.095	0.84
0.5	0.60	0.094	0.82
0.4	0.72	0.086	0.77
0.3	0.76	0.071	0.71
0.2	0.85	0.048	0.51
0.1	0.97	0.033	0.20
GP	0.14	0.04	0.87

much, in proportion, as the correct classifications), the overall accuracy has decreased. Given that our goal is to improve recall and precision, it is an acceptable price to pay.

Table 6.2: ROC curve, the tangent lines indicate the best trade off between misclassifications and false alarms costs

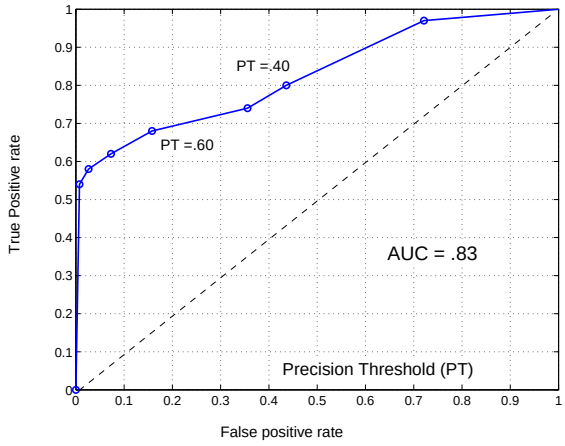


False Positive Cost	False Negative Cost	Slope Tangent
1	1	33.6
1	5	6.7
1	10	3.3
1	15	2.2
1	20	1.6
1	25	1.3
1	30	1.1

The AUC (see section 2.4.3) obtained by RM is 0.77, according to Giedrius [118], it describes a moderately predictive classifier. Figure 6.2 presents the ROC curve plotted by RM and the table with the slope of the tangent lines, which represent the best trade-off between misclassification and false alarms when the price of misclassifying is β =1,5,10,...,30 and the false alarm cost is μ=1.

6.3.2 Experiment: Comparison of RM and EDDIE-Arb

Data set:	Arbitrage
Tendency movement :	Increase
Training data set :	
Training examples:	1,006
Positive examples:	242 (24%)
Testing data set	
Testing examples:	635
Positive examples:	159 (25%)
Number of runs :	20
AUC :	83.5
RM Populations	
$P_{10}, P_{20}, \ldots, P_{100}$	



GP Parameters	
Population size:	1,000
Crossover probability:	.80
Mutation probability :	.06
Hill-climbing prob :	.05
Selection :	tournament (2)
Fitness function :	$\sqrt{Precision \cdot Recall}$

Figure 6.2: RM parameters and ROC curve using Arbitrage data set

Objective

The objective of this experiment is to compare the performance of RM with the performance of EDDIE-ARB [114]. For more information about the data set *Arbitrage* see Section 5.3.

Procedure

As in the previous experiment our approach was tested using different values for the precision threshold (PT ={10%,20%,...,100%}). The repository was formed gathering and accumulating rules from the populations $P_{10}, P_{20}, \ldots, P_{100}$.

Table 6.3: RM Results using Arbitrage data set

PT	*Recall*	*Precision*	*Accuracy*
1	0.00	0.00	0.75
0.90	0.54	0.96	0.88
0.80	0.58	0.88	0.88
0.70	0.62	0.74	0.85
0.60	0.68	0.59	0.80
0.50	0.74	0.41	0.67
0.40	0.80	0.38	0.62
0.30	0.97	0.31	0.45
0.20	1.00	0.25	0.25
0.10	1.00	0.25	0.25
EDIIE-Arb	0.42	1.00	0.85

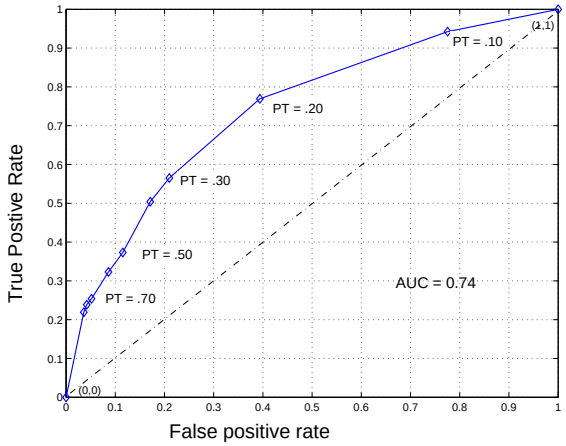
Observations

The recall, precision and accuracy of RM and EDDIE-Arb are presented in Table 6.3. As can be observed, EDDIE-Arb detected 42% of the positive cases with precision = 1. The result of EDIIE-Arb is plotted in the point (0, 0.42) in the ROC space. The result obtained by EDIIE-Arb is slightly better than RM results since this is plotted over the ROC curve generated by RM. However, RM offers other choices that detect more positive cases, for example: the recall and accuracy of RM when threshold =70%,80% and 90% exceed EDIIE-Arb recall and accuracy. However, the precision of EDDIE-Arb is better than RM precision. On the other hand, the solutions produced by RM are distributed along the ROC curve, which allows the user to choose the most suitable prediction according to their requirements. So it is possible to detect from 54% to 68% of the positive cases with an accuracy greater than or equal to 80% when the PT = .90,.80,.70,.60 (see Table 6.3).

6.3.3 Experiment: Comparison of RM and C5.0

Data set:	Barclays (400)
Rate of return :	15%
Number of days :	10
Tendency movement :	Increase
Training data set :	
Training examples:	400
Positive examples:	15 (3.7%)
Testing data set	
Testing examples:	400
Positive examples:	13 (3.2%)
Number of runs :	20
AUC :	0.74

RM Populations
$P_{10}, P_{20}, \ldots, P_{100}$



GP Parameters	
Population size:	1,000
Crossover probability:	.80
Mutation probability :	.05
Hill-climbing prob :	.08
Selection :	tournament (2)
Fitness function :	$\sqrt{Precision \cdot Recall}$

Figure 6.3: Comparison of RM and C5.0 parameters and results

Objective

The aim of this experiment is to compare the performance of RM with classifier C.5 developed by Quinlan [98]. C.5 is a machine learning system that extracts informative patterns from data; the classifier generated by C.5 is expressed as a decision tree or a sets of if-then rules. In the ML literature C4.5 (the previous version of C5.0) is considered one of the most relevant classifiers [67], [83], [121], [20].

Table 6.4: RM Results using Barclays 400.

Threshold	*Recall*	*Precision*	*Accuracy*
1.00	0.00	–	0.97
0.90	0.22	0.17	0.94
0.80	0.24	0.16	0.93
0.70	0.25	0.14	0.93
0.60	0.32	0.11	0.89
0.50	0.37	0.10	0.87
0.40	0.50	0.09	0.82
0.30	0.57	0.08	0.78
0.20	0.77	0.06	0.61
0.10	0.94	0.04	0.25
C5	0.00	–	0.97

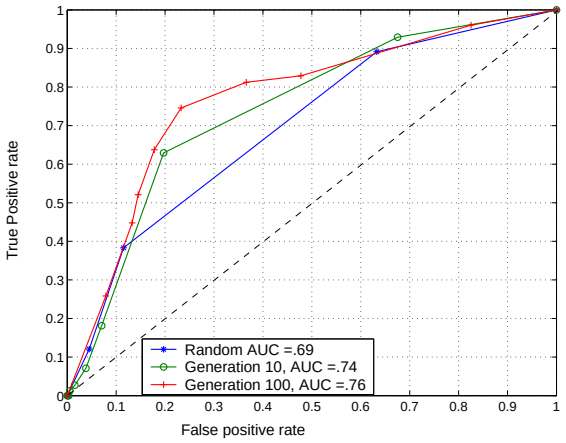
Procedure

To perform the experiment, we used the trial C.5 version. This demonstration version cannot process more than 400 training or testing cases. For that reason, the size of the training and testing data sets had to be adjusted to meet this constraint. Quinlan's algorithm was tested using ten-fold cross-validation and standard parameter settings.

Observations The recall, precision and accuracy obtained by RM and C.5 are presented in Table 6.4. As can be observed, the result obtained by C.5 was *true positive* =0, *false positive*=0, *false negatives* = 13, *true negative* = 387. This is plotted in the point (0,0) in the ROC space, which describes a conservative prediction. The classifier C.5 obtained an excellent accuracy (97%). However, it did not detect any positive cases. On the other hand, RM generated a range of solutions to satisfy different users' preferences (conservative or liberal), because the predictions are distributed in the ROC space. Thus, we can detect half of the positive examples with accuracy of 82%, using a PT=40%.

6.3.4 Experiment: Importance of the evolutionary process

Data set:	Barclays
Rate of return :	15%
Number of days :	10
Tendency movement :	Increase
Training data set :	
Training examples:	887
Positive examples:	39 (4.3%)
Testing data set	
Testing examples:	831
Positive examples:	24 (2.8%)
Number of runs :	20
Populations	
P_0	
P_{10}	
$P_{10}, P_{20} \ldots P_{100}$	



RM Parameters	
Population size:	1,000
Crossover probability:	.80
Mutation probability :	.05
Hill-climbing prob :	.05
Selection :	tournament (2)
Fitness function :	$\sqrt{Precision \cdot Recall}$

Figure 6.4: Experiment: Importance of the evolutionary process, parameters and results

Objective

The objective of this experiment is to show the importance of the evolutionary process as a generator of solutions. To achieve our goal, this section has been divided in two series of experiments.

Procedure (First part)

The performance of RM was measured using populations from different stages of the evolutionary process. Thus, RM was tested using the following populations:

1. A random population of decision trees P_0

Table 6.5: Barclays Recall (1), precision (2) and accuracy (3)

PT	Random decision trees (P_0)			Generation 10 (P_{10})			Generation 100 ($P_{10} \ldots P_{100}$)		
%	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)
100	0	–	.97	0	–	.97	0	–	.97
90	0	–	.96	0	–	.97	.26	.09	.90
80	0	–	.96	0	–	.97	.26	.09	.90
70	0	–	.96	0	–	.96	.45	.09	.86
60	0	–	.96	.01	.06	.96	.52	.10	.85
50	0	–	.96	.03	.05	.95	.64	.10	.83
40	0	–	.96	.07	.05	.93	.75	.09	.78
30	.12	.07	.93	.18	.07	.91	.81	.06	.65
20	.38	.09	.87	.63	.09	.81	.83	.05	.54
10	.89	.04	.38	.93	.04	.35	.96	.03	.20

2. A population that was evolved for 10 generations, let us call it P_{10}

3. The accumulation of ten populations from different generations $P_{10}, P_{20} \ldots P_{100}$.

Observations

The results obtained in the experiment are presented in Table 6.5. Figure 6.4 displays the ROC curves plotted by RM in the experiments described previously. Now, let us describe in detail the result in every experiment.

- **Random population** P_0 **-** Using a random population the AUC = .69, as can be observed from figure 6.4, the majority of the points are clustered in the conservative part of the ROC curve. It means that these did not classify any positive case. However, RM was able to generate an interesting choice for the investor, when PT=20% recall =38%, precision=9% and accuracy= 87%. Given that, the population was generated at random, the patterns captured by RM

were created by chance.

- **Population P_{10} -** Using a population evolved for ten generations the performance of RM improved, since the AUC increased from 0.69 to 0.74. In this experiment RM offers two valuable choices when PT=30% and PT=20%. However, one of the choices is on the conservative side and the other in the liberal part of the ROC curve as Table 6.5 shows.

- **Populations $P_{10}, P_{20} \ldots P_{100}$ -** Using the accumulation of ten populations from different stages of the evolutionary process, the AUC increased from 0.74 to 0.76, as was expected. However, the AUC increase is not as important as the fact that the predictions have been distributed along the ROC curve. This allows users to choose from a range of options the most suitable prediction according to their requirements. Thus, it is possible to detect from 26% to 75% of the positive cases with an accuracy greater or equal to 78% using the PT =.40,.50,. . . ,.90 (see Table 6.5).

When RM was tested in a set of decision trees that were randomly created the AUC drops to 0.69. When the evolutionary process advances, the quality of the predictions improves. It can be seen in: a) the AUC increase and b) the distribution of the predictions in the ROC space. These indicate that the evolutionary process really helps to create the rules for RM. As can be observed, the evolutionary process plays an important role in discovering patterns because it increases the AUC and distributes the predictions in the ROC space.

Procedure (Second part)

To provide a complete picture about how the evolutionary process helps to produce useful patterns, the previous experiment was repeated, but this time RM used the following sets of populations: $\{P_{10}\}$, $\{P_{10}, P_{20}\}$ ... $\{P_{10}, P_{20}, \ldots, P_{100}\}$

Observations

Table 6.6 presents the recall, precision and accuracy of the experiment. As can be seen, the best individual of the evolution is unstable because its recall and precision fluctuate from generation to generation. The following paragraphs analyse the results by indicators:

Recall - The recall obtained by RM increases consistently with the generations. Note that, when PT increases the recall declines, it is because the selection of rules becomes stricter and fewer rules are selected, this decreases the number and variety of the rules in the repository. Notice that, except for some cases in the earliest generations, RM shows better recall than the best GP individual. It might be due to, at the beginning of the evolutionary process, most of the trees are not too far from being random, it limits the production of useful rules. However, when the evolutionary process advances, it tends to generate more and better patterns, in consequence RM recall improves.

Precision - As can be seen in Table 6.6, the RM precision fluctuates slightly. However, it shows an upward trend when $PT \geq 40\%$ and a downward tendency when $PT \leq 20\%$. On the other hand, the GP precision oscillates significantly, but it does not exhibit any clear tendency. As can be observed, in the majority of cases the precision of the

standard GP is overcome by RM precision.

Accuracy - The improvement in recall and precision is paid for, in some cases, by decrease in accuracy. Thus, the GP has better accuracy than RM because it tends to predict negative classifications. Experiments show that RM picks out rules that together classify more positive cases. Most of the extra positive classifications were correctly made. This is reflected in both, increase in recall and precision. Since more errors were made (not as much, in proportion, as the correct classifications), the overall accuracy has decreased. If the goal is to improve the recall and precision, it is an acceptable price to pay. However, if the objective is to improve the accuracy, the user can choose a conservative prediction, which is produced using a high PT.

Table 6.6: GP and RM recall, precision and accuracy. RM was tested using different precision thresholds PT = 10%,20%, ..., 80%. In RM the generation x means that RM gathered rules from populations $P_{10}, P_{20} \ldots P_x$.

RECALL									
		Repository Method PT							
Gen	GP	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
10	1%	93%	65%	23%	7%	3%	1%	0%	0%
20	0%	94%	69%	41%	23%	9%	4%	3%	0%
30	12%	95%	75%	58%	42%	25%	14%	6%	2%
40	11%	95%	80%	66%	52%	38%	25%	15%	5%
50	11%	96%	80%	72%	59%	45%	34%	20%	8%
60	7%	96%	81%	72%	63%	50%	40%	30%	12%
70	16%	96%	82%	75%	65%	55%	45%	34%	14%
80	8%	96%	82%	75%	70%	58%	48%	39%	17%
90	6%	96%	82%	75%	71%	58%	48%	40%	19%
100	16%	96%	82%	76%	73%	60%	49%	43%	21%
PRECISION									
		Repository Method PT							
Gen	GP	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
10	1%	4%	9%	6%	3%	2%	1%	0%	0%
20	0%	4%	8%	8%	7%	4%	2%	2%	0%
30	2%	4%	7%	9%	9%	7%	5%	3%	1%
40	4%	4%	7%	9%	9%	9%	7%	5%	4%
50	6%	4%	6%	8%	9%	9%	8%	6%	6%
60	6%	3%	6%	8%	9%	10%	9%	8%	7%
70	5%	3%	6%	8%	9%	9%	9%	8%	6%
80	3%	4%	6%	8%	9%	9%	9%	8%	6%
90	7%	3%	5%	7%	9%	9%	9%	8%	7%
100	2%	3%	5%	7%	9%	9%	9%	9%	7%
ACCURACY									
		Repository Method PT							
Gen	GP	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
10	97%	34%	79%	90%	94%	96%	97%	97%	97%
20	97%	31%	75%	86%	91%	93%	95%	95%	97%
30	92%	28%	70%	83%	87%	90%	92%	93%	96%
40	92%	26%	67%	79%	85%	88%	90%	92%	95%
50	93%	24%	63%	77%	83%	87%	89%	90%	94%
60	94%	23%	61%	75%	82%	86%	88%	89%	93%
70	89%	23%	59%	72%	81%	85%	87%	88%	92%
80	93%	23%	58%	72%	78%	83%	86%	87%	91%
90	94%	21%	56%	72%	77%	83%	85%	87%	91%
100	86%	20%	55%	71%	77%	82%	85%	86%	90%

6.3.5 Experiment: Importance of the accumulation of rules through the evolutionary process

Objective

The previous experiment showed that the combination of diverse rules outperforms the precision and accuracy produced by a standard GP in an extreme imbalanced environment. At this point two important questions arise concerning the factors that favour RM:

1. Can RM obtain similar results taking rules from a smaller set of individuals, instead of the complete population?
2. Is it beneficial to gather rules from previous generations?

Those questions are answered by performing the following experiment.

Procedure

To uncover the effect of the accumulation of rules from previous generations, let Accumulated Repository Method (ARM) be the procedure that collects rules through different generations and let Simple Repository Method (SRM) be the process that compiles rules just from one generation. AR and SR denote two rules repositories created by ARM and SRM, respectively. Superscripts specify the generation of the population. Subscripts indicate the number of top individuals in the population. For instance, AR_{100}^{70} is a repository that compiles rules from the top 100 individuals in the population. These rules were accumulated during generations 10,20,...,70. On the other hand, SR_{10}^{70} is a repository that comprises rules from the top 10 individuals of the population at generation 70.

Table 6.7: Accumulated Repository Method results. Recall, precision and accuracy of (a) Best Individual, (b) AR_{10} (c) AR_{100} (d) AR_{1000}. Precision Threshold = 60%

Gen	Recall				Precision				Accuracy			
	(a)	(b)	(c)	(d)	(a)	(b)	(c)	(d)	(a)	(b)	(c)	(d)
10	1%	2%	2%	2%	1%	1%	2%	1%	97%	96%	96%	97%
20	0%	3%	3%	4%	0%	2%	2%	2%	96%	96%	96%	95%
30	12%	5%	8%	14%	2%	2%	3%	5%	94%	95%	93%	92%
40	11%	8%	11%	25%	4%	3%	4%	7%	90%	94%	92%	90%
50	11%	9%	17%	34%	6%	4%	5%	8%	93%	93%	91%	89%
60	7%	10%	20%	40%	6%	5%	6%	9%	94%	93%	91%	88%
70	17%	12%	25%	45%	6%	6%	7%	9%	87%	93%	90%	87%
80	10%	18%	32%	48%	4%	7%	8%	9%	93%	92%	89%	86%
90	6%	20%	34%	48%	7%	7%	8%	9%	94%	91%	88%	85%
100	14%	23%	36%	49%	2%	7%	8%	9%	88%	90%	88%	85%

To answer the first question, we have to investigate the effects of the number of individuals involved in the rule selection. For that purpose, ARM was used to create three accumulated repositories. The first repository gathers rules from the top ten individuals in the population, the second collects rules from the top 100 individuals. Finally, the third repository compiles rules in the complete population (1000 individuals), let us call them AR_{10},AR_{100} and AR_{1000}, respectively.

The populations used in this experiment were created by a standard GP using the Barclays data set described in section 6.3.1. This experiment produces a large amount of results, which is impractical to analyse. For that reason we focus our attention on just a single PT value. We decide to chose PT=60% because it is able to detect around 50% of the positive cases.

To answer the second question, we test the effects of the rule accumulation through previous generations. Thus, the last experiment was repeated, but this time SRM was applied.

Table 6.8: Simple Repository Method (SRM) results. Recall, precision and accuracy of (a) Best Individual, (b) SR_{10} (c) SR_{100} (d) SR_{1000}. Precision Threshold = 60%

Gen	Recall				Precision				Accuracy			
	(a)	(b)	(c)	(d)	(a)	(b)	(c)	(d)	(a)	(b)	(c)	(d)
10	1%	2%	2%	1%	1%	1%	2%	1%	97%	96%	96%	97%
20	0%	1%	3%	4%	0%	3%	3%	2%	96%	97%	96%	95%
30	12%	2%	7%	12%	2%	1%	3%	5%	94%	95%	94%	93%
40	11%	4%	7%	20%	4%	3%	3%	6%	90%	95%	93%	91%
50	11%	3%	14%	28%	6%	2%	5%	8%	93%	94%	92%	90%
60	7%	6%	13%	31%	6%	5%	5%	9%	94%	94%	92%	89%
70	17%	7%	17%	35%	6%	5%	6%	8%	87%	94%	91%	88%
80	10%	10%	21%	41%	4%	7%	6%	9%	93%	93%	90%	87%
90	6%	12%	20%	36%	7%	6%	6%	8%	94%	93%	90%	87%
100	14%	14%	23%	41%	2%	7%	7%	8%	88%	92%	90%	86%

Observations

The Effect of the number of individuals- This paragraph describes the impact of the number of decision trees in the population. As was mentioned, a series of experiments was carried out using: (a) the best individual, (b) the top 10 individuals, (c) the top 100 individuals of the population and (d) the complete population (1,000 individuals). Those decision trees were used to gather patterns. Table 6.7 shows the results (recall, precision and accuracy) of the experiment. In order to have a graphic idea about the results, these have been plotted in Figure 6.5. As can be seen from Figure 6.5, the recall, precision and accuracy of the best individual oscillates considerably through generations.

Recall- The recall obtained by ARM (Figure 6.5 i) steadily increases through the generations. This clearly improves when the number of individuals (decision trees), in the rule collection, increases. It indicates that there exists a great potential of patterns in the entire population that are not necessarily included in the best individual(s) of the

evolution.

Precision- The precision of ARM and SRM (Figure 6.5 iii,iv) shows an upward trend among generations. In addition, the number of individuals clearly improves the precision of the prediction.

Accuracy- The accuracy (Figure 6.5 v,vi) steadily decreases through generations. Furthermore the increase in the number of individuals affects negatively the accuracy of the classification. In other words, when the number of individuals in the population decreases, the number of patterns decreases too. Thus the predictions become conservative. The conservative predictions benefit accuracy, while liberal predictions worsen accuracy.

The effect of gathering rules from previous generations - In order to analyse the effects of the accumulation of rules from previous generations, we compare the results of ARM (Figure 6.5 i,iii,v) and SRM (Figure 6.5 ii,iv,vi). As can be seen, the trends in recall, precision and accuracy presented by SRM are similar to those in ARM. As can be seen from figure 6.5, the recall and precision reported by ARM (i),(iii) outperformed the SRM results (ii),(iv). Since ARM accumulates rules through generations and SRM does not, it indicates that the collection of rules from previous generations contribute to improve RM performance. It means that some useful rules could be lost during the evolutionary process. In addition, the collection of rules from preceding generations helps to maintain the variety and tackle the convergence (see section 3.3.4).

Reducing the number of evaluations required - An alternative perspective from these results is that RM can be used to reduce the number of evaluations in GP. For example, Figure 6.5 (i) shows that at generation 40, ARM achieved higher recall (0.25) than the best recall achieved by GP (0.18, which was achieved at generation 70). Figure 6.5 (iii) shows that at generation 40, ARM achieved precision (0.066) same as the best precision achieved by GP (which was achieved at generation 90). Similarly, at generation 50, SRM achieved accuracy, precision and accuracy better than or comparable to the best achieved by GP in later generations.

In many applications, evaluations are very expensive. In these problems, users will benefit from the reduction of evalutions. The above results show that RM could be an effective tool for reducing the number of evaluations in GP.

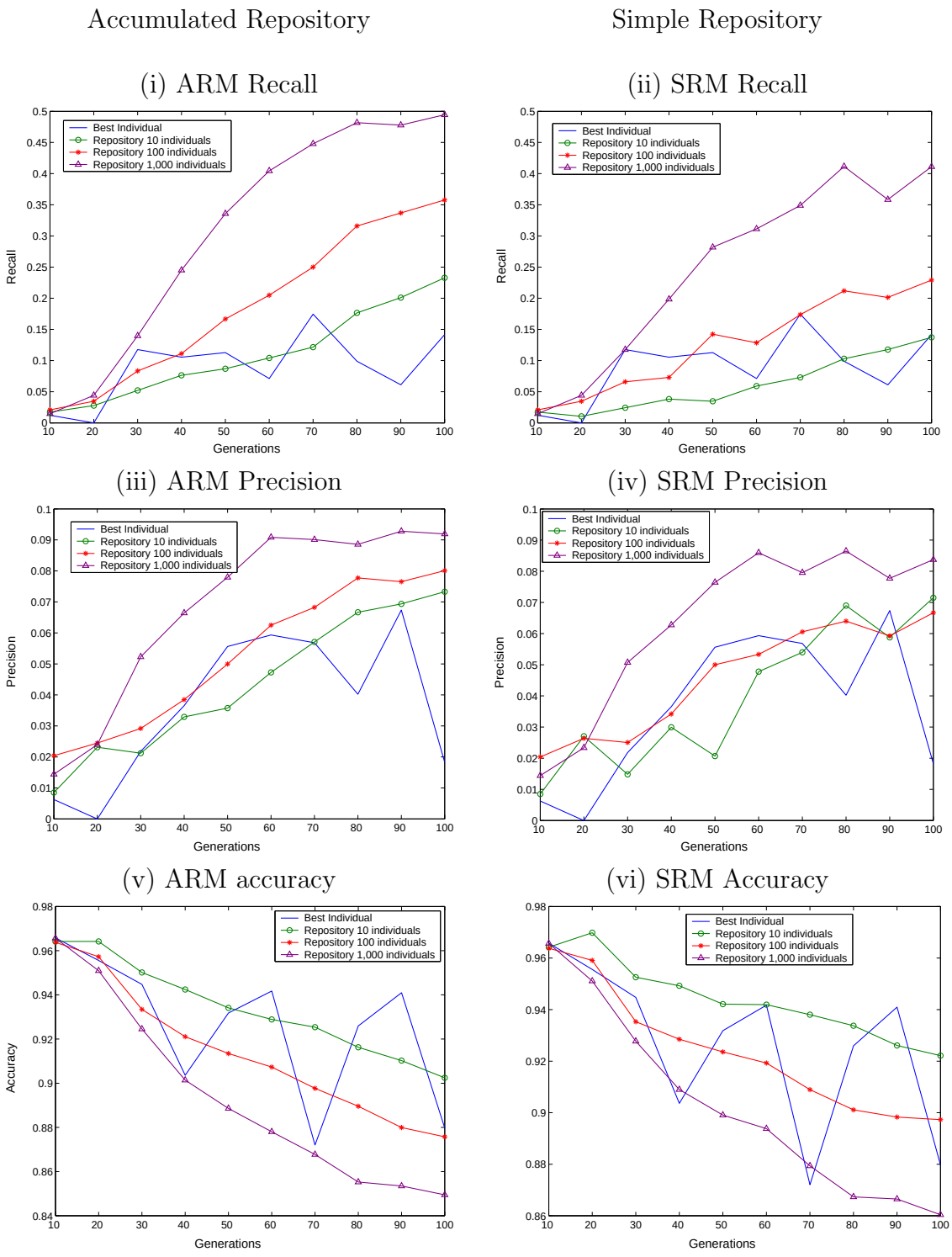


Figure 6.5: Results using Accumulated Repository Method (ARM) and Simple Repository Method (SRM), the experiment was performed using a PT=60%.

6.3.6 Experiment: Rule contribution per individual

Data set: BARCLAYS

Rate of return :	15%
Number of days :	10
Movement :	Increase

Training data set

Training examples:	887
Positive examples:	39

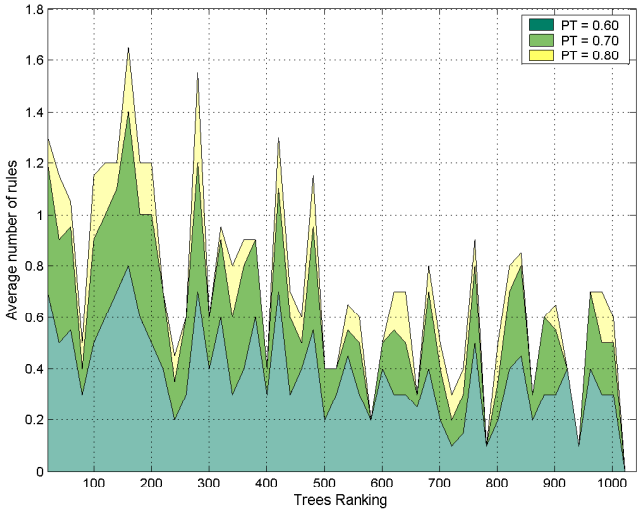
Testing data set

Testing examples:	831
Positive examples:	24

Number of runs : 20

Populations
$P_{10}, P_{20}, \ldots, P_{100}$,

Thresholds PT:
0.60, 0.70, 0.80



GP Parameters

Population size:	1,000
Crossover probability:	.80
Mutation probability :	.05
Hill-climbing prob :	.05
Selection :	tournament (2)
Fitness function :	$\sqrt{Precision \cdot Recall}$

Figure 6.6: Contribution of decision rules

Objective

The objective of the present experiment is to investigate the number of rules that each individual in the population provides to the repository of rules, we hypothesize that individuals with low fitness are able to provide good decision rules. To illustrate this point, consider the following example: let T be a decision tree, such as T follows the DG grammar and holds two rules, thus $T = \{R_1, R_2\}$. The confusion matrix of each rule is shown in figure 6.7.

Rule R_1

		Actual: Positive	Actual: Negative
	Positive prediction	10	0
	Negative prediction	0	90

Rule R_2

		Actual: Positive	Actual: Negative
	Positive prediction	10	0
	Negative prediction	90	0

Decision tree T

		Actual: Positive	Actual: Negative
	Positive prediction	10	0
	Negative prediction	90	0

Figure 6.7: Example: Confusion matrix of rules R_1 and R_2 and $T = R_1 \vee R_2$

As can be observed in figure 6.7, rule R_1 correctly classifies all the examples, thus the accuracy, precision and recall = 100%. In contrast R_2 classifies all the cases as positive, which is incorrect, then accuracy = 10%, precision = 100% and recall = 10%. As was mentioned in section 5.5, a DG decision tree can been seen as a collection of rules associated by disjunctions such as $T = \{R_1 \vee R_2\}$, it means that T is satisfied when at least one of its rules is satisfied. Thus, the tree T classifies an example as positive when R_1 is true or R_2 is true. Given that, R_2 classifies all the examples as positive then T classifies all the instances as positive. Thus the performance of the tree T is accuracy = 10%, precision = 100% and recall = 10%. The overall performance of the tree is very low, despite the fact that it holds a rule that classifies perfectly the data set. This example illustrates a low fitness decision tree that holds a valuable rule

that has been ignored due to the overall performance of the decision tree.

Procedure

In order to probe our claims, the following experiment was performed: the number of rules that every individual provides for the repository was counted. Figure 6.6 displays the number of rules that every individual in the population gives to the repository. The X-axis represents every individual in the population; these are ordered according to their fitness function. The best individual is in the first position, while the worst individual is in position 1,000 individuals. To smooth the graph, the number of rules was averaged on groups of twenty individuals. On the other hand, Y-axis represents the average number of rules that every individual provides for the rule repository. The experiment was repeated using the precision thresholds, PT =60,70,80. It is important to bear in mind that RM extracts rules starting from the best individual of the population, subsequently it takes the next individual in the ranking and so on. To count the number of rules that every individual provides for the rule repository, the following conditions were applied: 1) If a rule is present in more than one individual, that rule will be assigned to the first individual where it was found, it means that rules with the highest fitness will be benefited in the tally and 2) when a flexible rule $R_i \in T_a$ is replaced by a similar rule $R_k \in T_b$ because the latter performs better, the rule will be assigned to individual T_b.

Observations

As can be seen from Figure 6.6, RM picks rules from all sectors of the population, even from the lowest fitness individuals. The number of rules tends to reduce slightly when the ranking of the individual decreases. As can be observed, the graph oscillates considerably, it may be because the population has converged and individuals that have similar fitness could hold the same rules. Since novelty is a necessary condition to be selected, the contribution of rules is concentrated in the first individual of the group of alike individuals. As one might expect, the number of rules tends to decrease when PT increases due to the method becoming more selective and fewer rules are selected.

6.4 Conclusions

This chapter presented a novel approach called Repository Method (RM), whose target is to mine the knowledge acquired by a GP system to compile patterns of the minority class (positive cases) in imbalanced environments. RM has been designed to compile patterns from different individuals and stages of the evolutionary process.

Experimental results showed that RM detected more positive cases than a standard GP, classifier C5.0 and EDDIE-Arb (see sections 6.3.1, 6.3.2 and 6.3.3, respectively). From those experimental results we conclude that the first objective of RM has been achieved.

In addition, a series of experiments to analyse the factors that work in favour of RM was performed in section 6.3.5. The experimental results showed that gathering rules from several individuals of the population helps to collect patterns to classify

more positive cases, which is reflected in the recall improvement. The majority of the positive cases were correctly predicted, as it is shown in the improvement of the precision. On the other hand, the experiment in section 6.3.5 showed that some useful rules can be lost in the evolutionary process, for that reason the accumulation of rules, through generations helps to classify more positive cases. From those results we can conclude that the factors that benefit RM are 1) the collection of rules from several individuals of the population and 2) the accumulation of rules though generations. To reinforce the first argument, another experiment was performed in section 6.3.6 to count the number of rules provided by every individual in the population. This experiment showed evidence that there is a great potential of patterns in the complete population of decision trees, even in individuals with low fitness.

The experimental results in section 6.3.5, also suggests that RM can be used to reduce the required evaluations in GP. Fewer evaluations were required by RM in order to reach the precision, recall and accuracy achieved by the best individual of the GP population (see Figure 6.5).

From experimental results (see sections 6.3.1, 6.3.2 and 6.3.3) it was observed that RM offers a range of classifications to suit different user's preferences. Those results address the second goal of this method. In addition, the experiment in section 6.3.4 was designed to disclose the importance of the evolutionary process to produce useful patterns. As was observed, when RM was applied to a random population the majority of the predictions were clustered in the conservative part of the ROC space. However, when the evolutionary process advanced, this produced populations with more useful

patterns, thus it produced different classifications, from conservative to liberal, to suit different user's preferences.

The third goal of this method is to produce comprehensible solutions that can be understood by the user. For that purpose, a simplification process was included in RM in order to remove the conditions that are not affecting the decision of the rule. We shall leave it to Chapter 7, whose method used the same simplification procedure, to show an illustrative example about the interpretability of the decision rules.

RM is a general method, and therefore, this should not be limited to financial forecasting problems. It could be useful for classification problems where chances are rare (i.e. where the data set is extremely imbalanced).

Chapter 7

Evolving Decision rules

This chapter presents a new approach to generate and evolve a set of decision rules, this is called *Evolving Decision Rules* (EDR). The aims of this method are:

1. To classify the minority class in imbalanced environments

2. To produce a range of solutions to suit different user's risk-guidelines

3. To generate comprehensive decision rules that can be understood by the user

As will be realized Repository Method (see Chapter 6) and EDR share the same aims and some procedures, however, the mechanism of each approach is different. RM is a method that analyses decision trees which were produced by a GP system. This picks up useful rules (patterns) to predict the minority class in imbalanced data sets. RM is a deterministic process because given a specific precision threshold and a specific set of decision trees the result is deterministic. On the other hand, EDR is an evolutionary process that evolves decision rules, generating different solutions every time it is performed (see section 5.1.2)

This chapter is organised as follows: section 7.1 exposes the motivation of this work and provides a brief explanation of the previous works done in this area. This also discloses the main differences between those works and our approach. Section 7.2 describes the procedure of EDR. Next, section 7.3 describes the experiments and results to test our approach. It includes an analysis in order to compare our results with those found by a standard GP, EDDIE-ARB and C.5. Next, an illustrative example, that analyses the role of the set of decision rules produced by EDR, is given in section 7.3.5. Finally, the conclusions are given in Section 7.4.

7.1 Introduction and motivation

EDR [47], [48] is inspired by a previous work that is called Repository Method (RM) [46], [49], [43], [45].As was mentioned, many real world problems need the detection of the minority class. To deal with this problem we proposed Repository Method (RM) described in chapter 6. RM explores the solutions proposed by a GP in order to gather rules (patterns) that classify the minority class in imbalanced environments. This technique uses a threshold to produce several classifications. However, to obtain useful decision rules, RM has to rely on the results obtained by a GP system, such dependance could be a disadvantage. For that reason we propose EDR to evolve decision rules. EDR is an evolutionary process that is based on selection, mutation and hill-climbing[1]. This method has been designed to evolve a set of decision rules that holds different patterns of the positive cases. As was shown in section 6.3.5, some rules that classify

[1]Hill-climbing is a stochastic technique to find the global optimum where the search is given by trying one step in each direction and then choosing the steepest one[73]

positive cases could be eliminated during the evolutionary process. For that reason, we propose to collect rules in order to ensure that all the useful patterns produced by the evolutionary process, will be included in the final solution to the problem.

Previous works

Section 3.4 provides a survey of the literature on systems that evolve decision rules. The majority of those works have used genetic algorithms to evolve a complete set of rules (Michigan approach [58]) or a population of sets of rules (Pittsburgh approach [102], [64]).

Our approach has some similarities to the Michigan approach because EDR evolves rules and the solution of the problem is composed of a complete set of rules. In contrast in the Pittsburgh approach every individual in the population represents a complete set of rules, thus every individual holds an integral solution of the problem.

Some researches have created systems to evolve decision rules by using a normal GP [85], [12], [13]. Every decision tree classifies a single class, when the evolutionary process has been completed the best individual of the evolution is converted into classification rules. The authors claim that their approaches are able to generate comprehensible rules. However, GP tends to accumulate extra-code [69],[2],[107], [103] and unless the rules are simplified, these reveal the real variables and conditions in the rule. In contrast EDR provides a simplification of rules that helps to identify the real conditions of the rules. Given that our objective is to predict the minority class, in imbalanced environments, we believe that it is better to collect all the available patterns. Otherwise the best tree of the evolution would contain patterns that may not repeat themselves in

future data sets. The example in section 7.3.5 illustrates that situation. Finally, none of those applications are able to provide a range of solutions to suit different user's references.

7.2 Evolving Decision Rules Procedure

This section describes the EDR procedure, it starts by describing the general mechanism. Next, an overview of EDR is introduced and finally, a description of each step is given.

EDR evolves a set of decision rules by using GP, this receives feedback from a key element that is called *repository*. Let us define the repository as a structure, whose objective is to store a set of rules. Those rules classify or not a single class (binary classification) and these are used to create a range of classifications that allows the user to choose the best trade-off between the misclassifications and false alarms costs (see section 2.4.3). The list of procedures involved in EDR are the following:

1. creation of the initial population
2. Extraction of rules
3. Rule simplification 5.6).
4. Adding new rules to the repository
5. Creation of a new population
6. *Testing EDR:*

A general description of EDR has been introduced; in the following sections this approach will be described. However, the details of some procedures are explained in Chapter 5

Algorithm 1: EDR()

```
input  : integer PopulationSize,
         real PrecisionThreshold,
         integer NumInitialInstances,
         integer MaxNumberOfRules,
         real PercentInstances,
         real HillClimbingProb,
         real NumGenerations,
output : List
1  begin
2      List Population = CreateInitialPopulation(PopulationSize);
3      List Repository
4      for j=1 to NumRules do
5          for every T_i ∈ Pop do
6              T_i ← {R_a}; /* the set of rules in T_i*/
7              for every R_a ∈ T_i do
8                  if Precision(R_a) > PrecisionThreshold then
9                      R_a' ← Simplification(R_a);
10                     if Size(Repository) = MaxNumberOfRules then
11                         /* where Precision(R_w) < Precision(R_k) ∀
                           R_k ∈ Repository Repository ← Repository - R_w
12                     Repository ← Repository U R_a
13         CreatePopulation(PopulationSize,NumInitialInstances,PercentInstances,
14         HillClimbingProb);
15     return Population;
16 end
```

7.2.1 Initialization of Population

The objective is to create a collection of candidate solutions. We created an initial population of decision trees using the Discriminator Grammar (DG) (see section 5.4). The population is created at random using the growth method (see section 3.3).

At this point a question arises, if the system evolves decision rules why is the population composed of decision trees? As was pointed out, in section 5.5, a single individual (decision tree) could contain more than one rule, thus the creation of decision trees generates more solutions. Because disjunctions (OR) increases notably the number of rules in the decision tree, as a consequence the probability of finding valuable patterns increases too.

7.2.2 Rule extraction

This part of the process analyses every decision tree to define its rules and select those patterns that are useful for the classification. Once a rule $R_k \in T$ has been defined, it is evaluated. If the precision of R_k achieves a predefined Precision Threshold (PT), where $PT > 0$, then R_k is considered for the *Rule simplification* process, otherwise R_k is discarded. The details of this process are explained in section 5.5.

7.2.3 Rule Simplification

The aim of rule simplification is to remove vacuous and redundant conditions (definitions 4 and 3 respectively). The explanation of this process is detailed in section 5.6.

7.2.4 Adding new rules in the repository

Once a rule R_k has been simplified, we have to determine the novelty of that rule by comparing R_k to the existing rules in the repository. The explanation of this process is given in section 5.7.

7.2.5 Creation of a new population

This section describes the procedure to generate a new population of individuals. As was mentioned previously, the initial population was created randomly. However, in the subsequent generations the population will be totally replaced by a new population of decision trees. These are created by means of the mutation and hill-climbing of the existing rules in the repository. The number of rules in the repository is variable because it depends on the new patterns that have been found. The number of rules is limited by μ, which represents the maximum number of rules in the repository (see pseudo-code CreatePopulation() algorithm is in the appendix). The creation of a new generation of individuals is described below:

Let Rep be the repository of rules
μ be the maximum number of rules in Rep
φ be the number of initial descendants per rule in Rep
s be the current number of rules in Rep
ρ be the size of the population
β be the percentage of population created by the rules
h be the hill-climbing probability

- If $(s \cdot \varphi \leq \rho)$. At the beginning of the evolutionary process, when the product of the current number of rules in Rep multiplied by the number of initial descendant per rule is less than the population size, the system will replace the population in the following generation with φ offspring per rule. If the number of new offspring is less than the population size, then the remaining individuals will be created randomly by the grow-method.

- If ($s \cdot \varphi > \rho$). It is obvious that the repository is continuously growing and so, there is a maximum number of rules that can be stored. Now, we have to consider when the product of the current number of rules in *Rep* multiplied by the number of initial descendant per rule is greater than the population size ($\varphi \cdot s > \rho$), then the rules have to reproduce less and of course, the number of offspring is limited by $|\rho \ / \ s|$. As the value of the division is truncated, the number of offspring is less than the size of the population, thus the remaining individuals are created at random.

- If ($s = \mu$ and $s > \rho$) When the repository is totally full and the number of rules is greater than the population size, thus just a fraction of rules in *Rep* is allowed to produce one descendant. The rules to produce offspring are selected randomly without any type of elitism. However, those descendants will produce only β% of the population. The remaining individuals will be created at random in order to create variety. β is a parameter, which is determined by the user.

- The hill-climbing is applied randomly using a probability h, this will be one of the descendants of a rule, the remaining individuals, if any, are produced by mutation.

7.2.6 Rule evaluation

Once the evolutionary process has finished, the rules in the repository will be used to classify the testing data set as follows:

1. Sort the rules by precision in descending order

2. Define a set of thresholds $\tau = \{\tau_i\}$ between [0,1] separated at regular intervals for example: τ=$\{0,.05, \cdots,.95,1\}$

3. For each threshold τ_i, select those rules from *Rep* whose precision is greater or equal to τ_i, then store those rules in a sub-repository $Rep_{\tau_i} = \{R_{\tau_{ik}}\}$ where $R_{\tau_{ik}}$ is a rule, such as, $Precision(R_{\tau_{ik}}) \geq \tau_i \ \forall \ R_{\tau_{ik}} \in Rep_{\tau_i}$.

4. For each example in the data set, if at least one of the rules in the sub-repository satisfies the example, this is classified as positive, otherwise it is considered negative.

EDR has been designed to produce as many classifications as the number of thresholds τ. Each of those classifications can be plotted in ROC space (section 2.4.3), the result is a curve that can be used to measure the general performance of the classifier and to choose the best tradeoff between misclassifications and false alarms.

7.3 Experimental section

This section describes a series of experiments to test our approach. First, a series of experiments was performed to compare EDR performance with a standard GP, RM, EDDIE-Arb and classifier C.5 (sections 7.3.1 7.3.2 7.3.3, 7.3.3). Next, an experiment to test the performance of RM at different levels of complexity is described in section 7.3.4. Finally, an illustrative example that analyses a set of rules is shown in section 7.3.5. Preliminary experiments were performed to find out a good set of parameters for executing EDR.

7.3.1 Experiment: Comparison of EDR with RM

Name: BARCLAYS

March,1998-January,2005

Rate of return :	15%
Number of days :	10
Movement tendency :	Increase

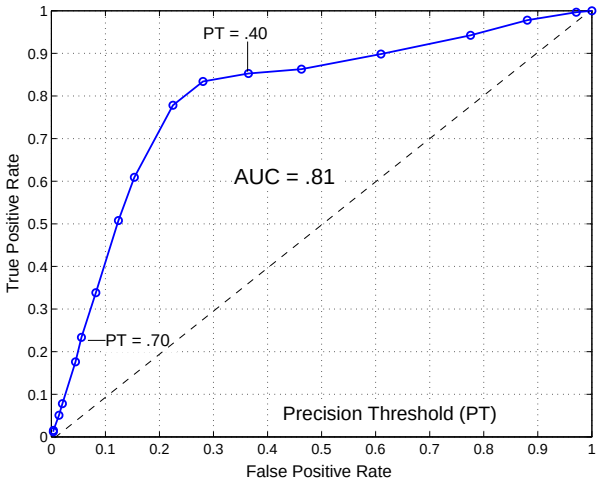
Training data set :

Training examples:	887
Positive examples:	39 (4.3%)

Testing data set

Testing examples:	831
Positive examples:	24 (2.8%)

Number of runs :	20
AUC :	0.81



EDR Parameters

Precision Threshold PT:	0.08
Population size ρ :	1,000
Number of initial offspring φ:	10
Individuals at random β :	20
Max number of rules μ :	2,500
Number of generations :	25
Hill-climbing prob :	0.14

Figure 7.1: Barclays parameters and results

Objective of the experiment

The objective of this experiment is to analyse the performance of EDR and to compare this with RM performance.

Procedure

The parameters used to run RM are listed in Figure 6.1. The set of rules generated by EDR, was used to create twenty classifications using the same number of thresholds τ.

Observations Figure 7.1 shows the ROC curve created by EDR. Notice that, the classifications provided by EDR are well-distributed over the ROC curve, thus, it is

Table 7.1: EDR Results using Barclays, τ is the minimum precision threshold

τ	*Recall*	*Precision*	*Accuracy*	τ	*Recall*	*Precision*	*Accuracy*
1.00	0.01	0.09	0.97	0.50	0.78	0.10	0.78
0.95	0.01	0.09	0.97	0.45	0.83	0.08	0.72
0.90	0.02	0.11	0.97	0.40	0.85	0.07	0.64
0.85	0.05	0.10	0.96	0.35	0.86	0.05	0.55
0.80	0.08	0.10	0.95	0.30	0.90	0.04	0.41
0.75	0.18	0.11	0.93	0.25	0.94	0.04	0.25
0.70	0.23	0.11	0.92	0.20	0.98	0.03	0.14
0.65	0.34	0.11	0.90	0.15	1.00	0.03	0.06
0.60	0.51	0.11	0.87	0.10	1.00	0.03	0.06
0.55	0.61	0.11	0.84	0.05	1.00	0.03	0.06

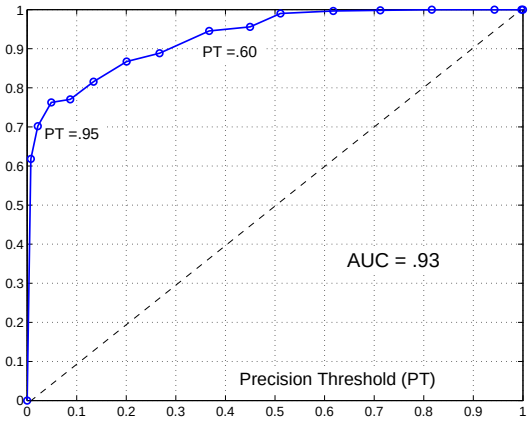
possible to find conservative and liberal predictions. Now, let us discuss the points that form the curve. For instance, when τ is equal to 0.50, the recall shows high performance because EDR was able to detect 78% of the positive examples, this result has not sacrificed accuracy (78%). When the investor's risk-guidelines are conservative, EDR offers a range of suitable classifications. For instance, with the threshold τ= 0.70 the system is able to classify 23% (almost a quarter) of positive cases with very high accuracy (92%). On the other hand, when τ is below 0.40, then the classifier's performance tends to decrease because the number of new positive cases that are detected are paid for with a serious decrease in accuracy and precision.

RM comparison - The result provided by RM and EDR is a set of classifications, which are distributed in the ROC space. Thus, we use the AUC to compare the performance of the classifiers. As was shown in section 6.3.1 the AUC obtained by RM was 0.77, which is outperformed by the AUC generated by EDR (0.81).

7.3.2 Experiment: Comparison of EDR with EDDIE-Arb

Name: Arbitrage

Movement tendency :	Increase
EDR Parameters	
Precision Threshold PT:	10%
Population size ρ :	1,000
Number of initial offspring φ	:10
Individuals at random β :	20%
Max number of rules μ :	2,000
Number of generations :	30
Hill-climbing prob :	5%
Number of runs :	20
AUC :	.93



Training data set		**Testing data set**	
Training examples:	1,006	Testing examples:	635
Positive examples:	242 (24%)	Positive examples:	159 (25%)

Figure 7.2: Barclays

Objective of the experiment

The objective of this experiment is to compare EDR performance with EDDIE-ARB. Additionally, EDR is compared to RM performance.

Procedure

As can be seen from Figure 7.2, the Arbitrage data set is imbalanced because the minority class is composed of 24% of the total cases in the training data and 25% in the testing data set. However, the imbalance is not so high as in the previous example.

Observations

Let us analyse the results obtained by EDR. As can be seen from Table 7.2, EDR detected 70% of the positive examples with a precision of 92% and an accuracy of 90% when $\tau = 0.95$. On the other hand, almost the totality (96%) of the positive cases is predicted with an accuracy of 65% using $\tau = 0.60$. When τ is smaller than 0.55 the performance of the classifier becomes liberal and the increase in the recall is paid for by a serious decrease in accuracy.

EDDIE-Arb - The performance of EDDIE-Arb was *True Positive*=66.8, *False Positive*=0 *False Negative*=92.2 and *True Negative*=476. Thus, the *false positive rate* =0 and the *true positive rate* = 0.42. It means that, EDDIE-Arb result is plotted at the point (0,0.42). It is fair to say that the approach proposed by Tsang et al. [114] achieved greater precision (100%), detecting 42% of the positive cases. However, EDR detected more positive cases moving towards the liberal predictions; obviously it is paid for by loosing precision. It is important to notice that, EDR offers a full-range of classifications to fulfill the user's requirements, this is not the case for GP systems, which offer a single result.

RM - The performance obtained by RM using the same data set produced a ROC curve whose AUC=83.5, while the AUC obtained by EDR was 93% (see Figure 7.2), which means that EDR outperformed the results of RM.

Table 7.2: EDR Results using Arbitrage data set, τ is the minimum precision threshold

τ	*Recall*	*Precision*	*Accuracy*	τ	*Recall*	*Precision*	*Accuracy*
1.00	0.62	0.95	0.90	0.50	1.00	0.35	0.53
0.95	0.70	0.90	0.91	0.45	1.00	0.32	0.46
0.90	0.76	0.83	0.90	0.40	1.00	0.29	0.38
0.85	0.77	0.74	0.88	0.35	1.00	0.26	0.29
0.80	0.82	0.66	0.85	0.30	1.00	0.25	0.25
0.75	0.88	0.58	0.81	0.25	1.00	0.25	0.25
0.70	0.90	0.53	0.77	0.20	1.00	0.25	0.25
0.65	0.95	0.46	0.71	0.15	1.00	0.25	0.25
0.60	0.96	0.41	0.65	0.10	1.00	0.25	0.25
0.55	0.99	0.39	0.61	0.05	1.00	0.25	0.25

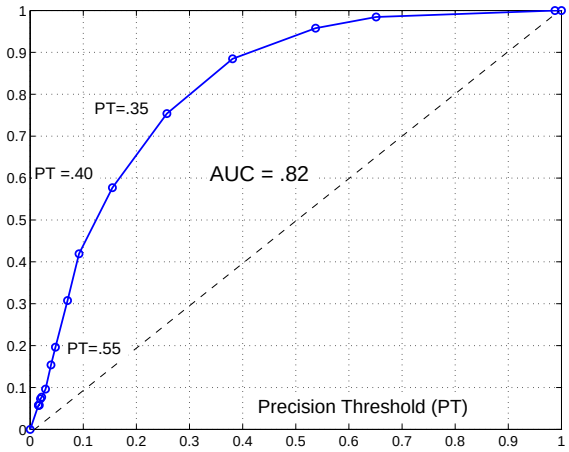
7.3.3 Experiment: Comparison of EDR with C5.0

Objective of the experiment

The aim of this experiment is to compare the performance of our approach with the performance of the trial version of C.5 developed by Quinlan. However, this version can not process more than 400 training or testing cases. For that reason, the size of the data sets had to be adjusted to meet this requirement. Quinlan's algorithm was tested using ten-fold cross-validation and standard parameters setting. The experiment was performed using two data sets Barclays and Tesco sections 7.3.3 and 7.3.3 respectively.

Name: Barclays 400

Rate of return :	15%
Number of days :	10
Movement tendency :	Increase
Training data set :	
Training examples:	400
Positive examples:	15 (3.7%)
Testing data set	
Training examples:	400
Positive examples:	13 (3.2%)
Number of runs :	20
AUC :	.82



EDR Parameters

Precision Threshold PT:	6%
Population size ρ :	1,000
Number of initial offspring φ:	10
Individuals at random β :	15%
Max number of rules μ :	700
Number of generations :	25
Hill-climbing prob :	15%

Figure 7.3: Barclays 400 records

Table 7.3: EDR Results using Barclays400 data set, τ is the minimum precision threshold

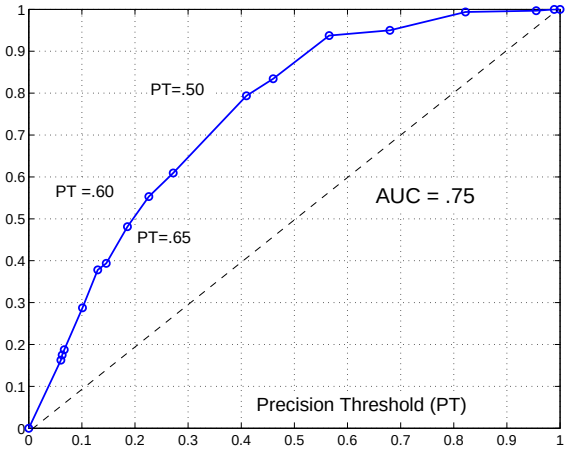
τ	*Recall*	*Precision*	*Accuracy*	τ	*Recall*	*Precision*	*Accuracy*
1.00	0.06	0.11	0.95	0.50	0.31	0.13	0.91
0.95	0.06	0.11	0.95	0.45	0.42	0.13	0.89
0.90	0.06	0.11	0.95	0.40	0.58	0.11	0.84
0.85	0.06	0.11	0.95	0.35	0.75	0.09	0.74
0.80	0.06	0.10	0.95	0.30	0.88	0.07	0.63
0.75	0.07	0.11	0.95	0.25	0.96	0.06	0.48
0.70	0.08	0.11	0.95	0.20	0.98	0.05	0.37
0.65	0.10	0.10	0.94	0.15	1.00	0.03	0.04
0.60	0.15	0.12	0.93	0.10	1.00	0.03	0.03
0.55	0.20	0.12	0.93	0.05	1.00	0.03	0.03

Observations

The result obtained by C.5 is the following: *True positive* = 0, *false positive*=0, *false negatives* = 13, *true negative* = 387. This classification is plotted at the point (0,0) in the ROC space. As can be seen, C.5 produced a highly accurate prediction (96.7%). However, it failed to detect the positive cases. In contrast EDR was able to detect more than a half (58%) of the positive examples using τ=40%. Since EDR offers liberal and conservative options for the user, it is possible to choose different types of classifications. For example, a liberal prediction is made by using τ =.35, the recall is 75% and the accuracy is still acceptable (74%). On the other hand, a conservative prediction is made when τ =.55, the recall is low (20%), but is highly accurate (93%). The ROC curve obtained by EDR is plotted in Figure 7.3. The results obtained by EDR (recall, precision and accuracy) are displayed in Table 7.3.

Name: TESCO 400

Rate of return :	10%
Number of days :	10
Movement tendency :	Decrease
Training data set :	
Training examples:	400
Positive examples:	34 (9%)
Testing data set	
Training examples:	400
Positive examples:	14 (4%)
Number of runs :	20
AUC :	.75



EDR Parameters	
Precision Threshold PT:	6%
Population size ρ :	1,000
Number of initial offspring φ:	10
Individuals at random β :	15%
Max number of rules μ :	2,000
Number of generations :	30
Hill-climbing probability :	15%

Figure 7.4: Tesco 400 records

Observations

C.5 comparison - The result obtained by C.5 is the following: *True positive* = 0, *false positive*=0, *false negatives* = 14, *true negative* = 386. As in the previous experiment it failed to detect the positive cases. On the other hand, EDR could detect more than a half (55%) of the positive examples when τ=60%. As can be seen in Figure 7.4, the points in the ROC curve are well distributed along this. It means that EDR is able to offer liberal and conservative options for the user. For example, conservative predictions are made by using a τ=1,.95,.90,.85 the recall is around 16% and 19%, and the accuracy between 90% and 91%. In contrast a liberal prediction is made when τ =50%, the recall is (79%) and the accuracy is acceptable (60%). The completed results

Table 7.4: EDR Results using Tesco 400 data set, τ is the minimum precision threshold

τ	*Recall*	*Precision*	*Accuracy*	τ	*Recall*	*Precision*	*Accuracy*
1.00	0.16	0.10	0.91	0.50	0.79	0.07	0.60
0.95	0.16	0.10	0.91	0.45	0.83	0.07	0.55
0.90	0.18	0.10	0.91	0.40	0.94	0.06	0.45
0.85	0.19	0.10	0.90	0.35	0.95	0.06	0.35
0.80	0.29	0.11	0.87	0.30	0.99	0.05	0.21
0.75	0.38	0.11	0.85	0.25	1.00	0.04	0.08
0.70	0.39	0.10	0.84	0.20	1.00	0.04	0.05
0.65	0.48	0.10	0.80	0.15	1.00	0.04	0.04
0.60	0.55	0.09	0.77	0.10	1.00	0.04	0.04
0.55	0.61	0.09	0.72	0.05	1.00	0.04	0.04

obtained by EDR are plotted in Figure 7.4, which shows that AUC = .75. The recall, precision and accuracy are displayed in Table 7.4.

7.3.4 Experiment to test different levels of complexity

Name: Artificial

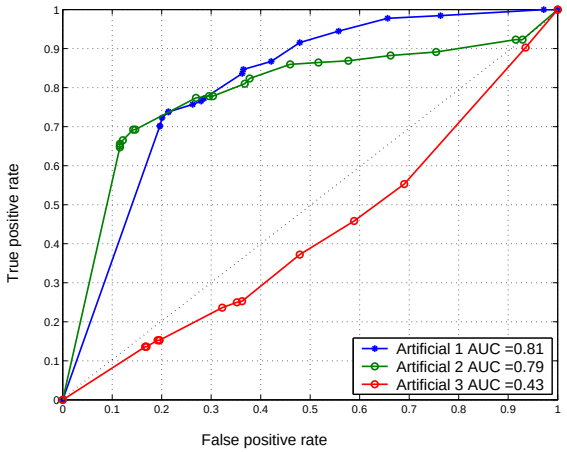
Training data set

Number of examples :	600
Positive examples	
$Artificial_1$:	28 (4.6%)
$Artificial_2$:	18 (3%)
$Artificial_3$:	18 (3%)

Testing data set

Number of examples :	600
Positive examples	
$Artificial_1$:	29 (4.8%)
$Artificial_2$:	17 (2.8%)
$Artificial_3$:	18 (3%)

Number of runs :	20
$Artificial_1$ AUC=	.81
$Artificial_2$ AUC=	.79
$Artificial_3$ AUC=	.43



EDR Parameters

Precision Threshold PT:	6%
Population size ρ :	1,000
Number of initial offspring φ:	10
Individuals at random β :	15%
Max number of rules μ :	2,000
Number of generations :	30
Hill-climbing probability :	15%

Figure 7.5: Results using three Artificial Data sets

Objective

The objective of this experiment is to test EDR using two data sets with different levels of complexity and a data set whose signal was labeled at random, which means that it does not have any pattern.

Procedure

We do not have a formal definition of rule complexity. However, for simplicity we measure the complexity of the data sets counting the number of conditions that are

involved in the solution. When the number of conditions in the solution increases, then we say that the solution is more complex. In order to control the complexity of the data in the experiment, we created three artificial data sets. The data set $Artificial_1$ was created as follows:

1. A set of 1,200 records was created, every record holds eight independent variables with real values. Every variable was randomly generated in a range of [0-1].

2. Every record was labeled with a class (*positive* or *negative*). The records that meet the requirements in at least one of the rules in S_1 (see Figure 7.6) is labeled as positive, otherwise the record is classified as negative.

3. The data was split in two data sets (training and testing) holding the same number of records (600).

The second artificial data set $Artificial_2$ was created repeating the steps 1-3 , but using S_2 instead of S_1 (see Figure 7.6). And the third data set ($Artificial_3$) the signal (label) was randomly categorised.

Observation

Before we start the analysis of the results in the testing data sets, let us present the results obtained by EDR in the training data set for each of the experiments. Using $Artificial_1$ AUC=.99, $Artificial_2$ AUC=.90 and $Artificial_3$ AUC=.91. As can be observed, EDR captures patterns from all the training data sets. However, let us analyse the results in the testing data sets:

Figure 7.6: Set of rules used to created the artificial data sets

S_1={ $R_1 = var_1 > 0.99$
$R_2 = var_2 < 0.009$
$R_3 = var_5 < 0.898$ and $var_5 > 0.89$
$R_4 = var_5 < 0.01$
$R_5 = var_6 > 0.88$ and $var_6 < 0.89$ }

S_2={ $R_1 = var_1 > 0.5$ and $var_1 < 0.58$ and $var_2 > 0.5$ and $var_3 < 0.7$ and $var_4 < var_3$
$R_2 = var_3 < 0.45$ and $var_3 > var_2$ and $var_3 > var_4$ and $var_3 > var_5$ and $var_3 > var_6$
$R_3 = var_8 < 0.898$ and $var_8 > 0.86$ and $var_5 > 0.065$ and $var_5 < 0.35$ and $var_3 > var_7$
$R_4 = var_1 > 0.5$ and $var_1 < 0.58$ and $var_2 > 0.5$ and $var_3 < 0.7$ and $var_4 < var_3$ and $var_4 < var_6$
$R_5 = var_6 > 0.56$ and $var_7 > var_6$ and $var_8 > var_6$ and $var_8 < var_1$
$R_6 = var_1 > var_7$ and $var_1 > var_6$ and $var_6 < 0.23$ and $var_5 < var_6$}

S_3= Random selection

$Artificial_1$ - According to our definition of complexity, Artificial$_1$ is a data set which has a low level of complexity. As can be observed from Figure 7.5, the AUC obtained by EDR is .81, surprisingly EDR does not offer any conservative prediction, as can de seen in Table 7.5. However, EDR found classifications that detect the 70% of the positive cases with accuracy of 80% when $\tau > .85$. On the other hand, when τ decreases the detection of positive cases increases steadily as the number of false alarms.

$Artificial_2$ - Artificial$_2$ is more complex than the previous data set. As can be observed from Figure 7.5, the AUC obtained by EDR is .79. It means that the performance of EDR in $Artificial_1$ was slightly better than in $Artificial_2$. As can be observed in the ROC curve, EDR was unable to classify a small part of the positive

cases. Those cases were classified just when τ was really low. Looking for an explanation of this phenomenon the data set was analysed, it was discovered that rule $R_3 \in S_2$ produces a single positive case in the testing data set but it did not generate any positive case in the training data set. It means that the instance was not identified because there was not a pattern to train EDR.

$Artificial_3$ - As was explained previously, this data set was labeled randomly. It means that, there are no patterns in the training data set that show the patterns to classify similar cases in future data sets. As was expected EDR gathered patterns from the training data set, but these were not repeated in the testing data set. Figure 7.5 shows the ROC curve plotted by EDR using $Artificial_3$. As can be seen, the AUC obtained is .43, the performance of EDR was very low, according to the ROC space described in Section 2.4.3, EDR produced a random classification because the patterns captured in the training data were not repeated in the testing data set. The main reasons for the low performance using EDR, which is a classifier based on supervised learning, can be summarized as follows:

1. The data set does not contain any patterns or the independent variables do not describe the behaviour of the data set.

2. The signal in the data set is labeled incorrectly

3. The patterns in the training data set do not repeat in the testing data set

The general observations of this experiment are the following: EDR is able to discover patterns to classify rare cases in imbalanced data sets. However, it is necessary

Table 7.5: EDR Results using Artificial 1 data set, τ is the minimum precision threshold

τ	*Recall*	*Precision*	*Accuracy*	τ	*Recall*	*Precision*	*Accuracy*
1.00	0.70	0.15	0.80	0.50	0.84	0.10	0.65
0.95	0.70	0.15	0.80	0.45	0.85	0.11	0.65
0.90	0.70	0.15	0.80	0.40	0.87	0.09	0.59
0.85	0.70	0.15	0.80	0.35	0.92	0.09	0.54
0.80	0.72	0.15	0.80	0.30	0.94	0.08	0.47
0.75	0.74	0.15	0.78	0.25	0.98	0.07	0.37
0.70	0.74	0.15	0.78	0.20	0.98	0.06	0.27
0.65	0.76	0.13	0.74	0.15	1.00	0.05	0.08
0.60	0.77	0.12	0.72	0.10	1.00	0.05	0.05
0.55	0.77	0.12	0.72	0.05	1.00	0.05	0.05

Table 7.6: EDR Results using Artificial 2 data set, τ is the minimum precision threshold

τ	*Recall*	*Precision*	*Accuracy*	τ	*Recall*	*Precision*	*Accuracy*
1.00	0.65	0.14	0.88	0.50	0.81	0.06	0.64
0.95	0.65	0.14	0.88	0.45	0.82	0.06	0.63
0.90	0.65	0.14	0.88	0.40	0.86	0.05	0.55
0.85	0.66	0.14	0.88	0.35	0.86	0.05	0.49
0.80	0.67	0.14	0.87	0.30	0.87	0.04	0.44
0.75	0.69	0.12	0.85	0.25	0.88	0.04	0.35
0.70	0.69	0.12	0.85	0.20	0.89	0.03	0.26
0.65	0.77	0.08	0.73	0.15	0.92	0.03	0.11
0.60	0.78	0.07	0.71	0.10	0.92	0.03	0.10
0.55	0.78	0.07	0.70	0.05	0.92	0.03	0.10

to provide a representative training data set in order to capture the patterns to predict future cases. The complexity of the rules does not seem to affect seriously the performance of EDR. However, more research needs to be done about this.

Table 7.7: EDR Results using Artificial 3 data set, τ is the minimum precision threshold

τ	*Recall*	*Precision*	*Accuracy*	τ	*Recall*	*Precision*	*Accuracy*
1.00	0.14	0.02	0.81	0.50	0.24	0.02	0.66
0.95	0.14	0.02	0.81	0.45	0.24	0.02	0.66
0.90	0.14	0.02	0.81	0.40	0.25	0.02	0.64
0.85	0.14	0.02	0.81	0.35	0.25	0.02	0.63
0.80	0.14	0.02	0.81	0.30	0.37	0.02	0.52
0.75	0.14	0.02	0.81	0.25	0.46	0.02	0.41
0.70	0.14	0.02	0.81	0.20	0.55	0.02	0.32
0.65	0.15	0.02	0.79	0.15	0.90	0.03	0.09
0.60	0.15	0.02	0.79	0.10	0.90	0.03	0.09
0.55	0.15	0.02	0.79	0.05	0.90	0.03	0.09

7.3.5 An illustrative example to analyse a set of decision rules produced by EDR

Objective

This section analyses a set of decision rules that were produced by EDR, the objectives of this study are:

1. To show how a larger collection of rules can help to detect the minority class in imbalanced environments

2. Shows the comprehensibility of the solutions provided by EDR

Procedure

The example was taken from BARC-400, it means that the training and the testing data sets are composed of 400 records each. The training data set has 15 positive examples, while the testing has 13. The set of rules for this analysis achieved a precision of 1 in the training data set, this set of rules is displayed in Table 7.8. The complete set of rules classifies eight positive examples, it means recall= 53% in the training data set. Table 7.9 shows the map of the positive examples that every rule classifies. As can be observed, every rule classifies three or four cases each. Given that the precision of the rules was 1, it means that the rules do not classify any negative case. Obviously there is overlapping in the classification. However, an important question arises here: Is it useful to keep a collection of rules that overlap their predictions?. This question is relevant because this will support one of the main claims of this method. This example was designed to answer that question.

Observations

As can be seen from Table 7.8, the set of rules classifies 4 true positives. Additionally it predicts 31 false alarms. Thus, the results obtained in the example are recall =23%, precision =09% and accuracy=.89.

Let us analyse the set of rules in Table 7.8, as the novelty is a basic condition for it to be included in the repository, notice that rules contain common conditions, but these are not identical rules. However, there is overlapping in the classification and even identical classification in rules R_6, R_7, R_8, R_9 and in rules R_{11}, R_{14}, R_{15} and R_{17}. It means that the genotype is different, but the phenotype is similar. Obviously we are taking as the phenotype the behaviour of rule. In the cases mentioned the rules produced the same results, let us analyse in detail the conditions and variables that are involved in each set of rules.

Let $S_a = \{R_6, R_7, R_8, R_9\}$ be the set of rules to analyse

As was expected S_a holds a set of different rules because EDR provides a mechanism to select different patterns, avoiding repeating the same rules in the repository (see section 5.7, 5.6). However, it is important to analyse the variables and relations that are involved in each rule in order to determine if those rules could be correlated.

As can be seen from Table 7.8, R_6 is different from the other rules in S_a, because R_6 does not have any equal hard condition or similar condition to those rules in S_a. On the other hand, R_7, R_8 and R_9 share the conditions: $var_{10} < var_{17}$ and $var_7 > 0.0727$, as the following paragraph shows:

$$R_6 = \quad var_3 > var_{18} \wedge var_{11} < -0.5056$$

$$\begin{array}{lll} & & \overbrace{\text{Common conditions}} \\ R_7 = & var_4 > var_{15} & \wedge \quad \overbrace{var_{10} < var_{17} \wedge var_7 > 0.0727} \\ R_8 = & var_{15} < var_{22} & \wedge \quad var_{10} < var_{17} \wedge var_7 > 0.0727 \\ R_9 = & \underbrace{var_{15} < var_{20}}_{\substack{\text{Different} \\ \text{condition}}} & \wedge \quad var_{10} < var_{17} \wedge var_7 > 0.0727 \end{array}$$

Let R_c be the rule that is formed by the common conditions in R_7, R_8 and R_9 thus $R_c = \{var_{10} < var_{17} \wedge var_7 > 0.0727\}$. A new evaluation was performed using R_c, the result was *True Positive* =4, *False Positive*=15, *False Negative*=11 and *True Negative*=370. It means that rules R_7, R_8 and R_9 are more specialized than R_c, because R_c classifies 15 false alarms.

Given that var_{15} is involved in *Different condition* in R_7, R_8, R_9 it is important to verify if the other variables (var_{22} and var_{20}) are correlated and if var_4 is inversely correlated to var_{22} and/or var_{20}. As can be seen from Table 7.10, the indicators are: var_4 =PTRB-50, var_{20}= LDNIB3MMA-50 and var_{22}=UK01Y00MA-50.

As can be seen, the decision rules produced by EDR can be understood by the user. However, the understanding of those rules depends on the expertise of the user to interpret the variables or attribute that were used to train the classifier. In this particular case the user has to be able to understand the financial indicators that were used to predict future movement in the stock prices. Thus, the financial and technical analysis knowledge of the user is crucial to understand the conditions in the rules. Other domains will require that users are familiar with the variables that are modeling the classifier. Now, let us to analyse the following set of rules:

$S_b = \{R_{11}, R_{14}, R_{15}, R_{17}\}$.

Obviously S_b holds a set of different rules. However, it is important to analyse the variables and relations that are involved in each rule in order to determine if those rules could be correlated. As can be seen from Table 7.8, R_{17} is different from the other rules in S_b. Because R_{17} does not have any equal hard condition or similar condition with another rule in S_b. On the other hand, R_{11}, R_{14} and R_{15} have in common the conditions: $var_7 < 0.0727$ and $var_{12} > Threshold.$ where $Threshold \in [-1185, -1082]$

$$R_{17} = \quad var_1 < -0.072 \wedge var_5 > -0.0445$$

$$\begin{array}{lllll} & & & & \overbrace{\qquad\qquad\qquad\qquad}^{\text{Common conditions}} \\ R_{11} = & var_8 < var_{13} & \wedge & \overbrace{var_7 > 0.0727 \wedge var_{12} > -1082.} \\ R_{14} = & var_{13} < -0.160 & \wedge & var_7 > 0.0727 \wedge var_{12} > -1082. \\ R_{15} = & \underbrace{var_4 < -0.213}_{\substack{\text{Different} \\ \text{condition}}} & \wedge & var_7 > 0.0727 \wedge var_{12} > -1185. \end{array}$$

Let R_d be the rule that is formed by the common conditions in R_{11}, R_{14}, and R_{15} thus $R_d = \{var_7 > 0.0727 \wedge var_{12} > -1185.\}$. A new evaluation was performed using R_d, the result was *True Positive* =3, *False Positive*=11,*False negative*=12 and *True Negative*=374. It means that R_{11}, R_{14} and R_{15} are more specialized than R_d because these do not classify any false alarm as R_d does. R_{11} differs because of the condition $var_8 < var_{13}$ where price volatility over 50 days is more than the moving average of 10 days of the momentum indicator of 10 days. The fact that R_{11} and R_{14} classify the same instances in the testing data set could suggest a correlation between them. Finally, it is important to verify if var_4 and var_{13} are correlated. As can be seen in

Table 7.10, the indicators are: var_4 =PTRB-50, var_{12}= MOM-10MA-10.

There is a big variety of decision trees that can be formed with a subset of rules in Table 7.8 that does not classify any positive case in the testing data set, for example:

$T_a = \{R_2, R_4, R_{17}\}$

$T_b = \{R_1, R_5, R_{15}\}$

$T_c = \{R_3, R_4, R_{11}\}$

As can be seen in Table 7.9, T_a, T_b, T_c are not able to classify any positive case in the testing data set. However, if we evaluate the fitness of T_a, T_b, T_c using the training data set, the performance is equal to the complete set of rules in Table 7.8, because every rule has precision =1. It means that the classification in the training data set for: T_a= T_b= T_c = {TP=8,FP=0,FN=5 and TN=387}. However, in the testing data set, T_a, T_b, T_c do not classify any positive example. It was shown that a bigger set of rules classifies more positive cases. It may be due to the fact that every positive instance could be classified using more than one pattern, as was shown in the example. However, it is difficult to predict which of the patterns are going to be repeated in future data sets.

Our analysis shows that in imbalanced data sets the reduction of rules could decrease the number of positive detections (it applies to the negative ones too). For that reason, we collect all the available patterns in the data set. Obviously the increase in recall causes a decrease in precision. However, the decision to increase the false alarms in order to improve the recall is finally made by the user.

Table 7.8: Set of rules for the example 1

Rule	Rule Description	Detections
R_1	$var_9 > var_{15}$ and $var_{10} < var_{17}$ and $var_7 > 0.0727$	4
R_2	$var_3 > var_{24}$ and $var_6 < var_7$ and $var_{21} > var_{24}$	4
R_3	$var_3 > var_{20}$ and $var_{13} < -527.9$	4
R_4	$var_3 > var_{18}$ and $var_6 > var_{13}$ and $var_9 < var_{18}$ and $var_{21} > var_{24}$	4
R_5	$var_3 > var_{24}$ and $var_6 < 0.0413$	4
R_6	$var_3 > var_{18}$ and $var_{11} < -0.5056$	4
R_7	$var_4 > var_{15}$ and $var_{10} < var_{17}$ and $var_7 > 0.0727$	4
R_8	$var_{10} < var_{17}$ and $var_{15} < var_{22}$ and $var_7 > 0.0727$	4
R_9	$var_{10} < var_{17}$ and $var_{15} < var_{20}$ and $var_7 > 0.0727$	4
R_{10}	$var_1 < var_{21}$ and $var_2 < var_{21}$ and $var_3 > var_{21}$ and $var_{24} < 0.0136$	3
R_{11}	$var_8 > var_{13}$ and $var_7 > 0.0727$ and $var_{12} > -1082.$	3
R_{12}	$var_{10} < var_{20}$ and $var_{15} < var_{23}$ and $var_7 > 0.0727$	3
R_{13}	$var_{10} < var_{21}$ and $var_6 < 0.0936$ and $var_{22} < -0.067$	3
R_{14}	$var_7 > 0.0727$ and $var_{12} > -1082.$ and $var_{13} < -0.160$	3
R_{15}	$var_4 < -0.213$ and $var_7 > 0.0727$ and $var_{12} > -1185.$	3
R_{16}	$var_{10} < var_{20}$ and $var_{15} < var_{20}$ and $var_7 > 0.0727$	3
R_{17}	$var_1 < -0.072$ and $var_5 > 0.0445$	3

There are many classifier systems based on GPs which claim to evolve a set of rules. These calculate the fitness of the individual by measuring the result of the classification and the "simplicity" of the solutions, for example [11], [12], [13], [59]. Other works just divide the resulting decision tree into rules, for example [85]. In those cases the GP is favouring the shortest solutions as T_a, T_b, T_c instead of a bigger tree that hold more rules. On the other hand, Yin et al [122] created a set of rules using GP discarding rules in order to find the minimal set of rules. That procedure was implemented to reduce the bloat in the evolutionary process.

Table 7.9: Positive instances classified, where e_i is a positive instance correctly classified in the training data set and e'_i is a positive instance correctly predicted in the training data set

Training data set									
Rule	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	Sum
R_1		X	X			X	X		4
R_2				X	X		X	X	4
R_3			X	X	X			X	4
R_4					X	X	X	X	4
R_5				X	X		X	X	4
R_6		X	X			X	X		4
R_7		X	X			X	X		4
R_8		X	X			X	X		4
R_9		X	X			X	X		4
R_{10}		X	X		X				3
R_{11}	X	X	X						3
R_{12}		X	X			X			3
R_{13}			X			X	X		3
R_{14}	X	X	X						3
R_{15}	X	X	X						3
R_{16}		X	X			X			3
R_{17}	X	X	X						3
Sum	4	12	14	3	5	9	9	4	

Testing data set				
Rule	e'_1	e'_2	e'_3	Sum
R_1				0
R_2				0
R_3				0
R_4				0
R_5				0
R_6		X		1
R_7				0
R_8				0
R_9				0
R_{10}				0
R_{11}	X			1
R_{12}				0
R_{13}			X	1
R_{14}	X			1
R_{15}				0
R_{16}				0
R_{17}				0
Sum	2	1	1	

Table 7.10: List of independent variables

$var_1 =$	Price moving average 12 days
$var_2 =$	Price moving average 12 days
$var_3 =$	Price Trading breaking rule 5 days
$var_4 =$	Price Trading breaking rule 50 days
$var_5 =$	Filter rule 5 days
$var_6 =$	Filter rule 63 days
$var_7 =$	Price volatility 12 days
$var_8 =$	Price volatility 50 days
$var_9 =$	Volume moving average 10 days
$var_{10} =$	Volume moving average 60 days
$var_{11} =$	Momentum indicator 10 days
$var_{12} =$	Momentum indicator 60 days
$var_{13} =$	Momentum 10 days moving average 10 days
$var_{14} =$	Momentum 60 days moving average 60 days
$var_{15} =$	Generalized Momentum indicator 10 days
$var_{16} =$	Generalized Momentum indicator 60 days
$var_{17} =$	FOOTSIE moving average 12 days
$var_{18} =$	FOOTSIE moving average 50 days
$var_{19} =$	LIBOR: 3 months moving average 12 days
$var_{20} =$	LIBOR: 3 months moving average 50 days
$var_{21} =$	UK01Y00 moving average 12 days
$var_{22} =$	UK01Y00MA moving average 50 days
$var_{23} =$	UK10Y00MA moving average 12 days
$var_{24} =$	UK10Y00MA moving average 50 days

7.4 Conclusions

In this chapter, we have presented a new approach, which we have called *Evolving Decision Rules* (EDR). This method was designed to classify the minority class in imbalanced data sets. The system's output is a set of decision rules, which based on a threshold τ produces a range of classifications to suit the investor's preferences. For a detail analysis, we have used the Receiver Operating Characteristic (ROC) curve, which helps to visualize the distribution of the classifications in the ROC space. In the same vein, we have used the Area Under the ROC curve (AUC) to measure the general performance of our approach and to compare this with Repository Method proposed in Chapter 6.

The core of our approach is based on GP, which is aided by a repository of rules. The aim of this repository is to collect useful patterns that are used to produce the following population in the evolutionary process. The main operators of EDR are mutation and hill-climbing, these produce offspring of the collected patterns. Furthermore a simplification process is used to simplify the rules in the repository in order to produce more comprehensible solutions. On the other hand, the removal of extra-code, allows a reduction in the computational effort.

The results obtained in this chapter show that EDR classifies more positive cases (minority class) than RM (section 7.3.1), EDDIE-Arb (section 7.3.2) and C5.0 (section 7.3.3, 7.3.3) in imbalanced environments. From experimental results we conclude that the first goal of this method has been achieved.

From experimental results (see sections 7.3.1, 7.3.2, 7.3.3, 7.3.3) we noticed that

EDR produces a series of classifications able to be adapted to the user's needs (from conservative to liberal predictions). From experimental results we observed that the second goal of EDR was accomplished.

Finally, an illustrative example is analysed in section 7.3.5 in order to 1) explain how a bigger collection of rules is able to classify more positive cases in imbalanced environments and 2) show the comprehensibility of the solutions provided by EDR. It was shown that a bigger set of rules has more chance of classifying the positive cases. Obviously the increase in the recall causes a decrease in precision. However, the decision to increase the false alarms in order to improve the recall is finally made by the user.

As can be observed from this example, EDR produces comprehensible rules that can be analysed by the user in order to understand the conditions and variables in the rule. Thus, users can combine their knowledge in order to make a more informed decision. The example discloses the understandability of the solutions proposed by EDR, thus we assume that the third goal of this method has been achieved.

Chapter 8

Scenario Method

The present chapter introduces a new approach, which is named Scenario Method (SM) [44]. The aim of this method is to prune decision trees produced by a GP system in order to:

1. improve the accuracy and precision of the classification

2. simplify the decision trees

This chapter is organized as follows: Section 8.1 presents the introduction and motivation of this approach, while section 8.2 describes the procedures of SM. Next, section 8.3 presents the experiments to test our approach. Finally, section 8.4 summarizes the conclusions.

8.1 Introduction and motivation

Decision trees have been widely used in Machine Learning (ML) for classification and prediction. However, the over-fitting and complexity of the resulting trees have shown the necessity for pruning procedures. Several classifiers have incorporated pruning

methods, for example: Classifier CART implemented *minimal cost-complexity pruning* [15], ID3 incorporated *reduced error pruning* and *pessimistic pruning* [97], while classifier C4.5 introduced *Error-based pruning* [98]. Breiman [15] and Quinlan [97] have asserted that tree simplification can benefit almost all decision trees when removing parts that do not contribute to the classification task. They argued that the resulting trees are less complex and more comprehensible, furthermore, the simplification helps to avoid the over-fitting.

Decision trees generated by GP [69] tend to grow [2, 88, 104, 74], many times this growth is not proportional to the quality of the solution (see section 3.3.3). To control the code growth, a variety of methods have been introduced, these are grouped in three main types: parsimony pressure, operator modification and code modification [105]. Parsimony pressure tries to evolve small solutions penalizing large individuals. Operator modification is represented by non-destructive crossover [103], whose objective is the preservation of the building blocks (see section 3.3.2) as the brood recombination method [1]. Code modification involves changing the structure of the code during or after the evolution, this technique has not been explored in depth since it uses more computational resources [103]. However, other ML techniques [15, 98] use pruning because it produces more exploration. We believe that decision tree simplification can be beneficial to trees produced by GP. But it is important to point out that the pruning procedures for statistical methods are not suitable for pruning GP

[1]The brood method is inspired by some natural species, which produce far more children than are expecting to live but just the fittest offsprings will survive. Selecting the best children ensures that the good blocks will be preserved [107]

decision trees and viceversa. Because GP decision trees are builded in different way and the pruning causes different effects as it is described in section 3.3.3. In our understanding the only method to prune decision trees produced by a GP system, before the submission of this thesis, is the approach proposed by Eggermont [33] (see section 3.3.3).

8.2 Scenario Method description

The aim of this approach is to simplify decision trees and improve their precision and accuracy. An analysis, based on hypothetical scenarios, is performed in order to identify the parts of the tree that contribute to the classification task. Those fragments that reduce the tree performance are removed. Let us introduce an overview of the main steps of SM:

1. *Class division*: to create decision trees that classify a single class.

2. *Rule extraction*: to identify every rule R_i in the decision tree T.

3. *Rule evaluation*: to evaluate individually every rule R_i.

4. *Rule selection*: to determine if the rule R_i is contributing to the classification task. If R_i is not good enough, R_i will be deleted at the next step.

5. *Tree pruning*: to remove the nodes that are involved in the rule, without affecting the other rules in the tree.

The next sections explain every procedure, but some detailed explanations are provided in Chapter 5.

8.2.1 Class division

To divide the classification problem, a population per each class is evolved independently. The class division has been used previously by [12], [12], [29]. For this purpose, decision trees are generated using Discriminator Grammar (DG), this grammar produces trees that classify or not a single class (see section 5.4).

8.2.2 Rule extraction

This process analyzes every decision tree T in the population in order to define its rules. The rule R_k can be expressed as a conjunction of conditions such as $R_k = (c_{k1} \wedge c_{k2} \wedge \cdots c_{kt})$ where kt is the number of conditions in rule R_k. Thus, to satisfy R_k, every condition in this rule has to be satisfied. Given that, our goal is to remove a specific rule without affecting the remaining ones, we need an structure to control the rules and their nodes.

Let *Rule Map* be a matrix that lists the decision rules of a tree T, as well as the conditions of each rule in T.

If $T = \{R_1 \vee R2 \vee \ldots\}$, where R_i is a rule. Then the k^{th}-row in the rule map is composed of the conditions in R_k. Given that the size of rules could be different, the size (number of nodes) of the matrix will be N x L, where N is the number of rules in the tree, and L is the size of the largest rule. When the length of a rule is smaller than

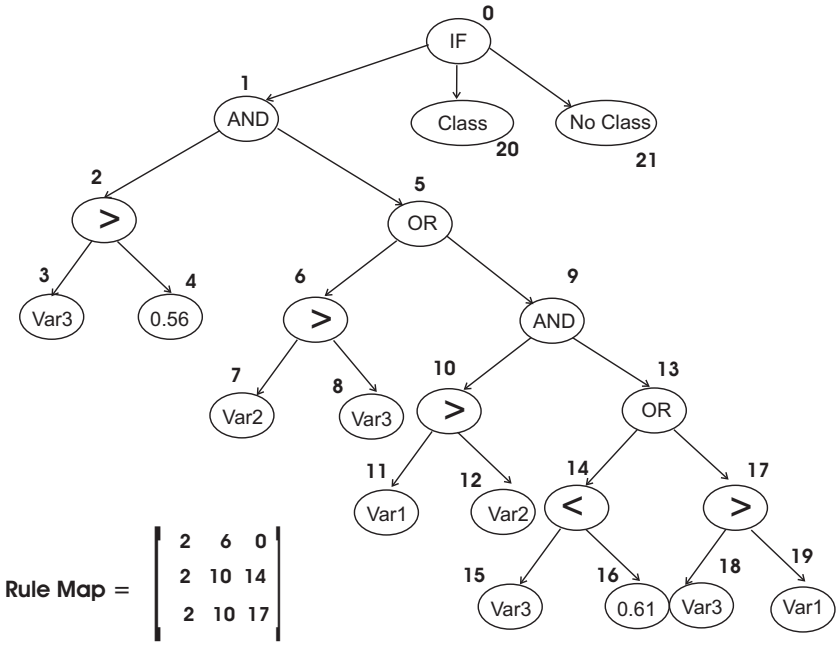


Figure 8.1: A decision tree and its rule map

L the empty spaces will be filled using 0. Figure 8.1 shows an example of a decision tree and its rule map. The conditions are represented by the number of the node in the operation (see section 5.4). A more detailed explanation about rule extraction is provided in section 5.5.

8.2.3 Rule evaluation

This section explains the procedure to evaluate the performance of each rule in the decision tree T. Every rule $R_i \in T$ is evaluated using the training data set, the result is registered in a confusion matrix (see section 2.4.2). Thus, there will be a confusion matrix M_i for each rule $R_i \in T$.

It is important to keep in mind that SM could be applied to prune trees that classify imbalanced data sets. According to Kubat *et al.* [71], in imbalanced environments, it

is not reliable to measure the performance of a classifier only by using the equation of accuracy. Let 8.2.1 be the equation to measure the rule contribution.

$$Ev(R_i) = \begin{cases} \left(\frac{TP_i}{TP_i+FN_i}\right)\left(\frac{TP_i-FP_i}{TP_i+FP_i}\right) & if \quad TP_i + FP_i > 0 \\ 0 & Otherwise \end{cases} \tag{8.2.1}$$

In previous chapters (6 and 7) the fitness function was measured by the recall multiplied by the precision. The mentioned formula was used because the methods that are proposed in those chapters were designed to detect the minority class in imbalanced data sets whose level of imbalanced is high (see section 2.4.2). On the other hand, the main objective of SM is to prune decision trees, and the data set can be or not imbalanced. It means that SM is not focused just on imbalanced data sets. Let us analyse the equation 8.2.1, as can be seen the first parenthesis encloses the recall and the second parenthesis encloses an expression similar to precision, but it penalizes the false positive cases. As can realise the change is basically the penalization of positive cases by subtracting the number of false positive to the number of true positives. The objective is to encourage the increase of the true positive cases and the decrease of the false positive cases. Notice that, $TP_i + FN_i$ is a constant number for every R_i because it depends on the data set.

8.2.4 Rule selection

Once the rules in tree T have been evaluated, the next step is to perform a rule selection based on hypothetical scenarios of the conjunction of rules. Let R_β be the rule with the highest evaluation $Ev()$, thus R_β is taken as a starting point. To disclose the potential

of the remaining rules $R_\eta \in T$, the *Best* and the *Worst scenario* for $R_{\beta\eta} = (R_\beta \vee R_\eta)$ are calculated, note that $R_{\beta\eta}$ is the disjunction of rules R_β and R_η. The best scenario is calculated assuming that the true positives cases in R_β and R_η are not overlapped and the false positive cases maximally overlap. Thus, the best scenario for the true positive and false positive cases are calculated as follows: ${TP_{\beta\eta}}^+ = TP_\beta + TP_\eta$ and ${FP_{\beta\eta}}^+ = max(FP_\beta, FP_\eta)$. Superscripts are used to indicate the scenario, it could be worst (-) or best (+), subscripts are used to denote rules. The worst scenario is calculated in exactly the opposite way, it assumes that true positive cases maximally overlap and false positive cases do not overlap. Thus, the worst scenario for true positive and false positive cases is ${TP_{\beta\eta}}^- = max(TP_\beta, TP_\eta)$ and ${FP_{\beta\eta}}^- = FP_\beta + FP_\eta$. Once the best and the worst scenarios are calculated, Formula 8.2.1 is applied to them as follows:

$$Ev({R_{\beta\eta}}^+) = \frac{TP_\beta + TP_\eta}{TP_\eta + FN_\eta} \cdot \frac{TP_\beta + TP_\eta - max(FP_\beta, FP_\eta)}{TP_\beta + TP_\eta + max(FP_\beta, FP_\eta)}$$

$$Ev({R_{\beta\eta}}^-) = \frac{max(TP_\beta, TP_\eta)}{TP_\eta + FN_\eta} \cdot \frac{max(TP_\beta, TP_\eta) - FP_\beta - FP_\eta}{max(TP_\beta, TP_\eta) + FP_\beta + FP_\eta}$$

Let us define the Potential of Improvement (PI) of a rule R_η as the capacity of R_η to improve the tree $T' = \{R_\beta\}$. Notice that T' is the tree that holds just the rule R_β, which is the best rule of the tree T. The potential of improvement pretends to measure the possibility that the performance of $R_\beta \vee R_\eta$ will be better than the rule R_β. We propose to measure this by calculating the worst and the best scenario of $R_{\beta\eta} = R_\beta \vee R_\eta$ and evaluating each of them. Thus, PI is calculated using the distance between the evaluation of R_β and the evaluation of the best scenario of $R_\beta \vee R_\eta$ as

Figure 8.2 shows. To normalize the measure, this is divided into the distance from the worst and the best scenario of $R_\beta \vee R_\eta$.

$$PI(R_\eta) = \begin{cases} \frac{Ev({R_{\beta\eta}}^+) - Ev(R_\beta)}{Ev({R_{\beta\eta}}^+) - Ev({R_{\beta\eta}}^-)} & if Ev(R_\beta) \leq Ev({R_{\beta\eta}}^+) \\ 0 & Otherwise \end{cases}$$

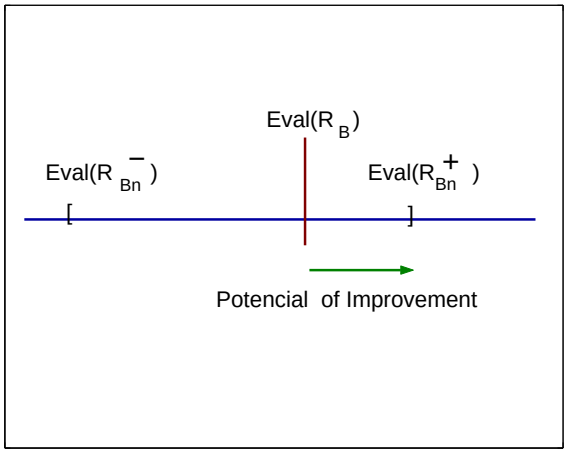


Figure 8.2: Interval of the worst and the best scenario of $R_{\beta\eta} = (R_\beta \vee R_\eta)$

Once the potential of improvement is calculated, it is necessary to decide whether or not the rule R_η is beneficial to the tree. A threshold from 0 to 100% is used to determine the level of pruning, this will be defined as the *Pruning Threshold* θ. If $(PI(R_\eta) < \theta)$ the rule R_η will be pruned. When θ is close to 0 the level of pruning is low, but when θ is close to 100% a hard pruning is performed.

At this point, an important question arises: why is the SM procedure preferred rather than the direct evaluation of the combined rule $R_{\beta\eta}$?. The reasons are the following: a) the direct evaluation of the new rule $R_{\beta\eta}$ consumes more computational resources and b) direct evaluation does not disclose the individual performance of the rule. To illustrate the last point, we present an example showing that the direct

Table 8.1: Example, where TP_β=40 and $FP_\beta = 20$

	R_η (TP_η, FP_η)	$R_\beta \cap R_\eta$	$R_\beta \vee R_\eta$ $(TP_{\beta\eta}, FP_{\beta\eta})$
$R_{\eta=1}$	$(1, 15)$	$(0, 15)$	$(41, 20)$
$R_{\eta=2}$	$(10, 2)$	$(10, 1)$	$(40, 21)$

evaluation of $(R_{\beta\eta} \vee R_\beta)$ does not give a good performance estimation of R_η. Imagine that the evaluation of R_β = (TP$_\beta$, FP$_\beta$) = (40,20). Next, we add the rules R_1 and R_2 from Table 8.1. The performance of R_1 = (TP=1,FP=15). As can be observed, R_1 produces more misclassifications than accurate predictions. However, direct evaluation suggests that R_1 is able to improve the tree (see Table 8.1). In contrast, SM discards this rule if we apply a low pruning threshold (greater than 7%). Next, let us analyze R_2, whose performance indicates that it is able to classify with a good rate of precision (83%). Nevertheless, direct evaluation discards R_2 because it classifies the same true positive cases that R_β (see Table 8.1). However, the fact that R_2 and R_η classify the same subset of examples in the training data set does not mean that those rules classify the same cases in other data sets (see example in section 7.3.5). SM only discards R_2 when a hard pruning threshold is applied (bigger than 82%).

8.2.5 Tree pruning

After the rule selection process is performed, those rules that do not achieve the expected θ are removed. During the pruning procedure, the condition map is used to detect the relations between rules and determine those nodes in bad rules that can be pruned without affecting the useful rules. The pruning pseudo-code is described in Figure 8.3. Notice that, the pruning procedure cannot be applied to all decision trees.

Thus, it is not possible to prune the tree in the following cases:

1. The tree is composed of a single rule.
2. SM does not suggest pruning to improve the tree.
3. SM suggests pruning but all conditions to prune are involved in good rules.
4. The evaluation of the best rule is inferior to zero, it occurs when the number of false positives exceeds the number of true positives.

Another way to eliminate the bad rules is to divide the tree into rules and just delete the unwanted ones. However, the method proposed in this chapter could be used to prune decision trees that later can be reintroduced in the evolutionary process.

8.3 Experimental section

To test our approach a series of experiments was performed. The first experiment analyses the number of decision trees that can be pruned using SM (see section 8.3.1). Next experiment (section 8.3.2), was designed to analyse a) the effects of SM on the performance of the decision trees, b) the impact of parameter θ and c) the effect of the SM through generations. Finally, the experiment in section 8.3.3 was designed to analyse the effect of SM on the size of the decision trees.

The data set that was used to train the GP system in the experiment came from the London stock market. The training data set contains 757 records (324 positive cases) that describe the behavior of the closing price of TESCO stock. The testing data set

```
PROCEDURE Prune( T, R_k , ConditionMap )
BEGIN
   /*Given the rule R_k =(n_k1 ∩ n_k2...∩n_kj...)
    where n_kj is a conditional node and R_k is
    the k^th-row in ConditionMap*/

     FOR each n_kj ∈ R_k
      IF (n_kj ∉ R_i where i ≠ k) THEN
          /* If n_kj is not part of other rule, delete it*/
         BEGIN
            n_p → Parent of node n_kj
            n_b →The other child of node n_p
            n_g →Parent of node n_p
            n_c1, n_c2 → The two children of node n_kj
            /* Delete n_kj, its parent and its children*/
            T → T − n_kj, n_p, n_c1, n_c2
            T → T  + Link between n_g and n_b
         END
       ConditionMap_kj → 0 /* Set 0 in node map */
     Return T;
END
```

Figure 8.3: Pruning Psedocode

is composed of 262 records (85 positive cases). In both cases we look for an increase in the closing price of at least 3% in ten days. The independent variables of the data sets are described in section 5.3.

Creation of populations

To test our approach in different stages of the evolutionary process, we generate and save populations from different points in the evolution. By doing so, a population of 1,000 individuals was created using DG, it was evolved during 100 generations. Every twenty generations the complete population was saved, therefore the result was five populations of 1,000 individuals each, let us call them $P_{20}, P_{40}, \cdots, P_{100}$. Subscripts

indicate the number of the generation.

Table 8.2 presents the GP parameters used to evolve the populations. These were selected by means of a series of preliminary experiments.

Table 8.2: Summary of Parameters.

Parameter	Value
Population size	1,000
Initialization method	Growth
Generations	100
Crossover Rate	0.8
Mutation Rate	0.05
Selection	Tournament (size 2)
Elitism	Size 1
Fitness function	Equation (8.2.1)
Control bloat growing	50% of those trees whose largest branch exceed 7 were penalized with 20% of the fitness for each node that surpassed the largest branch allowed.

8.3.1 Experiment: Number of pruned decision trees

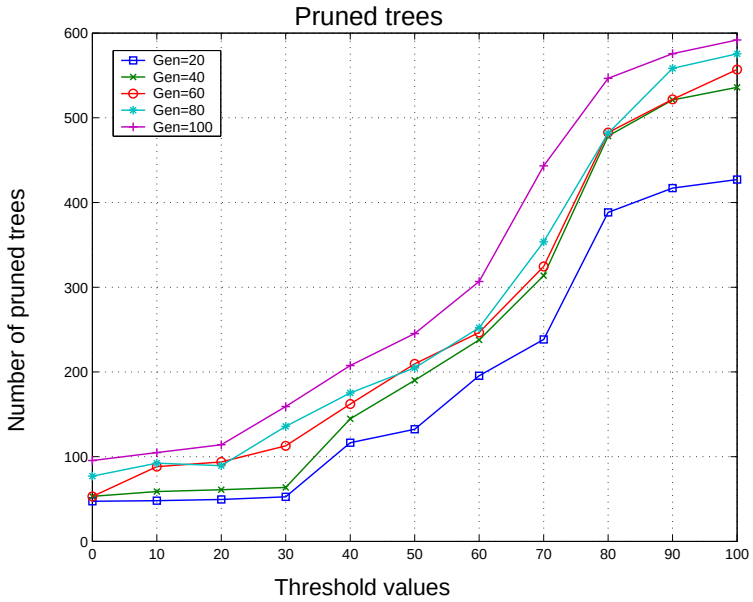


Figure 8.4: Pruned trees

Objective

The objective of this experiment is to analyse the number of decision trees that were pruned using SM. Specifically, we analysed the effects under the following conditions:

1. The effect of SM using different θ thresholds

2. The impact of SM when it is applied in different stages of the evolutionary process.

Procedure

SM was tested on a series of 25 runs, every series comprised five populations from different stages of the evolutionary process. In order to discover the impact of θ, the experiment was tested using different values for this parameter. The results of the

experiment were grouped and averaged by generation and pruning threshold.

Observations

Figure 8.4 plots the number of trees that were pruned by SM, every series represents a population in a specific generation $P_{20}, P_{40}, ..P_{100}$. As can be seen in X-axis, every population was tested using different pruning thresholds.

Not surprisingly, these results show that the increase in pruned trees is related to the number of generations. The number of pruned trees grows when the number of generations increases, this occurs because the tree size rises and there are more opportunities to perform a pruning. During earlier generations, the number of pruned trees is low because in the beginning of the evolutionary process, the tree size is small and trees must contain more than one rule in order to be pruned. As an example, the average number of rules per tree in a population of generation twenty is 1.9. It means that there are a high number of trees that hold only one rule, as a consequence these can not be pruned.

On the other hand, as one might expect, the number of pruned trees increases when θ increases, because the selection of rules of rules become stricter and more rules have to be pruned. Table 8.3 displays the number of pruned trees per population and threshold.

Table 8.3: Number of pruned trees by scenario method

θ	Gen=20	Gen=40	Gen=60	Gen=80	Gen=100
0	47.3	53.1	53.0	77.0	95.4
10	48.1	58.8	88.2	92.3	104.8
20	49.4	60.9	93.8	89.3	114.0
30	52.6	63.7	112.8	135.7	159.1
40	116.4	144.7	162.2	175.2	207.5
50	132.4	190.1	209.6	204.9	245.3
60	195.5	237.8	246.5	252.1	306.7
70	238.3	313.7	324.5	353.6	443.3
80	388.4	478.5	482.5	481.7	546.5
90	417.0	521.0	521.7	558.4	575.7
100	427.1	535.9	556.9	575.5	591.9

8.3.2 Experiment: Effects of the pruning procedure

Objective

This experiment was designed to analyse the following effects:

1. The impact of SM in the performance (precision, accuracy and recall) of the decision trees.

2. The effect of the θ parameter.

3. The effect of the pruning procedure through generations.

Procedure

In order to discover the impact of θ, the experiment was tested using different values for this parameter (θ = 0%,10%,..90%). To analyse the effect of SM through generations, SM was used in different populations from different points in the evolution. All figures given in this section denote average results from a series of 25 test runs.

Observations

Let us analyse the precision, accuracy and recall obtained by SM, using different θ values and populations.

Precision improvement- Table 8.4 describes the precision of a standard GP before and after SM was applied. As can be observed, the precision was improved in all cases. The average precision improvement is 9%, while the minimum improvement is 2% and the maximum is 18%. As can be seen from Table 8.4, the improvement precision is affected by the generation of the population and the θ value. The precision improvement declines when θ is close to 100%, or when the population starts to converge. When θ is close to 1 a hard pruning is performed. Thus, SM prunes many rules, some of them could be useful. On the other hand, the precision improvement tends to decrease when the evolutionary process advances. Because the decision trees are more evolved. In contrast at the beginning of the evolutionary process, most of the trees are not too far from being random. Thus the analysis of their components and the pruning of bad rules helps them to improve their precision.

Accuracy improvement- Figure 8.6 displays the accuracy improvement achieved by SM. Every curve represents a population tested with different pruning thresholds. As can be seen, SM helped to improve the accuracy in the majority of the cases. The average improvement in accuracy is 4.6%. The best results are obtained when PI is less than 60%. However, the number of pruned trees decreases when threshold does

the same. According to the experiment results, the best thresholds are between 40% and 60%. In this range the accuracy improvement and the number of pruned trees is high. The results of some experiments disclosed that it is possible to have a slight decrease in the accuracy when the threshold is close to 100% and the population has converged. This is because, when the PI is big the selection becomes stricter and some useful rules could be pruned. Table 8.5 shows the accuracy of a standard GP and the new accuracy when SM is used.

Recall decrease- As can be observed from Table 8.6, the recall has decreased in all cases. As it was mentioned previously the precision increases using SM. However, it is paid for a decrease in the recall. It means that the prediction is becoming conservative. As can be seen in Figure 8.8, the recall tends to decrease more when the θ increases. However, after SM has removed an important part of the rules, it tend to be stable. It may be due to the majority of the rules that help to improve the recall has been pruned.

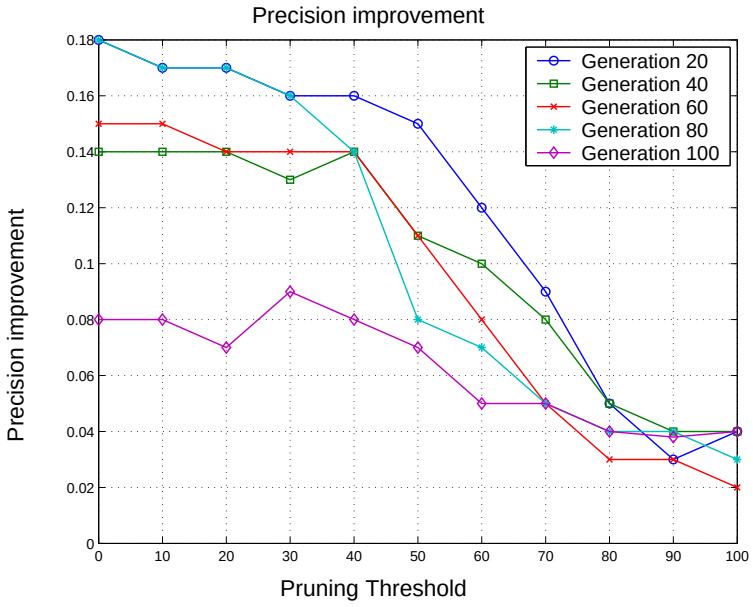


Figure 8.5: Precision improvement

Table 8.4: (a) Precision before SM, (b) Precision after SM

θ	Gen=20		Gen=40		Gen=60		Gen=80		Gen=100	
	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
0.00	0.41	0.59	0.43	0.57	0.47	0.62	0.46	0.64	0.50	0.58
10	0.42	0.59	0.46	0.60	0.47	0.62	0.46	0.63	0.51	0.59
20	0.42	0.59	0.46	0.60	0.48	0.62	0.47	0.64	0.51	0.58
30	0.43	0.59	0.47	0.60	0.49	0.63	0.53	0.69	0.52	0.61
40	0.44	0.60	0.47	0.61	0.50	0.64	0.55	0.69	0.54	0.62
50	0.46	0.61	0.51	0.62	0.54	0.65	0.59	0.67	0.55	0.62
60	0.54	0.66	0.57	0.67	0.58	0.66	0.60	0.67	0.59	0.64
70	0.61	0.70	0.62	0.70	0.64	0.69	0.64	0.69	0.66	0.71
80	0.69	0.74	0.69	0.74	0.71	0.74	0.70	0.74	0.71	0.75
90	0.72	0.75	0.72	0.76	0.74	0.77	0.72	0.76	0.74	0.76
100	0.72	0.76	0.72	0.76	0.75	0.77	0.73	0.76	0.74	0.78

(a) Precision before SM, (b) Precision after SM

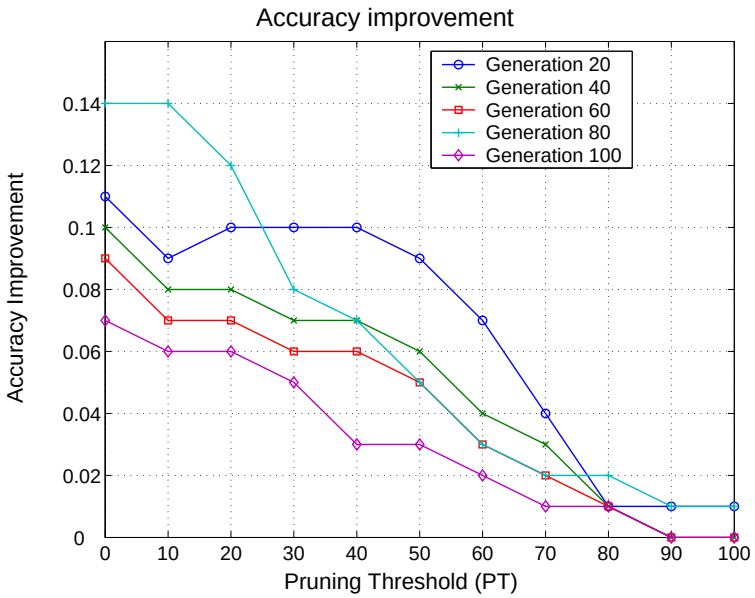


Figure 8.6: Accuracy improvement

Table 8.5: Accuracy before and after scenario method was applied.

θ	Gen=20		Gen=40		Gen=60		Gen=80		Gen=100	
	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
0	0.54	0.65	0.56	0.66	0.57	0.66	0.53	0.67	0.58	0.65
10	0.55	0.64	0.57	0.65	0.58	0.65	0.53	0.67	0.59	0.65
20	0.54	0.64	0.57	0.65	0.58	0.65	0.54	0.66	0.59	0.65
30	0.55	0.65	0.58	0.65	0.59	0.65	0.60	0.68	0.61	0.66
40	0.55	0.65	0.59	0.66	0.59	0.65	0.61	0.68	0.62	0.65
50	0.56	0.65	0.60	0.66	0.61	0.66	0.63	0.68	0.62	0.65
60	0.59	0.66	0.63	0.67	0.64	0.67	0.64	0.67	0.64	0.66
70	0.63	0.67	0.65	0.68	0.66	0.68	0.66	0.68	0.67	0.68
80	0.67	0.68	0.67	0.68	0.68	0.69	0.67	0.69	0.68	0.69
90	0.67	0.68	0.68	0.68	0.69	0.69	0.68	0.69	0.69	0.69
100	0.67	0.68	0.68	0.68	0.69	0.69	0.68	0.69	0.69	0.69

(a) Accuracy before SM, (b) Accuracy after SM

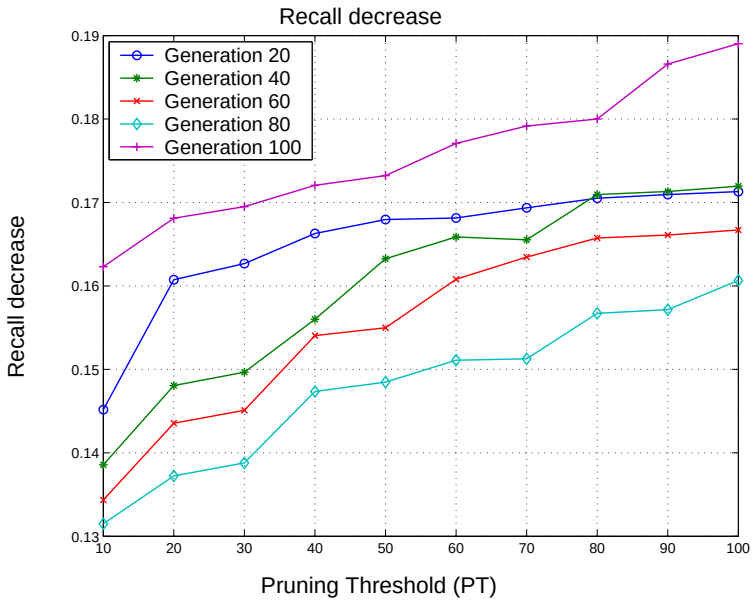


Figure 8.7: Recall improvement

Table 8.6: Recall before and after scenario method was applied.

θ	Gen=20		Gen=40		Gen=70		Gen=80		Gen=100	
	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
10	0.44	0.29	0.42	0.28	0.41	0.27	0.40	0.27	0.39	0.22
20	0.38	0.22	0.34	0.20	0.33	0.19	0.33	0.19	0.34	0.17
30	0.38	0.22	0.34	0.19	0.33	0.18	0.32	0.18	0.33	0.16
40	0.37	0.20	0.36	0.21	0.35	0.20	0.34	0.20	0.32	0.15
50	0.37	0.20	0.34	0.18	0.33	0.17	0.32	0.17	0.32	0.15
60	0.34	0.17	0.33	0.17	0.32	0.16	0.31	0.16	0.31	0.13
70	0.31	0.14	0.31	0.14	0.30	0.13	0.29	0.14	0.29	0.11
80	0.30	0.13	0.29	0.12	0.28	0.12	0.27	0.12	0.27	0.08
90	0.29	0.12	0.29	0.11	0.28	0.11	0.27	0.11	0.26	0.08
100	0.29	0.12	0.26	0.09	0.26	0.09	0.25	0.09	0.26	0.07

8.3.3 Experiment: Analysis of the tree size reduction

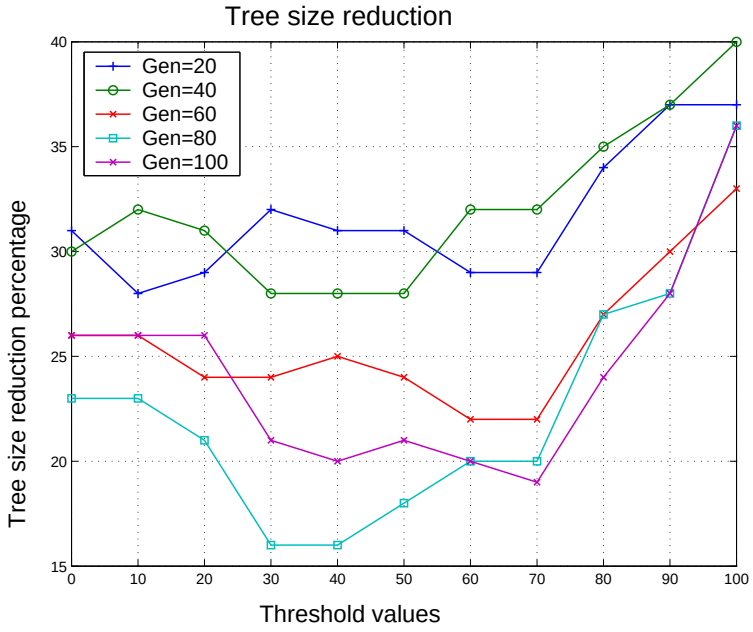


Figure 8.8: Tree size reduction

Objective

The objective of this experiment is to show the reduction in the tree size after SM was applied. In particular, we analyse the following effects:

1. The effect of SM using different θ thresholds

2. The impact of SM when it is used in different stages of the evolutionary process.

Procedure

In order to discover the impact of θ, the experiment was tested using different pruning thresholds (θ = 0%,10%,..90%) and using populations from different stages of the

evolutionary process. The results of the experiment were grouped and averaged by generation and pruning threshold.

Observations

Table 8.7 shows the percentage of the tree size reduction for generation. As can be seen, SM reduces considerably the size of the trees, the average reduction is 27%. Figure 8.8 shows the tree size reduction achieved in the experiment.

Table 8.7: Tree size reduction produced by scenario method

θ	Gen=20	Gen=40	Gen=60	Gen=80	Gen=100
0	31%	30%	26%	23%	26%
10	28%	32%	26%	23%	26%
20	29%	31%	24%	21%	26%
30	32%	28%	24%	16%	21%
40	31%	28%	25%	16%	20%
50	31%	28%	24%	18%	21%
60	29%	32%	22%	20%	20%
70	29%	32%	22%	20%	19%
80	34%	35%	27%	27%	24%
90	37%	37%	30%	28%	28%
100	36%	40%	33%	36%	36%

8.4 Conclusions

This chapter presented a pruning method for decision trees, which is named *Scenario Method* (SM). This pruning procedure was designed to prune decision trees generated by a GP system. The aim of this method is to identify rules that enhance the classification task as well as those rules that reduce the performance of the tree. This approach is based on the analysis of scenarios. The intensity of pruning is controlled

by a pruning threshold.

The experimental results in section 8.3.1 shows the number of decision trees that can be pruned using SM. It was observed that decision trees in later generations are more likely to be pruned than trees in earlier generations.

The experimental results in section 8.3.2, suggest that SM is able to identify useful parts of the tree. The pruning of non useful conditions helps to improve the accuracy and precision of some decision trees. However, the user has to bear in mind that such improvement is paid for with loss of recall. The improvement achieved by SM varies, it depends on the stage of the evolutionary process and the pruning threshold. From experimental results, we conclude that the first objective of SM has been achieved.

The experiment in section 8.3.3 shows that SM is able to reduce the tree size. In other words, the decision tree has been simplified. Since the introns (see section 3.3.3) have been removed, then the decision tree can show the real variables and conditions in the prediction. Those results prove that the second objective of SM has been achieved.

Chapter 9

Conclusions and Future Research

In this chapter we present the conclusions of the thesis and the directions of future work. This chapter is organised as follows: first, our research is summarized in section 9.1. Next, a description of the contributions presented in this thesis is provided in section 9.2. Finally, directions for further research are proposed in section 9.3.

9.1 Research summary

The research goals of this thesis are: 1) to classify the minority class in imbalanced data sets, 2) to generate a range of classifications to suit different users needs, 3) to create comprehensible solutions that can be understood by the user and 4) to improve the accuracy and precision of decision trees produced by a GP system. To achieve our goals, we have proposed three different methods, let us briefly summarize the work that has been done in each of them.

In Chapter 6 a novel approach, which is named Repository Method (RM) was introduced. The aims of this method are: a) to predict the minority class in imbalanced data sets, b) to provide a range of classifications and c) to generate comprehensible

solutions. To achieve our first goal, we proposed to collect different patterns that identify the cases of the minority class. RM analysed a set of solutions (decision trees) provided by a GP system in order to select and collect patterns (rules). A Precision Threshold (PT) is used to select the rules, these will be collected in a structure that is called *repository*. In order to create a range of solutions (second goal), the RM was run several times using different values for the PT. The result was a set of repositories, where each of them produces a different classification. Finally, in order to achieve our third goal, a simplification process was used to eliminate the extra-code and disclose the variables and conditions in the decision. The performance of RM was compared to a standard GP, the classifier C5.0 and EDDIE-Arb (see sections 6.3.1, 6.3.3, 6.3.2 respectively). A series of experiments was carried out in order to find out the factors that work in favour of RM. Our method was tested using populations of decision trees from different stages of the evolutionary process, as can be observed in section 6.3.4. Other series of experiments was performed in order to analyse the impact of the number of decision trees involved in the rule search (see section 6.3.5).

Chapter 7 presents the second method of this thesis, which is called Evolving Decision Rules (EDR). As in the previous method, the goals of EDR are: a) to predict the minority class in imbalanced data sets, b) to provide a range of classifications and c) to produce comprehensible solutions. We have achieved our first goal by collecting patterns from a GP system. This method has a key factor, called *repository*, this is a structure that stores rules that have been selected for their performance and novelty.

The new generations will be created using the rules in the repository as parents. The final solution is a set of decision rules, these are used to form different subsets of rules according to their precision. This produces a set of classifications (second goal), whose results can be plotted in the Receiver Operating Characteristic (ROC) curve. A simplification process was used in order to achieve our third goal. EDR performance was compared to a standard GP, the classifier C5.0 and EDDIE-Arb (see sections 7.3.1, 7.3.3, 7.3.2 respectively). Finally, an example that illustrated a) how a set of rules can detect more positive cases and b) the comprehensibility of the solutions, is described in section 7.3.5.

The third method proposed in this thesis is called Scenario Method (SM). The aim of this method is to improve the precision and the accuracy of decisions trees produced by a GP system. This procedure analyses decision trees in order to remove parts of the trees that do not contribute to the classification task. The improvement in the precision and accuracy is paid with loss of recall. However, the users decide which of these metrics is more important to them. An experiment was performed in order to find out the number of decision trees that are favoured by this method (see section 8.3.1). Section 8.3.2 presents an experiment that was designed to measure the performance of the decision tree when SM procedure was applied. Finally, an experiment was run to measure the percentage of reduction in the decision tree size after SM was used 8.3.3.

9.2 Contributions

The research presented in this thesis contributes to the fields of machine learning, genetic programming, chance discovery and financial forecasting. The five major contributions in this thesis are:

Minority class detection

We have proposed two different methods, RM and EDR, to detect the minority class (positive cases) in imbalanced data sets by means of the collection of patterns. Experimental results (see sections 6.3.1, 6.3.2 and 6.3.3, respectively) showed that RM detected more positive cases than a standard GP, classifier C5.0 and EDDIE-Arb. On the other hand, from experimental results it was shown that EDR classifies more positive cases than RM , EDDIE-Arb and classifier C5.0 in imbalanced environments (see sections 7.3.1, 7.3.2, 7.3.3, 7.3.3 respectively). From those results we conclude that the first goal of this thesis has been achieved.

Produce a range of solutions

Both, RM and EDR have been designed to produce a range of solutions. These approaches collect and store multiple patterns (rules). Next, the resulting rules are grouped by precision, in order to generate a range of classifications. From experimental results (see sections 6.3.1, 6.3.2 and 6.3.3 and 7.3.1, 7.3.2, 7.3.3, 7.3.3), it was observed that RM and EDR offer a range of classifications to suit different users preferences (from conservative to liberal predictions). Thus the users can choose the best

trade-off between misclassification and false alarms costs according to their requirements. From those experimental results we observed that the second goal of this thesis has been reached.

Simplification procedure

The third goal of this thesis is to produce comprehensible solutions that can be understood by the user. For that reason, a simplification process was included in RM and EDR in order to remove the conditions that are not affecting the decision of the rule. The simplification process is introduced in section 5.7. This helps to remove extra-code which is naturally produced by the GP process. Section 7.3.5 shows an example of the comprehensibility of the decision rules. Based on the simplification procedure and on the example in section 7.3.5 we have showed that our objective has been achieved. However, the understanding of the proposed solutions requires that users are familiar with the variables that form the classifier.

Precision and accuracy improvement

We have presented a pruning procedure, which is based on the analysis of decision trees. The idea behind this method is to improve the accuracy and the precision of the classification. From the experimental results in section 8.3.2 we conclude that SM is able to identify useful parts of the decision trees. The pruning of non useful conditions improved the accuracy and precision of some decision trees. However, the user has to bear in mind that such improvement is paid with loss of recall. The improvement achieved by SM varies; it depends on the stage of the evolutionary process and the

pruning threshold. Experimental results suggests that the improvement in precision and accuracy has been achieved.

Analysis in GP

A series of experiments to analyse the factors that work in favour of RM was performed in section 6.3.5. The experimental results showed that gathering rules from several individuals of the population helps to collect patterns to classify more positive cases, which is reflected in the recall improvement. The majority of the positive cases were correctly predicted, it is exhibited in the improvement of the prediction. On the other hand, the experiment in section 6.3.5 showed that some useful rules can be lost in the evolutionary process, for that reason the accumulation of rules, through generations aids to classify more positive cases. From those results we observed that the factors that benefit RM are 1) the collection of rules from several individuals of the population and 2) the accumulation of rules though generations. To reinforce the first argument, another experiment was performed in section 6.3.6 to count the number of rules provided by every individual in the population. The experiment showed evidence that there is a great potential of patterns in the complete population of decision trees, even in individuals with low fitness. From this analysis we conclude that:

1. some useful patterns can be lost during the evolutionary process

2. useful patterns can be found in low-fitness individuals

An alternative interpretation of our results is that RM (Chapter 6) can be used to reduce the number of evaluations required by GP to achieve the best precision, recall

and accuracy in imbalanced environments. This can be significant in applications where evaluations are expensive.

9.3 Future research

This section introduces recommendations that may enhance and extend the research in this thesis. Thus, the further work for Repository Method (RM) and Evolving Decision Rules (EDR) is presented in section 9.3.1. Next, section 9.3.2 suggests future research for EDR. Finally, the further work for Scenario Method (SM) will be proposed in section 9.3.3.

9.3.1 RM and EDR future research

As was mentioned in section 2.4.2, there are many problems that require the detection of the minority class in imbalanced data sets. A line of research is to test the applicability of RM and EDR in other domains.

According to japkowicz [63] the approaches to solve the imbalance data set problem work differently for every level of imbalance. Thus, another research direction would be to perform a series of experiments to test RM and EDR, using data sets with different levels of imbalance, to investigate their performance in each of them.

RM and EDR include a process to detect and eliminate the extra-code (introns), which is produced in the GP evolution, by removing the redundancy and the vacuous conditions (see definition 4 section 5.6). One of the advantages of removing the vacuous conditions is to eliminate those conditions that are not affecting the performance of

Table 9.1: Table of contributions: RM - Repository method, EDR - Evolving decision rules, SM - Scenario Method,

Contribution description	Chapter number	Method name	Empirical evidence	Other methods comparison
Minority class detection	6	RM	Barclays, Barclays 400, Arbitrage	Standard GP, C5.0, Eddie-Arb
	7	EDR	Barclays, Barclays 400, Arbitrage, Tesco, Artificial (1,2,3)	Standard GP, C5.0, Eddie-Arb
Produce a range of solutions	6	RM	Barclays, Barclays 400, Arbitrage	Standard GP, C5.0, Eddie-Arb
	7	EDR	Barclays, Barclays 400, Arbitrage, Tesco, Artificial (1,2,3)	Standard GP, C5.0, Eddie-Arb
	8	SM	Barclays, Barclays 400, Arbitrage, Tesco, Artificial (1,2,3)	Standard GP, C5.0, Eddie-Arb
Simplification procedure	6	RM	Barclays, Barclays 400, Arbitrage	Standard GP, C5.0, Eddie-Arb
	7	EDR	Barclays, Barclays 400, Arbitrage, Tesco, Artificial (1,2,3)	Standard GP, C5.0, Eddie-Arb
	8	SM	Barclays, Barclays 400, Arbitrage, Tesco, Artificial (1,2,3)	Standard GP, C5.0, Eddie-Arb
precision and accuracy improvement	8	SM	Barclays, Tesco	
Analysis in GP	6	RM	Barclays	

the classification in that data set, but that could be unpredictable in other data sets. An interesting future work would be to study the impact of removing the vacuous conditions.

Another line of research is to compare RM and EDR with other approaches designed to deal with imbalance data sets. These should be compared with techniques that work in the data set level as the re-sampling or other algorithms as the random forest.

An interesting future research would be tp investigate the interpretability of the decision rules produced by our approaches. An human expert can read the decision rules that classify the positive examples in the testing data set. The objective is to understand the conditions that are triggering the event, analyzing which of them are classifying the example correctly and which of them do not do it. The understanding of the decision rule can help to understand the solution of the problem.

9.3.2 Evolving Decision Rules (EDR) future research

EDR evolves a set of decision rules using the operators mutation and hill-climbing. Given that, EDR does not use the crossover operator an interesting line of research is to test the effect of this operator in EDR.

9.3.3 Scenario Method future research

Experimental results in chapter 8 suggested that SM is able to improve the precision and accuracy of the decision trees. More investigation needs to be done in order to find out the effects of SM when the pruned decision trees are reintroduced in the

evolutionary process.

Another research direction is to perform a series of experiments of SM using different values of the pruning threshold and graph the results in the ROC space. The objective of this experiment is to find out if the classifications, produced by the different levels of pruning, are able to generate a well distributed ROC curve to offer a range of predictions to suit different users' guidelines.

Part III

Appendix

This section describes in pseudo-code the procedures in Repository method. In order to describe clearly the experiment, the pseudo code for this procedure is included in this section. Let us define the following notation:

Algorithm 2: Selection()

```
/*Repository method, main procedure*/
input  : List Population
output: None
1  begin
2      NumVariables ← Number of variables in training/testing data set;
3      List Repository;
4      for each tree T ∈ Population do
5      begin
           /*Delimit the rule's tree*/
6          NodeMap ← ExtractRules(T); Alg11
           /*Number of rules in T is the number of rows in NodeMap*/
7          TotalRulesTree ← Number of rules in T;
8          for k=1 to TotalRulesTree do
9          begin
10             Rule_k ← GetConditionList(T,NodeMap,k); Alg12
11             GetAdmission(Rule_k, Repository); Alg4
12         end
13     end
14 end
```

Algorithm 3: GetConditionList()

```
/*Gets the conditions that compose a rule R_k. NodeMap is a matrix
  that lists the rules in T. Every row describes a rule by means of the
  nodes that identify that condition */

input  : Tree T, matrix NodeMap, int NumRule
output: List

1  begin
2      int NumCond ← 0;
3      PositionOperator ← 1; /*Constant values*/
4      PositionLeave1 ← 2;
5      PositionLeave2 ← 3;

       /*Counts the number of conditions in the rule*/
6      while NodeMap[NumRule, NumCond] ≠ 0 do
7          NumCond ← NumCond + 1
8
       /*Declares a matrix to store conditions*/
9      ConditionList ← matrix[NumCond, 3];
10     for k=1 to NumCond do
11     begin
           /*Extracts the operator and the variables*/
12         StartCond ← NodeMap[NumRuke,k];
13         ConditionList[k, PositionOperator] ← T[StartCond];
14         ConditionList[k, PositionLeave1] ← T[StartCond+1];
15         ConditionList[k, PositionLeave2] ← T[StartCond+2];
16     end
17     return ConditionList;
18 end
```

Algorithm 4: GetAdmission()

```
/*Determines if the rule will be part of the rule repository or not*/
input  : List Rule_new,
         List Repository
output: Boolean
1  begin
2      PrecisionThreshold ← Minimum precision required for a rule
3      TrainingData ← The data training data set
4      Different ← 0;
5      Equal ← 1;
6      Similar ← 2;
7      TrainingEval ← EvaluateRule(Rule_new,TrainingData); Alg5
8      if TrainingEval ≤ PrecisionThreshold then
9          return False
       /*Verify is the rule is contributing to the variety of the
         collection*/
10     CondMap_New ← CreateConditionMap(Rule_new); Alg6
11     for each Rule_k ∈ Repository do
12         begin
13             CondMap_k ← Gets R_k condition map from the Repository;
14             CompareRules ← (CondMap_k,CondMap_new);
15             case CompareRules = Similar
16                 Repository ← Repository - R_k;
17                 Repository ← Repository ∪ R_new;
18             case CompareRules = Different
19                 Repository ← Repository ∪ R_new;
20             case CompareRules = Equal
21                 Do nothing;
22         end
23     return;
24 end
```

Algorithm 5: EvaluateRule()

```
    /*Evaluates the rule*/
    input  : Tree T, List Data
    output: Decimal
 1  begin
        /*Initialize the confusion matrix values*/
 2      for j=1 to 2 do
 3          for k=1 to 2 do
 4              M_i(j,k) ← 0
        /*Evaluates the tree de data set Data*/
 5      if prediction = Class and instance = Class then
 6          M_i(1,1) ← M_i(1,1)+1
 7      if prediction = Class and instance = No Class then
 8          M_i(1,2) ← M_i(1,2)+1
 9      if prediction = No Class and instance = Class then
10          M_i(2,1) ← M_i(2,1)+1
11      if prediction = No Class and instance = No Class then
12          M_i(2,2) ← M_i(2,2)+1
        /*Estimates the precision and recall*/
13      Precision ← M_i(1,1) / (M_i(1,1)+M_i(1,2))
14      Recall ← M_i(1,1) / (M_i(1,1)+M_i(2,1))
        /*Estimates the evaluation*/
15      Evaluation ← sqrt(Precision × Recall)
16      return Evaluation;
17  end
```

Algorithm 6: CreateConditionMap()

```
/*Creates a condition map, this will be used to identify similarities in
  the rules*/
input  : List Rule
output: Matrix
1  begin
2      PositionOperator ← 1; /*Constant values*/
3      PositionLeave1 ← 2;
4      PositionLeave2 ← 3;
5      NumVar ← Number of variables in the training data;
6      NumConditions ← Number of elements in Rule;
       /*Creates the map to register the conditions of the rule*/
7      CondMap ← matrix[NumVariables, NumVariables + 2]
8      for j=1 to NumConditions do
9          for k=1 to NumConditions + 2 do
10             CondMap[j,k] ← ∅
11     for k=1 to NumConditions do
12     begin
           /*Extracts the conditions components*/
13         Operator ← Rule[k,PositionOperator];
14         Leave1 ← Rule[k,PositionLeave1];
15         Leave2 ← Rule[k,PositionLeave2];
           /*Register conditions composed of two variables*/
16         if (IsVar(Leave1)=True) and (IsVar(Leave2)= True) then
17             if Leave1 ≠ Leave2 then
18                 VarId1 ← GetVarId(Leave1);
19                 VarId2 ← GetVarId(Leave2);
20                 if VarId1 < VarId2 then
21                     CondMap[VarId1,VarId2] ← Operator
22                 else
23                     CondMap[VarId2,VarId1] ← Switch(Operator)
24         else
               /*conditions composed of a variable and a threshold*/
25             VarId1 ← GetVarId(Leave1);
26             Threshold ← Leave2;
27             CondMap ←
28             IdentifyRange(CondMap,Operator,VarId1,Threshold);
29     end
30     return ConditionMap
31 end
```

Algorithm 7: IsVar()

```
    /*Determines if the string is a variable or a threshold. All variables
      identification starts with the substring 'Variable' */
    input  : String Leave
    output: Boolean
1   begin
2       if (Leave = 'Variable'#) then
3           return True;
4       return False;
5   end
```

Algorithm 8: Switch()

```
    /*Changes the sense of the inequality*/
    input  : String Operator
    output: String
1   begin
2       if Operator = '<' then
3           return '>'
4       end
5       if Operator = '>' then
6           return '<'
7       end
8   end
```

Algorithm 9: IdentifyRange()

```
/*Identify the range of the threshold based in the inequalities of the
  rule*/
input  : matrix ConditionMap,
         String Operator,
         int VarId,
         number Threshold
output: Matrix ConditionMap
1  begin
2      NumVar ← Number of variables;
3      Th1 ← ConditionMap[VarId, NumVar + 1];
4      Th2 ← ConditionMap[VarId, NumVar + 2];
5      case Th1 = ∅ and Th2 = ∅ and Op = '<'
6          Th2 ← Threshold
7      case Th1 = ∅ and Th2 = ∅ and Op='>'
8          Th1 ← Threshold
9      case Th1 = ∅ and Th2 ≠ ∅ and Op='<' and Threshold < Th2
10         Th2 ← Threshold
11     case Th1 = ∅ and Th2 ≠ ∅ and Op='>' and Threshold < Th2
12         Th1 ← Threshold
13     case Th1 ≠ ∅ and Th2 ≠ ∅ and Op='<' and Th1 <Threshold
14         Th2 ← Threshold
15     case Th1 ≠ ∅ and Th2 = ∅ and Op='>' and Th1 < Threshold
16         Th1 ← Threshold
17     case Th1 ≠ ∅ and Th2 ≠ ∅ and Op = '<' and
18     Th1 < Threshold < Th2
19         Th2 ← Threshold
20     case Th1 ≠ ∅ and Th2 ≠ ∅ and Op = '>' and
21     Th1 < Threshold < Th2
22         Th1 ← Threshold
23     end
24     ConditionMap[VarId, NumVar + 1] ← Th1;
25     ConditionMap[VarId, NumVar + 2] ← Th2;
26     return ConditionMap
27 end
```

Algorithm 10: Equal()

```
/*Compares two condition map*/
input  : Matrix MapCond_1, Matrix MapCond_2
output: Number
1  begin
2      Different ← 0;
3      Equal ← 1;
4      Similar ← 2;
       /*Compares the conditions composed of two variables*/
5      NumVariables ← Number of columns in MapCond_1
6      for j=1 to NumVariables do
7          for k=1 to NumVariables do
8              if MapCond_1[j][k] ≠ MapCond_2[j][k] then
9                  return Different;
10         end
11     end
       /*Compares the conditions composed of a variable and a
         threshold*/
12     Result ← Equal;
13     for j = NumVariables + 1 to NumVariables + 2 do
14         for k=1 to NumVariables do
15             case MapCond_1[j][k] ≠ ∅ and MapCond_2[j][k] ≠ ∅
16                 Result ← Similar
17             case MapCond_1[j][k] ≠ ∅ and MapCond_2[j][k] = ∅
18                 return Diferent
19             case MapCond_1[j][k] = ∅ and MapCond_2[j][k] ≠ ∅
20                 return Different
21         end
22     end
23     return Result;
24 end
```

Algorithm 11: ExtractRules()

```
/*Delimit the tree's rules*/
input  : Tree T
output: Integer
1  begin
2      N ← 0;
3      L ← 0;
       /*Separates the rules of the tree in terms of its conditional nodes*/
4      for every R_j ∈ T do
5          R_j = {n_j1, n_j2, ···} N ← N+1; /*Stores the number of rules in
             T*/
6          NumNodes ← Number of conditional nodes in R_j
7          if L < NumNodes then
8              L ← Number of nodes in R_j
9      end
       /*Initialize the matrix*/
10     for i = 0 to N do
11         for j = 0 to L do
12             NodeMap[i, j] ← 0 ;
13         end
14     end
15     for each R_j where j = 1, 2, ··· N do
16         for each n_ji ∈ R_j do
17             NodeMap[j, i] ← n_ji
18         end
19     end
20     return NodeMap;
21 end
```

Algorithm 12: CreatePopulation()

```
input  : integer PopulationSize,
         integer NumInitialInstances,
         real PercentInstances,
         real HillClimbingProb,
output : List
1  begin
2      int NumGeneration ← 0; /* Variable initialization*/
3      int NumRules ← The number of rules in the Repository;
4      int NumIndividuals ← 0;;
5      List Population;
6      /* Calculates the number of instances*/
7      if NumRules > 0 then
8          if NumRules · NumInitialInstances > PopulationSize then
9              NumInstances = round(PopulationSize/NumRules);
10     if NumInitialInstances = 0 then
11         for k=1 to NumRules do
12         if PercentInstances > Random Number then
13             Rule_k ← Rule number k in the repository of rules
14             /* Apply the Hill-Climbing or Mutation*/
15             if HillClimbingProb < Random number then
16                 NewIndividual ← HillClimbing(Rule_k);
17             else
18                 NewIndividual ← Mutation(Rule_k);
19             Population ← Population ∪ NewIndividual;
20             NumIndividuals ← NumIndividuals + 1;
21     else
22         for k=1 to NumRules do
23             Rule_k ← Rule number k in the repository of rules
24             IniInstances ←0;
25             if HillClibingProb> Random Number then
26                 /* Apply the Hill-Climbing or Mutation*/
27                 NewIndividual ← HillClimbing(Rule_k);
28                 Population ← Population ∪ NewIndividual;
29                 IniInstances ← 1;
30             for k=1 to NumRules do
31                 NewIndividual ← Mutation(Rule);
32                 Population ← Population ∪ NewIndividual;
33             NumIndividuals ← NumInstances · NumRules;
34     for j=1 to PopulationSize do
35         NumIndividuals ← Mutation(Rule);
36         NewIndividual ← GenerateRandomTree();
37         Population ← Population ∪ NewIndividual;
38     return Population;
39 end
```

Bibliography

[1] Akinori Abe and Yukio Ohsawa, *Special issue on chance disocvery*, New Generation Computing, 1, vol. 21, Berlin: Springer and Tokyo: Ohmsha, November 2002, pp. 1–2.

[2] Peter Angeline, *Genetic programming and emergent intelligence*, Advances in Genetic Programming (Kenneth E., ed.), MIT Press, 1994, pp. 75–98.

[3] Andrew David Bain, *The economics of the financial markets*, Blackwell, Oxford UK and Cambridge USA, 1992.

[4] Wolfgang Banzhaf, *Genetic programming : an introduction on the automatic evolution of computer programs and its applications*, Morgan Kaufmann, 1998.

[5] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard, *A study of the behaviour of several methods for balancing machine learning training data*, SIGKDD Explor. Newsl. **6** (2004), no. 1, 20–29.

[6] Richard Ernest Bellman, *Adaptive control processes : a guided tour*, Princeton University Press, Princeton, N.J., 1961.

[7] Martin J. Bishop and Chris Rawlings (eds.), *Dna and protein sequence analysis: A practical approach*, Oxford University Press, 1997.

[8] Tobias Blickle and Lothar Thiele, *Genetic programming and redundancy*, Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrücken) (Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany) (J. Hopf, ed.), Max-Planck-Institut für Informatik (MPI-I-94-241), 1994, pp. 33–38.

[9] J. Bobbin and X. Yao, *Automatic discovery of relational information in comprehensible control rules by evolutionary algorithms*, Proceedings of the Third Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems (Canberra, Australia), 1999, pp. 117–123.

[10] Jason Bobbin and Xin Yao, *Evolving rules for nonlinear control*, 1999.

[11] Celia C. Bojarczuk, Heitor S. Lopes, and Alex A. Freitas, *Discovering comprehensible classification rules by using genetic programming: a case study in a medical domain*, Proceedings of the Genetic and Evolutionary Computation Conference (Orlando, Florida, USA) (Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, eds.), vol. 2, Morgan Kaufmann, 13-17 July 1999, pp. 953–958.

[12] Celia C Bojarczuk, Heitor S Lopes, and Alex A. Freitas, *An innovative application of a constrained-syntax genetic programming system to the problem of predicting survival of patients.*, Genetic Programming: Proc. 6th European Conference (EuroGP-2003) (C. Ryan, M. Keijzer, R. Poli, T. Soule, E. Tsang, and E. Costa, eds.), Lecture Notes in Computer Science, vol. 2610, Springer-Verlag, April 2003.

[13] Celia C. Bojarczuk, Heitor S. Lopes, Alex A. Freitas, and Edson L Michalkiewicz, *A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets*, Artificial Intelligence in Medicine **30** (2004), no. 1, 27–48.

[14] Leo Breiman, *Random forests*, Machine Learning **45** (2001), no. 1, 5–32.

[15] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles. J. Stone, *Classification and regression trees*, Wadsworth International Group, United States of America, 1984.

[16] E. Burke, S. Gustafson, and G. Kendall, *Survey and analysis of diversity measures in genetic programming*, 2002, (Accepted as a full paper) Proceedings of the Genetic and Evolutionary Computation Conference, 2002.

[17] Martin Volker Butz, *Rule-based evolutionary online learning systems: learning bounds, classification, and prediction*, Ph.D. thesis, Champaign, IL, USA, 2004, Adviser-David E. Goldberg.

[18] Robert Callan, *Essence of neural networks*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[19] Hongging Cao, Friedrich Recknagel, Gea-Jae Joo, and Dong-Kyun Kim, *Disovery of predictive rule sets for chlorophyll-a dynamics in the nakdong river (korea) by means of the hybrid evolutionaru algorithm hea*, Ecological Informatics **1** (2006), 43–53.

[20] Jaime Guillermo Carbonell, *Machine learning : paradigms and methods*, MIT Press, Cambridge, Mass, 1990.

[21] Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz, *Editorial: special issue on learning from imbalanced data sets*, SIGKDD Explor. Newsl. **6** (2004), no. 1, 1–6.

[22] Chao Chen, Andy Liaw, and Leo Breiman, *Using random forest to learn imbalanced data*, Technical Report 666, Department of Statistics, University of California, Berkeley, 2004.

[23] Shu-Heng Chen (ed.), *Genetic algorithms and genetic programming in computational finance*, Kluwer Academic, 2002.

[24] Shu-Heng Chen and Paul P. Wang (eds.), *Computational intelligence in economics and finance*, Springer, 2004.

[25] Noam Chomsky, *Syntactic structures*, The Hague, Mouton, 1957.

[26] Arthur L. Corcoran and Sandip Sen, *Using Real-Valued Genetic Algorithms to Evolve Rule Sets for Classification*, International Conference on Evolutionary Computation, 1994, pp. 120–124.

[27] R. Dahm, *Friedrich miescher and the discovery of dna*, Developmental Biology **13** (2005), no. 278(2), 274–88.

[28] Charles Darwin, *The origin of species*, Jonh Murray, 1859.

[29] I. De Falco, A. Della Cioppa, and E. Tarantino, *Discovering interesting classification rules with genetic programming*, Applied Soft Computing **1** (2001), no. 4, 257–269.

[30] K. De Jong, *Genetic-algorithm-based learning*, Machine Learning: An Artificial Intelligence Approach (Volume III) (Y. Kodratoff and R. S. Michalski, eds.), Kaufmann, San Mateo, CA, 1990, pp. 611–638.

[31] Rob Dixon and Phil Holmes, *Financial markets : an introduction*, series in accounting and finance, International Thomson Business Press, 168-173 High Holborn, London, 1996.

[32] Robert D. Edwards and John Magee, *Technical analysis of stock trends*, At. Lucie Press, Boca raton, Florida, 2001.

[33] Jeroen Eggermont, Joost N. Kok, and Walter A. Kosters, *Detecting and pruning introns for faster decision evolution*, The 8th International Conference of Parallel Problem Solving from Nature, Springer-Verlag, 2004.

[34] A. E. Eiben, *Introduction to evolutionary comptutation*, Springer-Verlag, Berlin, Germany, 1993.

[35] Peter Naur et al, *Revised report on the algorithmic language algol 60*, Communicacions of the ACM 6(1), vol. 1, 1963, pp. 1–17.

[36] E. F. Fama, *Efficient capital markets: A review of theory and empirical work*, The Journal of Finance **23** (1970), no. 383.

[37] Tom Fawcett, *Roc graphs: Notes and practical considerations for researchers*, Introductory paper, 2004.

[38] Tom Fawcett and Foster J. Provost, *Adaptive fraud detection*, Data Mining and Knowledge Discovery **1** (1997), no. 3, 291–316.

[39] M. V. Fidelis, H. S. Lopes, and A. A. Freitas, *Discovering comprehensible classification rules a genetic algorithm*, Proceedings of the 2000 Congress on Evolutionary Computation CEC00 (La Jolla Marriott Hotel La Jolla, California, USA), IEEE Press, 6-9 2000, pp. 805–810.

[40] Peter Flach and Nada Lavrac, *Intelligent data analysis*, Springer-Verlag, Berlin, Germany, 2003.

[41] Lawrence J Fogel, *Title intelligence through simulated evolution : forty years of evolutionary programming*, Wiley series on intelligent systems, Imprint New York, 1999.

[42] L.J. Fogel, A. J. Owens, and M.J. Walsh, *Artificial intelligence through a simulation of evolution*, Biophysics and Cybernetic Systems (Washington D.C.) (A. Callahan M. Maxfield and L.J Fogel, eds.), Spartan, 1965, pp. 131–156.

[43] Alma Garcia-Almanza and Edward Tsang, *Repository method to suit different investment strategies*, Congress on Evolutionary Computation (CEC), 2007.

[44] Alma Garcia-Almanza and Edward P.K. Tsang, *Simplifying decision trees learned by genetic algorithms*, Congress on Evolutionary Computation (CEC), 2006, pp. 7906–7912.

[45] Alma L Garcia-Almanza and Edward Tsang, *Detection of stock price movements using chance discovery and genetic programming*, Innovation in Knowledge-Based and Intelligent Engineering Systems (2007).

[46] Alma L Garcia-Almanza and Edward P.K. Tsang, *The repository method for chance discovery in financial forecasting*, KES2006 10th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (Springer-Verlag, ed.), 2006.

[47] Alma L Garcia-Almanza, Edward P.K. Tsang, and Edgar Galvan-Lopez, *Evolving decision rules to discover patterns in financial data sets*, Computational Methods in Financial Engineering (2007).

[48] Alma Lilia Garcia-Almanza and Edward Tsang, *Evolving decision rules to predict investment opportunities*, International Journal of Automation and Computing **05** (2008), no. 1, 22–31.

[49] Alma Lilia Garcia-Almanza and Edward P.K. Tsang, *Forecasting stock prices using genetic programming and chance discovery*, 12th International Conference On Computing In Economics And Finance, 2006.

[50] D.M. Green and J.A. Swets, *Signal detection theory and psychophysics*, Robert E. Krieger Publishing Co., New York, USA, 1974.

[51] M Greiner, D Pfeiffer, and RD. Smith, *Principles and practical application of receiver-operating characteristic analysis for diagnostic tests*, Prevent Veterinary Med **45** (2000), 23–41.

[52] Lawrence O. Hall and Ajay Joshi, *Building accurate classiffers from imbalanced data sets*, 2005.

[53] James A. Hanley and Barbara J. McNeil, *The meaning and use of the area under a reciever operating characteristic roc curve*, Radiology, vol. 143, W. Madison, 1998, pp. 29–36.

[54] P. E. Hart, *The condensed nearest neighbor rule*, IEEE Transactions on Information Theory, 1968.

[55] R. J. Henery, *Classification*, Ellis Horwood, Upper Saddle River, NJ, USA, 1994.

[56] J. H. Holland, *Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems*, Machine Learning: An Artificial Intelligence Approach: Volume II (R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds.), Kaufmann, Los Altos, CA, 1986, pp. 593–623.

[57] John Holland, *Adaptation*, vol. 4, Plenum, 1976.

[58] John H. Holland, *Adaptation in natural and artificial systems*, MIT Press, Cambridge MA, 1975.

[59] Jih-Jeng Huang, Gwo-Hshiung Tzeng, and Chorng-Shyong Ong, *Two-stage genetic programming (2SGP) for the credit scoring model*, Applied Mathematics and Computation **174** (2006), no. 2, 1039–1053.

[60] Jin Huang and Charles X. Ling, *Using auc and accuracy in evaluating learning algorithms*, vol. 17, IEEE Transactions on knowledge and Data Engineering, 2005, pp. 299–310.

[61] Hitoshi Iba, Hugo de Garis, and Taisuke Sato, *Genetic Programming using a Minimum Description Length Principle*, Advances in Genetic Programming (Kenneth E. Kinnear, Jr., ed.), MIT Press, 1994, pp. 265–284.

[62] Cezary Z. Janikow, *A knowledge-intensive genetic algorithm for supervised learning*, Machine Learning **13** (1993), no. 2-3, 189–228.

[63] Nathalie Japkowicz, *The class imbalance problem: Significance and strategies*, Proceedings of the 2000 International Conference on Artificial Intelligence (IC-AI'2000), vol. 1, 2000, pp. 111–117.

[64] Kenneth A. De Jong and William M. Spears, *Learning Concept Classification Rules using Genetic Algorithms*, Proceedings of the Twelfth International Conference on Artificial Intelligence IJCAI-91, vol. 2, 1991.

[65] Alain Rakotomamon Jy, *Optimizing area under roc curve with svms.*

[66] Michael N. Kahn, *Technical analysis plain and simple*, 1 ed., Pearson education limited, Great Britain, 1999.

[67] Yves Kodratoff, Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, *Machine learning : an artificial intelligence approach*, Morgan Kaufmann Publishers, inc., Palo Alto CA., 1990.

[68] Ron Kohavi and Foster Provost, *Glossary of terms*, Edited for the Special Issue on Applications of Machine Learning and the Knowledge Discovery Process, vol. 30, February 1998.

[69] John Koza, *Genetic programming: On the programming of computers by means of natural selection*, The MIT Press, Cambridge, Massachusetts, 1992.

[70] Miroslav Kubat, Robert Holte, and Stan Matwin, *Learning when negative examples abound*, ECML '97: Proceedings of the 9th European Conference on Machine Learning (London, UK), Springer-Verlag, 1997, pp. 146–153.

[71] Miroslav Kubat, Robert C. Holte, and Stan Matwin, *Machine learning for the detection of oil spills in satellite radar images*, Machine Learning, vol. 30, 195-215, 1998.

[72] Wojciech Kwedlo and Marek Kretowski, *An evolutionary algorithm using multivariate discretization for decision rule induction*, Principles of Data Mining and Knowledge Discovery, 1999, pp. 392–397.

[73] William Langdon and Riccardo Poli, *Foundations of genetic programming*, Springer, San Francisco, California, 2002.

[74] William B. Langdon, *Quadratic bloat in genetic programming*, Proceedings of the Genetic and evolutionary Computation Conference (GECCO-2000), 2000, pp. 451–458.

[75] William B. Langdon and Ricardo Poli, *Fitness causes bloat*, Soft Computing in Engineering Design and Manufacturing (London) (P. K. Chawdhry, R. Roy, and R. K. Pant, eds.), Springer-Verlag, 1997, pp. 13–22.

[76] Jorma Laurikkala, *Improving identification of difficult small classes by balancing class distribution*, AIME '01: Proceedings of the 8th Conference on AI in Medicine in Europe (London, UK), Springer-Verlag, 2001, pp. 63–66.

[77] Jin Li, *A genetic programming based tool for financial forecasting*, PhD Thesis, University of Essex, Colchester CO4 3SQ, UK, 2001.

[78] Charles X. Ling and Chenghui Li, *Data mining for direct marketing: Problems and solutions*, Knowledge Discovery and Data Mining, 1998, pp. 73–79.

[79] Charles X. Ling and Victor S. Sheng, *Cost-sensitive learning and the class imbalance problem*, Encyclopedia of Machine Learning. (C. Sammut, ed.), Springer, 2008.

[80] Honghu Liu and Tongtong Wu, *Estimating the area under a receiver operating characteristic curve for repeated measures design*, Journal of Statistical Software **8** (2003), no. 12, 1–18.

[81] Kate McCarthy, Bibi Zabar, and Gary Weiss, *Does cost-sensitive learning beat sampling for classifying rare classes?*, UBDM '05: Proceedings of the 1st international workshop on Utility-based data mining (New York, NY, USA), ACM Press, 2005, pp. 69–77.

[82] Zbigniew Michalewicz, *Genetic algorithms + data structures = evolution programs*, Springer, 2002.

[83] Tom M. Mitchell, *Machine learning*, McGraw-Hill, Boston, Mass, 1997.

[84] Joseph James Murray, *Genetic diversity and natural selection*, Oliver and Boyd, Edinburgh, 1972.

[85] Ayahiko Niimi and Eiichiro Tazaki, *Rule discovery technique using genetic programming combined with apriori algorithm*, Discovery Science, Third International Conference, DS 2000, Kyoto, Japan, December 4-6, 2000, Proceedings (Setsuo Arikawa and Shinichi Morishita, eds.), Lecture Notes in Computer Science, vol. 1967, Springer, 2000.

[86] Anton Nijholt, *Context-free grammars : covers, normal forms, and parsing*, Springer-Verlag, Berling, 1980.

[87] Lawrence O. Hall W. Philip Kegelmeyer Nitesh V. Chawla, Kevin W. Bowyer, *Smote: Synthetic minority over-sampling technique*, p. 321.

[88] Peter Nordin, Frank Francone, and Wolfgang Banzhaf, *Explicitly Defined Introns and Destructive Crossover in Genetic Programming*, Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications (Tahoe City, California, USA) (Justinian P. Rosca, ed.), 9 July 1995, pp. 6–22.

[89] Yukio Ohsawa and Peter McBurney, *Chance discovery*, Springer, 2003.

[90] Una-May O'Reilly and Franz Oppacher, *Hybridized crossover-based search techniques for program discovery*, Tech. Report 95-02-007, Santa Fe Institute, 1399 Hyde Park Road Santa Fe, New Mexico 87501-8943 USA, 1995.

[91] Seong Ho Park, Jin Mo Goo, and Chan-Hee Jo, *Receiver operating characteristic (roc) curve: Practical review for radiologist*, Korean Journal of Radiology **5** (2004), no. 1, 11–18.

[92] M. Pazzani, C. Merz, P. Murphy, T Hume K Ali, and C.Brunk, *Reducing misclassification costs*, Proceedings of the Eleventh International Conference on Machine Learning, 1994, pp. 217–225.

[93] Ricardo Poli, *A simple but theoretically-motivated method to control bloat in genetic programming*, Proceedings of the 6th European Conference, Springer-Verlag, 2003, pp. 204–217.

[94] Foster J. Provost and Tom Fawcett, *Robust classification for imprecise environments*, Machine Learning, no. 3, 203–231.

[95] Foster J. Provost and Tom Fawcett, *Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions*, Knowledge Discovery and Data Mining, 1997, pp. 43–48.

[96] Foster J. Provost, Tom Fawcett, and Ron Kohavi, *The case against accuracy estimation for comparing induction algorithms*, In Proc. Fifteenth International Conference in Machine Learning, W. Madison, 1998, pp. 445–553.

[97] John Quinlan, *Simplifying decision trees*, International Journal of Machine studies, 1986, pp. 221–234.

[98] John Ross Quinlan, *C.45 programs for machine learning*, Morgan Kaufmann, San Mateo California, 1993.

[99] Bhavani Raskutti and Adam Kowalczyk, *Extreme re-balancing for svms: a case study*, SIGKDD Explor. Newsl. **6** (2004), no. 1, 60–69.

[100] Conor Ryan and Michael O'Neill, *Grammatical evolution: A steady state approach*, Late Breaking Papers at the Genetic Programming 1998 Conference (University of Wisconsin, Madison, Wisconsin, USA) (John R. Koza, ed.), Stanford University Bookstore, 22-25 1998.

[101] Michael Sipser, *Introduction to the theory of computation*, Thomson, Boston, c2006.

[102] Stephen Smith, *Flexible learning of problem solving heuristics through adaptive search*, Proceedings 8th International Joint Conference on Artificial Intelligence, August 1983.

[103] Terence Soule, *Code growth in genetic programming*, PhD Thesis, College of Graduate Studies, University of Idaho, Moscow, Idaho, USA, 15 May 1998.

[104] Terence Soule and James Foster, *Code size and depth flows in genetic programming*, Proceeding of the Second Annual Conference (John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick R. Riolo, eds.), Morgan Kaufmann, 1997, pp. 313–320.

[105] Terence Soule and James A. Foster, *Effects of code growth and parsimony pressure on populations in genetic programming*, vol. 6, Winter 1998, pp. 293–309.

[106] John Arthur Swets, *Signal detection and recognition by human observers*, Wiley, New York, USA, 1964.

[107] Walter Alden Tackett, *Recombination, selection, and the genetic construction of computer programs*, Ph.D. thesis, University of Southern California, Department of Electrical Engineering Systems, USA, 1994.

[108] Astro Teller, *Algorithm evolution with internal reinforcement for signal understanding*, PhD Thesis, Carnegie Mellon University, 1998.

[109] Astro Teller and M. Veloso, *PADO: Learning tree structured algorithms for orchestration into an object recognition system*, Tech. Report CMU-CS-95-101, Pittsburgh, PA, USA, 1995.

[110] Astro Teller and Manuela Veloso, *Neural programming and an internal reinforcement policy*, In fisrt international Conference on Simulated Evolution and learning, Springer-Verlag, 1996, pp. 279–286.

[111] I. Tomek, *Two modifications of cnn*, IEEE Transactions on Systems Man and Communications, 1976, pp. 769–772.

[112] Edward P.K. Tsang, Jin Li, and J.M. Butler, *Eddie beats the bookies*, International Journal of Software, Practice and Experience, 10, vol. 28, Wiley, August 1998, pp. 1033–1043.

[113] Edward P.K. Tsang, Jin Li, Sheri Markose, Hakan Er, Abdel Salhi, and Giulia Iori, *Eddie in financial decision making*, Journal of Management and Economics, 4, vol. 4, November 2000.

[114] Edward P.K. Tsang, Sheri Markose, and Hakan Er, *Chance discovery in stock index option and future arbitrage*, New Mathematics and Natural Computation, World Scientific, 3, vol. 1, 2005, pp. 435–447.

[115] Edward P.K. Tsang and Serafin Martinez-Jaramillo, *Computational finance*, IEEE Computational Intelligence Society Newsletter, IEEE, 2004, pp. 3–8.

[116] Edward P.K. Tsang, P. Yung, and Jin Li, *Eddie-automation, a decision support tool for financial forecasting*, Journal of Decision Support Systems, Special Issue on Data Mining for Financial Decision Making, 4, vol. 37, 2004.

[117] Alan Mathison Turing, *The essential turing : seminal writings in computing, logic, philosophy, artificial intelligence, and artificial life, plus the secrets of enigma*, vol. 1, Oxford : Clarendon Press, 2004.

[118] Giedrius Vanagas, *Receiver operating characteristic curves and comparison of cardiac surgery risk stratification systems*, Interact CardioVasc Thorac Surg **3** (2004), no. 2, 319–322.

[119] Gary M. Weiss, *Mining with rarity: A unifying framework*, Special issue on learning from imbalanced data sets, vol. 6, 2004, pp. 7–19.

[120] S. W. Wilson, *Classifier fitness based on accuracy*, Evolutionary Computation **3** (1995), no. 2, 149–175.

[121] Ian H. Witten and Eibe Frank, *Data mining : practical machine learning tools and techniques with java implementations*, Morgan Kaufmann, San Francisco, Calif, 2000.

[122] Chuanhuan Yin, Shengfeng Tian, Houkuan Huang, and Jun He, *Applying genetic programming to evolve learned rules for network anomaly detection*, Advances in Natural Computation, First International Conference, ICNC 2005, Proceedings, Part III (Changsha, China) (Lipo Wang, Ke Chen, and Yew-Soon Ong, eds.), Lecture Notes in Computer Science, vol. 3612, Springer, 2005, pp. 323–331.