A New Method for Learning Decision Trees from Rules and its Illustration for Online Identity Application Fraud Detection

by

Amany Abdelhalim
B.Sc., Helwan University, 2000
M.Sc., Helwan University, 2003

A Dissertation Submitted in Partial Fulfillment
of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

Supervisory Committee

A New Method for Learning Decision Trees from Rules and its Illustration for Online Identity Application Fraud Detection

by

Amany Abdelhalim
B.Sc., Helwan University, 2000
M.Sc., Helwan University, 2003

Supervisory Committee

Dr. Issa Traore, Electrical and Computer Engineering
Supervisor

Dr. Kin Li, Electrical and Computer Engineering
Departmental Member

Dr. Mihai Sima, Electrical and Computer Engineering
Departmental Member

Dr. Jens Jahnke, Computer Science
Outside Member

Abstract

Supervisory Committee
Dr. Issa Traore, Electrical and Computer Engineering
Supervisor
Dr. Kin Li, Electrical and Computer Engineering
Departmental Member
Dr. Mihai Sima, Electrical and Computer Engineering
Departmental Member
Dr. Jens Jahnke, Computer Science
Outside Member

A decision tree is a graph or model for representing all the alternatives in a decision making process. Most of the methods that generate decision trees for a specific problem use examples of data instances in the decision tree generation process. We propose a new method called "RBDT-1"- rule based decision tree -for learning a decision tree from a set of decision rules that cover the data instances. RBDT-1 method uses a set of declarative rules as an input for generating a decision tree. The method's goal is to create on-demand a short and accurate decision tree from a stable or dynamically changing set of rules. The rules used by RBDT-1 could be generated either by an expert or induced directly from a rule induction method or indirectly by extracting them from a decision tree.

We conduct a comparative study of RBDT-1 with four existing decision tree methods based on different problems. The outcome of the study shows that in terms of tree complexity (number of nodes and leaves in the decision tree) RBDT-1 compares favorably to AQDT-1 and AQDT-2 which are methods that create decision trees from rules. RBDT-1 compares favorably also to ID3 while is as effective as C4.5 where both (ID3 and C4.5) are famous methods that generate decision trees from data examples. Experiments show that the classification accuracies of the different decision trees

produced by the different methods under comparison are equal. To illustrate how RBDT-1 can successfully be applied to an existing real life problem that could benefit from the method, we choose identity application fraud detection. We designed a new unsupervised framework to detect fraudulent applications for identity certificates by extracting identity patterns from the web, and crossing these patterns with information contained in the application forms in order to detect inconsistencies or anomalies. The outcome of this process is submitted to a decision tree classifier generated using RBDT-1 on the fly from a rule base which is derived from heuristics and expert knowledge, and updated as more information are obtained on fraudulent behavior. We evaluate the proposed framework by collecting real identity information online and generating synthetic fraud cases, achieving encouraging performance results.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I thank Almighty God for the completion of this thesis. I would like also to express my deep and sincere gratitude to my supervisor, Dr. Issa Traore. His wide knowledge and experience and his logical way of thinking have been of great value to me. It was a pleasure working under his supervision and a good opportunity to learn from him.

I owe my loving thanks to my wonderful husband Bassam Sayed for his understanding, encouragement, patience and help. I thank my sons Muhammad Sayed and Amr Sayed - who both were born during my PhD research - for making finishing the thesis a great challenge. My special thanks goes to my dear father Professor Mohammed Abdelhalim and my dear mother Mrs. Fatin Hussein, my sister Omniya and brothers Ahmed and Ayman for their loving support and encouragement. Finally, I would like to thank all my friends and family who wished me all the best.

Dedication

I dedicate this thesis to who raised me, loved me, encouraged me, prayed for me and always care about me my mum and dad.

I dedicate it also to my caring loving husband and my lovely sons.

Chapter 1.

Introduction

1.1 Context

Artificial intelligence (AI) can roughly be defined as the collection of problems and methodologies underlying or concerned with the automation of intelligent behavior. Over the last several decades, many AI methodologies have been developed to tackle various types of real-life problems. In general, these methodologies perform well when the problem domain is well-defined or a clear and accessible data source is available. Unfortunately, for specific types of problems the data may not be available or may simply be difficult to obtain. Examples of these types of problems include fraud detection and scientific discovery.

A straightforward approach to address these problems, when (instead of data) some common understanding, heuristic, or knowledge specific to these problems domains is available, consists of using rule-based methods.

However, in problem domains such as online identity fraud detection or other similar business applications, where the rule bases are submitted to constant changes and usually tend to grow at an accelerated rhythm, rule-based methods may not be suitable, because timely decision is essential in many business environments. In this case, a decision tree represents an effective alternative to rule-base reasoning for a quicker decision.

The Decision Tree is one of the most popular classification algorithms used in data mining and machine learning for creating knowledge structures that guide the decision making process. The creation of a good knowledge structure is the main step in the development of a decision making system. A decision tree or a tree diagram is a specific case of a decision structure used as a predictive model for mapping observations about an object to conclusions about its target class decision. It is used to describe and organize the steps of the decision making process. Among decision support tools, decision trees are simple to understand and interpret. They classify data instances by arranging the data attributes down the tree from the root node to some leaf node. Each node in the tree specifies a test of some attribute of the data instance, and each branch emerging from that node corresponds to one of its possible values [1], [2].

A decision tree represents an effective alternative to rule-base reasoning for quicker decision-making, because in order to be able to make a decision for some situation we need to decide the order in which tests should be evaluated. In that case a decision structure (e.g. decision tree) is much quicker than a rule engine. First, in the decision tree the order of checking conditions and executing actions is immediately noticeable. Second, conditions and actions of decision trees are found on some branches but not on others. Those conditions and actions that are critical are connected directly to other conditions and actions, whereas those conditions that do not matter are absent. In other words it does not have to be symmetrical. Third, decision trees are more readily understood by others in the organization than a set of rules. Consequently, they are more appropriate as a communication tool.

The most common methods for creating decision trees are those that create decision trees from a set of examples (data records). We refer to these methods as data-based decision tree methods. The attribute selection criteria are the essential characteristics in all those methods [3]. These criteria are used to choose the best attributes to be assigned to the nodes of the decision tree. Examples of such criteria include the entropy reduction [4], the gini index of diversity [5], and others [6], [7].

Methods that create decision trees from rules are referred to as rule-based decision tree methods. There is a major difference between building a decision tree from examples and building it from rules. When building a decision tree from rules the method assigns attributes to the nodes using criteria based on the properties of the attributes in the decision rules, rather than statistics regarding their coverage of the data examples [1].

Decision trees can be an effective tool for guiding a decision process as long as no changes occur in the conditions or dataset used to create the decision tree. When the conditions change, as this happens constantly with, for instance, online identity fraud cases, restructuring the decision tree becomes a desirable task. It is difficult, however, to manipulate or restructure decision trees. This is because a decision tree is a procedural knowledge representation, which imposes an evaluation order on the attributes. In contrast, a declarative representation, such as a set of decision rules is much easier to modify and adapt to different situations than a procedural one. This easiness is due to the absence of constraints on the order of evaluating the rules [8].

Rule-based decision tree techniques bridge the divide between rule-base system and decision tree, by allowing the creation on-demand of a short and accurate decision tree from a stable or dynamically changing set of rules. On one hand they easily allow

changes to the conditions (when needed) by modifying the rules rather than the decision tree itself. On the other hand they take advantage of the structure of the decision tree to organize the rules in a concise and efficient way required to take the best decision. So knowledge can be stored in a declarative rule form and then be transformed (on the fly) into a decision tree only when needed for a decision making situation [8].

In addition to that, generating a decision structure from decision rules can potentially be performed faster than generating it from training examples because the number of decision rules per decision class is usually much smaller than the number of training examples per class. Thus, this process could be done on demand without any noticeable delay [4], [9]. Methods that create decision trees from examples or data require examining the complete tree to extract information about any single classification. Even converting the tree into a set of individual rules could also result in a large amount of rules if the tree is large, which could be the case when the tree is based on a large dataset. Otherwise, with methods that create decision trees from rules, extracting information about any single classification can be done directly from the declarative rules rather than from the tree itself after it is built [1].

Although rule-based decision tree methods create decision trees from rules, they could be used also to create decision trees from examples by considering each example as a rule. In contrast, data-based decision tree methods create decision trees from data only. Thus, to generate a decision tree for problems where no data is available and only rules are provided, e.g. by an expert, rule-based decision tree methods are the only applicable solution. Another situation where rule-based decision tree methods could be helpful is after using decision tree pruning methods. Most of the techniques that are used to prune a

decision tree perform the pruning process and still end up with a tree but in the case of the production rules pruning method [10], it ends up with a set of rules with no capability of presenting them in a tree which is the limitation of this method. Considering this kind of situation a possible solution that we propose is to integrate a rule-based decision tree method with the production rule pruning method to convert the production rules into a decision tree.

There are a lot of powerful rule induction techniques for generating rules; however, in some cases their lack of structure causes a readability problem. Besides that, some conflicts could exist among rules that have to be resolved. Thus rules that are not expressed in a comprehensive form will add another burden on the users, causing an even more confusing situation. Here comes the benefit of the rule-based decision tree methods which are capable of organizing the knowledge (rules) in a tree structure, reveal/identify important features, and also filter those unimportant ones. The main idea comes from the decision tree, which is generally accepted to have the following advantages:

- A comprehensive data model: it is easy to represent and interpret.
- It doesn't have conflict knowledge: this is because it transforms conflict of decisions in both data and rules if any, into one decision chosen according to a certain function.
- It identifies and reveals important features of knowledge: by arranging attributes into a top-down structure according to their importance and effectiveness in producing a decision.
- It is also considered a concise and lossless rule representation approach: it organizes rules without any information loss.

1.2 Research Problem

To our knowledge, so far only few rule-based decision tree methods have been proposed in the literature including AQDT-1 [8], AQDT-2 [1], ACT [11], and Akiba *et al.* method [12]. Despite the advantages of the rule-based decision tree methods mentioned in the previous section there are still some important underlying limitations that need to be addressed for effective and efficient decision-making. We summarize these issues in the following points:

- Most of the existing rule-based decision tree methods, such as Akiba *et al.*'s method, and the ACT and AQDT-2 method, depend on both rules and data in the decision tree generation process. In such cases the methods are bounded to data as well as rules.
- Most of these methods are not capable of generating a decision tree using the data only by considering the data as special case of rules.
- Most of these methods are designed based on rules induced from a specific rule induction method. For instance, Akiba's method is based on C4.5 rules, ACT on association rules, AQDT-1 and AQDT-2 on AQ-type rules.
- Most of these methods produce decision trees that are more complex than the decision trees generated by data-based methods.
- Some of the existing rule-based decision tree methods such as AQDT-1 and AQDT-2 generate an unstable decision tree structure. That is, the structure of the final decision tree varies considerably in size each time the tree generation process is repeated for the same original training examples. Naturally, this

syntactical instability is undesirable, for instance, if the methods are applied in knowledge acquisition.

- The time required to generate a decision tree using the existing rule-based methods can be more than the data-based decision tree methods since the generation process is based on both data and rules while the latter methods are based on data only.
- The query complexity of the existing methods is higher due to the more complex decision tree that they produce.

We will be giving more details on the existing rule-based decision tree methods later in our discussion of the related work. The main rationale of this thesis is to develop a new rule-based decision tree method which allows efficient and effective decision-making by addressing the above mentioned limitations.

1.3 Proposed Approach

We propose in this work a new rule-based decision tree method named RBDT-1. We were successful in designing RBDT-1 with properties that overcome some of the research problems that plagued the previous rule-based decision tree methods. We summarize the advantages of RBDT-1 over existing rule-based methods in the following points:

Our proposed method RBDT-1:

- Requires only rules to generate a decision tree.
- Is capable of generating a decision tree from the data only.

- Is more general since it has been shown experimentally that it can be used to generate a decision tree from rules produced by a variety of approaches. These include rules induced from direct rule induction methods such as AQ-based methods, association rule methods such as the Apriori method and indirect rule induction methods specifically through rules extracted from decision trees generated using the ID3 and C4.5 methods.
- Produces a less complex decision tree than the existing rule-based methods. Smaller trees are more efficient both in terms of tree storage requirements and test time requirements and they tend to generalize better for the unseen samples.
- Is capable of summarizing a set of rules into a smaller set of rules extracted from the decision tree it generates without loosing any information. In other words the method is capable of transforming a set of rules into a smaller set of rules represented in a decision tree by removing some of the redundancies without decreasing the accuracy of the rules/tree.
- Is capable of generating a stable decision tree compared to other rule-based methods specifically, AQDT-1 and AQDT-2.
- The query complexity of the proposed algorithm for producing a decision (answering a query) is less than some of the other decision tree generation methods such as AQDT-1, AQDT-2, and ID3. The reason behind that is the capability of RBDT-1 to generate a smaller tree than the others not only in terms of the number of nodes and leaves but most importantly regarding the depth of the tree.

1.4 Method Illustration

In order to illustrate the applicability of the RBDT-1 method in problem-solving, we investigate in this thesis a new approach for online identity fraud detection, specifically application fraud detection.

Identity fraud occurs when a criminal impersonates another individual by taking on that person's identity for whatever reason [13], [14]. The range of information required to take on a person's identity or create a fake one vary greatly. For instance, opening an email account and obtaining a passport require different kinds of pieces of information. To open an email account, an applicant may simply need to provide the name he wants to be known by, a preferred username and a password. In contrast, a passport application requires much more personal information such as a photograph, a signature or some biometric sample [15]. Among all the pieces of identity information that a thief can steal, social security numbers are considered as one of the most sensitive. With these numbers, someone can access all the databases that use the social security number as their primary key [16].

With enough identifying information about an individual, a criminal can impersonate that individual and conduct a wide variety of unlawful activities in his name. These may include fraudulent applications for credit or loans, or Passports, etc. Unfortunately, it takes a lot of time, resources and efforts to undo the damage caused by identity fraud to a victim's finance or reputation.

Identity fraud is spreading fast and causing more and more damages both financially and sociologically. In many ways, the Internet serves as a key vehicle for such threat. The Internet represents an appealing place for fraudsters to collect a host of personal and

financial data related to many innocent users. Using the collected data they can impersonate the users and commit fraudulent activities using stolen or fake identities.

Identity fraud detection approaches fall in two different categories: approaches that focus on fraudulent transactions detection and those that target fraudulent identity application detection. The former category is referred to as transaction-based fraud detection while the latter is known as application-based fraud detection.

For research purposes in transaction fraud detection and prevention, the transaction data is available and easily used. On the other hand, research on application fraud detection and prevention is bounded by the unavailability of real identity application data, due to privacy reasons. Thus obtaining suitable data for carrying and evaluating our work on identity application fraud has been a challenge.

We propose, in this thesis, a new approach for online identity application fraud detection based on rules derived from heuristic and trivial understanding of the concept of fraud. In the proposed approach, the RBDT-1 method plays an important role in organizing the rules into a decision tree for flexible decision making as will be explained later in detail in chapter 5.

1.5 Research Contributions

We make two important contributions in this thesis outlined as follows:

1. We designed a new rule-based decision tree method RBDT-1, that is capable of overcoming several important limitations characterizing existing rule-based decision tree methods as summarized in section 1.3. The new method was published as a regular paper in the International RuleML Symposium on Rule

Interchange and Applications (RuleML09) [17]. The RuleML conference accepted only 25% of the submitted papers for review as regular papers. We also published two papers covering experiments comparing our proposed method RBDT-1 to some existing decision tree generation methods in the International Conference on Computer science and Applications (ICCSA09) [18] and in the International Conference on Machine Learning and Applications (ICMLA09) [19], respectively. We also submitted a paper presenting RBDT-1 along with an analysis of the methods complexity compared to other methods based on the combination of the experiments that we performed to the International Journal of Computational Intelligence (IJCI) that is currently under review [20]. We also prepared a book chapter [21] about RBDT-1 in a book titled as Machine Learning and Systems Engineering as an invitation from the (ICCSA09).

2. We developed a new approach for online identity application fraud detection using RBDT-1 which has been published in the International Journal of Computer and Network Security (IJCNS) [22]. This approach not only is the first unsupervised approach proposed in this area, but has been shown empirically to carry strong detection capability. A study of the impact of Google Hacking on identity fraud, with an emphasis on fraudulent applications as well as an outline of the architecture of our application fraud detector was published in the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (Pacrim 2007) [23].

1.6 Thesis Outline

The rest of the thesis is organized as follows.

Chapter 2 presents some background information on decision tree and a review of the literature covering both data-based decision tree methods and rule-based decision tree methods. We summarize and discuss related work on identity fraud detection, with a particular focus on both transaction fraud and application fraud.

Chapter 3 presents our proposed method RBDT-1, by covering the rule generation methods and notations, the attribute selection criteria, and the proposed decision tree building process. We also cover the pruning methodology adopted by RBDT-1, and analyze and compare the time complexity of RBDT-1 with AQDT-1 and AQDT-2.

Chapter 4 presents the experimental evaluation of the proposed method. Experiments for building decision trees based on both complete and non complete datasets and a comparison of tree complexities for RBDT-1, AQDT-1, AQDT-2, ID3 and C4.5 based on several public datasets are presented.

Chapter 5 introduces how RBDT-1 is applied to the field of identity application fraud detection. In this chapter, some background on application fraud is presented and the relevance of RBDT-1 to application fraud detection is discussed. This is followed by a presentation of the conceptual framework and architecture, and the design and evaluation of our Web-based identity information retrieval engine and the identity fraud detection engine.

Chapter 6 contains a conclusion of the research and contributions, a discussion of research limitations and recommendations for possible future work.

Chapter 2.

Background and Related Work

In this chapter, we give some background information on decision trees, and then we review and discuss previous research on data-based decision tree generation methods as well as rule-based decision tree methods. We also present a review of the literature on identity fraud detection, in which distinction is made between related works on transaction fraud detection methods and works related to application fraud detection.

2.1 Background

A decision tree is a special tree structure constructed to help with making decisions in complex problem-solving. In decision theory and decision analysis, a decision tree is a graph or model of conditions and their possible consequences. Typically a decision tree can be learned by splitting the source set of data or rules into subsets based on an attribute value condition. This process is repeated on each derived subset in a recursive way where each recursion is completed when all data or rules involved in the subset correspond to the same decision, or when splitting no longer adds value to the predictions [24].

A typical decision tree involves the following characteristics:

- The non leaf nodes are labeled with attributes.
- The arcs out of a node labeled with attribute A are labeled with each of the possible values of A.

- The leaves of the tree are labeled with classifications (decisions).

Amongst decision support tools, decision trees have several advantages. For instance, they are simple to understand and interpret as they can be used to break down a complex decision-making process into a collection of simpler decisions, thus providing a solution which is often easier to interpret. Decision trees could be easily extended to handle non numeric domains, where the attributes are categorical rather than numerical. They have value even with little hard data such as data generated based on experts describing a situation for a specific problem. They also can be combined with other decision techniques such as statistical learning methods or other AI techniques (e.g. neural networks) [25].

2.2 Related Work on Decision Tree Methods

The approaches proposed in the literature to create decision trees can broadly be categorized in two different types: data-based decision tree methods and rule-based decision tree methods. In this section, we present a review of the literature on each of these types of methods. We briefly review data-based decision methods and put more emphasis on rule-based methods, which represent the main focus of our work.

2.2.1 Data-based Decision Tree Methods

The most common methods for creating decision trees are those that create decision trees from a set of examples (data records). We refer to these methods in our research as data-

based decision tree methods while they are referred to in the literature as top-down decision tree induction methods.

The idea of top-down decision tree induction methods is that they are used to infer a set of rules from a decision tree generated from a set of labeled data (examples). The attribute selection criteria are the essential characteristics in all those methods [3]. These criteria are used to choose the best attributes to be assigned to the nodes of the decision tree and the data is partitioned based on the values of those nodes. New nodes are created to handle each partition of the data and then a leaf is created based on a stopping criterion. One of the important challenges for decision tree induction methods is the need to produce a small tree in size and depth. Smaller decision trees require less computational time to reach a decision and provide more generalization power [25].

A large number of data-based decision tree methods were designed under the above considerations. These methods are based on defining and using different criteria to divide/partition the data examples at each node of the tree. Examples of such criteria include the information gain and its variants [26], on which two of the classical algorithms such as ID3 [4] and C4.5 [27] are based. Other criteria in use include the gini index of diversity [5], chi-squared statistics [28], [29], exact probability metric [30], the orthogonality metric [31] etc. While any of the above mentioned node splitting criteria are good candidates for generating a decision tree there is no guarantee about which one will produce a smaller tree. This is due to the greedy local decisions made at each node.

There is a group of methods that support the data-based decision tree methods in producing a smaller tree which are the pruning methods. The goal of pruning is to remove sub-trees that have little statistical validity from the tree resulting in a smaller decision

tree. While stopping criteria can be viewed as a pruning method, pruning is usually performed after the decision tree construction. Pruning could also be combined with the decision tree construction itself. Examples of pruning methods include error complexity pruning method [5], minimum error pruning method [32], the critical value method [7] etc. While most of the pruning methods could lead to a smaller tree their computation complexity is high.

2.2.2 Rule-based Decision Tree Methods

There are few published works on creating decision structures from declarative rules. The AQDT-1 method introduced in [8] is the first approach proposed in the literature to create a decision tree from decision rules. The AQDT-1 method uses four criteria for selecting the fit attribute that will be placed at each node of the tree. Those criteria are the cost[1], the disjointness, the dominance, and the extent, which are applied in the same specified order in the method's default setting.

The AQDT-2 method introduced in [1] is a variant of AQDT-1. AQDT-2 uses five criteria in selecting the fit attribute for each node of the tree. Those criteria are the cost[1], disjointness, information importance, value distribution, and dominance, which are applied in the same specified order in the method's default setting. In both the AQDT-1 and AQDT-2 methods, the order of each criterion expresses its level of importance in deciding which attribute will be selected for a node in the decision tree. In both methods the first criterion for choosing the best fit attribute is the attribute disjointness. However,

[1] In the default setting, the *cost* equals 1 for all the attributes. Thus, the *disjointness* criterion is treated as the first criterion of the *AQDT-1* and *AQDT-2* methods in the decision tree building experiments throughout this thesis.

as we will show later, although the disjointness is an important criterion for choosing the best attribute, using it as the first criterion is not the optimal solution for creating the shortest tree.

Another point is that the calculation of the second criterion - the information importance - in AQDT-2 method depends on the training examples, which contradicts the method's fundamental idea of being a rule-based decision tree method rather than a data-based decision tree method. AQDT-2 requires both the examples and the rules to calculate the information importance at certain nodes where the first criterion-disjointness - is not enough in choosing the fit attribute. Thus, without the examples, AQDT-2 might not be able to create the decision tree as will be shown later in the experiments. AQDT-2 being both dependent on the examples as well as the rules increases the running times of the algorithm remarkably in large datasets especially those with large number of attributes.

In contrast the RBDT-1 method, proposed in this work, depends only on the rules induced from the examples, and does not require the presence of the examples themselves. The calculations of all the method's criteria are based on certain characteristics of the attributes intrinsic to the rules only.

Akiba *et al.* [12] proposed a rule-based decision tree method for learning a single decision tree that approximates the classification decision of a majority voting classifier. The method was proposed as a possible solution to solve the issues of intelligibility, classification speed, and required space in majority voting classifiers. In the proposed method, if-then rules are extracted from each classifier which is a decision tree generated using the data-based decision tree method C4.5. The extracted rules are used to learn a

single decision tree. Since the final learning result is represented as a single decision tree, problems of intelligibility and classification speed and storage consumption are improved. The goal of the method is to provide an approximation of the majority voting classifier rather than an exact matching behavior. The proposed method depends both on the real examples used to create the classifiers (decision trees) and on a set of training examples created using the rules extracted from the classifiers. The procedure followed in selecting the best attribute at each node of the tree is based on the C4.5 method as well. The size of a decision tree learned by the proposed method while using rules extracted from multiple classifiers built by C4.5 is about 1.2 to 4.2 times the size of a decision tree learned by C4.5 from the data. As will be shown in the experiments, the RBDT-1 method proposed in this work, when using rules extracted from a C4.5 decision tree, generates a tree that has the same size as the decision tree learned by C4.5 from data, and is even smaller in some of the cases with an equal accuracy.

In [11], the authors proposed a method called Associative Classification Tree (ACT) for building a decision tree from association rules rather than from data. Associative classification usually has a high accuracy, but the rules are not well-organized; they are difficult to understand and full of conflicts. On the other hand, decision tree representation is a very comprehensive way to organize knowledge. However, because of the greedy search strategy of decision tree, its classification accuracy is usually not as high as other approaches. Therefore, the goal of the authors was to use decision tree representation to summarize associative classification rule sets and generate a decision tree classification model that has a better accuracy than a decision tree model.

2.3 Related Work on Identity Fraud Detection

The related work on fraud detection can broadly be categorized into works that focus on fraudulent transactions detection and works targeting fraudulent identity application detection. In the following, we review and discuss existing works under each of these categories.

2.3.1 Transaction Fraud Detection

Credit card fraud is the most common type of identity theft. The number of transactions conducted using credit cards is continuously increasing so is the amount of fraud. A significant amount of research was devoted for developing techniques to analyze credit card transactions in order to compute fraud detectors [33].

Chan *et al.* proposed techniques for combining multiple learned fraud detectors. They developed a meta-classifier that can detect fraudulent transactions by separating fraudulent and legitimate transactions into subsets and applying mining techniques on each subset in parallel. That way, they compute multiple classifiers and then combine these classifiers through Meta learning process to form a meta-classifier [33].

Their goal is to execute learning-agents on remote data in different sites generating classifier agents and then transferring them among the different sites. The agents are implemented and demonstrated in a system that is called JAM for Java Agents for Meta-Learning [33].

Brause *et al.* showed that advanced data mining techniques and neural network algorithms can be combined to obtain high fraud coverage along with a low false alarm

rate. They mine the symbolic data in the credit card transactions in order to generate association rules that describe fraud transactions. The goal here is to combine the rules to come up with shorter and more general rules describing the most common types of fraud transactions. With this approach they are able to diagnose fraud in a more effective way [34], [35].

Aleskerov *et al.* proposed a neural network-based fraud detection approach, which is a part of CARDWATCH, a neural network-based database mining system for credit card fraud detection. The approach is based on the analysis of previous spending data and building a spending profile for each customer that describes his spending pattern. These profiles are used to train a neural network, and then letting the network analyze current spending patterns searching for any anomalies. A credit card institution can use this technique to compare new transactions of a customer against his profile on-line. In this case any suspicious transaction is reported and further checked [36].

Bolton *et al.* developed unsupervised profiling methods that detect fraudulent transactions by comparing any new transaction with previous spending clusters of transactions that they built from the database of transactions. An alarm is triggered if the transaction doesn't belong to any spending behavior cluster [37].

Bose presented an overview of how artificial intelligence technologies such as rule induction, neural networks, case-based reasoning, genetic algorithms and inductive logic programming can be used in the data mining process for discovering fraud in credit card and phone calling transactions. He discussed, by giving appropriate examples, the strengths and weaknesses of these machine-learning techniques. He also outlined that the

financial services industry has benefited from these machine-learning techniques in combating fraud [38].

Kou *et al.* conducted a survey of fraud detection techniques in credit card fraud. The authors outlined the differences between the supervised and unsupervised learning approaches for discovering fraud. They also outlined some products that use neural network techniques in detecting fraud in credit card transactions. The survey also presented computer intrusion and telecommunication fraud detection techniques [39].

Weatherford presented Falcon fraud manager used by a San Diego company. Supervised learning techniques were used to train a neural network for building models of fraudulent credit card usage. The neural network was trained using a large set of transactions including normal transactions and fraudulent ones. Falcon keeps monitoring the accounts of the company to detect credit card frauds and suspicious insurance claims. The merchants are notified immediately when any fraudulent transactions are detected [40].

Note that all the above works are devoted and focused on developing different techniques to discover fraudulent credit card transactions that have already taken place. In contrast in application fraud detection the focus is on studying the identity of the perpetrators by anticipating on their actions.

In [37], Bolton *et al.* define application fraud as the process of obtaining a credit card from a credit card issuer using someone else's identity. We discuss, in the next section, related work on application fraud.

2.3.2 Application Fraud Detection

Although there is a large amount of published works on identity fraud detection, only a few of these works focus specifically on application fraud detection. We review in this subsection representative samples of these works.

Wang *et al.* developed a general framework to analyze identity theft in the context of integrated multiparty perspectives [14]. Although their work is very important, it is mostly conceptual and focuses on identifying the stakeholders involved in identity fraud detection and protection, and on characterizing the interactions between them. According to their framework, there are five main stakeholders: identity owner, identity protector, identity issuer, identity checker, and identity thief. Although we use some of the concepts defined in their framework, our work is different from their work in the sense that we focus primarily on studying, implementing and evaluating a concrete and practical identity checker.

Burns and Stanley in [41] discuss the techniques used by credit card issuers to screen credit applications. Card issuers use multiple data sources (e.g., credit bureaus) to confirm the information listed in an application form. They monitor relevant databases for any changes to the consumer's credit records, personal address or phone number, which often give the earliest indication of identity theft. They also calculate spatial information such as the distance between the phone number and address presented in the application to determine if they originate from the same area code or not. Applications which are submitted from areas where a lot of fraud cases appeared before are also reviewed thoroughly.

Cross-referencing new applications with similar information from other databases is a common characteristic of most of the application fraud detection approaches proposed so far in the literature, including [42], [43] and [44].

Wheeler and Aitken in [42] applied case-based reasoning for credit card application fraud detection. The proposed system was used as reinforcement for an existing rule-based (RB) fraud detector. The input data to the system consists of pairs of database records, consisting in one hand of the application and on the other hand of fraud evidence produced by the RB system. The goal is to reduce the fraud investigations by combining the diagnosis of multiple algorithms in producing the final decision. The proposed case based reasoning framework consists of two decision-making modules, in charge of case retrieval and diagnosis, respectively. The retrieval component utilizes a weighting matrix and nearest neighbor matching to identify and extract appropriate cases to be used in the final diagnosis for fraud. The decision component utilizes a set of algorithms (i.e. probabilistic curve selection, best match algorithm, negative selection algorithm, density selection algorithm, and default goal) to analyze the retrieved cases and attempt to reach a final diagnosis. The results of the system showed that the performance of each algorithm employed in the fraud diagnosis process differs depending on the nature of the fraud case presented. In our work we also use application cross-referencing to capture similarities. However, our fraud detection algorithm does not need labels to make fraud or non-fraud decisions. While in the work of Wheeler and Aitken, the evidence has to be produced and tagged as fraud or non-fraud by a separate system, our proposed system uses unlabelled data in its decision-making process.

Phua *et al.* in [43] proposed a technique for detecting application fraud based on implicit links between new and previous applications. Using a communal suspicion scoring scheme, they classify a new application as belonging to one of three lists, a black list, a white list or an anomalous list. They performed the classification by finding a match between information contained in a new application and information from an existing application in one of the lists. Both the information in the new application and the corresponding linked application were represented by an attribute vector. The suspicious score is the summation of each pair-wise attributes which is expressed by either exact matching or fuzzy matching. They also assigned weights to the attributes depending on their nature. The proposed technique is similar to the weight matrix proposed in [42], except that in [43], in addition to basing their calculation of suspicion scores on matching attributes, they take into consideration temporal and spatial differences between the matching applications. A key difference between this approach and ours is that it requires labeling the data. Despite this important difference, we evaluate our work by mainly comparing it to the approach proposed by Phua *et al.*, as explained later.

In [44], a system that detects subscription fraud in fixed telecommunications was proposed. The system consisted of two main modules, a classification module and a prediction module. The purpose of the prediction module was to detect fraudulent customers when they attempt to subscribe to a fixed telecommunication service. To investigate the application for signs of fraud, the prediction module crosses the information available in the new application with information available in the account database. This allows linking the new application with existing fraudulent accounts.

Another source of information that was used was a public database that lists situations of insolvency mostly related to banks and department stores. The prediction module consisted of a multi-layer feed-forward neural network. The output units indicated one of two decisions for an application; either fraudulent or legitimate. A dataset of subscription examples was labeled and split into the following three sets: a training set, a validation set, and a testing set. From their experiments they outlined that the prediction module was able to identify only 56.2% of the true fraud cases while screening 3.5% of all the subscribers in the testing set. Like in [42] and [43], the proposed framework requires using labeled data and some form of supervision. Many criticisms can be made about using labeled data for fraud detection such as the limited efficiency of processing such data in event-driven environment, the cost, difficulty, and length of time needed to obtain such data, and the fact that the class labels can be inaccurate. In our work, we do not use or assume knowledge of existing fraudulent applications in the screening process. Instead, our proposed fraud detector processes unlabelled data from the identity information source.

In the Identity Angel approach proposed by Sweeney, identity information is extracted from the Web [45] and [46]. The goal of the Identity Angel project is to locate online resumes that contain sufficient information for a criminal to acquire a new credit card in the name of the owner of the resume, and then to notify the corresponding subject by sending him an email, encouraging him to remove such sensitive information.

Identity Angel is implemented using a Java program that uses filtered search and the Google API to identify resumes, and uses entity detectors for SSNs, dates of birth, and email addresses to extract information from online resumes. While the identity

information retrieval technique used by Sweeney is closely related to ours, an important difference is that the search space used in the identity angel project is limited to a specific kind of online documents, which are online resumes. In our case, our search space is the entire web and targets any kind of documents, which gives a wider scope for our knowledge source. Furthermore, the identity angel tool focuses only on information retrieval and does not implement any formal fraud detection process or algorithm.

2.4 Summary

In this chapter we introduced some background information on decision trees, and summarized and discussed related work on different categories of decision tree methods, namely data-based decision tree methods that are the standard decision tree methods and rule-based decision tree methods. We also presented related work on identity fraud detection by covering both transaction fraud and application fraud. In the next chapter, we will present a new method for rule-based decision tree generation which addresses many of the shortcomings of existing methods.

Chapter 3.

The RBDT-1 Method

In this chapter we present RBDT-1, our proposed rule-based decision tree method. The method involves three attribute selection criteria which will be presented in detail along with the steps of pruning and building the decision tree. We will illustrate the application of the method using the week-end problem, which is a publicly available dataset.

Using the same public dataset, we will also conduct a comparative study between RBDT-1 with other rule-based decision tree methods as well as data-based decision tree methods. The results will show that RBDT-1 produces favorable results regarding tree complexity over other existing methods.

3.1 Rule Generation and Notations

In order for our proposed method to be capable of generating a decision tree for a certain dataset, it has to be presented with a set of rules that cover the dataset. The rules will be used as input to RBDT-1 which will produce a decision tree as an output. The rules can either be provided up front, for instance, by an expert or can be generated algorithmically.

Let $a_1,...,a_n$ denote the attributes characterizing the data under consideration, and let $D_1,...,D_n$ denote the corresponding domains, respectively (i.e. D_i represents the set of

values for attribute a_i). Let $c_1,...,c_m$ represent the decision classes associated with the dataset.

The datasets that we use in our experiments are based on classification problems where each example in the dataset belongs to only one class. Thus, a desirable form of a rule-set would be a logically disjoint and complete family of rule-sets. Thus, given a collection of rule-sets, one for each class decision, no two rule-sets for two different classes shall logically intersect and the union of all the rule-sets shall cover the whole dataset. In such a case, each possible example in the dataset will belong to one of the predefined classes. So the decision classes induce a partition over the complete set of rules.

Let P denote the complete set of rules and R_i denote the set of rules associated with decision class c_i. Hence, we have the following: $i \neq j \Rightarrow R_i \cap R_j = \varnothing$, where $1 \leq i, j \leq m$;

$$\mathrm{P} = \bigcup_{1 \leq i \leq m} R_i$$

To illustrate our approach and compare it to existing similar approaches, we use in this chapter two methodologies for generating the set of disjoint rules for each dataset. The first algorithm that we used was to induce rules in an indirect way, where we extract rules from a decision tree generated by the ID3 or the C4.5 method by converting each branch – from the root to a leaf – of the decision tree to an if-then rule whose condition part is a pure conjunction. This scenario will ensure that we will have a collection of disjoint rules. We refer to these rules as ID3-based rules and C4.5-based rules.

In our experiments, we will compare the decision tree generated by our proposed method to the AQDT-1, AQDT-2, the ID3 and the C4.5 methods. Since all the methods under comparison except the ID3 and C4.5 methods are rule-based decision tree methods,

using the same rules extracted from the data-based decision tree itself (ID3-based rules and C4.5-based rules) is one of the best ways to perform a fair comparison. In addition to that it gives us a chance to illustrate our method's capability of producing a smaller tree while using the same rules produced by the data-based decision tree method without reducing the tree's accuracy.

The second methodology that we followed to obtain the rules was by inducing them in a direct way. We mean by a direct way the use of an induction rule program/method that induces rules given a set of data. We used AQ-type rule induction programs as an option to generate rules directly from datasets. AQ-type programs are a family of programs for machine learning and pattern discovery such as the AQ19 [47] and AQ21 [48], which are capable of creating logically disjoint rules. In the experiments performed in this chapter, we use the AQ19 program for creating logically disjoint rules which we refer to as AQ-based rules.

In our experiments, we also used the Apriori algorithm to generate rules from data. The Apriori algorithm is a classic algorithm for learning association rules which associate attribute-values together based on the frequency (support) and precision (confidence) of their occurrence together [49].

3.2 Proposed Method

In this section, we describe the RBDT-1 method by first outlining the required format of the input rules and the attribute selection criteria of the method, summarizing the main steps of the underlying decision tree building process and then finally, we present the pruning technique adopted by the method.

3.2.1 Preparing the Rules

For example, suppose that we have three attributes a_1, a_2 and a_3 with the same domain containing v_1, v_2 and v_3 as possible values.

Let us assume that the following rules correspond to class *c1*:

r1:c1 ← $a_1=v_1$ & $a_2=v_2$

r2: c1 ← $a_1=v_3$

The preparation of these two rules will result in the following formatted rules:

r1: c1 ← $a_1=v_1$ & $a_2=v_2$ & a_3="don't care"

r2: c1 ← $a_1=v_3$ & a_2="don't care" & a_3="don't care"

Each rule is submitted to RBDT-1 in the form of an attribute-value vector. This vector will contain first the values of the attributes that appear in the rule followed by the class decision representing the last element of the vector. Thus, accordingly the previous two rules will be presented as follows:

r1: (v_1, v_2, don't care, c1)

r2: (v_3, don't care, don't care, c1)

3.2.2 Attribute Selection Criteria

The RBDT-1 method applies three criteria on the attributes to select the fittest attribute that will be assigned to each node of the decision tree. These criteria are the Attribute Effectiveness, the Attribute Autonomy, and the Minimum Value Distribution.

3.2.2.1 Attribute Effectiveness (AE)

AE is the first criterion to be examined for the attributes. It prefers an attribute which has the most influence in determining the decision classes. In other words, it prefers the attribute that has the least number of "don't care" values for the class decisions in the rules, as this indicates its high relevance for discriminating among rule sets of given decision classes. On the other hand, an attribute which is omitted from all the rules (i.e. has a "don't care" value) for a certain class decision does not contribute in producing that corresponding decision. So it is considered less important than the other attributes which are mentioned in the rule for producing a decision of that class. Choosing attributes based on this criterion maximizes the chances of reaching leaf nodes faster which on its turn minimizes the branching process and leads to producing a smaller tree.

Using the notation provided above (see section 3.1), let V_{ij} denote the set of values for attribute a_j involved in the rules in R_i, which denote the set of rules associated with decision class c_i, $1 \le i \le m$. Let *DC* denote the 'don't care' value, we calculate $C_{ij}(DC)$ as follows:

$$C_{ij}(DC) = \begin{cases} 1 & if\ DC \in V_{ij} \\ 0 & otherwise \end{cases} \quad (3.1)$$

Given an attribute a_j, where $1 \le j \le n$, the corresponding attribute effectiveness is as follows:

$$AE(a_j) = \frac{m - \sum_{i=1}^{m} C_{ij}(DC)}{m} \quad (3.2)$$

(Where m is the total number of different classes in the set of rules).

The attribute with the highest *AE* is selected as the fit attribute. If more than one attribute achieve the highest *AE* score we will use the next criterion in our method, which is the *Attribute Autonomy* to determine the best attribute among them. Figure 3-1 presents an algorithm for the AE criterion and Figure 3-2 presents an algorithm for finding the candidate attributes.

Algorithm: Calculating The Attribute Effectiveness Criteria *Calculate-AE(A,R)*

Input: a set of rules *R* that correspond to the current branch where a node is to be constructed and a set of attributes *A* that didn't appear before on the current branch.

Output: the attribute that will be assigned to the node under construction.

Steps:

```
Let a_j ∈ A.                                                                    1
Let C be a set containing all class decisions appearing in R and c_i ∈ C.       2
Let m=number of class decisions in R.                                           3
Let R_i be a set containing the subset of rules that corresponds to the class decision c_i.   4
Let V_ij be a set containing the values for aj appearing in Ri                  5
Let Final_attributes =∅, which will contain a list of the attributes that achieved the highest AE.   6
Let DC="don't care"                                                             7

for each a_j ∈ A do:                                                            8
    C_ij(DC)=0                                                                  9
    for each c_i ∈ C do:                                                        10
            if DC ∈ V_ij:                                                       11
                C_ij(DC)= C_ij(DC)+1                                            12
            endif                                                               13
    endfor                                                                      14
     C_ij(DC)= (m-C_ij(DC))/m                                                   15
     AE[j][0]=a_j                                                               16
     AE[j][1]= C_ij(DC)                                                         17
endfor                                                                          18
Final_attributes = Find_Candidate_Attributes(AE, "max")                         19
Counter =0                                                                      20
for attr ∈ Final_attributes :                                                   21
    Counter= Counter+1                                                          22
endfor                                                                          23
if Counter=1:                                                                   24
    return Final_attributes [0]                                                 25
else                                                                            26
   return Calculate_AA(Final_attributes, R)                                     27
endif                                                                           28
```

Figure 3-1. The Attribute Effectiveness Algorithm.

Algorithm: Find_Candidate_Attributes(attr_and_value, condition)

Input: a multidimensional array of attributes and values corresponding to their (AE, AA or MVD) and a condition of "max" or "min" that will determine the bases on choosing the candidate attributes.

Output: a set of attributes that have the minimum MVD, maximum AE or maximum AA values.

Steps:	
Let Final_attributes =∅, which will contain a list of the candidate attributes based on the given condition.	1
Value=attr_and_value[0][1]	2
if condition="min":	3
for attr_value ∈ attr_and_value:	4
if attr_value[j][1] ≤ value:	5
Value = attr_value[j][1]	6
endif	7
endfor	8
elseif condition ="max":	9
for attr_value ∈ attr_and_value:	10
if attr_value[j][1] ≥ value:	11
Value = attr_value[j][1]	12
endif	13
endfor	14
endif	15
for attr_value ∈ attr_and_value:	16
if attr_value[j][1]=value:	17
Final_attributes =final_attributes ∪ attr_value[j][0]	18
endif	19
endfor	20
return final_attributes	21

Figure 3-2. Algorithm for finding the candidate attributes.

We illustrate how we calculate the AE values for some attributes through the following example. Consider that we have the following rules, and we want to choose the attribute with the maximum AE:

c1 ← a_1=v_1 & a_2= "don't care"

c2 ← a_1=v_2 & a_2= v_1

c3 ← a_1="don't care" & a_2=v_2

c3 ← a_1="don't care" & a_2=v_1

c3 ← a_1=v_1 & a_2="don't care"

Table 3-1. AE calculations for attributes A_1 and A_2.

Attribute	c1(DC)	c2(DC)	c3(DC)	ΣC_{ij}(DC)	AE(a_j)
a_1	0	0	1	1	0.67
a_2	1	0	1	2	0.33

Based on the calculations in Table 3-1, the attribute with the highest AE is attribute a_1.

3.2.2.2 Attribute Autonomy (AA)

AA is the second criterion to be examined for the attributes. This criterion is examined when the highest *AE* score is obtained by more than one attribute. This criterion prefers the attribute that will decrease the number of subsequent nodes required ahead in the branch before reaching a leaf node. Thus, it selects the attribute that is less dependent on the other attributes in deciding on the decision classes. We calculate the attribute autonomy for each attribute and the one with the highest score will be selected as the fit attribute. If more than one attribute achieve the highest *AA* score, we will use the next criterion in our method which is the *Minimum Value Distribution* to determine the best attribute among them.

For the sake of simplicity, let us assume that the set of attributes that achieved the highest *AE* score are $a_1,...,a_s$, $2 \le s \le n$. Let $v_{j1},...,v_{jp_j}$ denote the set of possible values for attribute a_j including the "don't care", and R_{ji} denote the rule subset consisting of the rules that have a_j appearing with the value v_{ji}, where $1 \le j \le s$ and $1 \le i \le p_j$. Note that R_{ji} will include the rules that have don't care values for a_j as well.

The *AA* criterion is computed in terms of the Attribute Disjointness Score (ADS), which was introduced by [1]. For each rule subset R_{ji}, let $MaxADS_{ji}$ denote the maximum ADS value and let ADS_List_{ji} denote a list that contains the *ADS* score for each attribute a_k, where $1 \le k \le s, k \ne j$.

According to [1], given an attribute a_j and two decision classes c_i and c_k (where $1 \le i,k \le m; 1 \le j \le s$), the degree of disjointness between the rule set for c_i and the rule set for c_j with respect to attribute a_j is defined as follows:

$$ADS(a_j, c_i, c_k) = \begin{cases} 0 & if\ V_{ij} \subseteq V_{kj} \\ 1 & if\ V_{ij} \supset V_{kj} \\ 2 & if\ V_{ij} \cap V_{kj} \neq (\varnothing\ or\ V_{ij}\ or\ V_{kj}) \\ 3 & if\ V_{ij} \cap V_{kj} = \varnothing \end{cases} \quad (3.3)$$

The *Attribute Disjointness* of the attribute a_j; $ADS(a_j)$ score is the summation of the degrees of class disjointness $ADS(a_j, c_i, c_k)$ as follows:

$$ADS(a_j) = \sum_{i=1}^{m} \sum_{\substack{1 \le k \le s \\ i \ne k}} ADS(a_j, c_i, c_k) \quad (3.4)$$

Thus, the number of *ADS_List* that will be created for each attribute a_j as well as the number of *MaxADS* values that are calculated will be equal to p_j. The $MaxADS_{ji}$ value as

defined by [1] is $3\times m\times(m-1)$ where m is the total number of classes in R_{ji}. We introduce the AA as a new criterion for attribute a_j as follows:

$$AA(a_j)=\frac{1}{\sum_{i=1}^{p_j} AA(a_j,i)} \quad (3.5)^2$$

Where $AA(a_j,i)$ is defined as follows:

$$AA(a_j,i)=\begin{cases} 0 & if \quad MaxADS_{ji}=0 \\ 1 & if \ \left(\left(MaxADS_{ji}\neq 0\right)\wedge\left(\left(s=2\right)\ \vee\ \left(\exists l: MaxADS_{ji}=ADS\_List_{ji}[l]\right)\right)\right) \\ 1+\left[(s-1)\times MaxADS_{ji}-\sum_{l=1,l\neq j}^{s-1} ADS\_List_{ji}[l]\right] & otherwise \end{cases} \quad (3.6)$$

The AA for each of the attributes is calculated using the above formula and the attribute with the highest AA score is selected as the fit attribute. According to the above formula, $AA(a_j,i)$ equals zero when the class decisions for the rule subset examined corresponds to one class. In that case $MaxADS=0$, which indicates that a leaf node is reached (best case for a branch). $AA(a_j,i)$ equals 1 when s equals 2 or when one of the attributes in the ADS_list has an ADS score equal to $MaxADS$ value (second best case).

[2] It is possible to get an attribute with $AA(a_j)$ as infinity (1/0) in this case it has the highest AA.

The second best case indicates that only one extra node will be required to reach a leaf node. Otherwise $AA(a_j, i)$ will be equal to 1 + (the difference between the *ADS* scores of the attributes in the *ADS_list* and the *MaxADS* value) which indicates that more than one node will be required until reaching a leaf node.

As an illustration of how we calculate the minimum AA, consider that we have the following 9 rules, and we want to choose the fit attribute among attributes [a_1, a_2, a_3]:

$c2 \leftarrow a_1 = v_1 \ \& \ a_2 = v_1 \ \& \ a_3 = v_1$

$c1 \leftarrow a_1 = v_1 \ \& \ a_2 = v_3 \ \& \ a_3 = v_1$

$c1 \leftarrow a_1 = v_1 \ \& \ a_2 = v_2 \ \& \ a_3 = v_1$

$c1 \leftarrow a_1 = v_2 \ \& \ a_2 = v_1 \ \& \ a_3 = v_1$

$c1 \leftarrow a_1 = v_2 \ \& \ a_2 = v_3 \ \& \ a_3 = v_1$

$c2 \leftarrow a_1 = v_2 \ \& \ a_2 = v_2 \ \& \ a_3 = v_1$

$c1 \leftarrow a_1 = v_3 \ \& \ a_2 = v_1 \ \& \ a_3 = v_1$

$c2 \leftarrow a_1 = v_3 \ \& \ a_2 = v_3 \ \& \ a_3 = v_1$

$c1 \leftarrow a_1 = v_3 \ \& \ a_2 = v2 \ \& \ a_3 = v_1$

As we can see from the calculations in Table 3-2, the attributes with the highest AA are a_1, and a_2.

Table 3-2. Example of AA calculations

Attribute a_1		
a_1= v1	MaxADS	6
	ADS_List	[6, 0]
a_1= v2	MaxADS	6
	ADS_List	[6, 0]
a_1= v3	MaxADS	6
	ADS_List	[6, 0]
AA(A_1)		0.33
Attribute a_2		
a_2= v1	MaxADS	6
	ADS_List	[6, 0]
a_2= v2	MaxADS	6
	ADS_List	[6, 0]
a_2= v3	MaxADS	6
	ADS_List	[6, 0]
AA(a_2)		0.33
Attribute a_3		
a_3= v1	MaxADS	6
	ADS_List	[0, 0]
AA(a_3)		0.077

Algorithm: Calculating the Attribute Autonomy Criteria ***Calculate_AA(A,R)***

Input: a set of attributes A that achieved the highest AE value and a set of rules R that correspond to the current branch where a node is to be constructed.

Output: an attribute that will be assigned to a node.

Steps:

```
Let a_j ∈ A.                                                                    1
Let v_j1,...,v_jp_j denote the set of possible values for attribute a_j         2
including the "don't care", and let Ra_j be a set of R_ji rules where
R_ji corresponds to the rule subset consisting of the rules that have
a_j appearing with the value v_ji.
Let attr_count=number of attributes in A.                                       3
Let m=number of decision classes in R_ji.                                       4
Let Final_attributes =∅, which will contain a list of the attributes that       5
achieved the highest AA.                                                        6
                                                                                7
for each a_j ∈ A do:                                                            8
    for each R_ji ∈ Ra_j do:                                                    9
            MaxADS=m ×3 × (m-1)                                                 10
            for each a_k ∈ A do:                                                11
              if a_k ≠ a_j:                                                     12
                 ADS_a_k=ADS(a_k, R_ji)                                         13
                 ADS_List=ADS_List ∪ ADS_a_k                                    14
              endif                                                             15
            endfor                                                              16
        endfor                                                                  17
    endfor                                                                      18
    if MaxADS=0:                                                                19
        AA_a_j=0                                                                20
    elseif ((MaxADS≠0) and (attr_count=2)) or (MaxADS∈ ADS_List ):               21
        AA_a_j=1                                                                22
    else:                                                                       23
        Sum_ADS_List=0                                                          24
        for ads ∈ ADS_List:                                                     25
            Sum_ADS_List= Sum_ADS_List+ads                                      26
        endfor                                                                  27
        AA_a_j=1+(attr_count-1) × MaxADS_sum_ADS_List                           28
    Endif                                                                       29
    AA_a_j=1/AA_a_j                                                             30
    AA[j][0]=a_j                                                                31
    AA[j][1]= AA_a_j                                                            32
endfor                                                                          33
Final_attributes = Find_Candidate_Attributes(AA, "max")                         34
Counter =0                                                                      35
for attr ∈ Final_attributes :                                                   36
    Counter= Counter+1                                                          37
endfor                                                                          38
if Counter=1:                                                                   39
    return Final_attributes [0]                                                 40
else                                                                            41
    return Calculate_MVD(Final_attributes, R)                                   42
endif                                                                           43
```

Figure 3-3. The Attribute Autonomy Algorithm.

Figure 3-3 presents an algorithm for the AA criterion and Figure 3-4 presents an algorithm for the ADS criterion.

Algorithm: Calculating the Attribute Disjointness *ADS(a_j, R)*

Input: an attribute a_j and a set of rules R.

Output: the attribute disjointness *ADS* of attribute a_j.

Steps:	
Let C be a set containing all the decision classes in R.	1
Let V_{ij} be a set of values for attribute a_j in R for decision class $c_i \in C$.	2
for $c_i \in C$:	3
for $c_k \in C$:	4
if $c_i \neq c_k$:	5
ADS_aj=0	6
If $V_{ij} \supset V_{kj}$:	7
ADS_aj=ADS_aj+1	8
elseIf ($V_{ij} \cap V_{kj} \neq$ ($\varnothing$ or V_{ij} or V_k)):	9
ADS_aj=ADS_aj+2	10
elseIf ($V_{ij} \cap V_{kj} \neq \varnothing$):	11
ADS_aj=ADS_aj+3	12
endif	13
endif	14
endfor	15
endfor	16
return ADS_aj	17

Figure 3-4. The Attribute Disjointness Algorithm.

3.2.2.3 Minimum Value Distribution (MVD)

The *MVD* criterion is concerned with the number of values that an attribute has in the current rules. When the highest *AA* score is obtained by more than one attribute, this criterion selects the attribute with the minimum number of values in the current rules. *MVD* criterion minimizes the size of the tree because the fewer the number of values of the attributes the fewer the number of branches involved and consequently the smaller the tree will become [1]. For the sake of simplicity, let us assume that the set of attributes that achieved the highest *AA* score are $a_1,...,a_q$, $2 \le q \le s$. Given an attribute a_j (where $1 \le j \le q$), we compute corresponding *MVD* value as follows:

$$MVD(a_j) = |\bigcup_{1 \le i \le m} V_{ij}| \quad (3.7)$$

(Where |X| denote the cardinality of set *X*).

When the lowest *MVD* score is obtained by more than one attribute, any of these attributes can be selected randomly as the fit attribute.

For example, given that:

AA (a_1) = AA (a_2),

The set of values for a_1 in the rules= {v_1, v_2},

The set of values for a_2 in the rules = {v_1, v_2, v_3},

Hence MVD (a_1) < MVD (a_2), the fit attribute =a_1

When the lowest MVD score is obtained by more than one attribute, any of these attributes can be selected randomly as the fit attribute. In our experiments in case where

more than two attributes have the lowest MVD score we take the first attribute. Figure 3-5 presents an algorithm for the MVD criterion.

Algorithm: Calculating the Minimum Value Distribution Criteria ***Calculate_MVD(A,R)***

Input: a set of attributes *A* that achieved the highest *AA* and a set of rules *R* that correspond to the current branch where a node is to be constructed.

Output: an attribute that will be assigned to a node.

Steps:

```
Let a_j ∈ A.                                                                    1
Let V_i be a set containing non-redundant values for a_j in R.                  2
Let Final_attributes =∅, which will contain a list of the attributes that achieved the   3
lowest MVD.
Let random be a random integer from 0 to the number of attributes in final attributes   4

for each a_j ∈ A do:                                                            5
    counter=0                                                                   6
    for each v ∈ V_i do:                                                        7
        counter=counter+1                                                       8
    endfor                                                                      9
    MVD[j][0]=a_j                                                               10
    MVD[j][1]= counter                                                          11
endfor                                                                          12
Final_attributes = Find_Candidate_Attributes(MVD, "min")                        13
Counter =0                                                                      14
For attr ∈ Final_attributes :                                                   15
    Counter= Counter+1                                                          16
endfor                                                                          17
if Counter=1:                                                                   18
    return Final_attributes [0]                                                 19
else                                                                            20
    return Final_attributes [random]                                            21
endif                                                                           22
```

Figure 3-5. The Minimum Value Distribution Algorithm.

3.3 Building the Decision Tree

We describe, in this section, the RBDT-1 approach for building a decision structure from a set of decision rules. In our case the decision structure is a decision tree which is a single-parent decision structure.

Figure 3-6 summarizes the main steps involved in this process. In building a decision tree, the method uses properties of the attributes involved in the decision rules to choose the attributes that will be assigned to the nodes.

We describe, in this section, the RBDT-1 approach for building a decision structure from a set of decision rules. In the decision tree building process, we select the fit attribute that will be assigned to each node from the current set of rules CR based on the attribute selection criteria outlined in the previous section. CR is a subset of the decision rules that satisfy the combination of attribute values assigned to the path from the root to the current node. CR will correspond to the whole set of rules at the root node. From each node a number of branches are pulled out according to the total number of values available for the corresponding attribute in CR. Each branch is associated with a reduced set of rules RR which is a subset of CR that satisfy the value of the corresponding attribute. If RR is empty, then a single node will be returned with the value of the most frequent class found in the whole set of rules. Otherwise, if all the rules in RR assigned to the branch belong to the same decision class, a leaf node will be created and assigned a value of that decision class. The process continues until each branch from the root node is terminated with a leaf node and no more further branching is required.

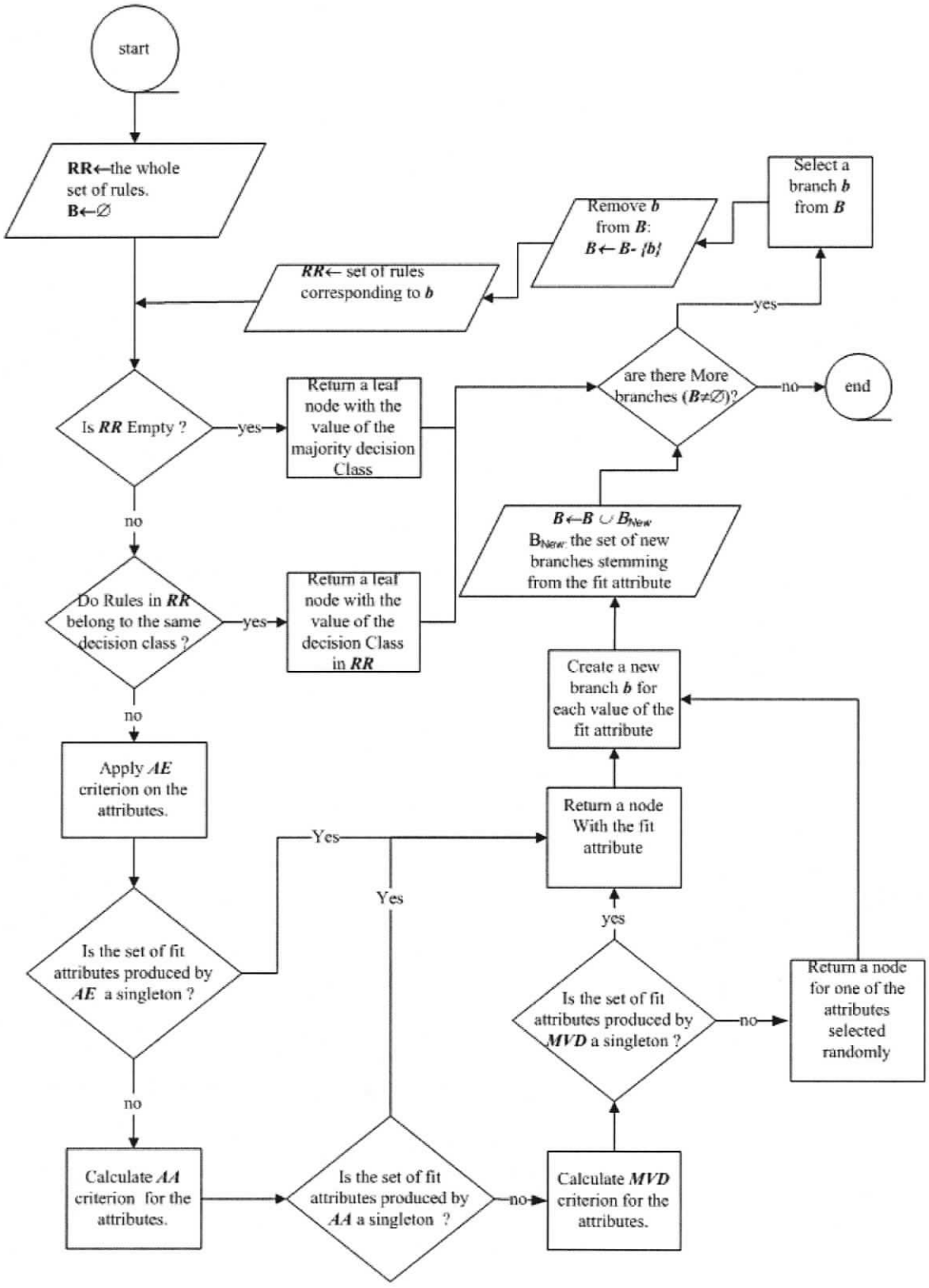

Figure 3-6. Flowchart of the decision tree building process employed by RBDT-1.

3.4 Pruning Decision Rules

Our proposed method not only generates a decision tree from a set of rules but it even could generate a smaller tree from the same set of rules extracted from a bigger tree without decreasing the accuracy (e.g. using the ID3-based rules). RBDT-1 multi criteria method rearranges the attributes in a way that will result in shorter branches (conditions) and/or a smaller number of branches (rules). In doing that, RBDT-1 performs some form of pruning on the rules to a certain extent without decreasing the accuracy of the rules.

On the other hand, another form of pruning is also required for handling the problem of learning from noisy training data. In RBDT-1, we handle noisy data by removing rules that cover only a small portion of the data that could be considered noise which is the same pruning technique adopted by [2]. The examples that were covered by the truncated rules can often be covered by applying an analogical matching procedure. The analogical matching procedure determines the degree of similarity between the examples to be classified and the rules of a given decision class, and selects the best matching decision [50]. In [51] experiments show that such a rule truncation method not only simplifies decision rules which could lead to a simpler decision tree, but could also improve their prediction accuracy in some cases.

In RBDT-1, rules are pruned if their support level is less than or equal to a predefined threshold. The support level of a rule is the percentage of the total number of examples covered by the rule (called the t-weight) to the total number of examples in the given decision class.

3.5 Illustration of the RBDT-1 Method and Comparative Study

In this section the RBDT-1 method is illustrated by a dataset named the weekend problem which is a publicly available dataset, and then compared with the AQDT-1, AQDT-2, ID3 and C4.5 methods in terms of the complexity and accuracy of the decision trees produced.

3.5.1 The Weekend Problem

In this subsection, the steps for generating a decision tree by the RBDT-1 method will be explained in detail using a small dataset called the Weekend problem. The Weekend problem is a dataset that consists of 10 data records listed in Table 3-3. The dataset describes different ways of spending the weekend. We obtained the dataset from an online document used in a lecture on decision trees [52].

Table 3-3. Weekend problem dataset.

# Records	Parents-visiting	Weather	Money	Decision
1	Yes	Sunny	Rich	Cinema
2	No	Sunny	Rich	Tennis
3	Yes	Windy	Rich	Cinema
4	Yes	Rainy	Rich	Cinema
5	No	Rainy	Poor	Stay-in
6	Yes	Rainy	Poor	Cinema
7	No	Windy	Poor	Cinema
8	No	Windy	Rich	Shopping
9	Yes	Windy	Rich	Cinema
10	No	Sunny	Rich	Tennis

The dataset involves the following three attributes: parents-visiting, weather, and money. The parents-visiting attribute has two values (yes and no); the weather attribute has three values (sunny, windy, and rainy) and the money attribute has two values (rich and poor). The decision class is one of four values (cinema, tennis, shopping, or stay-in). We used the AQ19 rule induction program to induce the rule set shown in Table 3-4 which will serve as the input to our proposed method RBDT-1 in this example. AQ19 was used with the mode of generating disjoint rules and with producing a complete set of rules without any truncation.

The weekend rule-set needs to be prepared before being used by the RBDT-1 method for building the decision tree. The rule-set in Table 3-4 is converted into the rule-set shown in Table 3-5.

Table 3-4. The weekend rule set induced by AQ19.

Rule#	Description
1	Cinema ← Parents-visiting="yes" & Money ="rich"
2	Tennis ← Parents-visiting="no" & weather ="sunny"
3	Shopping ← Parents-visiting="no" & weather ="windy" & Money ="rich"
4	Cinema ← Parents-visiting="no" & weather ="windy" & Money ="poor"
5	Stay-in ← Parents-visiting="no" & weather ="rainy" & Money ="poor"

Table 3-5. The weekend rule-set after preparation for RBDT-1.

Rule#	Description
1	Cinema ← Parents-visiting="yes" & weather ="don't care" & Money ="rich"
2	Tennis ← Parents-visiting="no" & weather ="sunny" & Money ="don't care"
3	Shopping ← Parents-visiting="no" & weather ="windy" & Money ="rich"
4	Cinema ← Parents-visiting="no" & weather ="windy" & Money ="poor"
5	Stay-in ← Parents-visiting="no" & weather ="rainy" & Money ="poor"

In order to choose the fit attribute for each node in our tree, we apply the three criteria of the RBDT-1 method on the attributes, in the same order explained in the previous

section. The candidate attribute with the highest AE as shown in Table 3-6, is the parents-visiting attribute. Two branches will be pulled out from the parents-visiting attribute corresponding to its two values, namely yes and no.

A subset of the weekend rule-set will be assigned to the branch where parents-visiting="yes" as shown in Table 3-7. This subset of rules will consist of all the rules with parents-visiting="yes" or "don't care". The corresponding subset contains only one rule with "cinema" as a decision. Thus, a leaf node "cinema" will be created and assigned to the branch where parents-visiting="yes".

Table 3-6. AE calculations for Parents-Visiting, Money and Weather attributes.

	Cinema (DC)	Tennis (DC)	Shopping (DC)	Stay-in (DC)	ΣC_{ij} (DC)	**AE(a_j)**
Parents-Visiting	0	0	0	0	0	1
Money	0	1	0	0	1	0.75
Weather	1	0	0	0	1	0.75

Another subset of the weekend rule-set will be assigned to the branch where parents-visiting="no" as shown in Table 3-8. This subset of rules will consist of all the rules with parents-visiting="no" or "don't care". The AE will be calculated for both the money and weather attributes.

Table 3-7. Subset of rules for branch parent-visiting="yes".

Cinema ← Parents-visiting="yes" & weather ="don't care" & Money ="rich"

Table 3-8. Subset of rules for branch parents-visiting="no".

Tennis ← Parents-visiting="no" & weather ="sunny" & Money ="don't care"
Shopping ← Parents-visiting="no" & weather ="windy" & Money ="rich"
Cinema ← Parents-visiting="no" & weather ="windy" & Money ="poor"
Stay-in ← Parents-visiting="no" & weather ="rainy" & Money ="poor"

Table 3-9. AE calculations for Weather and Money attributes.

	Cinema (DC)	Tennis (DC)	Shopping (DC)	Stay-in (DC)	ΣC_{ij} (DC)	**AE(a_j)**
Weather	0	0	0	0	0	1
Money	0	1	0	0	1	0.75

In Table 3-9, the candidate attribute with the highest AE is the weather attribute. So the weather attribute will be selected as the fit attribute for the branch parents-visiting="no". Three branches will be pulled out from the weather attribute corresponding to its three values: sunny, windy, and rainy. For the branch where parents-visiting="no" and weather="sunny", there is only one rule that corresponds to that branch with a class decision tennis as shown in Table 3-10. So a leaf node with tennis as a decision will be created and assigned to that branch.

Table 3-10. Subset of rules for Parents-visiting="no" & weather="sunny".

Tennis ← Parents-visiting="no" & weather ="sunny" & Money ="don't care"

There are two rules corresponding to the branch where parents-visiting="no" and weather="windy". As shown in Table 3-11, one of the rules corresponds to the decision

class shopping and the other rule corresponds to the decision class cinema. Both rules depend on the value of the money attribute for producing the decision. The money attribute will be chosen as the fit attribute since it is the only candidate attribute left. So a node will be created and assigned the attribute money. Two branches will be pulled out from the money node. A leaf node will be created and assigned the class decision cinema for the branch where money= "poor". Another leaf node will be created and assigned the class decision shopping for the branch where money="rich".

Table 3-11. Subset of rules for Parents-visiting="no" & weather="windy".

Shopping ← Parents-visiting="no" & weather ="windy" & Money ="rich"
Cinema ← Parents-visiting="no" & weather ="windy" & Money ="poor"

Table 3-12. Subset of rules for Parents-visiting="no" & weather="rainy".

Stay-in ← Parents-visiting="no" & weather ="rainy" & Money ="poor"

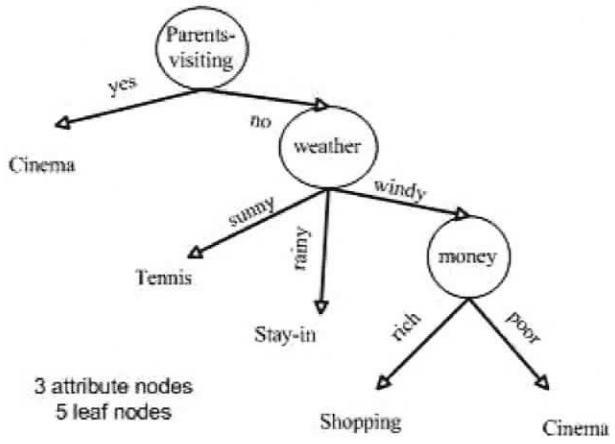

Figure 3-7. The decision tree generated by RBDT-1 for the weekend problem.

For the branch where Parents-Visiting="no" and Weather="rainy", there is only one rule that corresponds to that branch with a class decision stay-in as shown in Table 3-12. So a leaf node with a decision stay-in will be created and assigned to that branch. Overall, the corresponding decision tree created by the proposed RBDT-1 method for the weekend problem is shown in Figure 3-7. It consists of 3 nodes and 5 leaves with 100% classification accuracy for the data in Table 3-3.

3.5.2 Comparative Study

In this subsection, we apply the AQDT-1, AQDT-2, ID3 and C4.5 methods to the weekend problem, and compare the outcomes with the results obtained by the RBDT-1 method outlined in the previous section. Figure 3-8 illustrates the decision tree created using the AQDT-1 and AQDT-2 methods for the weekend problem using the same rule-set in Table 3-5 used by the RBDT-1. The attribute selection process for both the AQDT-1 and AQDT-2 methods identifies the fit attribute according to the score of the attribute disjointness as the first criterion. The attribute disjointness for the weather, parents-visiting and the money attributes in the weekend rule-set in Table 3-5 are [21, 3, 10] respectively. In that case the weather attribute will be selected as the fit attribute because it has the highest attribute disjointness among the rest. Starting with the weather attribute as a root node will result in a more complex tree (i.e. bigger tree) compared to the decision tree created using the RBDT-1 method.

By looking at the decision tree in Figure 3-8, we can see that the test parents-visiting="yes" results in a decision class cinema for all the branches (values) of the weather attribute. In that case starting with the weather attribute will result in redundant branches (bigger tree). On the other hand, starting with the attribute parents-visiting as

the root node as done by the RBDT-1 method avoids the redundancy happening in Figure 3-8, which results in a smaller tree. As explained earlier the parents-visiting was chosen as the root node because it had the highest AE score. The AE criterion which is the first criterion in the method selects the most effective attribute in classifying the data. Since the attribute parents-visiting has no don't care values in any of the class rule subsets, it was considered the most important attribute among the others in contributing to the data classification process.

Using the attribute disjointness as the first criterion for choosing the fit attribute is not the best criterion to start with. It only could be so in some cases as a coincidence when the fit attributes in the tree with the highest AE happen to have the highest ADS as well or when there is a tie between attributes and both method criteria end up choosing the same attributes. Examples of such situation are illustrated later where the same tree is obtained by all three methods (RBDT-1, AQDT-1, and AQDT-2). Although the tree created by AQDT-1 and AQDT-2 methods consists of 5 nodes and 7 leaves which is a bigger tree than the decision tree created by the RBDT-1 method, both have the same classification accuracy.

Although both of the AQDT-1 and AQDT-2 methods produce the same tree, AQDT-2 method from our point of view contradicts the method's idea of being a rule-based decision tree method. This is because the second criterion employed by the method for choosing the fit attribute is the information importance. The training examples will be needed to compute this criterion. Thus, the method is not independent from the examples as claimed. It appears that it depends on both the rules and the examples as well. So given a set of rules, the method would not be able to build a decision tree if the examples used

to induce the rules were unavailable. An example of that, is while building the decision tree for the weekend problem, selecting the fit attribute stemming from the branch weather="windy" required calculating the second criterion for both AQDT-1 and AQDT-2 methods. This is because the attribute disjointness scores for both the parents-visiting and the money attributes in the corresponding rule subset are the same. Both AQDT-1 and AQDT-2 had to compute the second criterion in each method to select the best between the two attributes. AQDT-1 will calculate the attribute disjointness for both attributes which is calculated using the rules. AQDT-2 will need to calculate the information importance for both attributes which is calculated using the data examples along with the rules. Unless the data in Table 3-3 is available, AQDT-2 will be stuck in the middle and will not be able to build the tree. Another remark on AQDT-2 is that because of the reliance on both the examples as well as the rules in the calculations of its criteria, the running time of the method increases for large datasets containing large number of attributes.

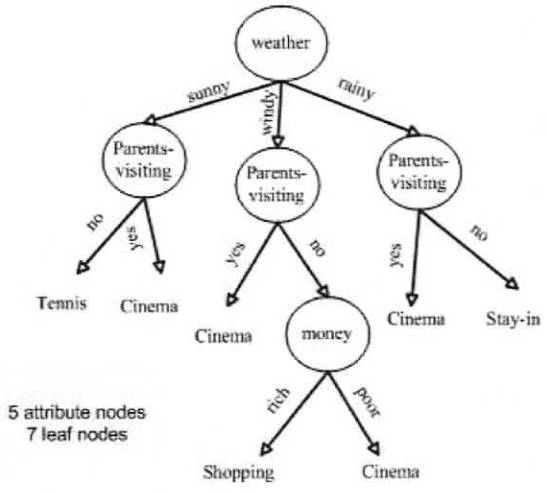

Figure 3-8. The decision tree generated by AQDT-1, AQDT-2 & ID3 for the weekend problem.

Considering the weekend example, using *RBDT-1* to generate a decision tree resulted in the tree in Figure 3-7. When using *AQDT-1* and *AQDT-2* to generate a tree for the same example the result was a bigger tree presented previously in Figure 3-8.

In the weekend example *RBDT-1* was capable of building the tree and choosing the best nodes throughout the tree using only the *AE* criteria.

The following is an illustration of the effect of not starting with the attribute scoring the highest *AE*. Based on the rules induced from the weekend example, the descending order of the attributes scoring the highest *AE* is *parents-visiting*, *weather*, then *money*. If we don't start with *parent-visiting* as the root and instead pick *weather* which has the highest *AA* we will end up with the *DT* in Figure 3-8. This is due to the redundancy of rules caused by starting with the *weather* attribute instead of the *parents-visiting*.

From the tree in Figure 3-8 we find the following rules:

Cinema ← weather ="sunny" & Parents-visiting="yes" & Money ="don't care"

Cinema ← weather ="windy" & Parents-visiting="yes" & Money ="don't care"

Cinema ← weather ="rainy" & Parents-visiting="yes" & Money ="don't care"

The above rules should have been replaced by a single rule as follows:

Cinema ← Parents-visiting="yes" & weather =" don't care" & Money ="don't care"

This only could have happened if we didn't pick the weather over the parents-visiting for the root node in the tree. As a consequence the number of attribute nodes would have been 3 instead of 5 by replacing the three parents-visiting nodes in the second level by only one in the first level (root).

Starting with the weather attribute as the root node could end up with some other DT's such as those shown in Figure 3-9. The DT's in Figure 3-8 and Figure 3-9 all have the same size but with different attributes in the second level.

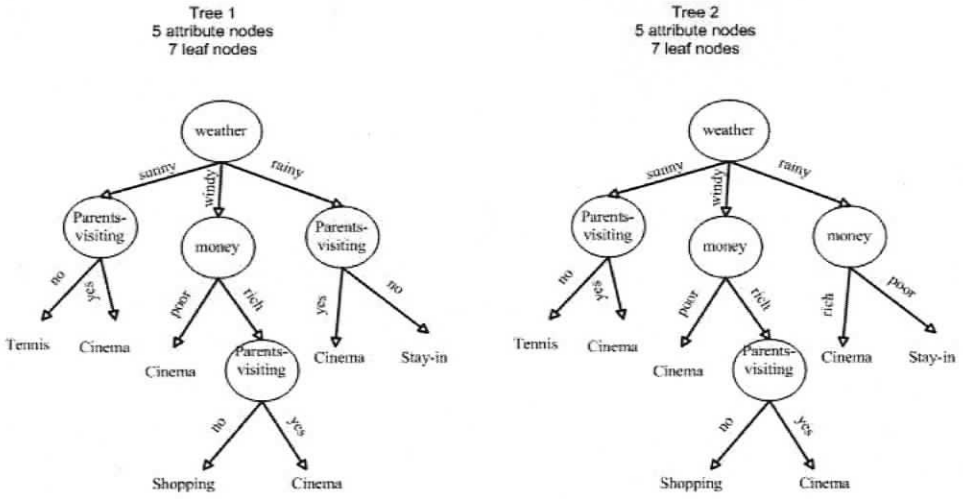

Figure 3-9. Other decision trees generated by AQDT methods for the weekend problem.

After analyzing the different trees we found out that the reason of such different trees is that there was a tie for both attributes (*parents-visiting and money*) in achieving the highest *ADS* score and in the following criteria as well - Attribute Importance in *AQDT-2* and Dominance in *AQDT-1* - and even finally a tie in the *MVD* score. In this case a random attribute is chosen and this is why we get different decision trees. The most remarkable point here is that in this example, as we can see, *AQDT-1 & 2* had to use all criteria in deciding the node to pick, while as mentioned before, *RBDT-1* used only the

AE in building the whole tree and yet produced a smaller tree than *AQDT-1 & 2* which required more effort (criteria) in the building process.

The following is another example illustrating the power of the *AE* criterion. Consider that we started with the parents-visiting as the root node which is the attribute that scored the highest *AE*. The rules for the branch stemming from parents-visiting corresponding to the value "no" will have the *weather* attribute scoring a higher *AE* than the *money* attribute. If we pick the *money* attribute instead of the *weather* attribute we will end up with the tree presented in Figure 3-10, which is still bigger than the optimal tree in Figure 3-7 produced by *RBDT-1*. The tree is bigger also due to the redundancy of paths in the tree.

If we look at the tree, the rules:

Tennis ← Parents-visiting="no" & Money ="rich" & weather ="sunny"

Tennis ← Parents-visiting="no" & Money ="poor" & weather ="sunny"

should have been replaced by a single rule which is:

Tennis ← Parents-visiting="no" & weather ="sunny" & Money ="don't care"

This only could happen if we didn't pick the *money* over the *weather* in the second level of the tree. As a consequence the number of attribute nodes would have been 3 instead of 4 by replacing the two weather nodes in the third level by only one in the second level.

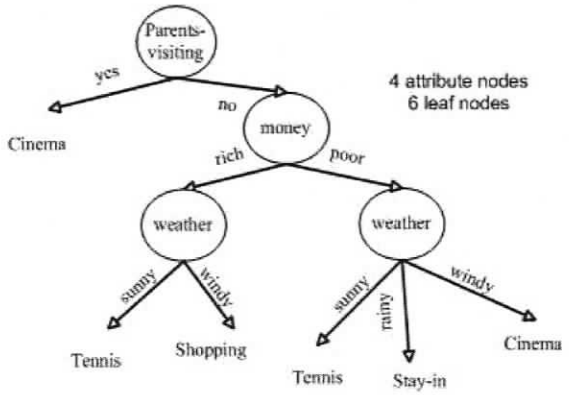

Figure 3-10. Decision tree illustrating the effect of not following RBDT-1criteria.

As indicated above, in RBDT-1, rules are pruned if their support level is less than or equal to a predefined threshold. Figure 3-11 shows a decision tree obtained after pruning the decision rules for the weekend problem in Table 3-5. In this case, we removed the rules with the lowest support level. The decision tree produced by RBDT-1 consists of 2 nodes and 4 leaves and misclassified one example out of 10 giving a predictive accuracy of 90%. Figure 3-11 shows as well the decision tree produced by AQDT-1 and AQDT-2 using the same pruned rules. Although the produced tree has the same accuracy as the tree produced with RBDT-1, it is bigger in size with 4 nodes and 6 leaves.

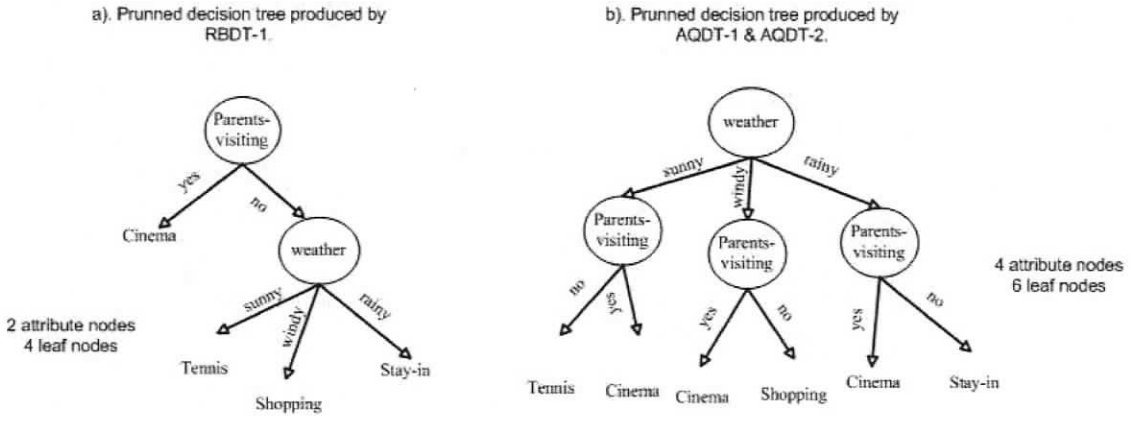

Figure 3-11. The pruned decision tree generated by RBDT-1, AQDT-1 and AQDT-2 for the weekend problem.

We also compared the RBDT-1 decision tree with the decision tree produced by the ID3 method. The ID3 method creates decision trees from data examples by calculating the information gain for each attribute and selects the attribute that has the highest information gain (Quinlan 1979). ID3 method creates the same redundant tree created by the AQDT-1 and AQDT-2 shown in Figure 3-8. Thus, the tree created by ID3 for the weekend problem is also bigger than RBDT-1. This occurs because ID3 employs the information gain which is biased towards choosing attributes with a large number of values. This is based on the assumption that such attributes divide the data examples into subsets that are more likely to be pure (correspond to only one decision class). A special case where the decision trees produced by both ID3 and RBDT-1 are the same occurs by coincidence when the fit attributes selected by the RBDT-1 criteria have the highest information gain as well. Applying C4.5-running under the default setting- to the dataset of the weekend problem resulted in a tree of the same size and accuracy as the pruned decision tree produced by RBDT-1 in Figure 3-9(a).

3.5.3 The Monks1 Problem

As an extended analysis to the performance of *RBDT-1* compared with the *AQDT* methods we will use another example.

We consider the MONKS1 dataset which was one of the datasets used by the authors of *AQDT* methods to prove the effectiveness of their methods over *C4.5* method.

The following analysis was based on rules induced by two different ways: directly and indirectly. As mentioned before, by directly we mean rules induced from a dataset by a rule induction method or program. By indirectly we mean extracting rules from a decision tree built by a DT generation method.

Comparison Based On Direct Induced Rule-Sets

The directly induced rules were based on rule sets produced by the association rule method "Apriori", the AQ19 program and AQ21 program.

The smallest accurate tree that could be produced from the MONKS1 dataset is a tree consisting of 10 attribute nodes and 27 leaf nodes.

The rule set induced from the MONKS1 data set using the Apriori method is composed of 1656 rules and 9678 conditions. Using the association-based rule set, *RBDT-1* was capable of producing a small accurate tree consisting of 10 attribute nodes and 27 leaf nodes which is the optimal tree. The *AQDT* methods produced a tree consisting of 13 attribute nodes and 28 leaf nodes which is bigger than the *RBDT-1* tree. We can see from this example the power of RBDT-1 in transforming 1656 rules into only 27 rules (leaf nodes), and 9678 conditions into 10 conditions (attribute nodes).

By taking a closer look at the root nodes of both trees, we found that for *RBDT-1* the attributes [a, b, e] scored the highest *AE*. Then employing the *AA* to break the tie, attribute '*e*' was eliminated and since both *'a'* and *'b'* attributes have the same *MVD*, the tree either had '*a*' as a root or '*b*'. This as a consequence resulted in the smaller tree.

With the *AQDT* methods, the *'e'* attribute was picked as the root node because it scores the highest *ADS*; the *'e'* attribute as mentioned before was eliminated by the *RBDT-1* method based on the *AA*. In the *AQDT* methods, as a consequence, starting with the *'e'* attribute we end up with a more complex tree.

The rule set induced using *AQ21* consists of 28 rules and 107 attribute nodes. *RBDT-1* based on the *AQ21* rules generates also a tree consisting of 10 attribute nodes

and 27 leaf nodes. The *AQDT* methods produce a tree consisting of 13 attribute nodes and 28 leaf nodes.

The rule set induced by *AQ19* consists of 28 rules and 110 conditions. *RBDT-1* based on the *AQ19* rules generates a tree consisting of 13 attribute nodes and 28 leaf nodes. Here, both *RBDT-1* and the *AQDT* methods produce a similar tree. This is because based on the *AQ19* rules, the attribute with the highest *AE* happened to be attribute *'e'* which has also the highest *ADS* based on the *AQDT* methods. Thus, as we can see the *RBDT-1* builds the shortest accurate tree based on the given rule set.

Comparison Based On Indirect Induced Rule-Sets

The rule sets induced indirectly were extracted from decision trees generated by the *ID3* method and *C4.5* method, respectively, both with and without pruning.

The *ID3* method generates a tree for the MONKS1 dataset that consists of 325 rules and 172 conditions. *RBDT-1* based on the ID3-based rules generates a tree consisting of 13 attribute nodes and 28 leaf nodes. The *AQDT* method produces an unstable tree with different tree syntax that has different sizes. The smallest tree size is 109 rules and 58 nodes which is the same size as the *ID3* tree. As we can see the smallest tree is still bigger than *RBDT-1's* tree.

By having a closer look at the root nodes of both trees, we found that for *RBDT-1* the attribute *'e'* scored the highest *AE* and was picked as the root node. For the second level, attributes [a, b, c, d, e] score the same *AE*. Thus by employing the *AA* to break the tie, attributes [a, b] win and since both attributes *'a'* and *'b'* have the same *MVD*, the tree either had '*a*' as a root or '*b*'. This resulted in the smaller tree.

For the *AQDT* methods, the *'e'* attribute was picked as the root node because it scores the highest *ADS*. For the second level, attributes [a, b, c, d, e] score the same *ADS*. Thus by employing the Attribute Importance for the *AQDT-2* or the *Dominance* for the *AQDT-1* to break the tie, attributes [a, b, d] get eliminated and since both attributes *'c'* and *'f'* have the same *MVD*, picking either result in a more complex tree. From this analysis it is also apparent that *RBDT-1* uses fewer criteria to pick the attributes at each node. This is an indication of how strong each of its criteria is.

C4.5 method, without employing any pruning, results in a tree that consists of 56 rules and 29 conditions. *RBDT-1*, from the non-pruned C4.5-based rules, generated a tree that consists of 38 rules and 19 conditions. The *AQDT* methods generate a tree of the same size as the C4.5 tree. Thus, *RBDT-1* based on the non-pruned *C4.5* rules was capable of producing a less complex tree than *AQDT-1*, and *C4.5* methods.

C4.5 method, when employing pruning, results in a tree that consists of 28 rules and 13 conditions. *RBDT-1* and the *AQDT* methods, from the pruned C4.5-based rules, generated a tree that consists also of 28 rules and 13 conditions which is the same size as the *C4.5* tree.

From these results, we can see that *RBDT-1*, without employing any pruning method, is capable of producing a less complex tree than *C4.5*, and that *C4.5* only produces a similar tree to *RBDT-1* after employing pruning.

Table 3-13 summarizes the comparison between *RBDT-1* and *AQDT* methods based on all the rule sets used. Table 3-14 summarizes the comparison between *RBDT-1* and *ID3* based on all the rule sets used as input to the *RBDT-1*. Table 3-15 summarizes the comparison between *RBDT-1* tree and *C4.5* tree with no pruning based on all the rule sets

used as input to the *RBDT-1*. Table 3-16 summarizes the comparison between *RBDT-1* tree and *C4.5* tree with pruning based on all the rule sets used as input to the *RBDT-1*.

Table 3-13. A comparison between RBDT-1 tree and the AQDT trees.

Rule-set	Less Complex Tree[3]
Association rules	RBDT1-1
AQ21 rules	RBDT1-1
AQ19 rules	RBDT1-1, AQDT
ID3 rules	RBDT1-1, AQDT
C4.5 rules (No pruning)	RBDT1-1
C4.5 rules (with pruning)	RBDT1-1, AQDT

Table 3-14. A comparison between RBDT-1 tree and ID3 tree.

Rule-set	Less Complex Tree
Association rules	RBDT1-1
AQ21 rules	RBDT1-1
AQ19 rules	RBDT1-1
ID3 rules	RBDT1-1
C4.5 rules (No pruning)	RBDT1-1
C4.5 rules (with pruning)	RBDT1-1

[3] The name of the method that generates the smallest tree will appear in the table.

Table 3-15. A comparison between RBDT-1 tree and C4.5 tree with no pruning.

Rule-set	Less Complex Tree
Association rules	RBDT1-1
AQ21 rules	RBDT1-1
AQ19 rules	RBDT1-1
ID3 rules	RBDT1-1
C4.5 rules (No pruning)	RBDT1-1
C4.5 rules (with pruning)	RBDT1-1

Table 3-16. A comparison between RBDT-1 tree and C4.5 tree with pruning.

Rule-set	Less Complex Tree
Association rules	RBDT1-1
AQ21 rules	RBDT1-1
AQ19 rules	RBDT1-1, C4.5 wp[4]
ID3 rules	RBDT1-1, C4.5 wp
C4.5 rules (No pruning)	RBDT1-1
C4.5 rules (with pruning)	RBDT1-1, C4.5 wp

[4] (wp) is short for with pruning.

3.6 Complexity of RBDT-1

We will present in this section the complexity of RBDT-1 method and compare it with the complexity of AQDT methods regarding Query complexity and tree construction complexity.

3.6.1 Query Complexity

Generally for any decision tree the process of constructing the tree is as follows: each node of the tree is labeled by an attribute and the branches from that node are labeled by the possible values of the attribute. The leaves are labeled by the output of the function. The process starts at the root, and works down the tree, choosing to learn the values of some of the attributes based on those already known and eventually reaches a decision.
The *decision tree complexity* of a function is the minimum depth of a decision tree that computes that function. A query cost is the maximum if it requires a number of nodes to be traversed equal to the number of attributes we have in the tree [53].

As illustrated in the weekend example and the upcoming experiments in chapter 4, RBDT-1 produces on average a less complex decision tree than the tree produced by the AQDT methods. As a result the query/classification complexity of RBDT-1 is also less than that of the AQDT methods. As an example, we will consider the decision trees presented in Figures 3-7 and 3-8 for the RBDT-1 and AQDT methods for the weekend problem illustrated in section 3.6.
Assuming that we have the following three queries, about what the decision is when:

- Weather is rainy and parents are visiting?
- Weather is sunny and parents are visiting?
- Weather is windy and parents are visiting?

For RBDT-1, each query will require a depth of one node to be traversed until we reach the decision which is cinema. Thus, the cost of the three queries = 1 node × 3 queries= 3 nodes. For AQDT methods, each query will require a depth of two nodes to be traversed until we reach the decision. Thus, the cost of the three queries = 2 nodes × 3 queries= 6 nodes which is double the query cost of the decision tree produced by RBDT-1.

We also calculated the query/classification cost for RBDT-1 and AQDT methods for the 10 example records presented in Table 3-3. We found that the query cost of AQDT methods is 23% higher than RBDT-1's query cost for the same records in this example. The query cost required (in terms of the number of nodes) for each of the 10 records to be classified by the methods is given in Table 3-17.

Table 3-17. The query cost for the weekend dataset by RBDT-1, AQDT1 & 2.

Record#	RBDT-1 Query Cost	AQDT Query Cost
1	1	2
2	2	2
3	1	2
4	1	2
5	2	2
6	1	2
7	3	3
8	3	3
9	1	2

10	2	2
Total cost	17	22

3.6.2 Tree Construction Complexity

For the calculation of the complexity of the decision tree construction for RBDT-1, consider the following:

- Let r be the number of rules in our rule set R.
- Let a be the number of attributes in R.
- Let K be the size of the largest domain attribute.

The decision tree T generated by RBDT-1 has the following characteristics:

- T is a *K-ary* tree.
- T is a tree with the maximum height = a.
- Each path in T is unique.
- An attribute a_j appears in a path P only once.

The decision tree has two types of nodes: the attribute node and the leaf node. Let c be the computation cost of constructing one attribute node in T, where c is at most $r \times a$. The total computational cost C for constructing T equals the total number of attribute nodes in T times the cost of constructing one attribute node c.

The attribute nodes AN = the total nodes in the trees N – the number of leaf nodes LN.

Thus, $C = r \times a \times AN$. Based on the characteristics of T that we mentioned, and given that for *K-ary* tree [54] the maximum number of nodes N in a tree with the maximum height is $N = \frac{K^{a+1} - 1}{K - 1}$, the maximum number of attribute nodes in T is $AN = \frac{K^{a} - 1}{K - 1}$. Assuming the tree will have the maximum number of attribute nodes, the worst case cost of constructing T by RBDT-1 is $O\left(r \times a \times \frac{K^{a} - 1}{K - 1} \right)$ which is the same as with the AQDT methods.

3.7 Summary

In this chapter we presented in detail the RBDT-1 method. We explained that RBDT-1 used both rules produced directly and indirectly from rule induction methods. We also presented the rule format required by RBDT-1 and explained in detail the attribute selection criteria adopted by the method.

We explained the process of Building and Pruning the decision tree by RBDT-1. We illustrated the different steps of the method using a simple example based on the week-end dataset. We also presented the complexity of RBDT-1.

In the next chapter, we will evaluate the RBDT-1 by comparing it to several existing decision tree method and by using several publicly available datasets.

Chapter 4.

Experiments

In order to evaluate the RBDT-1 method, we conducted some experiments using 16 publicly available datasets summarized in Table 4-1, including the weekend dataset used in the previous chapter. Other than the weekend dataset, all the datasets were obtained from the UCI machine learning repository [56]. First, the evaluation consisted mainly of comparing the RBDT-1 method with the AQDT-1, AQDT-2, and ID3 methods in terms of the complexity and accuracy of the decision trees produced. In view of the fact that the ID3 method doesn't handle datasets with missing values, we used only the complete datasets from Table 4-1 in this experiment which are 12 datasets. In this chapter, we present the experimental evaluation of the RBDT-1 method. Different experiments involving both complete and non complete data sets were conducted by comparing RBDT-1 to several existing decision tree methods. We first describe our evaluation setup and then summarize and discuss the obtained results. Since we were comparing our proposed method to AQDT-1 and AQDT-2 methods which are all rule-based decision tree methods, it was a good idea to compare their performance with rule-sets produced by different methods. Thus, besides using ID3-based rules we conducted an experiment comparing all three rule-based methods along with the ID3 using AQ-based rules generated by AQ19.

We also present a follow-up experiment in which we compare RBDT-1 to AQDT-1, AQDT-2, and C4.5. The input rules to RBDT-1, AQDT-1, and AQDT-2 in this experiment are two sets of C4.5-based rules. In this experiment the whole 16 datasets

summarized in Table 4-1 were used since C4.5 is capable of handling datasets with missing values. Finally we present a comparison between RBDT-1 and AQDT-1 based on an association rules method, namely the Apriori method.

4.1 Settings

We used the PYTHON language for implementing RBDT-1, AQDT-1, AQDT-2, and the rule extractor for the ID3-based rules and the C4.5-based rules. We also used PYTHON to implement a rule format-converter for converting rules produced by AQ19 into the rule format used by our method. An implementation for the ID3 method written in PYTHON was obtained from [56]. For C4.5 and the Apriori method, we used Orange [57], which is a component-based data mining software that includes a module for the C4.5 and the Apriori method as well. The experiments were conducted on a COMPAQ PC (2 Ghz, 512 Mb of RAM) with AMD Sempron 3000+.

4.2 Using Complete Datasets

In this section, we summarize and discuss the results obtained by applying all four methods to the 12 complete datasets. Our evaluation consisted of comparing the decision trees produced by the RBDT-1, AQDT-1, AQDT-2, and ID3 methods for each dataset in terms of tree complexity (number of nodes and leaves) and accuracy. All four methods run under the assumption that they will produce a complete and consistent decision tree yielding 100% correct recognition on the training examples. In the first part of our experiment, the rules used as input to RBDT-1, AQDT-1, and AQDT-2 were extracted from the ID3 decision tree. The whole data examples were used for building the decision tree by the ID3 method. Thus, the extracted ID3-based rules cover the whole set of

examples (100% coverage). The size of the rules is equal to the number of leaves produced by the ID3 tree. Please refer to Table A-1 in the Appendix for details. For 7 of the datasets listed in Table 4-2[5], the RBDT-1 produced a smaller tree than that produced by AQDT-1 & 2 with an average of 274 nodes less or a 29% smaller tree while producing a same tree for the rest of the datasets. Please refer to Table A-2 in the Appendix for details. On the other hand, RBDT-1 produced a smaller tree with 5 of these datasets compared to ID3 with an average of 142.6 nodes less or a 44% smaller tree while producing a same tree for the rest of the datasets.

It is clear based on the results of the comparison that the RBDT-1 method performs better than the other three methods in most cases in terms of tree complexity and achieves at least the same level of accuracy.

AQDT-1 and AQDT-2 methods in some cases such as Monk's 1, balance scale, car, and connect-4 problems led to a syntactical instable decision tree. That is, the structure of the final decision tree varied considerably in size each time the process is repeated for the same original training examples. Naturally, this syntactical instability is undesirable for instance if the methods are applied in knowledge acquisition. This could be illustrated using the MONK'S1 problem. For the MONK's1 problem, 325 ID3-based rules were used as input to the RBDT-1, AQDT-1 and AQDT-2 methods. These rules were extracted from the ID3 decision tree which consisted of 325 leaves and 172 nodes. RBDT-1 method generated a decision tree consisting of 28 leaves and 13 nodes. The AQDT-1 and AQDT-2 methods produced a tree that differs syntactically in size.

---

[5] In Tables 4-2, 4-3, 4-4 and 4-5 that present the results of the comparisons, the name of the method that produced a less complex tree appears under the method column; the "=" symbol indicates that the same tree was obtained by all methods under comparison.

Table 4-1. A summary of the datasets used in our experiments.

Dataset Name	Brief Description	# Records	# Attributes	# Attribute Values	# Class Decisions	Missing Values
Weekend	A dataset Describing different ways of spending the weekend	10	3	2 to 3	4	no
Lenses	A dataset for fitting contact lenses	24	4	2 to 3	3	no
King-Rook vs. King-Knight chess	A dataset for King-Knight-King-Rook chess end-game	2021	16	2 to 3	2	no
Car evaluation	A dataset for car evaluation	1728	6	3 to 4	4	no
Shuttle-Landing-Control	A dataset of a space shuttle auto landing domain	253	6	2 to 4	2	no
Connect-4	A dataset of all legal 8-ply positions in the game of connect-4 in which neither player has won yet, and in which the next move is not forced	67557	42	3	3	no
Nursery	Dataset derived from a hierarchical decision model originally developed to rank applications for nursery schools	12960	8	3 to 5	5	no
Balance Scale	A dataset based on balance scale weight and distance database	625	4	1 to 5	3	no
MONK's	Contains three different datasets ; MONK's1, MONK's2 and MONK's3 which were the basis of the first international comparison of learning algorithms	432 in each MONK's problem	6 in each MONK's problem	2 to 4 in each MONK's problem	2 in each MONK's problem	no
Zoo	A zoo dataset	101	16	2 to 6	7	no
Breast Cancer	Breast Cancer dataset	286	9	2 to 13	2	yes
Lung Cancer	Lung cancer dataset	32	56	3	3	yes
Primary Tumor	Primary Tumor dataset	339	17	2 to 3	22	yes
Voting	1984 United Stated Congressional Voting Records	435	16	2	2	yes

The minimum tree size observed is 109 leaves and 58 nodes. The maximum tree size observed is 172 leaves and 325 nodes, which is the same size for the ID3 tree. The reason

behind this fluctuation in size is the limitation of the Attribute disjointness (ADS) criterion in choosing the fit attribute when the number of attributes to choose from is more than two. Specifically in the MONK's1 problem, when building the tree, there were several nodes where the choice was between more than two attributes, all of which achieved the same ADS score. In which case, they were all considered fit from the criterion's perspective and passed to the subsequent criteria. At certain nodes the attribute that would lead to a smaller tree got eliminated and one of the other attributes was selected randomly.

Table 4-2. Comparison of tree complexities of the RBDT-1, AQDT-1, AQDT-2 & ID3 methods using ID3-based rules.

Dataset	Method
Weekend	RBDT-1
Lenses	RBDT-1, ID3
Chess	=
Car	RBDT-1, ID3
Shuttle-L-C	=
Connect-4	RBDT-1
MONK's 1	RBDT-1
MONK's 2	RBDT-1, AQDT-1 & 2
MONK's 3	=
Zoo	=
Nursery	RBDT-1
Balance	RBDT-1, ID3

Table 4-3. Comparison of tree complexities of the RBDT-1, AQDT-1, AQDT-2 & ID3 methods using AQ-based rules.

Dataset	Method
Weekend	RBDT-1
Lenses	RBDT-1, ID3
Zoo	RBDT-1
Car	RBDT-1
Monk's1	RBDT-1
Monk's2	RBDT-1
Monk's3	=
Chess	=
Balance	RBDT-1, ID3
Shuttle-L-C	RBDT-1, AQDT-1 & 2

Thus, using AQDT-1 and AQDT-2 methods to create a decision tree for the MONK's1 dataset could result in a different tree in terms of size each time (all sizes bigger than the RBDT-1 tree). In contrast, RBDT-1 method as described before calculates the AA criterion to select the fit attribute. In that case although all attributes could achieve the same ADS score, only the attributes with the maximum AA will be selected and passed to the final criterion. This is why the RBDT-1 decision tree was smaller than the other methods.

In our experiments we also compared the decision tree complexity and accuracy of the RBDT-1, AQDT-1, AQDT-2 and ID3 methods using AQ-based rules generated by the AQ19 rule induction program. The rules generated are complete rules with 100% correct recognition on each datasets. Results for some datasets are presented in Table 4-3.

Based on the results of the comparison in Table 4-3, the RBDT-1 method performed better than the AQDT-1 & 2 and the ID3 methods in most cases in terms of tree complexity by an average of 33.1 nodes less or a 17% smaller tree than AQDT-1 & 2 and by an average of 88.5 nodes less or a 37 % smaller tree than the ID3 method. The decision tree classification accuracies of all four methods were equal.

Based on the results of the comparison in Table 4-3, the RBDT-1 method performs better than the AQDT-1 and AQDT-2 methods in most cases in terms of tree complexity and achieves the same level of accuracy while using rules produced by the AQ19 program. Please refer to Table A-3 in the Appendix for details.

We also compared the decision tree complexity and accuracy of the RBDT-1, AQDT-1, and AQDT-2 methods using association rules generated by the Apriori algorithm. Despite the fact that association rules especially those produced by the Apriori algorithm suffer from redundancy causing an increase of the number of rules generated, RBDT-1 was capable of coping with this redundancy to a certain extent and minimizing it into a smaller number of rules represented in a decision tree. Based on the results of the comparison in Table 4-4, the RBDT-1 method performed better than the AQDT-1 & 2 methods in most cases in terms of tree complexity by an average of 57 nodes less than AQDT-1 or 16% smaller tree and by an average of 29 nodes less or an 11% smaller tree than AQDT-2. The decision tree classification accuracies of all four methods were equal.

Based on the results of the comparison in Table 4-4, the RBDT-1 method performs better than the AQDT-1 and AQDT-2 methods in most cases in terms of tree complexity and achieves the same level of accuracy while using association rules produced by the Apriori algorithm. Please, refer to Table A-4 in the Appendix for details.

Table 4-4. Comparison of tree complexities of the RBDT-1, AQDT-1, AQDT-2 methods using association rules.

Dataset	Method
Weekend	=
Lenses	RBDT-1
Car	RBDT-1
Shuttle-L-C	RBDT-1
Nursery	=
MONK's 1	RBDT-1
MONK's 2	RBDT-1
MONK's 3	=
Balance	RBDT-1

4.3 Using Incomplete Datasets

In this section, we assess the strength of our proposed method in handling datasets that have missing values.

In the previous experiments we used two different rule-sets, one extracted from the decision tree created using the ID3 method and another rule set produced by an AQ-type rule induction program. Here, we present an experiment in which we used two different C4.5-based rule sets. The two different C4.5-based rule sets are extracted from two decision trees generated using C4.5 method for each dataset, one with the Orange pruning option turned on and the other one without pruning. Unlike the ID3 method that handles only complete datasets, C4.5 is capable of handling datasets with missing values. Thus,

we were capable of experimenting with both rules extracted from complete and incomplete datasets. The comparison was based on 16 datasets; so accordingly 32 C4.5-based rule sets were extracted and used in this experiment. The rule sets are given as input to the three rule-based methods under comparison.

In Table 4-5, we illustrate the results of the comparison between RBDT-1, AQDT-1, AQDT-2 and C4.5 in two experiments named; experiment1 and experiment2. In each experiment the C4.5 decision tree was generated from the whole set of examples of each dataset and the rules extracted from that tree served as input to the other three rule-based decision tree methods. In experiment1, the pruning option was turned off, while in experiment2 the pruning option was turned on. Based on the results in Table 4-5, AQDT-1 & 2 produce a larger tree than RBDT-1 by an average of 146.33 nodes with the exception of 3 rule sets for which our proposed method is larger by an average of 3 nodes. In addition, the results illustrate that RBDT-1 is at least as effective as C4.5. In experiment2 both methods achieved the same results, while in experiment1 RBDT-1 performed better than C4.5 by producing a smaller tree for the connect-4, nursery and the MONK's 1 data sets. While for the connect-4 and nursery data sets RBDT-1 tree was only slightly smaller than the C4.5 tree, it was 33% smaller than the C4.5 tree for the MONK's 1 data set. In terms of accuracy, the four methods have equal performance which is equal to the classification accuracy of the C4.5-based rules. Please refer to Tables A-5 and A-6 in the Appendix for details.

4.4 Summary

In this chapter we presented several experiments to compare RBDT-1 to other decision tree generation methods, such as ID3, the C4.5, AQDT-1 and AQDT-2 methods. The

experiments was based on 16 public datasets and on rules generated by different rule induction methods. The results presented by the experiments show that RBDT-1 on average creates a less complex decision tree than the other methods.

In the next chapter we will illustrate the application of the proposed RBDT-1 method to an emerging research area, namely the field of *application fraud detection.*

Table 4-5. Comparison of tree complexities of the RBDT-1, AQDT-1, AQDT-2 & C4.5 using C4.5-based rules.

Dataset	Method (Experiment1)	Method (Experiment2)
Weekend	RBDT-1, C4.5	RBDT-1, C4.5
Lenses	RBDT-1, C4.5	RBDT-1, C4.5
Chess	=	=
Car	RBDT-1, C4.5	RBDT-1, C4.5
Shuttle-L-C	=	=
Connect-4	RBDT-1	RBDT-1, C4.5
Nursery	RBDT-1, AQDT-1 & 2	=
Balance	=	=
MONK's 1	RBDT-1	=
MONK's 2	=	=
MONK's 3	AQDT-1 & 2	AQDT-1 & 2
Zoo	RBDT-1, C4.5	RBDT-1, C4.5
Breast-C	=	=
Lung-C	RBDT-1, C4.5	RBDT-1, C4.5
Primary-T	RBDT-1, C4.5	RBDT-1, C4.5
Voting	AQDT-1 &2	=

Chapter 5.

Use of RBDT-1 for Application Fraud Detection

In this chapter, we illustrate the applicability of the RBDT-1 method for real-life problem solving by proposing a new approach for identity application fraud detection which is relatively an unexplored research area. We discuss the challenges involved in identity fraud detection and describe the important role played by RBDT-1 in the design of the proposed application fraud detector. We then present the elements of our fraud detection framework and conduct empirical evaluation by collecting real identity information online and generating synthetic identity fraud samples.

5.1 Background

Identity frauds can be categorized into two different types: transaction frauds and application frauds. Application fraud occurs when an individual or an organization applies for an identity certificate (e.g., passport, credit card etc.) using someone else's identity. Transaction fraud, also known as behavioral fraud, occurs when an identity thief performs some operations or transactions using fake or stolen identity. Most of the research in identity fraud detection has focused so far on credit transactional fraud detection. Limited attention has been paid to application fraud detection, where only few papers have been published so far. Application fraud detection, however, is an important aspect of any sound and global strategy to combat identity fraud. Application fraud detection is a proactive measure that allows early screening of fraudsters, contributing as

a result to cutting down significantly the effort and resource required to detect fraudulent transactions [37].

Application frauds share many of the characteristics of transaction frauds, including the large amount of data processed, the need to make an acceptance or rejection decision in a short time span, and the significant imbalance between normal and fraudulent data (fraud cases represent only a very small fraction of performed operations). These issues have widely been studied in the transaction fraud literature. The open issues concern the differences between application and transaction frauds.

Transactional operations benefit typically from significant historical records collected over time for a single person. Many Artificial Intelligence (AI) techniques can be used to process past usage history and provide fairly accurate fraud detection results. In contrast historical information related to application operations for a specific person is in general very limited if non existent. For instance, there is a significant delay between consecutive applications for passport renewal in most countries; in some countries it takes between 5 to 10 years before having to renew a passport. As a result the background historical data per individual is very limited. Furthermore the application file contains only sparse identity attributes, which by themselves might not be enough to make a proper decision. So application fraud detection requires specific analysis techniques which can compensate for the weak characteristics of the available data.

Another key difference between transaction and application fraud data is the breadth of the identity information covered. While as discussed above, transaction fraud data involve significant depth (at least from historical standpoint), application fraud data have wider scope. Application forms may potentially be linked to a wider variety of data

sources than what typical transactional operations could enjoy. For instance, information contained in a passport application may be crossed with collateral information in many other databases like vital statistics, death records, social security database, birth records etc. In contrast transactional operations typically cover only a limited number of different data sources maintained internally by the issuer or available externally. For instance, for legal and competitive reasons, credit card transactions can be cross-checked against only few external databases such as credit bureaus.

An alternative identity information source that transcends the above restriction is the web. The web is actually a federation of many different identity information data sources. Using the web, it may be possible to cross-check application forms with data from various collateral identity information sources; even though such information might be sparse or useful information could be missing.

The web is one of the richest and diverse sources of identity information. But at the same time it is one of the most challenging to deal with because of the unstructured nature of the data involved. To our knowledge none of the application fraud detection frameworks proposed so far has explored the strength of such data source. All of them use traditional offline or private data sources. In many ways, the Internet serves as a key vehicle for identity fraud. The Internet represents an appealing place for fraudsters to collect a host of personal and financial data related to many innocent users. Using the collected data they can impersonate the users and commit fraudulent activities using stolen or fake identities. Mining Internet data for fraudulent purposes using a search engine is commonly referred to as black hat Google hacking.

5.2 Relevance of RBDT-1 to Application Fraud Detection

In this work, we propose an application fraud detection framework that consists of two main components: an online identity mining module and a fraud detector. Our proposed online identity mining scheme is based on white hat Google hacking, in which identity information is collected through online search targeting a specific individual (i.e. the applicant). The fraud detector is an intelligent unsupervised decision model that analyzes extracted online identity information related to the applicant and crosses such information with information contained in the application form, in order to detect and report possible inconsistencies or anomalies. We designed our fraud detector using rule-based decision tree generation technique specifically RBDT-1 fed with a set of simple heuristics that define general and common understanding of the notion of fraudulent and normal behaviours.

Although various machine learning techniques (supervised or unsupervised) may be used in designing such kind of detector, the lack of genuine fraud data tends to hinder such process. A common challenge of data-mining based fraud detection research is the lack of publicly available real data for model building and evaluation. For privacy and competitive reasons, organizations are reluctant to release data related to fraudulent activities.

In order to compensate for the fact that labels may not be readily available, ideally such techniques should be unsupervised. Although many unsupervised detection frameworks have so far been proposed for transaction fraud detection, none of them apply specifically to application fraud detection. To our knowledge, the application fraud

detection techniques proposed so far in the research literature are either supervised or semi-supervised.

The solution commonly adopted in the industry and in the literature to address this issue is to encode expert knowledge and past knowledge of fraudulent behavior into rule bases. Likewise, the application fraud detection techniques used in commercial applications and in the research literature include rule-based matching of credit application and credit history, fraud matching using black lists (based on previous fraudulent applications), and supervised model based on labeled data. However, due to the fast pace at which new fraud methods are created and used by fraudsters, the rule bases are submitted to constant changes and usually tend to grow at an accelerated rhythm, quickly reaching unmanageable size. This might not be conducive to timely decision, which is required in many business environments. In this case, a decision tree represents an effective alternative to rule-base reasoning for a quicker decision. This is because in order to be able to make a decision for some situation we need to decide the shortest most efficient order in which tests should be evaluated. In that case a decision structure (e.g. decision tree) is much quicker than a rule engine to reach a decision. In the decision tree the order of checking conditions and executing actions is immediately noticeable. Second, conditions and actions of decision trees are found on some branches but not on others. Those conditions and actions that are critical are connected directly to other conditions and actions, whereas those conditions that do not matter are absent. In other words it does not have to be symmetrical. Third, decision trees are more readily understood by others in the organization than a set of rules.

As we mentioned before, rule-based decision tree techniques are the solution to bridge the divide between rule-base systems and decision trees, by allowing the creation on-demand of a short and accurate decision tree from a stable or dynamically changing set of rules. So fraud knowledge can be stored in a declarative rule form and then be transformed (on the fly) into a decision tree only when needed for judging an upcoming application. Thus, RBDT-1 is used for transforming the fraud rules into a comprehensive decision tree once a new application requires a decision.

5.3 Conceptual Framework and Architecture

In this section, we describe basic identity concepts and give an outline of our identity fraud detection framework design.

5.3.1 Identity Concepts

Identity can be defined "as one or more pieces of information that cause others to believe they know who someone is" [33]. We divide such pieces of information into two categories: basic identity information and specific identity information. Basic identity information simply refers to the first name and last name of an individual. Specific identity information refers to additional identifying information other than the basic identity information, such as birth date, social security number, address, credit card number, etc. A targeted search strategy typically consists of using basic identity information to obtain specific identity information for a particular individual.

In [14], the notion of identity certificate is introduced. An identity owner applies for an identity certificate for various purposes in his life, either administrative or business related. Examples of identity certificates include passport, social security card, driver's

license, health card, credit card, digital (security) certificate, etc. Identity issuers, represented by trusted government or private institutions, deliver identity certificates.

An identity certificate usually includes at least one or several of the following pieces of identity information: certificate number, owner's information, the purpose of the certificate (social security or credit card), issuer's information, validity time period, issuer's certification, and owner security information such as his signature or the security number used for credit card. An identity owner is usually characterized by providing at least one or several of the following pieces of identity information: the first and last names, the parents' names with a particular emphasis on mother's maiden name, the address, the date and place of birth and the telephone number.

Although identity issuers have various and more or less sophisticated ways to check the validity of identity certificates, identity certificates still represent the main vehicle used to conduct identity fraud. The delivery and use of identity certificates rely on a chain of trust. For instance, the delivery of a credit card relies on the social security card, which in its turn relies on the passport, which again relies on the birth certificate. This chain of trust can be broken, when a fraudster is able to steal one's identity or to fake a new one. To obtain an identity certificate an individual submits an application to a certificate issuer providing required identification information. In doing so, she makes an identity claim. So, an identity claim occurs when an individual declares a specific identity, for instance, on an official document like a passport application, or a business document like a credit card application.

5.3.2 General Architecture

Our application fraud detection framework consists of three main modules as illustrated by Figure 5-1: the identity information source, the identity information retrieval engine, and a fraud detector. The identity information source can be a single or a combination of several identity data sources such as credit bureaus databases, previous applications databases, vital statistics, or the web.

The identity information retrieval engine queries the identity information source by targeting the identity of the applicant, and feeds the targeted search results into the identity fraud detector. The identity fraud detector crosses and analyzes, using heuristics, the search results with background identity information fetched from the application file (e.g., credit card application) being checked.

The identity information retrieved for a particular individual consists of a collection of identity information patterns. Each pattern may consist of a sequence of identity information pieces such as address, credit card number, mother maiden name, social security number etc. Resulting from a targeted search process, the patterns may belong to different individuals that simply happen to share the same name. So it is necessary to develop a strategy to sort and narrow down the returned patterns for the target individual. The sorted information can then be analyzed to detect possible identity fraud.

Identity fraud detection will consist of screening a particular identity claim for possible inconsistencies with the information in the search results. The system will extract the identity information for the individual found in the search results and check the results against the identity claim, reporting any discrepancy as an anomaly. This may

either lead to a rejection of the application or to further investigations with some follow-up questions.

We will describe in subsequent sections the identity information retrieval engine and the identity fraud detector.

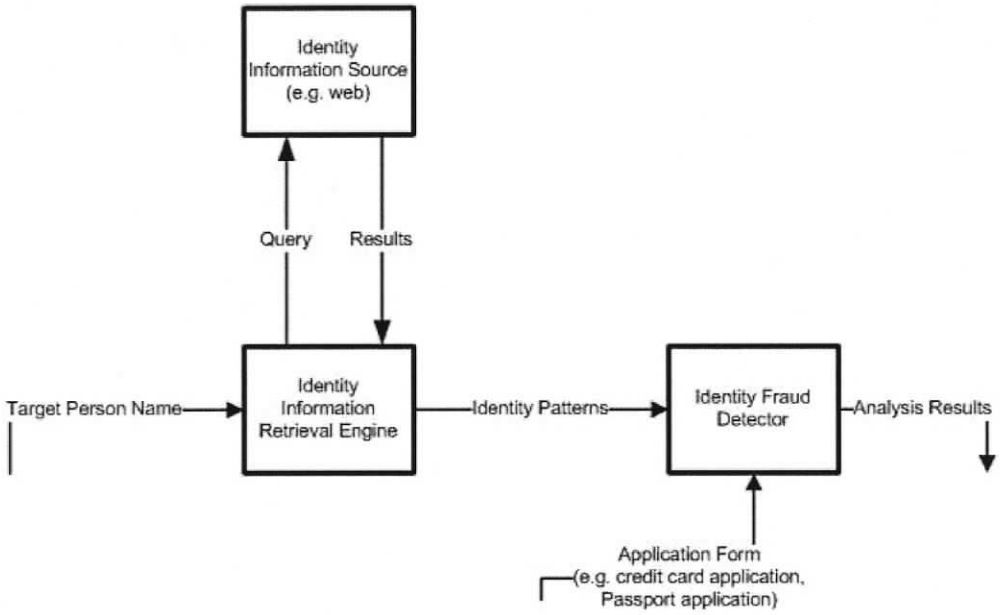

Figure 5-1. Fraud Detection Framework.

5.4 Web-based Identity Information Retrieval Engine

The design of the identity information retrieval engine depends on the identity information source. Unlike with other identity information sources, identity information retrieval on the web involves a lot of challenges because of the scale, complexity, and diversity of the information provided. On the other hand, the web represents a powerful source of identity information that can be used effectively in combating identity fraud.

We describe in this section the challenges and algorithms involved in the design of our web-based identity information retrieval engine.

5.4.1 Design Challenges and Strategies

Although many tools are readily available for information retrieval, identity information retrieval based on targeted search is a challenging task. Because of the huge amount of information available online, locating and sorting identity information related to a specific individual is a daunting task. We discuss in the following paragraphs some of the challenges involved in this task.

A key challenge is that a wide variety of document formats are used on the Internet (e.g., PDF, Doc, Excel, or Html). Some of these formats are not directly searchable, and require some form of pre-processing. Also our search space is not limited to a specific type of documents. As a matter of fact searching and extracting identity information from a document, which has only identity values for the target name such as the curriculum vitae is easier than doing the same with a document that could have more than one person's identity information such as a company employee list. For the latter, the challenge is to avoid extracting identity information for someone different from our target.

Furthermore it is difficult to establish the existence of values corresponding to the identity keywords located in the documents. The reason for such difficulty is that although some of the identity keywords targeted in our search have a fixed number of characters, such as social security number or telephone number, no standard format is used to express corresponding values in the documents on the Internet. So, it is necessary to design a general value filter that is capable of finding and extracting identity values

based on the common ways that people usually tend to express them online. One of the proposed solutions is to limit the search area in the document starting from the target name and ending at the next existing name.

Another challenge is that some of our identity keywords have homonyms. In many cases, sorting information pertaining to different homonyms is challenging. In the literature, one of the main approaches used to handle homonyms consists of associating semantic information with the keyword. The semantic-based approach, however, does not make sense when the homonyms are proper nouns. Since we are dealing, in our work, with targeted search, homonyms based on proper nouns are very important. We handle this kind of homonyms using our identity profile derivation strategy, outlined in Section 5.5, which allows sorting and separating identity information belonging to different individuals who happen to be homonyms.

Besides that, each of the identity information pieces, searched for, has synonyms. Because our search space is the Internet, there is no standard word for expressing these keywords. So we have to come up with all the possible synonyms for each of the identity keywords and expand the search parameters in the identity keyword filter to include those synonyms.

Algorithm: The Identity Information Detector IID
Input: The target person's name n.

Output: A list of documents containing at least one value for any of the keywords detected, along with those keywords and their values.

Uses: Set of search keywords Keys and the web.

Steps:	
let D= ∅, a set that will contain all the documents returned from the search.	1
let D_t = ∅, a set that will contain all the documents returned from the search after converting them into text.	2
for each s ∈ Keys do:	3
D= SearchEngineApi (s+n)	4
endfor	5
for each d_i ∈ D do:	6
D_t= ConvertToText(d_i) ∪ D_t	7
endfor	8
for each d_i ∈ D_t do:	9
ArrayofKeys= KeywordFilter (d_i)	10
KeyValue = ValueFilter (d_t, ArrayofKeys)	11
if KeyValue ≠ ∅	12
result[i][1] = d_i	13
result [i][2] = KeyValue	14
endif	15
endfor	16
return result	17

Figure 5-2. Identity Information Detection (IID) Algorithm.

5.4.2 Search Algorithm

In order to illustrate our approach, we limit (in our proof of concept) our search keywords to only credit card and social security information, which correspond to two of the most popular and sensitive identity certificates available online. We think that this does not affect in anyway the generality of our framework.

Our identity information retrieval scheme will use targeted search based on finding the following specific identity information:

- Social security number

- Date of birth
- Home address
- Home telephone number
- Mother maiden name
- Credit card number
- Credit card expiry date
- Credit card type
- Credit card security number

Through manual search using Google, different synonyms corresponding to the above identity information were learned and were used in implementing our identity information search module.

Table 5-1 depicts the selected keywords. All these keywords will be referred to, in the rest of the chapter, as the identity search keywords.

Figure 5-2 depicts our identity information detection (IID) algorithm. The searching mechanism of the IID algorithm has three phases. The first phase is described in steps 1 through 8. During this phase, first of all the search engine's API is invoked to find web pages containing the name of the target user and at least one of the identity search keywords. In steps 6 and 8, the content of each document returned from the search is converted into text and saved in a text document. This is because the returned documents could be in different formats (e.g., PDF, DOC, XLS, HTML) and converting them into text enables us to easily parse them.

Table 5-1. List of Keywords for Automated Identity Information Retrieval.

Identity Information	Search Keywords
Social security number	[ssn| social security number| social security no| social security #| ssn#| ssnum| ssno]
Date of birth	[date of birth| born| dob| birthplace and date| d.o.b| birthdate]
Home address	[address| add| home| home address]
Home phone number	[Phone | telephone| Phones| home |tel. | tel | ph]
Mother's maiden name	[mothers maiden name |mother's maiden name | mother maiden name | mmn]
Credit card number	[credit card number| creditno| creditnum| ccnum| ccno| cc#| card number| amex| master card]
Credit card expiry date	[credit expire date |expire date| e-date| expdate| expiration date]
Credit card type	[credit card type| card type| Ctype]
Credit card security number	[security number| verification number]

The second phase is described in steps 9 to 11. In this phase, a function named KeywordFilter described in Figure 5-3 is invoked in order to identify the exact list of identity search keywords appearing in each document. The keyword filter is designed while taking into consideration possible synonyms for each of the identity keywords described earlier.

The third phase covers steps 12 to 17. In this phase a (keyword) value filter function named ValueFilter described in Figure 5-4 is used to identify the exact list of values corresponding to the identity search keywords appearing in each document. The value

filter is designed while taking into consideration all the different formats used to express the identity keyword values.

Finally, document names along with a list of identity keywords and values are displayed for only those documents that at least have one value corresponding to an identity keyword. The rest of the documents are discarded.

The prototype of our retrieval engine was implemented and tested using Google API. Figure 5-5 represents a screen shot of one of the files created by our tool as a result of searching for an individual named John Q. Jones. Each block of information corresponds to the identity information found in a single document for this particular individual. Each block of information corresponds to an identity information pattern, which as mentioned earlier will be processed (by the fraud detector) to detect possible inconsistencies with submitted applications by John Q. Jones.

Algorithm: keywordFilter(d)

Input: A document d.

Output: a list Keys of keywords found in the given document.

Uses: Set of regular expressions that returns a keyword if found or ∅ otherwise, and a list of keywords ArrayofK that we are going to search for in the document.

Steps:	
let ArrayofKeys = ∅, a placeholder for the list of keys found in the document	1
for each $k_i \in$ ArrayofK do:	2
KeyFound=KeyRegularExpression(d, k_i)	3
if KeyFound ≠ ∅	4
ArrayofKeys [i]=k_i	5
endif	6
endfor	7
return ArrayofKeys	8

Figure 5-3. Keyword filtering algorithm that uses a set of regular expressions to identify the identity keywords appearing in the document.

5.5 Identity Fraud Detector

We illustrate in this section our proposed identity fraud detection strategy and algorithms. Our identity fraud detection approach consists of two major phases: the derivation of individual profiles and the analysis of the derived profiles using rule-based decision tree. We illustrate each of the two phases in the following.

5.5.1 Shared Identity Information

Let P be the set of identity patterns returned by a targeted search. Each identity pattern $p_i \in P$ is represented by a k-dimensional attribute vector $<p_{i1},..., p_{ik}>$. Possible examples of features include the following:

<Social security number, Date of birth, Address, Telephone number, Mother maiden name, Credit card number, Credit card expiry date, Credit card type, Credit card security number>

We exclude the name simply because at this stage we are dealing with identity information belonging to homonyms. A typical identity pattern returned from a targeted search might include only a subset of the k attributes set. The missing (or undefined) fields will simply be considered non-applicable (NA), and will not be used for the matching.

Algorithm: ValueFilter(d, ArrayofKeys)

Input: A document d, along with the list of keywords ArrayofKeys found in that document.

Output: An array of m rows and 2 columns, each row has one of the keywords found in the document along with its value, or an empty array otherwise if no keyword-values were found.

Uses: Set of regular expressions that returns a value of a keyword if found or ∅ otherwise.

Steps:	
let ArrayofValues = ∅, a list that will contain the keywords found in the document along with their values.	1
for each $k_i \in$ ArrayofKeys do:	2
ValueFound=ValueRegularExpression(d, k_i)	3
if ValueFound ≠ ∅	4
ArrayofValues [i][1]= k_i	5
ArrayofValues [i][2]= ValueFound	6
endif	7
endfor	8
return ArrayofValues	9

Figure 5-4. Value Filtering algorithm that uses a set of regular expressions to identify the values for the identity keywords appearing in the document.

Figure 5-6 illustrates sample identity patterns. In this example the following five identity attributes are considered: social security number (ssn), date of birth (dob), mother maiden name (mmn), address (addr) and phone number (tel).

The retrieval engine uses a name and a set of keywords to search the identity information source (e.g., the Web), and returns a collection of identity information related to the identity claim corresponding to the application being checked.

As a result the returned identity information may correspond to several different individuals, all having the same name. For instance, we can have in the search result several social security numbers corresponding to the same name, which necessarily means that different individuals actually share the searched name. But this could also

mean that the same person is impersonating all these different individuals by simply changing some of the identifying attributes. For the purpose of fraud detection, we need to identify and isolate the trail of identity patterns that relate to the individual submitting the application. We determine the patterns related to the applicant by determining direct and indirect connections between the application pattern and the patterns retrieved from the identity information source.

```
JOHN Q. JONES

-------------------------------------------------
the keywords and their values found in the file named: html42.html
born
this could be a date of birth value 07 23 1832
1
-------------------------------------------------
the keywords and their values found in the file named: html43.html
born
this could be a date of birth value 03 23, 1944
1
-------------------------------------------------
the keywords and their values found in the file named: html44.html
0
-------------------------------------------------
the keywords and their values found in the file named: pdf25.txt
this could be a telephone number value (858) 534-3750
1
-------------------------------------------------
the keywords and their values found in the file named: pdf26.txt
0
-------------------------------------------------
the keywords and their values found in the file named: doc4.doc
social security number
this could be a social security number value 123-45-5678
date of birth
this could be a date of birth value 06/01/1993
home address
phone
this could be a telephone number value 410-272-1234
1
-------------------------------------------------
the keywords and their values found in the file named: html45.html
social security number
this could be a social security number value 123-45-6789
phone
this could be a telephone number value (410) 306-0229
1
-------------------------------------------------
the keywords and their values found in the file named: pdf27.txt
social security number
this could be a social security number value 123-45-6789
phone
this could be a telephone number value (410) 306-0229
1
-------------------------------------------------
the keywords and their values found in the file named: html46.html
ssn
this could be a social security number value 123456789
this could be a telephone number value 907-345-1234
1
```

Figure 5-5. Sample search results returned by our tool for an individual named John Q. Jones. Each block of information represents an identity information pattern in our framework.

Two patterns are directly connected if they share at least one attribute value. Two patterns p and q have an indirect connection, if there is a sequence of directly connected

patterns linking them. In other words, there exists a sequence of patterns p1… pn, such that (p, p_1), (p_n,q), and (p_i, p_{i+1}), $\forall i \in \{1, \ldots, n-1\}$, are each a direct connection.

For the purpose of fraud detection, we trim the returned set P of identity patterns, and retain only the patterns having direct or indirect connections with the application pattern; let P^+ denote the subset of P containing all such patterns. Let p_0 denote the application pattern and let G denote a set of patterns such that $G = P^+ \cup \{p_0\}$.

As an example, let's consider the sample patterns depicted by Figure 5-6. Figure 5-7 depicts a tree structure exposing the direct and indirect connections between the sample patterns shown in Figure 5-7. We assume in this example that p_0 is the application pattern and the set of retrieved patterns is P, P= {p_1, p_2, p_3, p_4}. The set of connected patterns is G, G= {p_0, p_1, p_2, p_3}. For instance, patterns p_0 and p_2 share two attributes, namely ssn and mmn. Pattern p_4 has no connection (direct or indirect) with p_0, so it is excluded from the fraud analysis. It is assumed in this case that p_4 does not belong to the applicant but belongs to another person that shares his name.

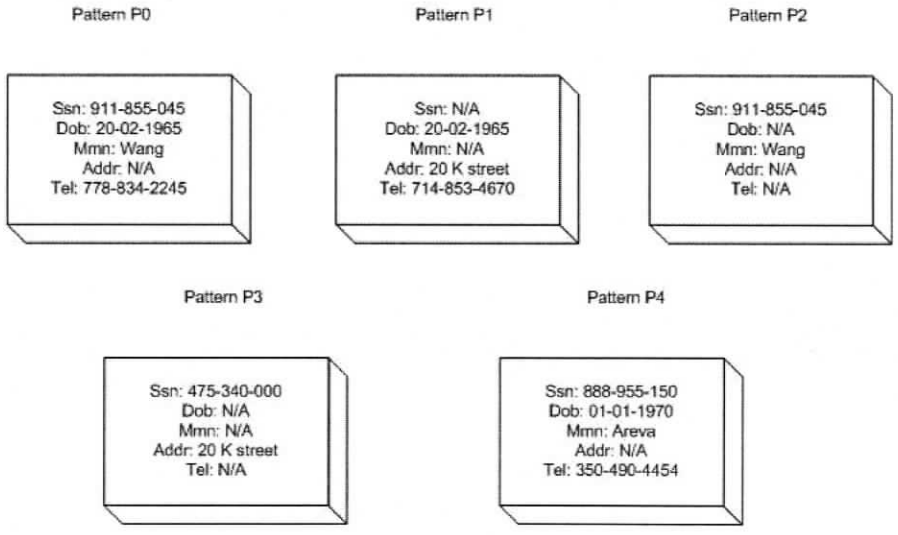

Figure 5-6. Examples of identity patterns based on 5 attributes.

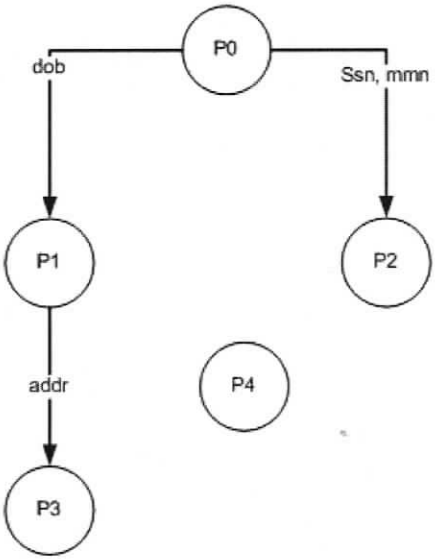

Figure 5-7. Direct and indirect connections between identity patterns.

For the purpose of fraud detection, we analyze the link between every pattern-pair from G which are directly connected. Specifically we convert every pattern-pair that are directly connected into a single feature vector characterizing the underlying relationship. For each pattern-pair $< p_i, p_j > \in G \times G$, we derive a feature vector $v_{ij} = [\delta_{ijl}]_{1 \le l \le k}$, such that:

$$\delta_{ijl} = \begin{cases} ? \ if \ ((p_{il} = na) \ or \ (p_{il} = na)) \\ 1 \ if \ p_{il} = p_{jl} \\ 0 \ otherwise \end{cases} \qquad (5.1)$$

Where "?" denote a missing value, which may be either 1 or 0, but this is unknown. The derived feature vector captures numerically the existence (or lack) of shared identity information in the corresponding pattern-pair. Let V denote the set of all the feature vectors derived from G. The derived set of feature vectors corresponding to the previous example (see Figure 5-7) is as follows:

$$V = \left(v_{01} = \begin{vmatrix} ? \\ 1 \\ ? \\ ? \\ 0 \end{vmatrix}, v_{02} = \begin{vmatrix} 1 \\ ? \\ 1 \\ ? \\ ? \end{vmatrix}, v_{03} = \begin{vmatrix} 0 \\ ? \\ ? \\ 0 \\ ? \end{vmatrix}, v_{12} = \begin{vmatrix} ? \\ ? \\ ? \\ ? \\ ? \end{vmatrix}, v_{13} = \begin{vmatrix} ? \\ ? \\ ? \\ 1 \\ ? \end{vmatrix}, v_{23} = \begin{vmatrix} 0 \\ ? \\ ? \\ ? \\ ? \end{vmatrix} \right)$$

5.5.2 Fraud Detection

We present, in this subsection, our fraud detection approach and the initial rule base used along with the RBDT-1 method.

5.5.2.1 Approach

Our fraud detection approach consists of analyzing the coherence or consistency of the shared identity information between patterns. For instance, two identity certificates attributed to the same individual are expected to bear the same birth date and mother maiden name, when such information is available. The feature vectors describing patterns connections are used as basis for such analysis. As an outcome of the analysis, patterns connections are classified in one of four categories: normal, suspicious-low, suspicious-high, or fraudulent represented by the labels N, S-, S+, and F, respectively. A normal connection is a connection for which no anomaly has been found. We consider two strains or levels of suspicion, suspicious-low and suspicious-high that refer to unlikely and highly unlikely situations, respectively. A suspicious connection may potentially involve some fraud; the judgment is not definitive, and as such it calls for further inspection. In contrast, a fraudulent connection is a profile involving definitely some fraud; the judgment here is final.

Our fraud detector is implemented as a decision tree that takes as an input pattern-pair feature vectors, and provides as outputs fraud decisions. The tree is built and updated dynamically based on a rule base which encodes expert knowledge or common sense understanding of the notion of fraudulent or inconsistent behaviour. No previous fraud instances should be required. The size of the decision tree resulting from such process is expected to be manageable because typically application forms involve sparse and limited identity attributes.

We use our proposed rule-based decision tree method RBDT-1 to transform the set of rules into a decision tree, which is updated dynamically as more rules are found.

5.5.2.2 Rule Base

Based on the five attributes patterns considered in our previous example, we derived 82 rules for the initial implementation of our proof concept. Table 5-2 illustrates a decision table for a sample of the rules. In the rules, "yes" corresponds to $\delta_{ijl} = 1$, "no" corresponds to $\delta_{ijl} = 0$, "na" corresponds to $\delta_{ijl} = ?$ and * is a placeholder for all possible values of δ_{ijl} (i.e. 1, 0, "?").

While some of the rules are straightforward, many require some explanations. A straightforward fraud case is when the social security numbers match and either the dates of birth or the mother maiden names do not such as rules #5, #6 and #7 in Table 5-2. Some other straightforward cases of fraud are when the home telephone numbers match while the addresses do not such as rule #8.

In both cases one could assume that the same individual is impersonating two different individuals: in the first case by changing either the mother maiden name or the

date of birth, and in the second case by using different locations. Two straightforward normal cases are when either none of the attributes match or all the attributes match such as rules #10 and #15, respectively. We assume in the case when none of the attributes match that the non-matching patterns belong to different individuals and have not been used by the applicant (in some past attempt to defraud the system). Obviously, the case when all the attributes match is regular. A variant of this case is when the telephone numbers do not match; this is considered normal because it could simply be the case that the same individual owns several telephone lines.

According to the degree of rarity the case is classified as either highly suspicious or lightly suspicious. For instance, considering that we are dealing with homonyms, a case where the social security numbers do not match, while the addresses, dates of birth, and mother maiden names match, such as rule #4, is extremely unusual. Because this simply means that there are two different individuals (different ssn), who are homonyms, born the same day, from homonym mothers and living at the same location. This is highly suspicious and requires further verification. If these two individuals (born on the same day with the same mother maiden name) were living in different locations, with different phone numbers, then the situation could be considered as a coincidence, although still uncommon. We classify such occurrence as lightly suspicious. Further investigation would also be required in this case as well, but such investigation does not have to be as thorough as in a highly suspicious case.

Table 5-2. Sample of the rules in the rule-base.

Attributes / Rule#	ssn	mmn	dob	add	tel	decision
1	no	yes	yes	na	na	S-
2	no	yes	yes	na	no	S-
3	na	no	yes	yes	*	S-
4	no	*	yes	yes	*	S+
5	yes	*	No	*	*	F
6	yes	no	yes	*	*	F
7	yes	no	Na	*	*	F
8	*	*	*	no	yes	F
9	no	*	*	na	yes	F
10	no	no	*	no	no	N
11	na	no	*	no	no	N
12	no	na	*	no	no	N
13	na	na	*	no	no	N
14	no	no	*	na	no	N
15	yes	yes	yes	yes	yes	N

The decision tree obtained, by applying the RBDT-1 method to the rule base, is illustrated in Figure 5-7. The resulting tree consists of 57 rules; the distribution of the different decisions is summarized in Table 5-3.

Table 5-3. Rules distribution summary for the RBDT-1 decision tree.

Decision	#Rules
Normal (N)	39
Fraud (F)	10
Suspicious-Low (S-)	7
Suspicious-High (S+)	1
Total	57

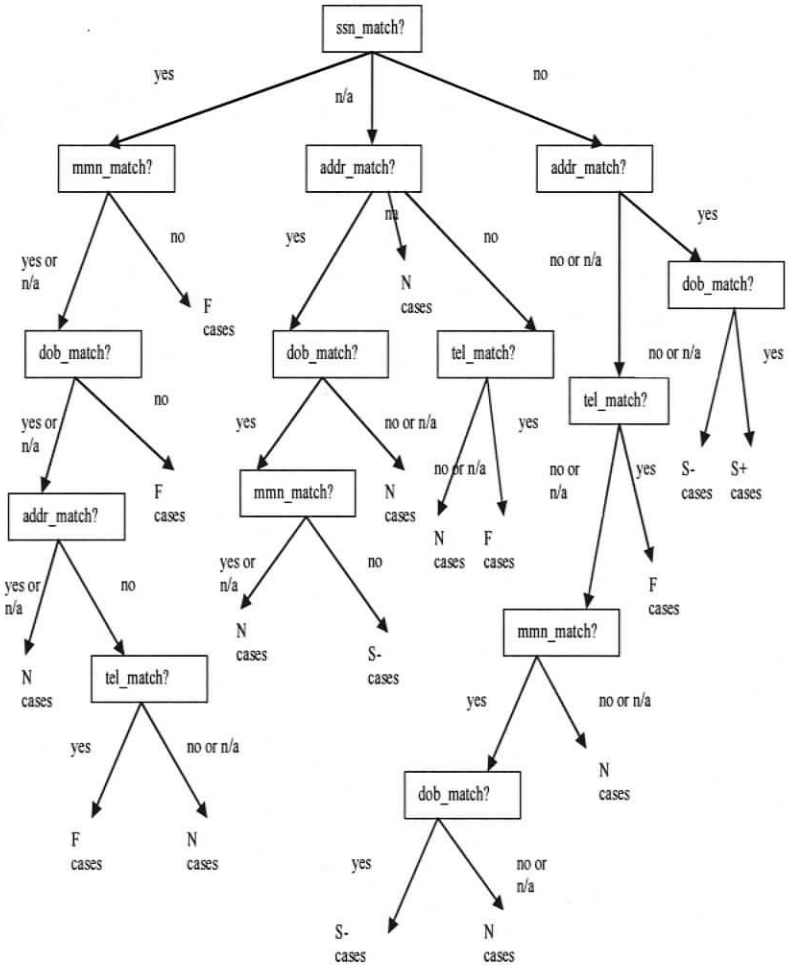

Figure 5-8. The decision tree produced by RBDT-1.

5.6 Evaluation

We present, in this section, the evaluation of the different components of our fraud detection framework. We start by presenting the evaluation method and data collection, and then we present and discuss the evaluation results obtained for the identity information retrieval engine, followed by the fraud detector.

5.6.1 Evaluation Method and Data

The most common way of evaluating frameworks like the one proposed in this thesis consists of finding and using some public dataset containing real identity fraud information. As expected, however, the evaluation of our framework faces difficulty in obtaining real data to effectively analyze and test our system. Usually data containing real identity (fraud) information is restricted from being used due to privacy issues. An alternative approach may consist of generating synthetic fraud data. In this work, we collected real identity information online and then generated and injected synthetic fraud information in the data. Having, however, a dataset is not enough to carry out the evaluation. We need an unbiased mechanism to label the data. In the absence of a domain expert to label the data, we have decided to adopt a comparative labelling mechanism in this work. More specifically, we used the communal scoring approach by Phua *et al.* described in the related work to produce an initial set of labels, and then we compared the produced labels to the labels obtained while using our proposed approach. Using such comparative evaluation allows us to assess the relative strength of our approach.

To obtain real identity information, we conducted some white hat Google hacking experiments over two weeks. During this study, we searched manually for identity information using Google. We used different words that express identity information such as the social security number and searched in different types of documents, and were able to collect sensitive identity information for living as well as for dead persons. The study allowed us to establish a database of online identity information related to 154 different individuals, which serve as basis for the validation of our identity fraud detection framework. Details of the study and corresponding results can be found in [22].

5.6.2 Evaluation of the Information Retrieval Engine

In order to evaluate our search algorithms, described earlier, firstly, we used Google search engine to manually search for pages containing identity information for only 70 different names out of the 154 names identified in our initial manual search described in the previous section. Those 70 names were the subset of the 154 names for which identity information were still available online at the time of this evaluation; the information corresponding to the remaining names were removed between the time of the initial data collection and the online evaluation reported here. In our search query we used the search identity keywords listed in Table 5-1 along with each of the 70 names.

The search results for each name were classified into two categories:

- Keyword-only documents, which are documents that contain some keywords but no values for any of them.
- Keyword-value documents, which are documents that contain at least one value corresponding to an identity keyword.

Secondly, we ran our identity information retrieval tool for each of the 70 names used in the previous manual search. The results of each run consist of document names along with a list of identity keywords and values displayed only for those documents that contain at least one value corresponding to an identity keyword. The numbers of such documents, which are categorized as Keyword_Value documents, as well as the execution time that each search takes are also displayed in the output.

These experiments were conducted on a COMPAQ PC (2 Ghz, 512 Mb of RAM) with AMD Sempron 3000+ and the programs were written using PYTHON language.

We used three metrics to assess the performance of our automated information retrieval scheme, namely recall, precision and false alarm. In the context of our identity information retrieval scheme, these metrics can be defined as follows:

Recall: is the proportion of the correct Keyword_Value documents retrieved by the application from the set of total Keyword_Value documents calculated in the manual search. Recall can be calculated as follows:

$$Recall = \frac{Proportion\ of\ Correct\ Keyword\_Value\ documents\ calculated\ by\ the\ tool(CKVT)}{Total\ number\ of\ Keyword\_Value\ documents\ calculated\ manually(KVM)} \quad (5.2)$$

Precision: is the proportion of the correct Keyword_Value documents retrieved by the application from the set of total Keyword_Value documents claimed in the application output. Precision can be calculated as follows:

$$Precision = \frac{Proportion\ of\ Correct\ Keyword\_Value\ documents\ calculated\ by\ the\ tool(CKVT)}{Total\ number\ of\ Keyword\_Value\ documents\ calculated\ by\ the\ tool(KVT)} \quad (5.3)$$

False Alarm: is the proportion of the false Keyword_Value documents retrieved by the Identity Information Detector (IID) from the set of total Keyword_Only documents calculated in the manual search. False alarm can be computed as follows:

$$False\ Alarm = \frac{Proportion\ of\ False\ Keyword\_Value\ documents\ calculated\ by\ the\ tool(FKVT)}{Total\ number\ of\ Keyword\_Only\ documents\ calculated\ manually(KOM)} \quad (5.4)$$

We calculated the recall, precision, and false alarm rates for each of the 70 names, based on corresponding search results. Overall we obtained the following results:

- an average recall of 97.16% where the maximum recall was 100% and the minimum was approximately 78%,
- an average precision of 81.82 % where the maximum precision was 100% and the minimum was approximately 34%,
- an average false alarm of 13.33% where the worst false alarm rate was approximately 45% and the minimum was 0%,.

In general these figures are acceptable since we are collecting information from documents with different formats and structures. For instance, comparable results were obtained in [45] and [46] although the information in their case was collected only from one source of documents which are curriculum vitas.

5.6.3 Evaluation of the Fraud Detector

We present, in this subsection, the application fraud data used to validate our fraud detector and the technique used to label the data. Finally we present the results produced by our fraud detector based on the labelled data.

5.6.3.1 Application Fraud Data

In order to evaluate our application fraud detector, we need instances of labelled data both with normal, fraud and suspicious cases and then we need to label the data with our proposed fraud detector and compare both labels.

As mentioned above, due to privacy and confidentiality reasons, obtaining real identity fraud information is extremely difficult. Usually the available real data is encrypted and key identity attributes are removed which makes it ineffective for evaluating our detector. To overcome this issue we assume that the data that we collected in our white hat Google hacking experiments are application forms submitted by applicants applying for an identity certificate, which yields 154 normal applications. In addition we used the names of each of the 154 applicants to search for identity patterns containing identity information that share the same name.

Since the data that we collected does not include fraud instances, we created some synthetic fraud data based on obvious fraud concepts. Specifically, the main obvious fraud concept used in our evaluation was based on the case where pair of applications bear the same social security number and have different dates of birth or mother maiden names. Based on this fraud concept, we created synthetic data based on all the possible value combinations of the identity attributes involved. The synthetic data produced is $(2\times 3^3) - 9$(redundant cases) = 45 fraud data instances. So, overall our evaluation dataset involved 199 data instances consisting of 154 normal cases and 45 fraud cases.

5.6.3.2 Data Labelling

We used a methodology based on a technique proposed by Phua *et al.* in [43] for labelling the data instances described in the previous subsection as normal, fraud or suspicious.

In order to make the thesis self-contained, we briefly describe Phua *et al.* technique which we will refer to as CASS for communal analysis suspicion scoring. CASS is a technique for generating numeric suspicion scores for credit applications based on implicit links to other previous applications over both time and space.

Every new application v_i is pair-wise matched against previous scored applications within a window W.

The attribute vector y_{ij} between v_i and v_j which captures the relationship for an application-pair is defined as:

$$y_{ij} = \{y_1^{ij}, y_2^{ij}, ..., y_N^{ij}\}, \forall i, j$$

Where N is the total number of attributes and y_k^{ij} is the individual match score of each pair-wise attribute: $y_k^{ij} = y_k\left[a_{ik}, a_{jk}\right]$, $1 \le k \le N$.

The match score of each pair-wise attribute y_k could be Boolean 1 for a match or 0 for a non match. In this case the maximum possible score of an application–pair is $\sum_{k=1}^{N} y_k^{ij} = N$. By assigning weights to the attributes, the maximum possible score of an application –pair would be $\sum_{k=1}^{N} y_k^{ij} = 1$.

With their technique, Phua *et al.* label a new application v_i by first linking it with an existing application v_j from either a black list or a white list or as anomalous, or by considering it unlinked. Linkage between v_i and v_j is denoted as $v_i \xrightarrow{L} v_j$, where L is a label, corresponding to fraud, or normal or anomalous. After establishing the linkage, the communal suspicion score $W_{communal_{ij}}$ for the linked application-pair is computed accordingly.

Phua *et al.* assume that the black list is made up of actual identity applications previously found to be fraudulent. As a result, any new application that contains similar identity information to those applications in the black list is considered as a fraud.

For the black list, Phua *et al.* define the communal link derived from pair-wise matching of attributes $W_{communal_{ij}}$ as follows:

$$W_{communal_{ij}} = \begin{cases} 1 & \text{and } \mathrm{v}_i \xrightarrow{fraud} v_j \text{ if } \mathrm{v}_j \text{ is a known fraud} \\ & \text{and } \sum_{k=1}^{N} S(y_k^{ij}) \geq T_{fraud} \\ 0 & \text{otherwise} \end{cases} \quad (5.5)$$

Where T_{fraud} is an integer threshold for linking a new application to the black list ($0 \leq T_{fraud} \leq N$) and S is a string similarity metric, which can be exact matching or fuzzy matching. We use in our evaluation exact matching.

Although the white list is the opposite of the black list, it is not made up necessarily of the actual identity applications that are not fraudulent. The white list consists of a set

R of N relationships (or rules) defined as normal: $R = [R_1, R_2, R_3, ..., R_N]$. A weight WR_k is associated with each relationship R_k $(1 \le k \le N)$, where $0.5 \le WR_k \le 1$. The set of weights $W_{normal} = [WR_1, WR_2, ..., WR_N]$ is sorted in ascending order while the set of relationships R is ranked in descending order of WR_k.

For the white list, the communal score $W_{communal}$ is defined as follows:

$$W_{communal_{ij}} = \begin{cases} [WR_x * \sum_{k=1}^{N} S(y_k^{ij})] \text{ and } \mathrm{v}_i \xrightarrow{normal} v_j \text{ if } y_{ij} \in R \\ \qquad \text{and } \sum_{k=1}^{N} S(y_k^{ij}) \ge T_{normal} \\ 0 \qquad \text{otherwise} \end{cases} \quad (5.6)$$

Where T_{normal} is an integer threshold for linking a new application to the white list $(0 \le T_{normal} \le N)$, and WR_x is the weight associated with relationship $R_x \in R$ matching with y_{ij}.

Phua *et al.* assume that anomalous application pairs correspond to linked applications that are in neither the black list nor the white list. The communal score $W_{communal}$ is defined in this case as follows:

$$W_{communal_{ij}} = \begin{cases} \sum_{k=1}^{N} S(y_k^{ij}) \text{ and } \mathrm{v}_i \xrightarrow{anomalous} v_j \text{ if } y \notin R \\ \qquad \text{and } \sum_{k=1}^{N} S(y_k^{ij}) \ge T_{normal} \\ 0 \qquad \text{otherwise} \end{cases} \quad (5.7)$$

Unlinked application-pairs are the results of too few attributes (below the selected thresholds) or no attribute match at all. Phua *et al.* state that there is a strong possibility that many fraudulent applications that share few (below the fraud threshold) identity attributes or no attributes with those in the black list would not be detected and would be accepted as normal.

In order to make a decision for a new application, the total suspicion score for that application is calculated based on all the linked application pairs' scores, and then a final classification as fraud, normal or anomalous is made [43].

In the process of labeling our data using Phua *et al.*'s technique, we populated the black list with identity information of four synthetic fraud applications crafted based on the obvious fraud case explained in the previous subsection. These will be used to match against our 45 synthetic fraud applications mentioned in the previous subsection. We populated the white list with 15 normal relationships defined by Phua *et al.* found in [58] after adapting them to fit the five attributes used in our fraud detector. Table 5-4 presents a sample of the normal relationships in the white list. We randomly selected 49 patterns from the identity patterns that we collected online and labeled them as normal to link them to the white list.

We assumed that the suspicion scores of the applications linked to the black list were very high, and those linked to the white list were very low. Thus, in this case we labeled a new application linked to an application in the black list with $W_{communal} = 1$ as a fraud application. We labeled the applications linked to the white list with $W_{communal} > 0$ and the unlinked applications as normal applications and labeled those linked as anomalous with $W_{communal} > 0$ as suspicious.

Table 5-4. Sample of the normal relashionships in the white list.

Attributes / Rule#	ssn	mmn	dob	add	tel	decision
1	yes	yes	yes	yes	Yes	N
2	yes	yes	yes	na	Na	N
3	yes	yes	yes	yes	Na	N
4	yes	yes	yes	na	Yes	N
5	na	yes	yes	yes	Yes	N
6	yes	na	yes	yes	Yes	N
7	yes	na	yes	na	Na	N
8	yes	na	yes	yes	Na	N
9	yes	na	yes	na	Yes	N
10	na	na	yes	yes	Yes	N

5.6.3.3 Evaluation and Results

To evaluate our application fraud detector we removed the labels of the data and allowed our system to produce its own labels, and then compared them to the data labels produced by CASS. The outcome of the comparison is the Match Rate (MR), when the labels produced by both techniques coincide. The Non Match Rate (NMR), which is the complement of the MR (NMR=1-MR) measures the disagreement between both techniques. Note that a high NMR doesn't necessarily mean a weakness of either of the techniques. A closer error analysis needs to be done to find out which technique is actually at fault.

We chose the values 2 and 3 for the normal and fraud thresholds, which are considered intermediate values since the application-pair matching score is based on five attributes. As a result of setting the thresholds for fraud and normal to a combination of the above two values (i.e. 2 and 3) we produced four different sets of labels for our data. We compared the labels produced by our proposed fraud detector to each of the four sets of labels and computed the match rate by our method which is summarized in Table 5-5.

Table 5-5. Summary of the match rates produced by our fraud detector when compared to CASS-labelled data.

T_{fraud}	T_{normal}	Match Rate
2	2	87%
3	3	82%
3	2	77%
2	3	92%

Based on the selected thresholds, the best match rate produced by our proposed fraud detector was 92% for the data labelled by CASS using a combination of $T_{normal} = 3$ for the white list and the anomalous and $T_{fraud} = 2$ for the black list.

To analyze this result, we have to discuss the labels produced by the CASS approach. By reviewing the 45 synthetic fraud applications that we created, we discovered that 12 fraud applications were labelled as unlinked by CASS, and as a result were considered as normal and got accepted. These 12 applications are cases where two applications share only one attribute value (below the black list threshold) that happens to be the social security number. Although these applications share only one attribute, they correspond to

an obvious fraud case because the social security number is a unique attribute and hence requires that the rest of the attributes to be identical. Thus, in that sense our application fraud detector produces a correct fraud label for all the 45 fraud applications which, however, does not match the CASS labels. CASS produces the wrong labels because, as explained above, the method depends on a threshold which is not effective in detecting such fraud cases.

For the 154 normal applications, while our application fraud detector labels them as normal, CASS labelled 3 of the 154 applications as anomalous and the rest as normal. Examination of these 3 CASS-labelled anomalous applications shows that while the threshold for a linkage between each of the applications and a normal application is met there is no corresponding matching relationship in R. This could possibly be addressed by expanding the initial set of normal relationships used for the evaluation.

Overall our proposed system performs well when assessed against data labelled using CASS, which underscores its potential as a strong fraud detector.

5.7 Summary

We have presented in this chapter an unsupervised application fraud detection framework that analyzes online identity patterns using an intelligent decision model to identify fraudulent applications based on the RBDT-1 method. We designed a web-based identity information retrieval engine capable of finding and collecting online identity information. We defined heuristic rules for detecting fraudulent applications and used the RBDT-1 method to transform the rules into a decision tree. To evaluate our fraud detector we used a mix of real data collected online and synthetic data to induce some

fraud cases. The data was treated as identity applications and labelled using a technique called CASS as normal, fraud or suspicious. The data was labelled four times based on the CASS approach by using different combinations of normal and fraud thresholds and compared to the labels produced by our fraud detector. The best match rate produced by our detector was 92%. The non-matching cases happened to be related labelling errors by the CASS approach. Overall our approach shows strong promise for online identity application fraud detection.

This confirms the applicability of the RBDT-1 method to tackle real-life problems for which data may not easily be available or accessible.

Chapter 6.

Conclusion

In this chapter we will present a summary of the work presented and the contributions. We will also specify the limitations and the planned future work.

6.1 Summary and Contributions

The RBDT-1 method proposed in this work allows generating a decision tree for a specific decision making situation from a set of rules covering the different possible actions. Following this methodology, knowledge can be stored in a declarative rule form and transformed into a decision structure when it is needed for decision making. Generating a decision structure from decision rules can potentially be performed much faster than by generating it from training examples.

Modifications to the data are handled easier in rule-based methods than in data-based methods. This is because the modifications are applied to the rules rather than the decision tree itself. At the same time rule-based methods could transform the rules to a decision tree once we need to decide the order in which tests should be evaluated. Rule-based decision tree methods although designed to create decision trees from rules, could also generate decision tress from data examples. Rule-based decision tree methods are the only solution for generating a decision tree for applications where no data is available and only rules exist. The price of the RBDT-1 advantages is the need to generate rules first before being capable of generating the tree. However, there are efficient rule learning systems available.

In our experiments, our proposed method was compared to two other rule-based decision tree methods; AQDT-1 and AQDT-2. RBDT-1 was also compared to the ID3 method which is a data-based decision tree method. Experiments were done using public datasets from the UCI repository and results for the RBDT-1 were better than the other methods in most of the cases in terms of tree complexity. The accuracy of the decision trees generated from all the methods was the same. In our experiments we used rules extracted from the ID3, C4.5 decision trees and the AQ19 rule induction programs as input to the rule-based decision tree methods that we were using in the comparison.

We also presented a technique for pruning decision trees using the proposed method and conducted some experiments on datasets that have missing values using the RBDT-1 method. It was shown that RBDT-1 was as effective in terms of accuracy and the tree complexity as C4.5.

We chose identity application fraud detection as an application for RBDT-1. In the last several years; identity theft has been on the rise. Unfortunately, the Internet has been facilitating this phenomenon since it represents a tremendous and open repository for sensitive identity information available for those who know how to find them, including fraudsters. We have presented in this work an application fraud detection framework that analyzes online identity patterns using an intelligent decision model to identify fraudulent applications. We designed a web-based identity information retrieval engine capable of finding and collecting online identity information.We defined heuristic rules for detecting fraudulent applications and used a rule-based decision tree method for transforming the rules into a decision tree. For evaluating our fraud detector we used a mix of real data collected by the web-based identity information retrieval engine and synthetic data to

induce some fraud cases. The data both (real/synthetic) was treated as identity applications and labelled using a technique called CASS as normal, fraud or suspicious. The data was labelled four times by using different combinations of normal and fraud thresholds and compared to the labels produced by our fraud detector. The best hit rate produced by our detector was 92% based on the labels produced by the thresholds that we used in this comparison. This underscores the ability of RBDT-1 to be used for real-life problems.

6.2 Other Areas of Application for RBDT-1

There is a wide range of applications that are rule dependent and could benefit from rule-based decision tree methods especially RBDT-1 in their approaches. Such applications could be machine language translation systems which generally require rule-based methods to parse the source text to create an intermediary, symbolic representation, from which the text in the target language is generated. Law systems could be another example of an application which is rule dependent and could benefit from transforming the rules into a concise meaningful decision tree once a judgment needs to be made.

Other rule-based applications that could benefit from rule-based decision tree methods are air traffic control systems, intrusion detection systems, firewall systems, syntax analysis applications and clinical decision support systems.

6.3 Limitations and Future Work

The following limitations have been identified for RBDT-1:

- Although RBDT-1 is capable of accepting data as input, it performs better with rules.

- RBDT-1 using rule sets with high redundancy (e.g Apriori association rules) rate can result in a bigger tree than rule sets with less redundancy (e.g. AQ-based rules).

Investigating these limitations will be the purpose of our future work. Other aspects that we intend to study in the future include the following:

- Experimenting with other rule inducing programs that are known for their ability to produce a shorter set of rules that is known to be accurate as well such as AQ17 [59]. Doing so, there is a possibility of producing a tree that has even less tree complexity than already produced.
- Exploring how to improve RBDT-1 so that the tree accuracy can be better than similar algorithms (based on the experiments that we performed it appeared to be equivalent).
- We have started:
    - Exploring the capability of adding new criteria or adapting the existing criteria of RBDT-1 to overcome the above limitations.
    - Examining the very few DT's that RBDT-1 produced which were slightly bigger than AQDT DT's.

Publication List

Journal Papers:

1. A. Abdelhalim and I. Traore, "Identity application fraud detection using web mining and rule-based decision tree", In the International Journal of Computer and Network Security, Vol. 1, No. 1, pp.31-44, 2009.
2. A. Abdelhalim and I. Traore, "Creating decision trees from rules using RBDT-1", submitted to the International Journal of computational Intelligence. (Under review)

Conference Papers:

1. A. Abdelhalim and I. Traore, "The impact of Google hacking on identity and application fraud", In Proceedings of the IEEE Pacific Rim Conference, Communications, Computers and Signal Processing, pp. 240-244, 2007.
2. A. Abdelhalim, I. Traore, B. Sayed. "RBDT-1: a new rule-based decision tree generation technique", In the 3rd International Symposium on Rules, Applications and Interoperability, Las Vegas, Nevada, USA: Nov 5-7, 2009.
3. A. Abdelhalim, I. Traore. "Converting declarative rules into decision trees", In the International Conference on Computer Science and Applications, San Francisco, USA, 20-22 October, 2009.
4. A. Abdelhalim and I. Traore, "A new method for learning decision trees from rules", In the Eighth International Conference on Machine Learning and Applications (ICMLA'09), Miami, Florida, USA, December 13-15, 2009.

Book Chapters:

1. A. Abdelhalim and I. Traore, "RBDT-1 method: Combining rules and Decision tree capabilities", Machine Learning and Systems Engineering, Springer (under press).

Bibliography

[1] R. S. Michalski and I. F. Imam, "Learning problem-oriented decision structures from decision rules: the AQDT-2 system", In Proceedings of 8th International Symposium on Methodologies for Intelligent Systems. Lecture Notes in Artificial Intelligence 869. Springer Verlag, Heidelberg, pp. 416-426, 1994.

[2] R. S. Michalski and I. F. Imam, "On learning decision structures. Fundamenta Informaticae", 31(1): pp. 49-64., 1997.

[3] I. F Imam, "An empirical comparison between learning decision trees from examples and from decision rules", In Proceedings of the Ninth International Symposium on Methodologies for Intelligent Systems, Zakopane, 1996, http://digilib.gmu.edu:8080/xmlui/handle/1920/1830.

[4] J. R. Quinlan, "Discovering rules by induction from large collections of examples". In D. Michie (Edr), Expert Systems in the Microelectronic Age, Edinburgh University Press, pp. 168-201, 1979.

[5] L. Breiman, J. H. Friedman, R. A. Oishen, and C. J. Stone, "Classification and regression structures", Belmont, California: Wadsworth Int. Group, 1984.

[6] B. Cestnik and A. Karalie, "The estimation of probabilities in attribute selection measures for decision structure induction", In Proceeding of the European Summer School on Machine Learning, Priory Corsendonk, Belgium, pp. 22-31, 1991.

[7] J. Mingers, "An empirical comparison of selection measures for decision-structure induction", Machine Learning, Kluwer Academic Publishers, 3(3): pp. 319-342, 1989.

[8] I. F. Imam and R. S. Michalski, "Should decision trees be learned from examples of from decision rules?", Lecture Notes in Computer Science. In Proceedings of the 7th International Symposium on Methodologies, 689: pp. 395-404, 1993.

[9] I. H. Witten and B. A. MacDonald. "Using concept learning for knowledge acquisition", International Journal of Man-Machine Studies, pp. 349-370, 1988.

[10] J. R. Quinlan, "Simplifying decision trees", International Journal of Man-Machine Studies. 27, 3, pp. 221-234, 1987.

[11] Y. Chen, L. T. Hung, "Using decision trees to summarize associative classification rules". Expert System Application Pergamon Press, Inc. Publisher, 36(2): pp. 2338-2351, 2009.

[12] Y. Akiba S. Kaneda, and H. Almuallim, "Turning majority voting classifiers into a single decision tree", In Proceedings of the 10th IEEE International Conference on Tools with Artificial Intelligence, pp. 224-230, 1998.

[13] "Identity fraud: A study", Economic and Domestic Secretariat Cabinet Office, Crown copyright, www.statewatch.org/news/2004/may/id-fraud-report.pdf, 2002.

[14] W. Wang; Y. Yuan; N. Archer, "A contextual framework for combating identity theft", Security & Privacy Magazine, IEEE Volume 4, pp. 30-38, 2006.

[15] A. M. Marshall, B. C. Tompsett, "Identity theft in an online world", Computer Law & Security Report, pp. 128-137, Elsevier Ltd, 2005.

[16] H. Berghel , "Identity theft, social security numbers, and the Web", Communications of the ACM, Volume 43 Issue 2. ACM Press, pp. 17-21, 2000.

[17] A. Abdelhalim, I. Traore, B. Sayed. "RBDT-1: a new rule-based decision tree generation technique", In the 3rd International Symposium on Rules, Applications and Interoperability, Las Vegas, Nevada, USA: Nov 5-7, 2009.

[18] A. Abdelhalim, I. Traore. "Converting declarative rules into decision trees", In the International Conference on Computer Science and Applications, San Francisco, USA, 20-22 October, 2009.

[19] A. Abdelhalim and I. Traore, "A new method for learning decision trees from rules", In the Eighth International Conference on Machine Learning and Applications (ICMLA'09), Miami, Florida, USA, December 13-15, 2009.

[20] A. Abdelhalim and I. Traore, "Identity application fraud detection using web mining and rule-based decision tree", In the International Journal of Computer and Network Security, Vol. 1, No. 1, pp.31-44, 2009.

[21] A. Abdelhalim and I. Traore, "RBDT-1 method: Combining rules and Decision tree capabilities", Machine Learning and Systems Engineering, Springer (under press).

[22] A. Abdelhalim and I. Traore, "Creating decision trees from rules using RBDT-1", submitted to the International Journal of computational Intelligence. (Under review)

[23] A. Abdelhalim and I. Traore, "The impact of Google hacking on identity and application fraud", In Proceedings of the IEEE Pacific Rim Conference, Communications, Computers and Signal Processing, pp. 240-244, 2007.

[24] Y. Yuan and M.J. Shaw, "Induction of fuzzy decision trees", Fuzzy Sets and Systems, 69, pp. 125-139, 1995

[25] R. Kothari and M. Dong, "Decision trees for classification: a review and some new results", Lecture Notes in Pattern Recognition, World Scientific Publishing Company, Singapore, 2001.

[26] J.R. Quinlan, "Induction of decision trees", In Machine Learning Journal, (1), pp. 81-106, 1986.

[27] J. R. Quinlan, "C4.5: Programs for Machine Learning", San Mateo, CA Morgan Kaufmann, 1993.

[28] A. E. Hart, "Experience in the use of an inductive system in knowledge engineering", in Bramer, M.A. (Eds),Research and Development in Expert Systems, Cambridge University Press, Cambridge, UK, pp.117-26, 1985.

[29] J. Mingers, Expert systems - Experiments with Rule Induction. Journal of the Operational Research Society, 37 (11). pp. 1031-1037. ISSN 0160-5682, 1986.

[30] J. K. Martin, "An Exact probability metric for decision tree splitting and stopping". In Machine Learning Journal 28, 2-3, pp. 257-291, 1997.

[31] U.M. Fayyad and K.B. Irani, "The attribute selection problem in decision tree generation", in Proceedings of the Tenth National Conference on Artificial Intelligence, AAAI Press, pp. 104-110, 1992.

[32] T. Niblett and I. Bratko, "Learning decision rules in noisy domains", Proceeding of Expert Systems 86, Cambridge University Press, Cambridge, pp. 25-34, 1986.

[33] P.K.Chan, W.Fan, A.L.Prodromidis, and S.J.Stolfo. "Distributed data mining in credit card fraud detection", Intelligent Systems, IEEE, Volume: 14, Issue: 6, pp. 67-74, Nov.-Dec. 1999.

[34] R.Brause, T.Langsdorf, and M.Hepp, "Neural data mining for credit card fraud detection", Proceedings of 11th IEEE International Conference on Tools with Artificial Intelligence, pp. 103-106, 1999.

[35] R.Brause, T.Langsdorf, and M.Hepp. "Credit card fraud detection by adaptive neural data mining", www.informatik.unifrankfurt. de/fbreports/fbreport07-99.pdf.

[36] E.Aleskerov, B.Freileben, and B.Rao. "Cardwatch: a neural network based database mining system for credit card fraud detection", Computational Intelligence for Financial Engineering (CIFEr), Proceedings of the IEEE/IAFE, pp. 220-226, March 1997.

[37] R. J. Bolton and D. J. Hand. "Unsupervised profiling methods for fraud detection", Credit Scoring and Credit Control Volume 2, Edinburgh, UK, 2001.

[38] R. Bose. "Intelligent technologies for managing fraud and identity theft", Third International Conference on Information Technology: New Generations, pp. 446-451, 2006.

[39] Y. Kou, C. Lu, S. Sirwongwattana and Y. Huang, "Survey of fraud detection techniques". Proceedings of the 2004 IEEE, International Conference on Networking, Sensing & Control, pp. 749-754, 2004.

[40] M. Weatherford, "Mining for fraud", Intelligent Systems, IEEE, Volume: 17, Issue: 4, pp. 4-6, 2002.

[41] P. Burns, A. Stanley. "Fraud management in the credit card industry", Payment Cards Center Discussion Paper number 025, Federal Reserve Bank of Philadelphia, 2002, http://www.phil.frb.org/pcc/papers/2002/FraudManagement_042002.pdf

[42] R. Wheeler, S. Aitken. "Multiple algorithms for fraud detection", Knowledge-Based Systems, Vol. 13, No. 3, pp. 93-99, 2000.

[43] C. Phua, V. Lee, K. Smith, R. Gayler, "A comprehensive survey of data mining-based fraud detection research", Clayton School of Information Technology, Monash University, 2005.

[44] P. A. Estevez, C. M. Held, C. A. Perez, "Subscription fraud prevention in telecommunications using fuzzy rules and neural networks", Expert Systems with Applications, Vol. 31, pp. 337-344, 2006.

[45] L. Sweeney. "AI technologies to defeat identity theft vulnerabilities", AAAI Spring Symposium on AI Technologies for Homeland Security, 2005.

[46] L. Sweeney. "Protecting job seekers from identity theft", IEEE Internet Computing, Vol. 10, No. 2, pp.74-78, 2006.

[47] R. S. Michalski and K. Kaufman, „The AQ19 system for machine learning and pattern discovery: a general description and user's guide". Reports of the Machine Learning and Inference Laboratory, MLI 01-2, George Mason University, Fairfax, VA, 2001.

[48] J. Wojtusiak, "AQ21 user's guide", Reports of the Machine Learning and Inference Laboratory, MLI 04-5, George Mason University, 2004.

[49] R. Agrawal, T. Imielinski, A. N. Swami, "Mining association rules between sets of items in large databases", In ACM SIGMOD., 22(2), pp. 207-16, June 1993.

[50] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac, "The multi-purpose incremental learning system AQ15 and its testing application to three dedical domains", In Proceedings of AAAI-86, Philadelphia, PA, pp. 1041-1045, 1986.

[51] Bergadano, F., S. Matwin, R. S. Michalski and J. Zhang. "Learning two-tiered descriptions of flexible concepts: the POSEIDON system", Machine Learning, Vol. 8, No. 1, pp. 5-43, 1992.

[52] S. Colton, 2004, OnlineDocument.
www.doc.ic.ac.uk/~sgc/teaching/v231/lecture11.html.

[53] M. Yu. Moshkov, "Time complexity of decision trees," in Transactions on Rough Sets III, Lecture. Notes Computer Science 3400, pp. 244-459, Berlin; Heidelberg: Springer-Verlag, 2005.

[54] B. R. Preiss, "Data structures and algorithms with object-oriented design patterns in java", available at: http://www.brpreiss.com/books/opus5/html/page257.html.

[55] A. Asuncion and D.J. Newman, 2007. UCI Machine Learning Repository [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science.

[56] K. Roach, "ID3implementationwithPython", 2006.
http://www.onlamp.com/pub/a/python/2006/02/09/ai_decision_trees.html.

[57] J. Demsar, B. Zupan and G. Leban, "Orange: From Experimental Machine Learning to Interactive Data Mining", White Paper (www.ailab.si/orange), Faculty of Computer and Information Science, University of Ljubljana, 2004.

[58] C. Phua, R. Gayler, V. Lee and K. Smith-Miles, "On the Communal Analysis Suspicion Scoring for Identity Crime in Streaming Credit Applications", European Journal of Operational Research, 195, pp. 595-612, 2009.

[59] E. Bloedorn, J. Wnek, R. S. Michalski, and K. Kaufman, "AQ17: A Multistrategy Learning System: The Method and User's Guide", Report of Machine Learning and Inference Laboratory, MIL-93-12, George Mason University, 1993.

Appendix A

Table A-0-1. Complexity of the decision trees created using the ID3 method.

Dataset Name	Complexity		Accuracy[6]
	#Nodes	# Leafs	
Weekend	5	7	100%
Lenses	6	9	100%
King-Rook vs. King-Knight chess	5	10	100%
Car evaluation	112	296	100%
Shuttle-landing-control	8	15	100%
Connect-4	13924	20193	100%
Nursery	320	839	100%
Balance Scale	100	401	100%
MONK's 1	172	325	100%
MONK's 2	171	308	100%
MONK's 3	5	12	100%
Zoo	9	14	100%

[6] The accuracy of the decision tree in the tables is calculated as the percentage of the number of examples correctly classified by the tree to the total number of examples.

Table A-0-2. Decision tree complexities for the RBDT-1, AQDT-1, and AQDT-2 methods using ID3-based rules.

Method	Complexity		Accuracy
	# Nodes	# Leafs	
The weekend problem			
RBDT-1	3	5	100%
AQDT-1, AQDT-2	5	7	100%
The fitting contact lenses problem			
RBDT-1	6	9	100%
AQDT-1, AQDT-2	7	10	100%
The MONK's1 problem			
RBDT-1	13	28	100%
AQDT-1,AQDT-2	Min=58, Max=172	Min=109, Max=325	100%
The MONK's 2 problem			
RBDT-1,AQDT-1, AQDT-2	131	308	100%
The balance scale problem			
RBDT-1	100	401	100%
AQDT-1, AQDT-2	Min=111, Max=114	Min=445, Max=457	100%
The nursery problem			
RBDT-1	318	839	100%
AQDT-1, AQDT-2	320	839	100%
The King-rook vs. king-knight chess problem			
RBDT-1,AQDT-1, AQDT-2	5	10	100%

Table A-2. Continued.

Method	Complexity		Accuracy
	# Nodes	# Leafs	
The car evaluation problem			
RBDT-1	112	296	100%
AQDT-1, AQDT-2	Min=137, Max=139	Min=389, Max=395	100%
The connect-4 problem			
RBDT-1	13718	20188	100%
AQDT-1, AQDT-2	Min=14390, Max=14420	Min=20902, Max=20902	99%
The MONK's3 problem			
RBDT-1,AQDT-1, AQDT-2	5	12	100%
The zoo problem			
RBDT-1,AQDT-1, AQDT-2	9	14	100%
The shuttle-control-landing			
RBDT-1,AQDT-1, AQDT-2	8	15	100%

Table A-3. Decision tree complexities for the RBDT-1, AQDT-1, AQDT-2 methods using AQ-based rules.

Method	Complexity		Accuracy
	# Nodes	# Leafs	
The weekend problem			
RBDT-1	3	5	100%
AQDT-1, AQDT-2	5	7	100%
The lens problem			
RBDT-1	6	9	100%
AQDT-1, AQDT-2	7	10	100%
The MONK's1 problem			
RBDT-1	10	27	100%
AQDT-1, AQDT-2	13	28	100%
The MONK's2 problem			
RBDT-1	125	293	100%
AQDT-1, AQDT-2	171	311	100%
The MONK's3 problem			
RBDT-1, AQDT-1, AQDT-2	5	12	100%

Table A-3. Continued.

Method	Complexity		Accuracy
	#Nodes	#Leafs	
The car			
RBDT-1	295	109	100%
AQDT-1, AQDT-2	Min=139, Max=141	Min=393,Max= 399	100%
The shuttle-control-landing			
RBDT-1, AQDT-1, AQDT-2	7	14	100%
The zoo problem			
RBDT-1	10	12	100%
AQDT-1, AQDT-2	9	14	100%
King-rook vs. king-knight chess problem			
RBDT-1, AQDT-1, AQDT-2	5	10	100%
The balance			
RBDT-1	Min=100, Max=106	Min=401,Max= 425	100%
AQDT-1, AQDT-2	Min=111, Max=117	Min=445,Max= 469	100%

Table A-4. Tree complexities and accuracy of RBDT-1, AQDT-1, AQDT-2 association rules.

Method	Complexity		Accuracy
	# Nodes	# Leaves	
The weekend problem			
All three methods	9	7	100%
The fitting contact lenses problem			
RBDT-1	6	9	100%
AQDT-1, AQDT-2	7	10	100%
The MONK's1 problem			
RBDT-1	10	27	100%
AQDT-1, AQDT-2	13	28	100%
The MONK's 2 problem			
RBDT-1	113	290	100%
AQDT-1	244	330	100%
AQDT-2	117	290	100%
The MONK's 3 problem			
All three methods	5	12	100%

Table A-4. Continued.

Method	Complexity		Accuracy
	# Nodes	# Leaves	
The balance scale problem			
RBDT-1	105	421	100%
AQDT-1, AQDT-2	111	445	100%
The car evaluation problem			
RBDT-1	108	297	100%
AQDT-1, AQDT-2	140	394	100%
The shuttle-landing-control problem			
RBDT-1	12	19	100%
AQDT-1	15	22	100%
AQDT-2	13	20	100%
The nursery problem			
All three methods	7	14	74%

Table A-5. Tree complexities and accuracy of RBDT-1, AQDT-1, AQDT-2 using C4.5-based rules produced without pruning and the C4.5 method.

Using C4.5-based rules without pruning			
Method	**Complexity**		**Accuracy**
	# Nodes	**# Leaves**	
The weekend problem			
RBDT-1,C4.5	2	4	90%
AQDT-1, AQDT-2	3	5	90%
The fitting contact lenses problem			
RBDT-1, C4.5	3	4	91%
AQDT-1, AQDT-2	4	5	91%
The MONK's1 problem			
RBDT-1	19	38	97%
AQDT-1,AQDT-2, C4.5	29	56	97%
The MONK's 2 problem			
All four methods	88	166	85%
The balance scale problem			
All four methods	22	89	80%
The car evaluation problem			
RBDT-1, C4.5	52	134	96%
AQDT-1, AQDT-2	65	180	96%
The connect-4 problem			
C4.5	5317	10635	91%
RBDT-1	5316	10633	91%
AQDT-1, AQDT-2	5701	11403	91%

Table A-5. Continued.

Using C4.5-based rules without pruning			
Method	**Complexity**		**Accuracy**
	# Nodes	**# Leafs**	
The chess problem			
All four methods	5	10	100%
The Monks3			
RBDT-1, C4.5	5	14	100%
AQDT-1, AQDT-2	5	12	100%
The breast-cancer problem			
All four methods	27	113	85%
The lung-cancer problem			
RBDT-1,C4.5	7	12	90%
AQDT-1, AQDT-2	8	14	90%
The primary-tumor problem			
RBDT-1, C4.5	56	67	57%
AQDT-1, AQDT-2	114	133	53%
The voting problem			
RBDT-1, C4.5	18	19	92%
AQDT-1, AQDT-2	17	18	92%
The zoo problem			
RBDT-1, C4.5	8	13	99%
AQDT-1, AQDT-2	10	15	99%
The shuttle-control-landing			
All four methods	6	13	99%
The nursery problem			
C4.5	264	680	99%
RBDT-1,AQDT-1, AQDT-2	262	680	99%

Table A-6. Tree complexities and accuracy of RBDT-1, AQDT-1, AQDT-2 using C4.5-based rules produced with pruning and the C4.5 method.

Using C4.5-based rules with pruning option set to true			
Method	**Complexity**		**Accuracy**
	# Nodes	**# Leaves**	
The weekend problem			
RBDT-1, C4.5	2	4	90%
AQDT-1, AQDT-2	3	5	90%
The fitting contact lenses problem			
RBDT-1, C4.5	3	4	91%
AQDT-1, AQDT-2	4	5	91%
The MONK's1 problem			
All four methods	13	28	100%
The MONK's 2 problem			
All four methods	0	1	67%
The balance scale problem			
All four methods	8	33	75%
The car evaluation problem			
RBDT-1, C4.5	51	131	96%
AQDT-1, AQDT-2	64	177	96%
The connect-4 problem			
RBDT-1, C4.5	2142	4285	87%
AQDT-1, AQDT-2	2394	4789	87%
The nursery problem			
C4.5	152	359	98%
RBDT-1,AQDT-1, AQDT-2	150	359	98%

Table A-6. Continued.

Using C4.5-based rules with pruning option set to true			
	Complexity		
Method	**# Nodes**	**# Leaves**	**Accuracy**
The lung-cancer problem			
RBDT-1, C4.5	6	10	87%
AQDT-1, AQDT-2	7	12	87%
The primary-tumor problem			
RBDT-1, C4.5	41	47	58%
AQDT-1, AQDT-2	47	53	55%
The voting problem			
All four methods	5	6	93%
The zoo problem			
RBDT-1, C4.5	8	13	99%
AQDT-1, AQDT-2	10	15	99%
The shuttle-landing-control problem			
All four methods	5	10	98%
The chess problem			
All four methods	5	10	100%
The Monks3 problem			
RBDT-1, C4.5	5	14	100%
AQDT-1, AQDT-2	5	12	100%
The Breast-cancer problem			
All four methods	2	4	74%