Linköping Studies in Science and Technology

Dissertation No. 1086

Obtaining Accurate and Comprehensible Data Mining Models – An Evolutionary Approach

by

Ulf Johansson

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2007

Abstract

When performing predictive data mining, the use of ensembles is claimed to virtually guarantee increased accuracy compared to the use of single models. Unfortunately, the problem of how to maximize ensemble accuracy is far from solved. In particular, the relationship between ensemble diversity and accuracy is not completely understood, making it hard to efficiently utilize diversity for ensemble creation. Furthermore, most high-accuracy predictive models are opaque, i.e. it is not possible for a human to follow and understand the logic behind a prediction. For some domains, this is unacceptable, since models need to be comprehensible. To obtain comprehensibility, accuracy is often sacrificed by using simpler but transparent models; a trade-off termed the accuracy vs. comprehensibility trade-off. With this trade-off in mind, several researchers have suggested rule extraction algorithms, where opaque models are transformed into comprehensible models, keeping an acceptable accuracy.

In this thesis, two novel algorithms based on Genetic Programming are suggested. The first algorithm (GEMS) is used for ensemble creation, and the second (G-REX) is used for rule extraction from opaque models. The main property of GEMS is the ability to combine smaller ensembles and individual models in an almost arbitrary way. Moreover, GEMS can use base models of any kind and the optimization function is very flexible, easily permitting inclusion of, for instance, diversity measures. In the experimentation, GEMS obtained accuracies higher than both straightforward design choices and published results for Random Forests and AdaBoost. The key quality of G-REX is the inherent ability to explicitly control the accuracy vs. comprehensibility trade-off. Compared to the standard tree inducers C5.0 and CART, and some well-known rule extraction algorithms, rules extracted by G-REX are significantly more accurate and compact. Most importantly, G-REX is thoroughly evaluated and found to meet all relevant evaluation criteria for rule extraction algorithms, thus establishing G-REX as the algorithm to benchmark against.

Contents

List of Figures

List of Tables

List of Publications

Thesis

Johansson, U., Rule Extraction - the Key to Accurate and Comprehensible Data Mining Models, Licentiate thesis, Institute of Technology, Linköping University, 2004.

Journal papers

Johansson, U., Löfström, T., König, R. and Niklasson, L., Why Not Use an Oracle When You Got One?, *Neural Information Processing - Letters and Reviews*, Vol. 10, No 8-9:227-236, 2006.

Löfström, T. and Johansson, U., Predicting the Benefit of Rule Extraction - A Novel Component in Data Mining, *Human IT,* Vol. 7.3:78-108, 2005.

König, R., Johansson, U. and Niklasson, L., Increasing rule extraction comprehensibility, *International Journal of Information Technology and Intelligent Computing*, Vol. 1, No. 2:303-314, 2006

International conference papers

Johansson, U. and Niklasson, L., Predicting the impact of advertising - a neural network approach*, The International Joint Conference on Neural Networks*, IEEE Press, Washington D.C., pp. 1799-1804, 2001.

Johansson, U. and Niklasson, L., Increased Performance with Neural Nets - An Example from the Marketing Domain*, The International Joint Conference on Neural Networks*, IEEE Press, Honolulu, HI, pp. 1684-1689, 2002.

Johansson, U. and Niklasson, L., Neural Networks - from Prediction to Explanation*, IASTED International Conference Artificial Intelligence and Applications,* IASTED, Malaga, Spain, pp. 93-98, 2002.

Johansson, U., König, R. and Niklasson, L., Rule Extraction from Trained Neural Networks using Genetic Programming, *13th International Conference on Artificial Neural Networks,* Istanbul, Turkey, supplementary proceedings pp. 13-16, 2003.

Johansson, U., Sönströd, C., König, R. and Niklasson, L., Neural Networks and Rule Extraction for Prediction and Explanation in the Marketing Domain, *The International Joint Conference on Neural Networks,* IEEE Press, Portland, OR, pp. 2866-2871, 2003.

Johansson, U., König, R. and Niklasson, L., The Truth is in There - Rule Extraction from Opaque Models Using Genetic Programming, *17th Florida Artificial Intelligence Research Society Conference (FLAIRS) 04*, Miami, FL, AAAI Press, pp. 658-662, 2004.

Johansson, U., Niklasson, L. and König, R., Accuracy vs. Comprehensibility in Data Mining Models, *7th International Conference on Information Fusion*, Stockholm, Sweden, pp. 295-300, 2004.

Johansson, U., Sönströd, C. and Niklasson, L., Why Rule Extraction Matters, *8th IASTED International Conference on Software Engineering and Applications*, MIT, Cambridge, MA, pp. 47-52, 2004.

Löfström, T., Johansson, U. and Niklasson, L., Rule Extraction by Seeing Through the Model, *11th International Conference on Neural Information Processing (ICONIP)*, Calcutta, India, pp. 555-560, 2004.

Johansson, U., König, R. and Niklasson, L., Automatically Balancing Accuracy and Comprehensibility in Predictive Modeling, *8th International Conference on Information Fusion*, Philadelphia, PA, 2005.

Johansson, U., Löfström, T. and Niklasson, L., Obtaining Accurate Neural Network Ensembles, *International Conference on Computational Intelligence for Modelling Control and Automation - CIMCA'2005*, Vienna, Austria, IEEE Computer Society, Vol. 2:103-108, 2005.

Johansson, U., Löfström, T., König, R. and Niklasson, L., Introducing GEMS - a Novel Technique for Ensemble Creation, *19th Florida Artificial Intelligence Research Society Conference (FLAIRS) 06*, Melbourne Beach, FL, AAAI Press, pp. 700-705, 2006.

Johansson, U., Löfström, T., König, R. and Niklasson, L., Genetically Evolved Trees Representing Ensembles, *8th International Conference on Artificial Intelligence and Soft Computing*, Zakopane, Poland, Lecture Notes in Artificial Intelligence, Springer-Verlag, pp. 613-622, 2006.

Johansson, U., Löfström, T., König, R. and Niklasson, L., Building Neural Network Ensembles using Genetic Programming, *The International Joint Conference on Neural Networks,* IEEE Press, Vancouver, Canada, pp. 2239- 2244, 2006.

Johansson, U., Sönströd, C. and Niklasson, L., Explaining Winning Poker - A Data Mining Approach, *6th International Conference on Machine Learning and Applications*, Orlando, FL, IEEE press, pp. 129-134, 2006.

Johansson, U., Löfström, T., König, R., Sönströd, C. and Niklasson, L., Rule Extraction from Opaque Models - A Slightly Different Perspective, *6th International Conference on Machine Learning and Applications*, Orlando, FL, IEEE press, pp. 22-27, 2006.

Johansson, U., Löfström, T. and Niklasson, L., The Importance of Diversity in Neural Network Ensembles - An Empirical Investigation, *The International Joint Conference on Neural Networks,* IEEE Press, Orlando, FL, 2007, To appear.

Johansson, U., König, R. and Niklasson, L., Inconsistency - Friend or Foe, *The International Joint Conference on Neural Networks,* IEEE Press, Orlando, FL, 2007, To appear.

Sönströd, C. and Johansson, U., Concept Description - A Fresh Look, *The International Joint Conference on Neural Networks*, IEEE Press, Orlando, FL, 2007, To appear.

National conference papers

Johansson, U. and Sönströd, C., G-REX: Rule Extraction from Opaque Models Using Genetic Programming, *21st Annual Workshop of the Swedish Artificial Intelligence Society*, Lund, Sweden, 2004, pp. 114-129.

Johansson, U., Löfström, T., König, R. and Niklasson, L. Accurate Neural Network Ensembles Using Genetic Programming, *23rd Annual Workshop of the Swedish Artificial Intelligence Society*, Umeå, Sweden, 2006, pp. 117-126.

Johansson, U., Löfström, T. and Niklasson, L. Accuracy on a Hold-out Set: The Red Herring of Data Mining, *23rd Annual Workshop of the Swedish Artificial Intelligence Society*, Umeå, Sweden, 2006, pp. 137-146.

Acknowledgement

First of all, very special thanks to my main supervisor Professor Lars Niklasson, University of Skövde. You're the best! I could not have done this without your help and support.

Special thanks to my good friends in the Artificial Intelligence and Mining group, University of Borås. Rikard König (lead G-REX programmer), Tuve Löfström (MatLab guru) and Cecilia Sönströd (my prime discussion partner); you have all contributed greatly to this thesis. I hope I will be able to return the favor in the near future.

I also want to thank my primary supervisor Professor Tom Ziemke, University of Skövde, especially for important and valuable feed-back given.

Obviously I am very grateful to University of Borås for funding my PhD studies, especially to Romulo Enmark and Birgitta Påhlsson for making the initial arrangements.

Last but not least, I want to thank Cecilia Sönströd for helping me proofread this thesis, all grammatical mistakes and typos were introduced by me after her last reading.

At the end of this long journey, I send an appreciative thought to my parents, who encouraged me to study natural science in upper secondary school and after that computer engineering.

Pia and Kalle: I hope you still remember me. I'm the guy you used to do a lot of fun things together with before I decided to spend all my spare time on writing a stupid book. Now, Kalle, that the book is finally finished, we have a lot of catching up to do on the Xbox and on the golf course.

Finally I want to thank a lot of people, cats and organizations who have all, in one way or another, helped me through this sometimes exhausting process:

Niclas Åstrand, Anna Palmquist, Henrik Carlsson, Patrik Hedberg, Mikael Lind, Malin Nilsson, Rolf Appelquist, Lillemor Wallgren, Christian Bennet, Jörgen Sjögren, Helge Malmgren, Orvar, Maja, Pelle, Tarzan, Tequila, IFK Göteborg, Anaheim Angels, Kungsbacka Baseball Club, Chalmers, Livregementets husarer, Johan Kinde, Ralph Hütter, Florian Schneider, Richard Wagner, Coca-Cola, Samuel Adams, Bowmore, Zoegas and, of course, all the kind fish at Party Poker.

Chapter 1

Introduction

Recent advances in data collection and storage technology has made it possible for organizations to accumulate huge amounts of data at moderate cost. While most data is not stored with predictive modeling or analysis in mind, the collected data could contain potentially valuable information. Exploiting this stored data, in order to extract useful and actionable information, is the overall goal of the generic activity termed *data mining*. Although several definitions of data mining exist, they are quite similar. In [BL97], the following definition is given:

> *Data mining is the process of exploration and analysis, by automatic or semi-automatic means, of large quantities of data in order to discover meaningful patterns and rules.* (p. 5)

Since data mining is used in many domains, the exact purpose of an individual data mining project can vary a great deal. The information discovered is, however, almost always intended as a basis for human decision making.

At the core of the data mining process is the use of a data mining technique. Some data mining techniques directly obtain the information by performing a descriptive partitioning of the data. More often, however, data mining techniques utilize stored data in order to build *predictive models*. The purpose of a predictive model is to allow the data miner to predict an unknown (often future) value of a specific variable; the *target variable*. If the target value is one of a predefined number of discrete (class) labels, the data mining task is called *classification*. If the target variable is a real number, the task is *regression*.

From a general perspective, there is strong agreement among both researchers and executives about the criteria that all data mining techniques must meet. Most importantly, the techniques should have *high performance*. This criterion is, for predictive modeling, understood to mean that the technique should produce models that will generalize well, i.e. models having high *accuracy* when performing predictions based on novel data.

Two general techniques for predictive modeling available in many data mining tools are *neural networks* and *decision trees*[1]. Comparing neural networks and decision trees, the prevailing opinion is that neural networks most often will obtain more accurate models; see e.g. [SMT91]. As a matter of fact, the key quality of neural networks is their robustness; enabling them to produce very accurate models on most data sets. Consequently, neural networks have been successfully used for predictive modeling in a variety of domains.

Within the machine learning research community it is, however, also well known that it is possible to obtain even higher accuracy, by combining several individual models into *ensembles*; see e.g. [HS90] and [KV95]. A key result, derived in [KV95], is that the ensemble error depends not only on the average accuracy of the base models but also on their diversity[2]. Informally, diversity means that the base classifiers make their mistakes on different instances. So, the overall goal when creating an ensemble is to combine models that are highly accurate, but differ in their predictions. Unfortunately, base classifier accuracy and diversity is highly correlated, so maximizing diversity would most likely reduce the average accuracy. In addition, diversity is, for predictive classification, not uniquely defined. Because of this, there are several diversity measures, and, to further complicate matters, no specific diversity measure has shown high correlation with accuracy on novel data.

---

[1] It should be noted that not everyone would agree to that decision tree techniques should (or even could) be used for predictive modeling. In their opinion, the sole purpose of a decision tree model is to *explain* the relationship between variables in the model. In this thesis, however, decision tree techniques are primarily regarded as techniques for predictive modeling. The obvious motivation is that decision tree techniques often, in practice, *are* used for predictive modeling. Most general purpose textbooks about data mining, see e.g. [TSK06], also share this opinion, describing decision trees as a tool for predictive modeling.

[2] Krogh and Vedelsby used the term *ambiguity* instead of diversity in their paper. In this thesis, the more common term diversity is, however, used exclusively.

The most renowned ensemble techniques are probably *bagging*, *boosting* and *stacking*, all of which can be applied to different types of models and perform both regression and classification. Most importantly; bagging, boosting and stacking will, almost always, increase predictive performance over a single model. Unfortunately, machine learning researchers have struggled to understand why these techniques work; see e.g. [WF05]. In addition, it should be noted that these general techniques must be regarded as schemes rather than actual algorithms since they all require several design choices and parameter settings.

From a high level perspective, any ensemble creation algorithm must both generate and combine individual models. It is fair to say that most existing algorithms focus on the creation of the base models, rather than the combination. As a matter of fact, a majority of algorithms either just average the predictions from the individual models, or use some kind of voting scheme. Sometimes, outputs from individual models are weighted, typically based on model accuracy. When generating the base models, diversity is either explicitly or implicitly targeted. Explicit methods somehow measure and balance diversity against accuracy, while implicit methods obtain diversity without actually targeting it. One very common procedure, aimed at producing implicit diversity, is to generate each model using a different part of the available data. Another, frequently used, option is to generate models using different parameters or with varied architectures.

So, due to promising theoretical results and several existing successful algorithms for ensemble creation, coupled with the fact that no specific algorithm is recognized as superior to all others, there is a constant flow of research papers suggesting novel algorithms for constructing ensembles. Although most of these algorithms do target diversity, either explicitly or implicitly, there is no agreement on which diversity measure to use or how diversity should be balanced against accuracy. Instead, most algorithms are based on more or less ad hoc methods for obtaining diversity. Furthermore, for algorithms generating or searching for ensembles using some kind of optimization function, it is far from obvious which measure to optimize. Arguably, the most common procedure is to set aside a part of the available data and use ensemble accuracy on this holdout set as the optimization function.

Finally, the actual use of ensembles in applications is surprisingly limited. Two possible reasons for this are insufficient knowledge about the benefits of using ensembles and limited support in most data mining tools. In addition, using ensembles is far from straightforward, although this is sometimes hidden from the data miner by the software.

Although accuracy is the prioritized criterion for predictive modeling, the *comprehensibility* of the model is often very important. A comprehensible model makes it possible for the data miner to understand not only the model itself but also why individual predictions are made. Traditionally, most research papers focus on high accuracy, although the comprehensibility criterion is often emphasized by business representatives; see e.g. [BL00]. CRISP-DM[1] points out the advantage of having "a verbal description of the generated model (e.g. via rules)", thus acknowledging the importance of comprehensibility. Only with this description is it possible to "assess the rules; are they logical, are they feasible, are there too many or too few, do they offend common sense?"

Clearly, comprehensibility is tightly connected to the choice of data mining technique. As a matter of fact, the most often cited drawback with neural networks is that the models produced are *opaque*; i.e. they do not permit human inspection or understanding. A decision tree model, on the other hand, is regarded as comprehensible since it is *transparent*, making it possible for a human to follow and understand the logic behind a prediction. These descriptions are, however, too simplified. Comprehensibility is, at least, also dependent on the size of the model. For example; the comprehensibility of an extremely bushy decision tree is clearly questionable. It should also be noted that ensembles in general must be considered incomprehensible; it would be quite hard to grasp an ensemble model, even if it consisted of only a few small decision trees.

Since techniques producing opaque models normally will obtain highest accuracy, it seems inevitable that the choice of technique is a direct trade-off between accuracy and comprehensibility. With this trade-off in mind, several researchers have tried to bridge the gap by introducing techniques for transforming opaque models into transparent models, keeping an acceptable

[1] CRoss-Industry Standard Process for Data Mining was an ESPRIT project that started in the mid-1990's. The purpose of the project was to propose a non-proprietary industry standard process model for data mining. For details see www.crisp-dm.org.

accuracy. Most significant are the many attempts to extract rules from trained neural networks. Several authors have discussed key demands on a reliable rule extraction method; see e.g. [ADT95] and [CS99]. The most common criteria are: *accuracy* (the extracted model must perform almost as well as the original on unseen data), *comprehensibility* (the extracted model should be easily interpretable) and *fidelity* (the extracted model should perform similarly to the original model). Two other very important criteria are *scalability* (the method should scale to networks with large input spaces and large numbers of weighted connections) and *generality*; i.e. the method should place few restrictions on network architectures and training regimes. Some papers (see e.g. [TS93]), also mention the criterion *consistency*. A rule extraction algorithm is consistent if the rules extracted from a specific model are similar between runs.

Although proposed rule extraction algorithms often show good performance in reported case studies, there is still no rule extraction method recognized as superior to all others. As a matter of fact, the opinion is that no existing method meets all criteria; see e.g. [CS99][1]. A clear indication of the status of the different rule extraction algorithms is the fact that none of the major data mining software tools (e.g. Clementine[2] and Enterprise Miner[3]) includes a rule extraction option.

1.1 Problem statement

The description in the introduction leads to the following key observations:

1. When used for predictive modeling, data mining techniques must produce highly accurate models. Although single models can, on occasion, be very accurate, the use of ensembles all but guarantees increased accuracy. Despite the solid theoretical foundation and a large number of existing algorithms for ensemble creation, there is currently no agreement on which research direction to pursue. The reason for this, and the most important problem identified, is the fact that the relationship between diversity and ensemble accuracy is not completely understood, especially for classification problems.

[1] In their paper, Craven and Shavlik argue that their rule extraction method TREPAN meets most criteria. It should be noted, however, that TREPAN is restricted to classification problems, thus showing poor generality, and also lacks a method for directly controlling the trade-off between accuracy and comprehensibility.

[2] www.spss.com

[3] www.sas.com

Because of this, there is no widely accepted diversity measure that can be used for designing classifier ensembles. Presently, various researchers instead try very different approaches, resulting in many extremely varied algorithms. These algorithms are often quite complex and very specialized. More specifically; they require homogenous base models of a particular kind and they use predefined, fixed combination rules and optimization functions.

2. Sometimes accuracy is not the only relevant criterion. Often, there is a need for comprehensible models. When this is the case, the easiest solution is to use a technique directly producing transparent models; most often decision trees. This will, however, normally lead to a loss of accuracy; a trade-off termed the *accuracy vs. comprehensibility* trade-off. Because of this trade-off, rule extraction, which is the task of transforming opaque models with high accuracy to comprehensible models while retaining the level of accuracy, is important. There are well-established criteria to evaluate rule extraction algorithms against. These criteria are evident, objective and inclusive. Although several rule extraction techniques exist, the opinion is that no specific technique meets all criteria.

Based on the key observations, this thesis addresses the following two main problems:

1. How to optimize accuracy when creating ensembles.

2. How to extract accurate and comprehensible rules from opaque predictive models.

Clearly these problems are beyond the scope of a single thesis, so instead some important sub-problems are identified and investigated. The first three sub-problems relate to optimizing ensemble accuracy and the last five to rule extraction from opaque models.

1.1 What is the relationship between diversity and ensemble accuracy on novel data? More specifically; is it beneficial to use a diversity measure as part of the optimization function when generating or searching for an ensemble?

1.2 How strong are straightforward approaches, like averaging a fixed number of neural networks, and how important is implicit diversity for such approaches?

1.3 How should available data be used when constructing ensembles? In particular; is it advantageous to use ensemble accuracy on a part of the data set not used when creating the base models for ranking of ensembles, or as part of an optimization function?

2.1 How significant is the difference in accuracy between opaque and transparent models; i.e. how severe is the accuracy vs. comprehensibility trade-off?

2.2 What are the relevant criteria for evaluating rule extraction algorithms?

2.3 How well do existing techniques meet the criteria?

2.4 Is it possible to constantly obtain higher accuracy and comprehensibility by using rule extraction, compared to techniques directly inducing decision trees from the data set?

2.5 Can unlabeled data instances be used to increase accuracy when performing rule extraction? More specifically; would rules extracted using opaque model predictions provide better explanations of the predictions made?

1.2 Main contributions

In this thesis, two novel algorithms are suggested. The first algorithm, named GEMS (Genetic Ensemble Member Selection), is used for ensemble creation, and the second, named G-REX (Genetic Rule EXtraction), is used for rule extraction from opaque models.

The most important property of GEMS is generality. More specifically; GEMS can work with any kind of base models while permitting extremely flexible and complex combination rules. As a matter of fact, GEMS has the inherent ability to combine smaller ensembles and individual models in an almost arbitrary way. Moreover; the optimization function is easily adaptable, making it uncomplicated to include, for instance, diversity measures. Regarding performance, GEMS in the experimentation obtains accuracies that compare favorably to both straightforward design choices and published results for Random Forests and AdaBoost.

G-REX is also extremely general since it can extract rules in a variety of representation languages from arbitrary opaque models. The key quality is, however, the inherent ability to explicitly control the trade-off between accuracy and comprehensibility. In this thesis, G-REX is thoroughly evaluated using the standard criteria. In addition, G-REX performance is systematically compared to both standard decision tree inducers and some well-known rule extraction algorithms. With the exception of consistency and possibly scalability, G-REX clearly meets all criteria. Most importantly, the studies indisputably show that rules extracted by G-REX are both very accurate and very compact. More specifically, extracted G-REX rules are significantly more accurate and compact than decision trees induced directly from the data sets using C5.0 or CART. Moreover, G-REX also obtains significantly higher accuracy than the two well-known rule extraction algorithms RX and Trepan. Arguing that consistency is not vital for a rule extraction algorithm, and that experimentation has shown G-REX scalability to be at least acceptable, the overall picture is that G-REX meets all important criteria.

In addition to the algorithms, some important insights regarding ensemble creation and rule extraction were obtained:

- All diversity measures evaluated show low or very low correlation with ensemble accuracy on novel data. Nevertheless, the inclusion of a diversity measure in the optimization function when searching for accurate ensembles proved to be beneficial.
- Implicit diversity, as a result of slightly different architectures or training using different parts of the data set, is clearly beneficial for neural network ensemble creation. If individual neural networks are accurate and at least slightly diverse, even straightforward techniques like averaging a fixed number of neural networks, often obtained very high accuracy.
- Several techniques for ensemble creation optimize ensembles based on accuracy on a specific validation set. In this thesis it is shown that the correlation between accuracy on such validation set, and accuracy on another fresh set of data (a test set) often is very low.
- Experimentation show that the use of unlabeled instances together with predictions from the opaque model generally increase rule extraction accuracy. The technique suggested means that the same novel data instances used for actual prediction also are used by the rule extraction algorithm. For problems where predictions are made for sets of instances rather than one instance at the time, this is a way of obtaining better explanations of predictions made.

1.3 Thesis outline

The purpose of the first chapter; *Introduction*, is to present the two key criteria for data mining techniques recognized in this thesis; i.e. *accuracy* and *comprehensibility*. The first chapter also includes the problem statement, main contributions and this outline.

The second chapter, *Data Mining*, gives an overview of data mining as an activity. The purpose is to introduce the reader to data mining, including the relevant terminology. The chapter gives a fairly detailed description of the general tasks *predictive regression* and *predictive classification*. In addition, the important question how performance of different techniques or algorithms should be compared, especially over several data sets, is discussed.

The third chapter named *Data mining techniques*, gives a thorough description of some important basic data mining techniques. The purpose is to give the reader some necessary background theory. Most techniques described here are later used in the experimentation. The most important techniques covered are *neural networks*, *evolutionary algorithms* and *decision trees*.

The fourth chapter; *Rule extraction*, describes the problem of extracting knowledge from opaque models, especially neural networks. An established taxonomy for rule extraction approaches is presented together with some widely accepted criteria for evaluation of rule extraction algorithms. The evaluation criteria are used in the experimentation to evaluate the novel algorithm for rule extraction presented later. In addition, three existing rule extraction algorithms are presented in detail. Although the algorithms are not extensively evaluated, some main advantages and drawbacks are identified and discussed. The purpose is to familiarize the reader with some typical algorithms for rule extraction and lay the foundation for later comparisons.

Chapter 5 introduces *ensembles*, both basic theory and related work. The purpose of this chapter is to introduce the reader to ensembles in preparation for the presentation of a novel technique for ensemble creation.

Chapter 6 presents a novel rule extraction algorithm named *G-REX*. The algorithm is first described in detail and G-REX is then evaluated based on five experiments using public data sets.

Chapter 7 contains a lengthy case study called *The Impact of Advertising*. This case study, which was undertaken over a period of three years, illustrates the use of neural networks and rule extraction in the marketing domain. Results from this case study are, among other things, used to evaluate G-REX; especially regarding the criterion *generality*.

The eighth chapter; *A novel technique for ensemble creation*, contains six studies, all about ensembles. Here, the novel technique *GEMS* is introduced and evaluated in several experiments.

The ninth and final chapter; *Conclusions and future work*, reports the overall conclusions of this thesis. Naturally these conclusions are based on the problems identified in the problem statement. Finally, several suggestions for future work are given, regarding both rule extraction and ensemble creation.

Chapter 2

Data mining

The definition of data mining given in chapter 1 (as well as most other definitions) emphasizes that data mining is an activity with a clear goal. The overall purpose is to support decision making by turning collected data into actionable information. Using this perspective, data mining is the key activity in a larger process called *knowledge discovery in databases (KDD)*. A generic description of KDD is given in Figure 1.

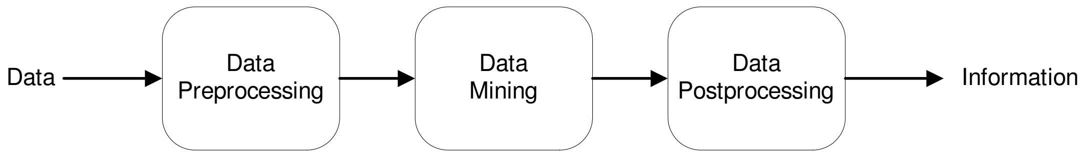


Figure 1: The KDD process.

Data can come from several different sources and in a variety of formats. The purpose of preprocessing is to transform the input into an appropriate form for data mining. Preprocessing typically involves steps like fusing data from multiple sources, selecting the data relevant for the mining task and cleaning data; e.g. handling missing values and outliers. The output of preprocessing is a *standard data matrix*; i.e. a vector of objects (*tuples* or *instances*) where each instance is a set of *attribute* values. If there are n instances and each instance has p attributes, the standard data matrix thus has n rows and p columns. Data mining uses the preprocessed data to produce *models*, typically used for either description or prediction. The purpose of the postprocessing step is to make sure that only valid and useful data mining results are actually used. Postprocessing often includes activities like hypothesis testing and visualization.

The process of transforming data into information is, in practice, always context dependent; i.e. KDD is at all times performed in a specific situation and with a specific purpose. Although most research focuses on data mining techniques (the technical context), it is important to realize that for executives the ultimate goal is to add business value. This viewpoint is sometimes referred to as the *business context* of data mining.

KDD, as described above, is actually part of a larger process called *the virtuous cycle of data mining* [BL97]; see Figure 2. In the virtuous cycle, KDD represents the activity *transform data into actionable information using data mining techniques*. To exploit the full potential of the techniques, data mining must be part of a company's strategy; i.e. data mining could typically be considered as part of customer relationship management.

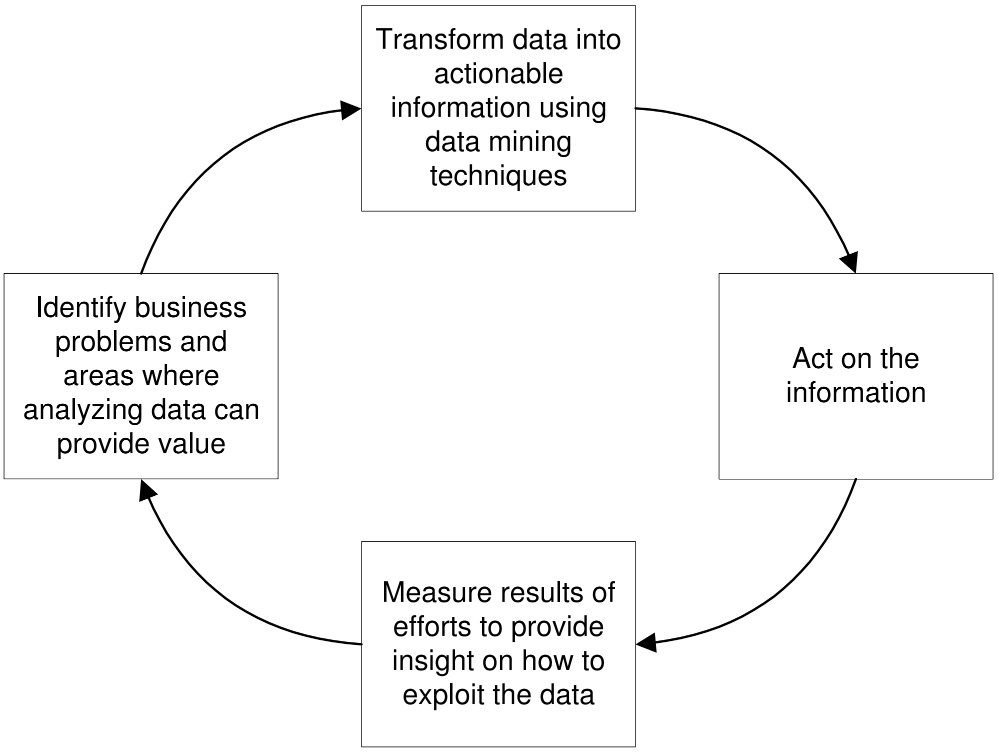

Transform data into actionable information using data mining techniques
Act on the information
Measure results of efforts to provide insight on how to exploit the data
Identify business problems and areas where analyzing data can provide value

Figure 2: Virtuous cycle of data mining (adopted from [BL97]).

Whether the term *data mining* should be used for the entire virtuous cycle, the *transform data into actionable information* activity or just as one part of the KDD process, depends on the abstraction level. In this thesis, data mining mainly refers to applying different data mining techniques, but the business context and its demands are also recognized. More specifically, the fact that results from data mining techniques ultimately should be used by human decision-makers places some demands on the data mining models. Since different techniques produce different kinds of models, these demands will in fact often determine which data mining technique to use.

One particular and important demand, introduced in chapter 1 (arguably also following from the fact that most business executives still are unfamiliar with data mining and data mining techniques) is the fact that transparent models are preferred to *black-box* models. Black-box models are models that do not permit human understanding and inspection, while *open-box* methods produce, at least, limited explanations of their inner workings.

Data mining combines techniques from several disciplines. Many algorithms and techniques come from the field of *machine learning (ML)*, i.e. the sub-field of artificial intelligence focused on programs capable of learning. In the data mining context, learning most often consists of establishing a (general) model from examples; i.e. data instances where the value of the target variable is known.

Statistics is the other main contributor to the data mining field. Predictive algorithms, sampling methodologies, experimental design and metrics to capture the performance of the data mining effort are some important examples.

Other important subjects are *computer technology*, *decision support systems* and *database technology*. Since data mining requires complex calculations to be applied to large quantities of stored data, only the recent advances in computer technology have made large-scale data mining practical and profitable. Decision support systems is a term covering all information technology used by companies to make informed and better decisions. In [BL00], the authors point out the need for two different databases; one operational system that handles transactions, and one decision support system where historical records can be studied. A special case of a decision support system database, called a *data warehouse* is a large database fed by several operational systems. When incorporated into the warehouse, data is normally cleaned, transformed and often even summarized and aggregated. For data mining this is a double-edged sword; the data becomes readily available, but sometimes valuable information is destroyed in the process. Historically, data warehouses have been used mainly for reporting and not mining. The trend during the last decade is, however, that increasingly, data warehouses also store non-aggregated data, and that data warehouses are built with data mining in mind [BL00].

2.1 A generic description of data mining algorithms

In [HMS01] the authors describe data mining algorithms in terms of four aspects:

- **Model or patterns structure**: determining the underlying structure or functional forms from the data.
- **Score function**: judging the quality of a fitted model.
- **Optimization and search method**: optimizing the score function and searching over different model and pattern structures.
- **Data management strategy**: handling data access efficiently during the search and optimization.

Model or pattern structures represent the general functional forms, e.g. a neural network with certain architecture or a linear regression model with unspecified parameter values. A fitted model or pattern has specific values for its parameters; e.g. a *trained* neural network.

Score functions quantify how well a model or parameter structure fits a given data set. Optimally the score function should measure the utility, but usually some simple generic score function based on accuracy is used.

The goal of optimization and search is to find the structure and parameter values that maximize the score function. The optimization and search procedure is the key element of the data mining algorithm and determines how the algorithm actually operates.

The data management strategy determines how data is stored, indexed and accessed. Most data mining algorithms assume that all data tuples can be accessed quickly and efficiently in the primary memory, which clearly is an oversimplification when using really large data sets. As a matter of fact, many algorithms do not even include a data management strategy. Some algorithms, like decision trees, scale very poorly when applied directly to data residing in secondary storage [HMS01].

In the context of this thesis a *model* is a global summary of a data set; it makes statements about every possible point in the input space. A *pattern structure*, on the other hand, makes statements about restricted regions of the input space. Model building in data mining is data-driven. The purpose is to capture the relationships in the data and create models for, typically, prediction or description. The validity of the data mining process thus depends on some basic, and most often not explicitly expressed, assumptions. First of all, the past must be a good predictor of the future since most data mining models are built from historical data. Second, the necessary data should be readily available. Finally, the data must, of course, contain the "relationship" that should be mined.

The purpose of a data mining effort is normally either to create a *descriptive model* or a *predictive model*. A descriptive model presents, in concise form, the main characteristics of the data set. It is essentially a summary of the data points, making it possible to study important aspects of the data set. Typically, a descriptive model is found through *undirected* data mining; i.e. a bottom-up approach where the data "speaks for itself". Undirected data mining finds patterns in the data set but leaves the interpretation of the patterns to the data miner. The data miner must also determine the usability of the patterns found. The most characteristic descriptive modeling task is *clustering*, i.e. decomposing or partitioning a data set into groups. Typically, points inside a group should be similar to each other and, at the same time, as different as possible from points in other groups.

Normally, a predictive model is found from *directed* data mining; i.e. a top-down approach where a mapping from a vector input to a scalar output is obtained by applying some supervised learning technique on historical (training) data. Most supervised learning techniques require that the correct value of the target variable is available for every training instance.

The predictive model is thus created from given known values of variables, possibly including previous values of the target variable. The training data consists of pairs of measurements, each consisting of an input vector $x(i)$ with a corresponding target value $y(i)$. The predictive model is an estimation of the function $y=f(x;\theta)$ able to predict a value y, given an input vector of measured values x and a set of estimated parameters θ for the model f. The process of finding the best θ values is the core of the data mining technique. As mentioned in the introduction, when the target value is one of a predefined number of discrete (class) labels, the data mining task is called classification. If the target variable is a real number, the task is called regression. Predictive regression and predictive classification are described in detail in chapters 2.3 and 2.4, respectively.

Figure 3 shows how data from both a *data warehouse* and *operational databases* is fed to the data mining algorithm in use. The data mining algorithm uses a *score function* to produce a *model*, which is used on *novel data* (a *production set*) to produce predictions.

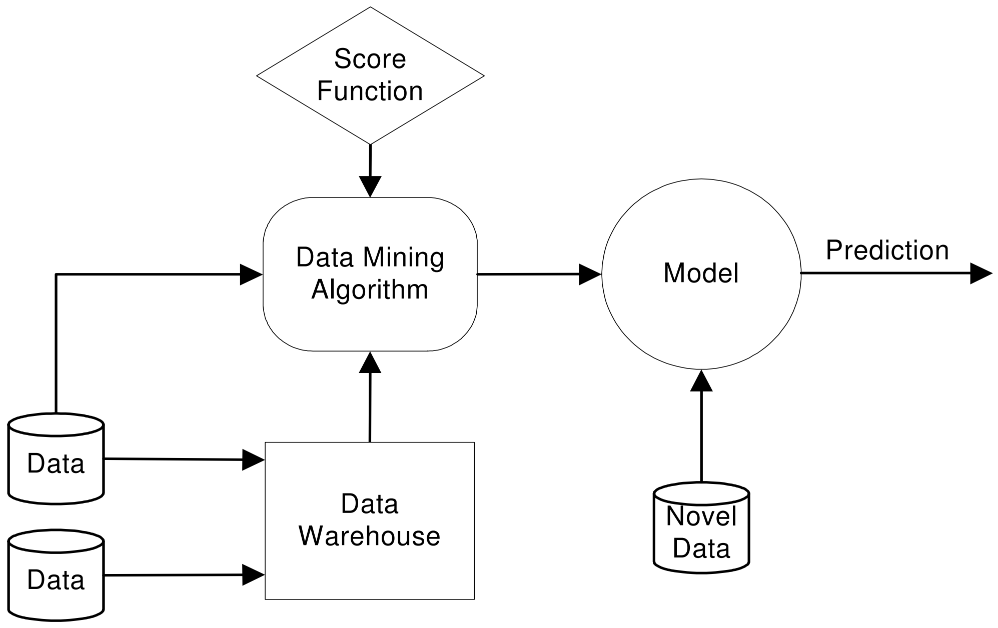


Figure 3: Predictive modeling.

It should be noted that the purpose of all predictive modeling is to apply the model on novel data (a *test* or *production* set). It is therefore absolutely vital that the model is general enough to permit this. One particular problem is that of *overfitting*; i.e. when the model is so specialized on the training set that it performs poorly on unseen data.

Naturally, descriptive models and predictive models could (and often should) be used together in data mining projects. As an example, it is often useful to first search for patterns in the data using undirected techniques. These patterns can suggest segments and insights that improve the results of directed modeling.

Score functions are used to determine the utility of the data mining model. Ultimately the purpose of the score function is to rank models based on their performance. Most score functions focus on the accuracy of the model. Typically, well-defined statistical measurements are used. Both predictive models and descriptive models have natural score functions. For predictive models the score function clearly should measure the error; i.e. the difference between predictions and targets. For descriptive models it is slightly harder to define obvious score functions, but they normally capture the discrepancy between the observed data and the proposed model.

Whether to use a score function measuring only goodness-of-fit or also trying to capture generalization performance (i.e. how well the model describes or predicts data outside the training set) is a subtle issue. Another, related, question is if simpler models should take precedence over more complex; something that could be achieved by using a score function penalizing high complexity. A straightforward way is to minimize a score function of the form in Figure 4 below. The penalty function puts a premium on simplicity by measuring the complexity of the model.

$$score(model)= error(model) + complexity(model)$$

Figure 4: A generic score function balancing accuracy and complexity.

2.2 Data

Obviously, data is central to all data mining activity. First of all, data must be available and in a suitable format. In most real-world, larger scale, applications the necessary data is initially stored in several different relational databases and data warehouses. For data mining purposes it is, however, often assumed that the data has been preprocessed into a standard data matrix.

Some data sets, however, do not fit well into the table format. One example is a time series where consecutive values correspond to measurements taken at consecutive times. If a time series is stored as a two-variable matrix the ordered aspect of the data is lost, something that would probably lead to a poor model.

Each attribute (column in the standard data matrix) represents a specific property of the objects; i.e. each property is described by the values in that column. Obviously it is important to distinguish between different measures. Is, for instance, a specific property *ordered*, i.e. is it natural to say that one value is smaller or greater than another?

Columns with no natural order are normally referred to as *categorical* or *nominal*. Categorical attributes have a well-defined set of values, usually categorical labels. The values are in general represented as strings, but could easily be transformed into numbers if so required. The naïve approach to assign a number to each value does, however, introduce a spurious ordering not present in the original data, which could alter the problem. The *binary scale* where the measure is either 0 or 1, is a special case of categorical data. One general approach to eliminating the problem of spurious ordering is to introduce a *flag variable* for each value; i.e. the categorical column is replaced by one binary column for each value originally present. This coding is normally referred to as a *localist representation*, or simply *1-of-C*.

There are different kinds of ordered columns; i.e. the scale types have different properties.

- Ordinal scale. Data elements may be ordered according to their relative size or quality. It is possible to rank ordinal data, but not to quantify differences between two ordinal values. Operations such as conventional addition and subtraction are meaningless.
- Interval scale. It is possible to quantify the difference between two interval scale values but there is no natural zero. Operations such as addition and subtraction are meaningful. Since the zero point on the scale is arbitrary, ratios between numbers on the scale are not meaningful; i.e. operations such as multiplication and division cannot be carried out directly. But ratios of *differences* can be expressed; for example, one difference can be twice as large as another.
- Ratio scale. The numbers assigned to objects have all the features of interval measurement and also have meaningful ratios between arbitrary pairs of numbers. Operations such as multiplication and division are therefore meaningful. The zero value on a ratio scale is non-arbitrary. Variables measured at the ratio level are called ratio variables, or often simply *true numeric*.

If, for some reason, an ordered attribute has to be binary coded, *thermometer coding* is an option. Before the coding, the attribute has to be discretisized into intervals. If there are *n* intervals, thermometer coding would use *n-1* binary digits. The lowest interval is coded with all zeroes, while the second lowest has all but the last digit set to 0, and so forth up to the highest interval, which consists of all ones. With this coding, each digit represents a specific interval boundary. For an example showing how five intervals are coded using thermometer coding; see Table 1 below.

Interval	**Thermometer coding**
Lowest	0000
	0001
	0011
	0111
Highest	1111

Table 1: Thermometer coding.

Most data mining techniques require the data to be in some specific format; e.g. numbers. Thus, one important step before applying the data mining techniques is to understand what the data represents and possibly convert it to a specific format. Two other common problems for the data miner are those of *missing data* and *outliers*. Missing data is a field without a value, which can occur for several reasons and could be handled in different ways. Some techniques, like for instance decision trees, do not require any specific handling. For other techniques, the best approach could be to insert derived values like the mean value or the mode value. An outlier is a data point that lies outside the expected values of the data. The outlier is in some way very different from the other data points. Since data mining builds models from data sets, outliers could severely influence the model, especially if the outlier is in some way incorrect. On the other hand, it should be noted that outliers sometimes contain very important information. The most common approach to handling outliers is to replace them with an "appropriate" value. As described above, data preparation is a key process when performing data mining. There are many steps prior to applying the data mining technique to the standard data matrix. Gathering relevant data from different sources, handling missing values and outliers, preprocessing the data to suit the technique etc. are all activities that will ultimately determine the success of many data mining projects. As the authors of [BL00] frankly put it:

> *Data is always dirty.* (p. 181)

2.2.1 Terminology for the different data sets used during data mining

When performing data mining, the data set is normally split in several parts. Each part is used differently and for different purposes. Unfortunately, the names of the data sets are not fully standardized. A first, important, division is to separate the parts of the data set used to build the model from the parts used to evaluate the model.

- The *training set* is the part of the data set used to build the model.
- The *validation set* is used for model (or parameter) selection. The purpose of the validation set is to enable selection by scoring the model on a part of the data set not used for building the model.
- The *test set* is used to evaluate the fitted model. The test set is in no way used to build the model, but must be kept "hidden" until the model is completely finished. One very important property of the test set is that it should produce results similar to what can be expected from a fresh data set. With this in mind, performance on the test set can also be used to estimate data quality.
- The *production set* is the data set on which the model is actually used.

With these definitions, the test set (but not the production set) is assumed to be available when data mining is performed. Specifically, correct target variable are on hand. Although this is probably the standard terminology used by data miners, it is not the terminology used in most research papers. Researchers instead normally report results on a test set, and if a specific holdout set is used to somehow rank and select models, this is referred to as a validation set. Sometimes multiple holdout sets are used. One holdout set could be used for "early stopping" of neural network training, another for the ranking of models and a third for evaluation on unseen data. In that case most researchers would call the first two holdout sets for validation set 1 and validation set 2, and the third for the test set. This is also the terminology used in this thesis; i.e. the important results are test set results, which are always on a part of the data set not used at all during model building and selection.

2.3 Predictive regression

Regression is the task of learning a target function f that maps each instance x to a continuous output y. The input variables $x_1, x_2, \ldots, x_p$ are variously referred to as features, attributes, explanatory variables and so on. Each attribute can be continuous or discrete, but most regressive techniques internally handle all attributes as real-valued.

When using supervised learning to obtain a predictive regression model, the purpose is to find a regressive function minimizing an error function over all training instances. Here, the assumption is that the target variable is a numeric scalar. When using supervised learning, the data miner has access to a training set where both input vectors and target values are known. The training set is therefore

$$D = \{(\mathbf{x}(1), y(1)), \ldots, (\mathbf{x}(N), y(N))\} \tag{1}$$

where $x \in R^p$, y is a scalar and N is the size of the training set. The functional relationship between x and y can be expressed as:

$$d = f(\mathbf{x}) + \varepsilon \tag{2}$$

where $f(\mathbf{x})$ is some function of vector x and ε is a random variable representing noise. The statistical model described by (2) is called a *regressive model*. In this model the function $f(\mathbf{x})$ is defined by

$$f(\mathbf{x}) = E[y \mid \mathbf{x}] \tag{3}$$

where E is the statistical expectation operator. The exact functional relationship between x and y is usually unknown. The purpose of the supervised learning is to build a model representing the function f. This model could then use x to predict y. One very natural criterion for optimizing the parameters Θ of the model is the minimization of the expected difference between the target value and the prediction from the model. Writing the prediction from the model as $\hat{y} = F(\mathbf{x}, \Theta)$, the score function to be minimized becomes the *mean square error (MSE)* between the prediction and the target.

$$MSE = E[(y - F(\mathbf{x}, \Theta))^2] = E[(y - f(\mathbf{x}))^2] + E[(f(\mathbf{x}) - F(\mathbf{x}, \Theta))^2] \tag{4}$$

Here, the first term is independent of Θ, making it sufficient to minimize the second term. To make the dependence on the training set D explicit, the approximating function may be rewritten as $F(x, D)$. The mean-squared error of using $F(x, D)$ as an estimator of the regression function $f(x)$ is

$$E_D[(E[y \mid \mathbf{x}] - F(\mathbf{x}, D))^2] \tag{5}$$

where the expectation operator E_D represents the average over all training sets D of given size N. Taking expectations with respect to D, MSE can be decomposed into two terms; *bias* and *variance*; see (6).

$$\begin{aligned} MSE = E_D[(E[y \mid \mathbf{x}] - F(\mathbf{x}, D))^2] = (E_D[F(\mathbf{x}, D)] - E[y \mid \mathbf{x}])^2 + \\ E_D[(F(\mathbf{x}, D) - E_D[F(\mathbf{x}, D)])^2] \end{aligned} \tag{6}$$

The first term is the square of the bias of the approximating function $F(x, D)$, measured with respect to the regression function $f(x)$, and the second term represents the variance of the approximating function. Very often this bias-variance decomposition of MSE is written as

$$MSE = E[(\hat{y} - y)^2] = (E[\hat{y}] - y)^2 + E[(\hat{y} - E[\hat{y}])^2 \tag{7}$$

or just

$$MSE = bias^2 + variance \tag{8}$$

The bias term measures how much the average prediction deviates from the target value, while the variance term measures how much predictions fluctuate around the expected value; see Figure 5 below. The bias term reflects the systematic error in the predictive model; i.e. how far the average prediction is from the corresponding target. The variance term determines how much the prediction will vary across different potential data sets of size N; i.e. it measures the sensitivity of the prediction to the particular training set used.

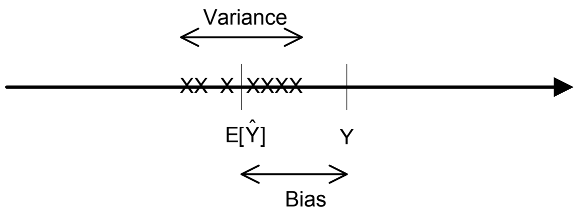


Figure 5: Bias and variance.

To decrease bias, the model should be flexible, but this obviously is at the risk of having high variance. If the variance is low, on the other hand, the model may not be able to represent the relationships in the data; making the bias high. So, there is a potential trade-off between using simple models with few parameters (high bias but low variance) and more complex models (low bias but high variance.) This dilemma is sometimes referred to as the *bias/variance dilemma*.

2.3.1 Prediction of time series

A special case of predictive regression is time series; i.e. to predict a future value in a series where previous values are known. If $Y(t)$ is a time series, to predict at horizon H is the problem of finding, at a given instant t_0, an estimate $\hat{Y}(t_0+H)$ of $Y(t_0+H)$. $Y(t_0+H)$ depends on previous values of Y, and possibly previous values of other variables.

When it is assumed that $Y(t)$ only depends on previous values of Y, the problem is referred to as *autoregression*, and is typically handled by using a Box-Jenkins autoregressive model; see equation (9).

$$Y(t) = a_0 + a_1 Y(t-1) + a_2 Y(t-2) + \ldots + a_p Y(t-p) + \varepsilon(t) \tag{9}$$

where $a_0, a_1, a_2, \ldots, a_p$ are regression coefficients and $\varepsilon(t)$ represents random noise not covered by the model.

To find coefficient values, a nonlinear least squares method is normally used. Most often this is done using an iterative solution technique rather than direct computation, see e.g. [BJ76].

2.3.2 Score functions for regression

Score functions used for regression minimize the error; i.e. the difference between the prediction $\hat{y}$ and the target value y. The most common error function is probably MSE, see (10) below.

$$MSE = \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n} \tag{10}$$

Although MSE is normally the score function during model construction, different error functions could be used when evaluating and reporting final results. If the error should have the same unit as the predicted entity, either *mean absolute deviation* (MAD) or *root mean square error* (RMS), could be used; see equations (11) and (12).

$$MAD = \frac{\sum_{i=1}^{n}\left|(\hat{y}_i - y_i)\right|}{n} \tag{11}$$

$$RMS = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}} \tag{12}$$

Another option is to report the correlation between the prediction and the actual value. One standard formula for measuring linear correlation is the *correlation coefficient r*; see equation (13).

$$r = \frac{\sum_{i=1}^{n}(\hat{Y}_i - \overline{\hat{Y}})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(\hat{Y}_i - \overline{\hat{Y}})^2 \sum_{i=1}^{n}(Y_i - \overline{Y})^2}} \tag{13}$$

The correlation coefficient is a real value $r \in [-1, 1]$. A positive number indicates a positive correlation, whereas a negative number indicates a negative correlation. The closer r is to 0 the smaller the correlation. A perfect correlation exists when $r = \pm 1$.

The *coefficient of determination* (R^2), defined as the square of the correlation coefficient, thus is a real value $R^2 \in [0, 1]$. R^2 is interpreted as the proportion of the variance in the dependent variable explained by the model; see equation (14).

$$R^2 = \left(\frac{\sum_{i=1}^{n}(\hat{Y}_i - \overline{\hat{Y}})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(\hat{Y}_i - \overline{\hat{Y}})^2 \sum_{i=1}^{n}(Y_i - \overline{Y})^2}} \right)^2 \tag{14}$$

2.4 Predictive classification

Predictive classification is the task of learning a target function f that maps each instance x to one of the predefined class labels y. The target attribute y (the class label) is a discrete attribute and can only take values from a predefined set $\{C_1, C_2,\ldots,C_n\}$. Since a predictive classification model should be able to assign every possible input vector to one of the predefined classes, the classes must partition the input space; i.e. the classes are non-overlapping and exhaustive.

As discussed in [KLR+98], there are three basic approaches to the building of predictive classification models:

- **Specifying boundaries**: The classification is performed by dividing the input space into regions, where each region is associated with one class. No direct attempt is made to model either the class-conditional or posterior class probabilities. Instead the decision boundaries are modeled directly; i.e. there is a direct mapping from inputs x to one of N class labels $C_1, C_2, \ldots, C_n$. Decision trees and support vector machines [Vap95], are two examples of techniques using this approach.
- **Using probability distributions:** The class-conditional distributions $P(\mathbf{x}|C_j)$ are modeled explicitly, and along with estimates of $P(C_j)$ are inverted via Bayes rule to give $P(C_j|x)$ for each class. When predicting, the class with the highest probability is chosen. Classifiers using this approach are normally referred to as *Bayesian classifiers* because of the use of Bayes theorem.
- **Using posterior probabilities:** The posterior class probabilities $P(C_j|x)$ are modeled explicitly, and for prediction the maximum of these probabilities is chosen. The standard feed-forward neural network is, under certain assumptions, an example of this approach; see 3.3.2.

All approaches above consist of a two-step process: first an inductive step, where a model is constructed from data, and then a second, deductive, step where the model is applied to test instances. Learners using this scheme are sometimes called *eager learners*.

An alternative approach is to omit the model building and directly classify novel instances based on available training instances. Such approaches are named *lazy learners* or *instance based learners*. The most common lazy approach is *nearest neighbor* classification. Given an instance to classify, the algorithm first finds the majority class C_m among the k closest (according to some distance measure) data points in the training set. The new instance is then classified as belonging to class C_m. The value k is a parameter, and the entire technique is known as *k-Nearest Neighbor* (kNN) classification.

kNN is a simple and frequently used technique for classification problems. In spite of its simplicity, kNN often performs quite well. One purpose of using kNN is to get an idea of the classification rate that should, at the very least, be achieved by more powerful methods like neural networks and decision trees; see e.g. [Bis95].

2.4.1 Score functions for classification

The obvious way to evaluate a classification model is the *misclassification* (or *error*) rate; i.e. the proportion of instances classified incorrectly; see (15) below.

$$Error\,rate = \sum_{i=1}^{n} I(\hat{y}_i, y_i) = \frac{Number\,of\ wrong\ predictions}{Total\,number\,of\ predictions} \qquad (15)$$

where $I(x, y) = 1$ if x is not equal to y and 0 otherwise. The proportion of instances classified correctly is referred to as the *accuracy*.

$$Accuracy = 1 - Error\,rate = \frac{Number\,of\ correct\ predictions}{Total\,number\,of\ predictions} \qquad (16)$$

There are, however, a couple of subtleties involved. First of all, for many problems, some misclassifications are more significant than others. One way to handle this is to have different penalties (costs) in the error function for different mistakes. Since the purpose of constructing a classifier is to minimize the error function used, this will solve the problem, assuming the costs are correctly assigned. A special case is when one class dominates the data set, with maybe 99% of all instances. Obviously a classifier assigning every new instance to the majority class will be correct 99% of the time, an error rate probably hard to improve. One way of handling this could be to use *oversampling*; i.e. the proportions in the training set are altered by adding copies (possibly slightly distorted) of instances from the dominated classes.

When presenting classification results, the performance metrics normally used are either *error rate* or *accuracy*, as defined above. One common way to visualize classification results is *confusion matrices*. In a confusion matrix, predictions are tabled as rows and the actual as columns; see an example with two classes (yes/no) in Figure 6. In the example, 87 of 100 instances are classified correctly.

		ACTUAL	
		Yes	No
PREDICTED	Yes	50	3
	No	10	37

Figure 6: A sample confusion matrix.

Although accuracy is the most widely used measure for classification, there are alternative metrics. These metrics are used particularly when the data set is imbalanced. For binary classification, the rare class is often more interesting than the majority class. With this in mind, the rare class is referred to as the *positive class*. Based on this, the numbers in a confusion matrix are called:

- True positive (TP): The number of positive instances correctly predicted by the model.
- False positive (FP): The number of positive instances wrongly predicted as negative by the model.
- False negative (FN): The number of negative instances wrongly predicted as positive by the model.
- True negative (TN): The number of negative instances correctly predicted by the model.

Based on these numbers, some commonly used percentages can be calculated. The *sensitivity* (or the *true positive rate*) is the proportion of positive examples correctly predicted; i.e. *TP/(TP+FN)*. Similarly, the *specificity* (*or true negative rate*) is the proportion of negative examples correctly predicted; i.e. *TN/(TN+FP)*. The *false positive rate* is *FP/(TN+FP)* and *false negative rate* is *FN/(TP+FN)*. Two even more common metrics are *precision* (*p*) and *recall* (*r*); see (17) and (18) below.

$$p = \frac{TP}{TP + FP} \tag{17}$$

$$r = \frac{TP}{TP + FN} \tag{18}$$

Precision determines the proportion of instances classified as positive that actually are positive. Recall measures the proportion of positive instances that are actually classified as positive.

A *receiver operating characteristic* (*ROC*) curve is a graphical representation of the trade-off between the true positive rate and the false positive rate. In ROC curves, the true positive rate is plotted along the *y* axis and the false positive rate is shown on the *x* axis. Each point on the curve corresponds to a specific model generated by the classifier. A good classifier should be located as close to the upper left corner as possible. The area under the ROC curve, (*AUC*) is a measure that could be used for comparing different models. If the model is perfect, the area will be 1, while a model performing random guessing will have an area of 0.5. Figure 7 below shows a sample ROC curve where Classifier 1 clearly outperforms Classifier 2.

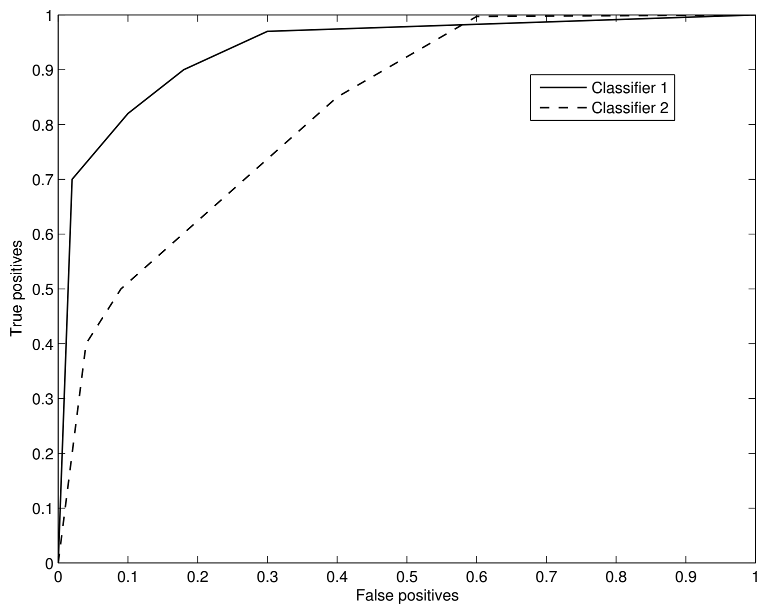


Figure 7: A sample ROC curve.

ROC curves can be generated if a classifier produces a continuous output that can be used to sort the predictions, from the least likely instance to be classified as positive, to the most likely. Very briefly, the procedure consists of stepping through the sorted instances in order, starting with the least likely. At each step, all lower ranked instances are assigned to the negative class, while the current instance and all higher ranked are assigned to the positive. Using this procedure, each step will produce a point on the ROC curve.

Another situation where ROC curves can be generated is when the classifier uses a threshold to determine how to interpret the continuous output. If, as an example, a classifier produces a continuous output between zero and one, a threshold of 0.5 is often used; i.e. everything above 0.5 is positive and everything below is negative. But, if this threshold is varied, each threshold value would produce a point on the ROC curve.

2.5 Clustering

Clustering is the activity of segmenting a diverse data set into a number of clusters, based on similarity. Clustering is distinguished from classification by having no predefined classes.

Clustering requires that the term similarity must have a meaning for the data set at hand; i.e. it must be possible to determine if two data points are "close" or "distant". There are several distance measures that could be used for clustering, but the most common is Euclidean distance; see equation (19).

$$d(i,j)=\sqrt{\sum_{k=1}^{p}(x_k(i)-x_k(j))^2} \tag{19}$$

Some clustering algorithms measure the distance between single elements, some between clusters. Distance between clusters could in turn be measured in several different ways; e.g. distance between centroids or the smallest distance between a data point in one cluster and a data point in the other (*single link*).

Clustering algorithms can be divided into *hierarchical* and *partitional*. Hierarchical algorithms gradually merge points or divide clusters into subclusters. For partitional algorithms the user must first supply the desired number of clusters to the algorithm. The set of clusters is then created directly and the algorithm iteratively move points to more appropriate clusters.

Hierarchical algorithms that merge are called *agglomerate* and algorithms that divide are called *divisive*. Agglomerate algorithms normally start with each data point in an individual cluster and then iteratively merge clusters. The agglomerate algorithms differ in the way they merge clusters at each level, but the purpose is always to merge clusters with similar data points. Divisive algorithms, on the other hand, initially place all data points in one cluster and then repeatedly split clusters. Obviously the idea is to split clusters where data points are not sufficiently close.

Partitional algorithms use a score function to determine the goodness of a proposed clustering. Partitional clustering is thus a search through the space of possible assignments of data points into a predetermined number of clusters; i.e. a combinatorial optimization problem with the purpose of finding a clustering C of n data points into K clusters that maximizes a score function. A typical score function should minimize the *within-cluster variation* and maximize the *between-cluster variation*. As an example, if the within-cluster variation, *wc(C),* is measured as the sum of squares of Euclidean distances from each point to the center of the cluster and the between-cluster variation *bc(C)* is the sum of the Euclidean distances between cluster centers, the score function could be *bc(C)/wc(C)*. Obviously, for larger data sets the number of possible assignments makes exhaustive methods useless.

K-means clustering [McQ67] is an iterative clustering algorithm in which instances are moved between clusters until a desired set of clusters is reached. The algorithm assumes that the desired number of clusters, *k*, is given as input. The algorithm iteratively first assigns each instance to the cluster which has the closest (according to some distance measurement) mean and then recalculates the mean of every cluster. This is repeated until some convergence criterion is met. Although K-means has several flaws, including the inability to handle categorical data, and exhibits problems with outliers, it is often used as a first clustering algorithm.

2.6 Concept description

The data mining task *concept description*, as defined by the CRISP-DM consortium [Cri00], aims at generating an understandable description of concepts or classes. The purpose is not to generate high performance predictive models, but to gain insights. From this it is obvious that concept description is closely related to both classification and clustering. One typical example could be to "explain" the clusters found by a clustering algorithm. With this viewpoint, conceptual clustering techniques perform both clustering and concept description at the same time.

Models produced by classification algorithms can also be regarded as concept description *as long as they are comprehensible*. The results from a concept description project would most often be new information, typically presented as verbal descriptions or rules.

According to the CRISP-DM manual one important difference between classification and concept description is that the classification model must be global (i.e. applicable to all possible instances), while a concept description only has to describe important parts of concepts or classes.

2.7 Evaluation and comparison of classifiers

When discussing evaluation and comparison of classifiers, there are three major questions:

- How should the future error rate (i.e. on novel data) of a specific classifier be estimated using only results on available data?
- How should the results of two classifiers or two different algorithms be compared against each other on a specific data set?
- How should the results of two or more classifiers or algorithms be compared against each other over several data sets?

The error rate on the training set, sometimes referred to as the *apparent error rate*, is almost guaranteed to underestimate future misclassification since the predictive model is built to fit the training set. So, to estimate future error rate, it is necessary to evaluate the model on data not used during construction. Typically, two-thirds of the data available for model building is used for training and one-third (a validation set) is used for error estimation. The standard procedure is to alter specific parameters (like number of hidden units in a neural network) and ultimately choose the setting obtaining lowest error on the validation set. Naturally, once all parameters have been set, both training and validation data is available for the construction of the final model. With this approach, it must be noted that the validation set becomes part of the data used for building and selecting a specific model. The implication is that if an unbiased estimation of the future error rate for the final model is needed, this requires the evaluation of that model on yet novel data, a *test set*. Accuracy on test sets is also often used to compare the relative performance of different algorithms on the same data set. To iterate, training and validation data is used to build and select models. Test data is used to estimate future error rates, making it suitable for comparisons.

When comparing classifiers, either using validation data during construction and selection or on test data after final models are chosen, the important point is that all classifiers should be applied to the same, previously unused, data. The simplest way of achieving this is to directly set aside a part of the available data, as described above. This is called the *holdout method* and the data set used for evaluation is often referred to as a *holdout set*. The holdout method has several limitations. First of all, fewer instances are available for actual training. Second, the model is highly dependent of how the data is divided into training and holdout sets. A relatively small training set increases the variance of the model, while a relatively small holdout set increases the confidence interval of the estimated accuracy. In addition, the two sets are no longer independent. Since both parts were drawn from the original data set, the fact that one class is underrepresented in the training set automatically means that it is overrepresented in the holdout set.

If the holdout method is repeated a number of times, each using a random part of the available data as holdout set, the approach is called *random subsampling*. The estimated accuracy is of course the mean accuracy over all repetitions. Random subsampling still suffers from some of the problems of the holdout method since it does not use all available data for training. In addition, there is no control how often a specific instance is chosen for the holdout set, so some instances could be chosen more often, something that might affect the result slightly.

Cross-validation uses each of the N instances the same number of times for training and exactly once in the holdout set. The available data is first divided in k subsets (called *folds*) and then at each stage one fold is used as the holdout set and all others for training. This procedure is repeated k times, and it is ensured that each fold is used as holdout set exactly once. Again the overall estimated error is the average over all runs. Specifically the *leave-one-out* method, where $k = N$ (i.e. the holdout set consists of only one instance) is the least biased method for estimating future performance. The reason is that as much data as possible is utilized for all training, and that the holdout sets are mutually exclusive, while covering the entire data set. The obvious drawback is that it is very computational expensive since the procedure must be repeated N times. The most common value for k is instead 10. When $k = 10$ the entire procedure is called *10-fold cross-validation*. Although k-fold cross-validation is extremely common when comparing classifiers or reporting results, it should be noted that there are several inherent problems. First of all, each fold is for many data sets rather small, resulting in small holdout sets and therefore large confidence intervals for the estimated accuracy. Second, training sets overlap considerably. Because of this, results for individual classifiers or algorithms may very well depend on the exact folding used. With this in mind, it is sometimes suggested that a series of cross-validation tests should be performed, each using different, randomized, folding. One typical example would be to run 10-fold cross-validation ten times on a specific data set; a method referred to as 10x10-fold cross-validation. Naturally, 10x10-fold cross-validation is rather computationally intensive.

As an alternative to cross-validation, training instances can be sampled with replacement; i.e. the training set may contain several duplicates of one instance. This procedure is called *bootstrapping*. Instances not chosen for the training set become part of the test set. Often, the size of the training set is fixed to the original size of the entire data set. This will lead to, on average, approximately 63% of all instances being present in the training set. Normally this is repeated a number of times and then the overall accuracy is obtained by combining the accuracies of each bootstrap sample. Specifically, in the widely used *.632 bootstrap*, the accuracy is calculated using (20) below. *b* is the number of bootstraps, acc_i is the accuracy on bootstrap *i*, and acc_{tot} is the apparent accuracy; i.e. the accuracy on the original data set when all instances are used for training.

$$Accuracy_{.632\,bootstrap} = \frac{1}{b}\sum_{i=1}^{b} 0.632\,acc_i + 0.368\,acc_{tot} \tag{20}$$

2.7.1 Statistical techniques for error rate estimation

When discussing accuracy or error rates of models, it is important to distinguish between the *sample error* and the *true error*. The sample error of a model, with respect to some sample of instances *S*, drawn from the space of possible instances *X*, is the fraction of *S* that the model misclassifies. The true error *p*, is the probability that the model misclassifies an instance draw at random from the distribution *D*. The distribution *D* specifies for each possible instance $x \in X$ the probability that, if just one instance is drawn randomly from *X*, this instance is *x*.

When estimating the error rate of a certain classifier, two slightly different techniques could be used; either a confidence interval is calculated for the true error, or a hypothesis test is carried out to determine if the true error is less than or equal to some probability p_0.

If there is just one training set and one holdout set, the only information used is the number of errors *e* on the holdout set of size *N*. The correctness when classifying a novel instance (from the holdout set) can be regarded as a random variable with a *Bernoulli distribution*. The number of errors, *Y*, on a holdout set of size *N*, thus is a sum of Bernoulli distributed random variables, making *Y* a random variable with a *binomial distribution*.

The sample error $Y = e/N$, is an unbiased estimator for p. The variance for e is $Np(1-p)$, where p has to be substituted with the estimator e/N. Since the parameters for the binomial distribution governing the sample error are known, a confidence interval for p is established by finding the interval centered around the sample error holding an appropriate amount (e.g. 95%) of the total probability under this distribution. Normally, this calculation is not performed directly; i.e. actually using the binomial distribution. The standard procedure is instead to use the *normal distribution* as an approximation; leading to the confidence interval for p given in equation (21).

$$p = \frac{e}{N} \pm Z_{\alpha/2}\sqrt{\frac{\frac{e}{N}(1-\frac{e}{N})}{N}} \qquad (1-\alpha) \tag{21}$$

It should be noted that this includes two approximations. First, the standard deviation of the sample error is used instead of the (unknown) standard deviation of the true error. Second, the normal distribution is used to approximate the binomial distribution. Both approximations are very good as long as N is sufficiently large (~30) or $Np(1-p) \geq 5$.

If hypothesis testing should be used, the *null hypothesis* is normally that the true error rate is higher than a specific value, p_0. The reason is that if the null hypotheses can be rejected, at a certain significance level, this means that the true error, at the same significance level, is smaller than p_0. The test statistic, if the same approximations as above are used, is found in equation (22) below.

$$Z = \frac{\frac{e}{N} - p_0}{\sqrt{\frac{p_0(1-p_0)}{N}}} \tag{22}$$

When using cross-validation over, for instance, ten folds, it is also possible to obtain either a confidence interval for the mean error or to perform hypothesis testing determining if the error rate is smaller than a value p_0, at a specific significance level. Since the number of observations (folds) is rather small, the *Student's t-distribution* is used instead of the normal distribution. More specifically, the quantity $\frac{\bar{X}-\mu}{S/\sqrt{k}}$, will follow a t-distribution with k-1 degrees of freedom. $\bar{X}$ is the average error rate over the folds, μ is the (true) mean error rate, S is the sample standard deviation and k is the number of folds.

2.7.2 Statistical techniques for comparison of classifiers or algorithms

When comparing two algorithms on one data set, the standard approach is to use *k-fold cross-validation* and then apply a *paired t-test*. The difference in error rate (or equivalently accuracy) between the two classifiers for fold *i* is called d_i. Given that both error rates are approximately normal, their difference d_i is also normal. So, if $\bar{d}$ is the mean difference over all *k* folds, $\frac{\sqrt{k} \cdot \bar{d}}{S}$ is *t-distributed* with *k-1* degrees of freedom. Naturally, the null hypothesis is that this distribution has a zero mean; i.e. that there is no difference in error rate between the two classifiers.

It should be noted, however, that the paired t-test over cross validation folds, must be regarded as overly confident. In [Die98], Dietterich warns against using t-tests after either repetitive random sampling or cross-validation. The reason is that the variance is underestimated and thus the Type I error (i.e. rejecting a true null hypothesis) is elevated. Dietterich instead suggests an alternative procedure, called the *5x2cv paired t-test* that overcomes this problem. When using the *5x2cv paired t-test*, five replications of two-fold cross-validation is performed. In each replication, the data set is divided into two equal-sized sets. If $d_i^{(j)}$ is the difference in error rate between the two classifiers on fold *j* and replication *i*, the average on replication *i* is $\bar{d}_i = (d_i^{(1)} + d_i^{(2)})/2$ and the estimated variance is $s_i^2 = (d_i^{(1)} - \bar{d}_i)^2 + (d_i^{(2)} - \bar{d}_i)^2$. Under the null hypothesis that the two algorithms have the same error rate, $d_i^{(j)}$ is the difference of two identically distributed proportions, and can be treated as approximately normal distributed with zero mean and unknown variance. Using the (somewhat questionable) assumptions that $d_i^{(1)}$ and $d_i^{(2)}$ are independent normals, and that each s_i^2 is independent, the sum of variances is chi-square with five degrees of freedom. So, the t-statistic (with five degrees of freedom) for the *5x2cv paired t-test* is:

$$t = \frac{d_1^{(1)}}{\sqrt{\sum_{i=1}^{5} s_i^2 / 5}} \tag{23}$$

Actually, the numerator in equation (23) above is arbitrary; any $d_i^{(j)}$, *j=1, 2; i=1,…,5* could be used, leading to ten different statistics. With this in mind, Alpaydın [Alp99] suggested an extension combining all ten possible statistics. Using this extension, the test variable is *F-distributed* with *(10, 5)* degrees of freedom. The test statistic is calculated using equation (24) below. The Alpaydın version of the test is more robust, has a lower type I error and higher power.

$$f = \frac{\sum_{i=1}^{5}\sum_{j=1}^{2}(d_i^{(j)})^2}{2\sum_{i=1}^{5} s_i^2} \tag{24}$$

An even more interesting question is how several algorithms should be compared using a number of data sets. Despite the need for such comparisons, this problem remains unsolved. As a matter of fact, there is no standard technique commonly accepted. Instead, several more or less ad hoc, common sense procedures are used. Demšar in [Dem06] analyzes and discusses how classification algorithms should be compared over a number of data sets. First of all, he distinguishes between the case when there are exactly two algorithms and the more general case comparing multiple algorithms.

When comparing exactly two algorithms, Demšar starts by discussing the naïve choice of averaging results from several data sets. This approach is, according to Demšar, not useful for statistical inference, since results on different data sets are normally not comparable. In addition, averages are very susceptible to outliers. The use of averages over data sets is consequently very uncommon in research papers. On the other hand, the use of paired t-tests, where the paired results are results from one data set, is rather frequent. Despite this, Demšar points out that this approach suffers from three weaknesses. First of all: the t-test only makes sense when the differences over the data sets are commensurate. Second: unless the sample size is large enough (~30 data sets), the paired t-test requires that the difference between the two random variables compared are distributed normally. The nature of the problem does not give any provisions for normality, and the number of data sets is usually less than 30. Third: t-tests are, just like averages, seriously affected by outliers. With this in mind, Demšar does not recommend the use of t-tests over several data sets.

Demšar instead recommends a non-parametric test called the *Wilcoxon Signed-Ranks Test*, which is an alternative to the paired t-test. The main principle is that the differences in performance between the two classifiers for each data set is ranked, ignoring signs. Then the sums of ranks for positive and negative differences are compared. More specifically, let d_i be the difference in performance (e.g. error rate or accuracy) on **data set** *i*, $i=1, 2, \ldots, N$. All differences are now ranked according to their absolute values. The smallest difference gets rank 1 etc. In case of ties, average ranks are used. Let R^+ be the sum of ranks for the data sets on which the second algorithm outperformed the first, and R^- the sum of ranks for the opposite; see equations (25) and (26) below. Ranks of d_i=0 are split evenly among the sums; if there is an odd number of them, one is ignored.

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2} \sum_{d_i=0} rank(d_i) \qquad (25)$$

$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2} \sum_{d_i=0} rank(d_i) \qquad (26)$$

Now, the test statistic $T = min(R^+, R^-)$. Most books on general statistics include a table of exact critical values for T for N up to 25. Table 2 below shows some critical values for two-tailed tests with α=*0.05*.

#Data sets	6	8	10	12	15	17	20	22	25
Critical value	1	4	8	14	25	35	52	66	90

Table 2: Critical values of the Wilcoxon *T* statistic, two-tailed test.

When comparing the Wilcoxon signed ranks test to the t-test, Demšar points out that it assumes commensurability but only qualitatively: greater differences still count more, but absolute magnitudes are ignored. The Wilcoxon signed ranks test is also safer, since it does not assume normal distributions. Finally, the outliers have less effect on Wilcoxon than on the t-test.

When it comes to comparing more than two classifiers, it must be noted that none of the tests above was designed for reasoning about the means of multiple random variables. Despite this, such usage is very common in research papers. One example of a very questionable procedure, given by Demšar, is when seven algorithms are compared by conducting all 21 paired t-tests. When so many tests are made, a certain proportion of the null hypotheses will be rejected due to pure chance. Demšar instead recommends using the non-parametric *Friedman test* [Fri37]. When using the Friedman test to compare k algorithms over N data sets, the algorithms are ranked for each data set separately, with the best performing algorithm getting rank 1 etc. In case of ties, average ranks are assigned. The test then compares the average rank for each algorithm over all data sets. Let R_j be the average rank for algorithm j. Under the null hypothesis that all algorithms are equivalent, the Friedman statistic is distributed according to χ_F^2 with k-1 degrees of freedom, when N and k are big enough ($N>10$ and $k>5$). For a smaller number of algorithms and data sets, exact critical values have been computed. The Friedman test statistic is given in (27) below.

$$\chi_F^2 = \frac{12N}{k(k+1)}\left(\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right) \tag{27}$$

Iman and Davenport [IM80], however, showed that this test is undesirably conservative and derived a better statistic given in (28) below. This statistic is distributed according to the *F-distribution* with *k-1* and *(k-1)(N-1)* degrees of freedom.

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2} \tag{28}$$

If the null hypothesis is rejected, this means that not all algorithms have performed equally well. To determine which algorithms that have, in fact, statistically different performance, post-hoc tests are used. The *Nemenyi test* [Nem63] is similar to the Tukey test for ANOVA, and is used when all algorithms are compared to each other. The performance of two algorithms is significantly different if the corresponding average ranks differ by at least the critical difference given in (29).

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{29}$$

Critical values for α=*0.05* are given in Table 3 below.

#Algorithms	2	3	4	5	6	7	8	9	10
Critical value	1.960	2.343	2.569	2.728	2.850	2.949	3.031	3.102	3.164

Table 3: Critical values for the two-tailed Nemenyi test.

When, instead, one specific algorithm is compared with all others, it is better to use one of the general procedures for controlling the family-wise error; i.e. the probability of making at least one Type I error in any of the comparisons. For the test statistic for comparing algorithms *i* and *j* using these methods, see (30) below.

$$z = \frac{(R_i - R_j)}{\sqrt{\frac{k(k+1)}{6N}}} \tag{30}$$

The z value is used to find the corresponding probability from the table of normal distribution, which is then compared to an appropriate α. The tests differ in the way they adjust the value of α to compensate for multiple comparisons. When using the *Bonferroni-Dunn test* [Dun61], the same equation as for the Nemenyi test is used to calculate the CD. The only difference is the critical values; see Table 4 below.

#Algorithms	2	3	4	5	6	7	8	9	10
Critical value	1.960	2.241	2.394	2.498	2.576	2.638	2.690	2.724	2.773

Table 4: Critical values for the two-tailed Bonferroni-Dunn test.

By comparing Table 3 and Table 4, it is obvious that the power of the post-hoc test is much greater when one specific algorithm is compared to all others.

One last interesting observation, made by Demšar, is the fact that the procedure of acknowledging only *significant differences* (where the significance is determined by using a statistical test on each data set) when comparing or ranking algorithms over several data sets, does in fact make the test *less* reliable. The reason is simply that such procedure draws an arbitrary threshold of $p<0.05$ between what counts and what does not.

So, in summation, Demšar recommends the Wilcoxon signed ranks test for comparison of exactly two classifiers, and the Friedman test for comparison of multiple classifiers, when the evaluation is performed over many data sets.

Chapter 3

Basic data mining techniques

There are many different data mining techniques. Some techniques are tailored directly to a specific data mining problem; e.g. market basket analysis. Other techniques, like, for instance, neural networks and genetic algorithms are general techniques originally used in other areas.

Most data mining techniques come from either statistics or machine learning. Examples of techniques based on statistics are linear regression, multiple linear regression and logistic regression, but also Bayesian classification, k-nearest-neighbor classification, K-means clustering and parametric models for describing probability distributions and density functions. Other important techniques from the field of statistics, like principal component analysis, independent component analysis and wavelet transformations, are mainly used during preprocessing, or to gain an initial understanding of the data set. Techniques from machine learning include decision trees and neural networks, but also evolutionary algorithms. In addition, there are more recent predictive modeling techniques like Support Vector Machines [Vap95] and Random Forests [Bre01], which, despite very good results in studies, are rarely used in data mining projects.

From the description above, it is obvious that a data miner must be familiar with many techniques from different areas, not only to choose the correct tool for a specific project, but also to be able to combine techniques in a useful way. There are several software packages created with this in mind, but even so, data mining is still a demanding task, requiring knowledge in many fields.

3.1 Linear regression

Statistical regression is a parametric, supervised technique that generalizes a set of numeric data by creating a mathematical equation relating the input variables to the single output variable. With *linear regression* the model describes the dependent variable as a linear combination of the independent variables. Thus, the linear regression equation is of the form:

$$Y = f(x_1, x_2, ..., x_n) = a_1 x_1 + a_2 x_2 + ... + a_n x_n + c \qquad (31)$$

where x_1, x_2, … x_n are the independent variables and a_1, a_2, … a_n and c are constants. Obviously, linear regression is only appropriate if the relationship between the dependent and independent variables is nearly linear.

The simplest form of linear regression uses only a single independent variable as the predictor of the dependent variable. This type of regression is actually called *simple linear regression*, but most often the phrase *linear regression* would mean simple linear regression, and the term *multiple linear regression* is used for multiple independent variables.

The constants of equation (31) are found by minimizing the sum of squared errors between the data points in the training set and the corresponding points of the estimated regression model, something that all statistical packages have built-in functionality for. In a simple regression model, the least-squares estimators minimize the sum of squared errors from the estimated regression line. In a multiple regression model, the least-squares estimators minimize the sum of squared errors from the estimated regression hyperplane.

Once the regression model is found, it can easily be used to obtain predictions of the dependent variable. A single-valued prediction of Y is obtained by inserting the input values in the estimated regression equation.

3.2 Decision trees

A *decision tree* is a predictive model most often used for classification. Decision trees partition the input space into cells where each cell belongs to one class. The partitioning is represented as a sequence of tests. Each interior node in the decision tree tests the value of some input variable, and the branches from the node are labeled with the possible results of the test. The leaf nodes represent the cells and specify the class to return if that leaf node is reached. The classification of a specific input instance is thus performed by starting at the root node and, depending on the results of the tests, following the appropriate branches until a leaf node is reached.

The decision tree is created from examples (the training set) with the obvious requirement that it should agree with the training set. The basic strategy for building the tree is to recursively split the cells of the input space. To choose the variable and threshold on which to split, a search over possible input variables and thresholds is performed to find the split that leads to the greatest improvement of a specified score function. Typically this score function is based on some information theory measurement, like *information gain* or *entropy*. The overall idea is to minimize the depth of the final tree by always choosing splits that make the most difference to the classification of an instance. The splitting procedure could in principle be repeated until each cell contains instances from one class only. At the same time the decision tree must not simply memorize the training set, but should be capable of generalizing to unseen data; i.e. the decision tree should not overfit. The goal is thus to have a decision tree as simple (small) as possible, but still representing the training set well.

Two basic strategies for avoiding overfitting are to stop growth of the tree when some criterion has been met, or to afterwards reduce (prune) a large tree by iteratively merging leaf nodes. Obviously, goodness-of-fit could also be balanced against model complexity by using a score function based on the generic score function in Figure 4.

A specific technique for increasing accuracy is *boosting*; i.e. to combine several trees. Boosting works by creating multiple training sets from the original training set, where each instance has a weight determining how important that instance is for the entire classification. Initially, all weights are equal. First a decision tree is constructed and evaluated on the training set. Instances classified incorrectly are then assigned higher weights and the process is repeated a number of times. The final classification is a weighted-majority combination of all trees, each weighted according to how well it performed on the training set. It should be noted that boosting does not produce *one* model but several, which hampers inspection of the relationships found.

Actual decision tree algorithms differ in stopping criteria, choice of score function, how continuous data is handled, use of pruning etc.

Regression trees represent an alternative to statistical regression and take the form of decision trees where the leaf nodes contain numerical rather than categorical values. Regression trees are more accurate than linear regression equations when the data to be modeled is nonlinear. However, complex regression trees are quite awkward to interpret. For this reason, regression trees are normally combined with linear regression to form what is sometimes called *model trees* or *rule-based linear models*. With model trees each leaf node represents a linear regression equation.

3.2.1 C4.5 and C5.0/See5

Arguably the most popular tool for decision trees is C4.5 [Qui93]. C4.5 is based on the predecessor ID3 [Qui86] which, however, handled categorical data only. Both ID3 and C4.5 produce trees with varying number of branches per node. The ID3 approach to splitting favors attributes with many divisions, thus easily leading to overfitting.

C4.5 allows continuous data and also has improved techniques for splitting. More specifically, C4.5 uses *information gain ratio* instead of information gain as the score function for splitting, resulting in fewer branches and less bushy trees. The gain ratio takes the cardinality of each division into account to eliminate the bias towards choosing attributes with many outcomes. Using original information gain, attributes with unique values for each instance will always be selected.

More formally, the gain ratio is the ratio between the total information gain due to the split and the information gain attributable solely to the number of subsets; see equation (32).

$$GainRatio = \frac{Gain(D,S)}{H\left(\frac{|D_1|}{|D|},\ldots,\frac{|D_S|}{|D|}\right)} \tag{32}$$

where *Gain(D,S)* is the information gain for the split *S* on the data set *D*, and *H* is the standard entropy function.

C4.5 uses two primary strategies for pruning:

- *Subtree replacement* works bottom up, by iteratively replacing a subtree with a leaf node if the replacement results in an error rate close to that of the original tree.
- *Subtree raising* replaces a subtree by its most used subtree, meaning that this subtree is raised one level in the tree. Error rate is again used as the criterion to determine appropriate pruning.

C4.5 allows classification via either decision trees or rules generated from the trees. C4.5 also uses techniques to simplify complex rules. First, paths of the tree are combined in ways that do not change the behavior of tree. After that C4.5 tries to generalize each rule by removing clauses, comparing the predicted error rate with that of the original rule. After this process the rules might not be mutually exclusive and some tuples may not be covered by any rule. The second problem is handled by the introduction of a *default* class, assigned to any instance not covered by a rule. Finally C4.5 groups the rules for each class and eliminates those that do not seem to contribute much to the accuracy of the entire rule set. The final, reduced rule set is obviously less complex than the original rule set and provides easier inspection. How much accuracy on unseen data (if any) that is lost compared to the original decision tree is problem dependent. The question of how much accuracy it is reasonable to trade for increased understanding of the model must also be answered for each specific problem.

C5.0[1] [Qui98] is a commercial version of C4.5 widely used in many data mining packages such as Clementine and RuleQuest[2]. The algorithm for inducing the decision tree is very similar to that of C4.5, but the rule generation is different. Unlike C4.5, the exact algorithms used by C5.0 have not been revealed. One major improvement in C5.0 is boosting, which clearly increases the accuracy, but requires training of several models and thus fails to produce a model suitable for inspection.

CUBIST (called the "sister program of C5.0" on the RuleQuest website) is a commercial system performing predictive modeling for regression. The appearance of CUBIST is very similar to that of C5.0. CUBIST learns a set of unordered IF-THEN-ELSE rules with linear models in the conclusion part. There is no published work on CUBIST, but the system is essentially a rule-based version of MS [Qui92]. CUBIST can deal with both continuous and nominal variables and obtains a piecewise linear model of the data. In addition, CUBIST can combine prediction of this model with kNN predictions (as MS does).

[1] C5.0 is called See5 on the Windows platform. In this thesis the name C5.0 is used regardless of the platform actually used.

[2] http://www.rulequest.com

3.2.2 CART

Classification and Regression Trees (CART) [BFO+84] is a technique generating binary decision trees. Each internal node in the tree specifies a binary test on a single variable, using thresholds on real and integer-valued variables and subset membership for categorical variables. Entropy is used as a measure for choosing the best splitting attribute and criterion. Unlike C4.5, only two children are created at each split, however. The splitting is performed around what is determined to be the best split point. At each step, an exhaustive search is used to determine the best split. For details about the function used to determine the best split, see [BFO+84]. The score function used by CART is misclassification rate on an internal validation set.

CART demands that an ordering of the attributes is used. CART handles missing data by ignoring that instance when calculating the goodness of a split on that attribute. The tree stops growing when no split will improve the performance. CART also contains a pruning strategy which can be found in [KLR+98].

3.3 Neural networks

Artificial neural networks (ANNs) are loosely based on the function of the human brain. ANNs have proved to be successful in numerous data mining and decision-support applications, as well as in many other areas. ANNs are considered to be very powerful, general-purpose and readily applicable to predictive regression and classification. Some ANNs, however, like self-organizing maps [Koh82], are used for undirected clustering.

From a high level perspective, ANNs used for prediction utilize examples to establish a functional mapping between input and output. The function is realized by the *architecture* or *topology* (the number of units used and how they are connected) and a set of *weights*, i.e. how much adjacent units tend to activate or deactivate each other. The architecture is normally chosen before experimentation starts, but the weights are calculated during *learning.* For obvious reasons, this supervised learning is often called *neural network training*. The purpose of the training is to adjust the weights to minimize the error function over the training set. There are many different learning algorithms for different architectures. Once the network is fitted to the training set, it may be regarded as a function mapping an input pattern to an output and could be used on novel data. If the ANN is good at generalization it should produce reasonable outputs when presented with previously unseen input patterns.

3.3.1 Multilayer perceptrons

An ANN is often pictured as a graph with neurons as nodes and connections as weighted arcs. Figure 8 shows a sample layered ANN with *n* inputs to the input layer, an interior (called hidden) layer with *m* units, and *k* output units in the output layer.

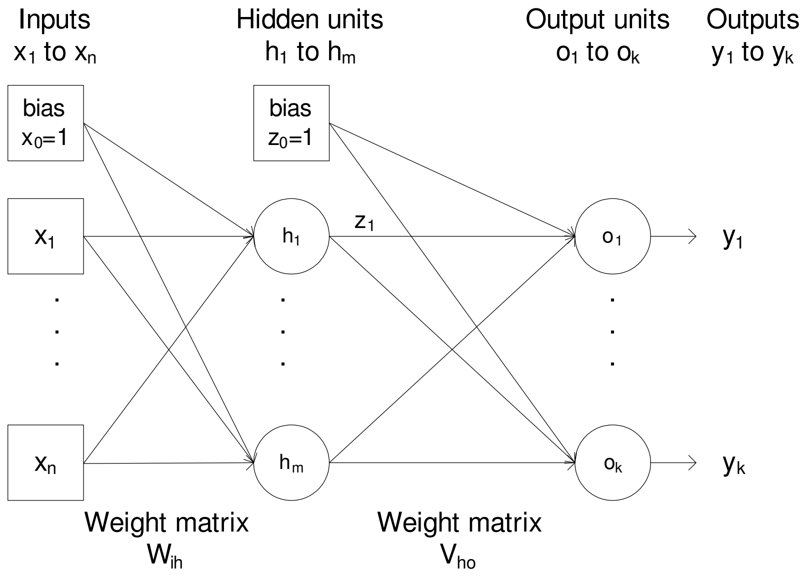


Figure 8: A sample ANN.

The output z_j from one hidden unit is calculated using equation (33)

$$z_j = f(\sum_{i=0}^{n} w_{ij} x_i) \tag{33}$$

where ***w*** is a matrix of weights, ***x*** the input vector (augmented with a bias x_0 which always is set to 1.0) and *f* the *activation function*. The activation function is most often non-linear to make the ANNs capable of handling non-linear relationships. The most common activation function is the *sigmoid*; see equation (34). An alternative is the *hyperbolic tangent* function; see equation (35).

$$f(x) = \frac{1}{1+e^{-x}} \tag{34}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{35}$$

The outputs y_i of the ANN in Figure 8 above are calculated using equation (36).

$$y_i = f(\sum_{h=0}^{m} v_{hi} z_h) \tag{36}$$

where f is the activation function for the unit in the output layer. This activation function is normally linear (i.e. the output is just the sum in the equation above) if the ANN is supposed to learn a regression problem. When performing classification, it is often desirable to be able to treat the outputs as posterior probabilities, in which case the activation function in the output layer should be either *sigmoid* (for binary problems) or *softmax* (for multiclass problem); see 3.3.2 below.

The example above, which is a layered feed-forward neural network, is the most common ANN architecture. This architecture is normally referred to as *multilayer perceptrons* (MLPs). Distinguishing for MLPs is that the signal propagates only in the forward direction and that the architecture consists of one input layer, one or more hidden layers and one output layer. A feed-forward network represents a function of its current inputs since it has no internal state other than the weights themselves.

If each unit is connected to every unit in the next layer the network is said to be *fully connected*. Normally units with a non-linear activation function are used for the hidden layers. The output layer could have a non-linear activation function or a linear activation function, depending on the range of the desired outputs. In theory, two hidden layers are enough to represent any function and a single hidden layer is enough to represent any continuous function; for details see [Cyb88] and [Cyb89]. *Generalized feed-forward nets* are a special case of MLPs, in which connections can bypass one or more layers.

It should be noted that when using fully-connected MLPs, the only design choices regarding architecture are the number of hidden layers and the number of units in each hidden layer. The number of input units is, of course, determined by the number of inputs in the data set. Similarly, most regression problems naturally have only one output unit. When performing binary classification, the standard procedure is to also use just one output unit. If so, all output values over a certain threshold level denote one class, while all values under that level represent the other class. If the classes are coded as 0/1, which is standard, the threshold is normally 0.5. If there are more classes than two, a localist coding is normally used, so the number of output units becomes equal to the number of classes. Localist coding thus has target patterns in the training set consisting of zeroes for every class but the associated class, which is represented with a '1'.

When predicting, the output unit with the highest output value determines the class associated with that input pattern.

3.3.2 Backpropagation training

The MLP is the function f that produces outputs values from the equation $y=f(x;\theta)$. The purpose of the training is consequently to find the best θ values, which for MLPs correspond to the architecture and the weights associated with the arcs. Since the architecture is often chosen before training starts, the term neural network training normally refers to the process of changing the weights from observed differences between desired and actual output. The overall purpose of supervised MLP training is to minimize an error function over the training set. Although there are many different algorithms for MLP learning, the most common is still some variation of *backpropagation* [Wer74], [RHW86]. When using backpropagation training, the error is based on the difference between the target value and the output for instance t; i.e. $r^t - y^t$. Backpropagation is a gradient descent algorithm where the error is minimized by moving weights in the direction that produces the steepest descent along the error surface; i.e. the negative gradient. The overall procedure is that this error is first used to change the weights between the hidden layer and the output units. After that, the error is backpropagated and weighted by the responsibility of the hidden unit when changing the weights between the input layer and the hidden layer. The responsibility is given as the weight v_h; i.e. the weight between the hidden layer and the output.

Regression

When using backpropagation to learn a regression function, there is normally only one output unit and the error function to minimize is MSE. To simplify the derivatives the exact error function is:

$$E(\mathbf{W},\mathbf{v}\,|\,\mathbf{x}) = \frac{1}{2}\sum_t (r^t - y^t)^2 \tag{37}$$

To update the weight between the hidden units and the output unit, the following equation is used:

$$\Delta v_h = \eta \sum_t (r^t - y^t) z_h^t \tag{38}$$

and to update the weights between the input layer and the hidden layer:

$$\Delta w_{jh} = \eta \sum_t (r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t \tag{39}$$

η is a constant, called the *learning rate*, which will determine the step size for the weight change.

Here, the direction and magnitude of each weight is calculated for every training instance before any actual updating is performed. This is called *batch learning*. A complete pass of all training instances is called an *epoch*. The alternative is to use *online learning*, where the weights are updated after every training instance. If online training is used, the learning factor should be smaller and the order of instances should be randomized between epochs.

Binary classification

When performing binary classification, a single output unit with a sigmoid activation function is used.

$$y^t = \frac{1}{1 + e^{-\left(\sum_{h=0}^{m} v_h z_h^t\right)}} \tag{40}$$

y^t approximates $P(C_1 | \mathbf{x}^t)$ so $\hat{P}(C_2 | \mathbf{x}^t) = 1 - y^t$. The error function now is *cross entropy*:

$$E(\mathbf{W}, \mathbf{v} | \mathbf{x}) = -\sum_t r^t \log y^t + (1 - r^t) \log(1 - y^t) \tag{41}$$

The updates become:

$$\Delta v_h = \eta \sum_t (r^t - y^t) z_h^t \tag{42}$$

$$\Delta w_{jh} = \eta \sum_t (r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t \tag{43}$$

which is identical to the ones used for regression. The *values* are, of course, different though.

Multiclass classification

Normally, a localist coding is used and each output unit represents one class. To make it possible to interpret the outputs as posterior probabilities, it must be ensured that the outputs sum to one. In practice this is, for multiclass problems, achieved by using a *softmax* activation function in the output layer. More specifically:

$$o_i^t = \sum_{h=0}^{m} v_{hi} z_h^t \tag{44}$$

$$y_i^t = \frac{e^{o_i^t}}{\sum_{j=1}^{k} e^{o_j^t}} \tag{45}$$

where y_i^t approximates $P(C_i \mid \mathbf{x}^t)$. The error function is:

$$E(\mathbf{W}, \mathbf{v} \mid \mathbf{x}) = -\sum_t \sum_i r_i^t \log y_i^t \tag{46}$$

and the update equations:

$$\Delta v_{hi} = \eta \sum_t (r_i^t - y_i^t) z_h^t \tag{47}$$

$$\Delta w_{jh} = \eta \sum_t \left(\sum_i (r_i^t - y_i^t) v_{hi} \right) z_h^t (1 - z_h^t) x_j^t \tag{48}$$

Richard and Lippman [RL91] showed that an MLP of enough complexity and given sufficient training data will, after suitable training, estimate posterior probabilities.

3.3.3 Alternative training procedures

Standard gradient descent converges rather slowly, but it is possible to improve performance significantly by using *momentum* or *adaptive learning rate*.

The key idea with momentum is to take a running average by incorporating the previous update in the current change, as there is a momentum due to the previous update. This is achieved by adding a term $\alpha \Delta w_i^{t-1}$ to the weight update equation in use. α is a constant value, often between 0.5 and 1.0. When using an adaptive learning rate, η is increased (by a constant factor) if the error over the training set decreased and is decreased geometrically if the error increased.

There are also numerous more sophisticated optimization methods. Bishop [Bis95] discusses several conjugate gradient and second-order methods. One, often significantly faster, alternative to backpropagation, is the Levenberg-Marquardt training algorithm [Lev44], [Mar63].

A totally different approach transforms the weights into a representation suitable for *genetic algorithms* (see 3.4) and treats the task of finding optimal parameters as a standard combinatorial optimization problem, for which genetic algorithms are well suited.

The complexity of neural network learning has been heavily investigated by researchers. It should be noted that the general problem of finding weights consistent with the training set is NP-complete [Jud90]. Baum and Haussler showed that the number of instances in the training set necessary for efficient learning grows as $O(W \log W)$, where W is the number of weights [BH89].

3.3.4 Overfitting

Since ANNs are so powerful, they are prone to overfitting. An overly complex model will memorize the noise in the training data and therefore generalize poorly. For an ANN, this will happen frequently if the architecture is too complex or if training is continued for too long.

One straightforward technique used to avoid overfitting is to continuously evaluate the network on a validation set during training. The training is stopped when the error on the validation set starts to rise, with the motivation that the network has started to memorize non-general details in the training set.

Other methods for avoiding overfitting try to find a network as small as possible, since smaller networks are less powerful and thus "must" focus on the general patterns to minimize the error function. Obviously, parameters like number of hidden layers and units per layer could be chosen from experimenting ("trial-and-error") with a validation set. More structured approaches either start with a large trained network and removes (*prunes*) unnecessary connections, or grow a larger network from a smaller one. The *optimal brain damage* algorithm [LJB+89] and the *tiling* algorithm [MN89] are examples of pruning and growing algorithms, respectively.

Regularization is yet another approach to avoid overfitting. It is based on the assumption that a network with smaller weights will be smoother and less likely to overfit. Training with regularization uses an error function similar to the generic score function in Figure 4 above. Here, the MSE is balanced against a term (MSW) that consists of the mean of the sum of squares of the network weights; see equation (49).

$$Error = \gamma MSE + (1-\gamma) MSW \tag{49}$$

γ is a parameter called the performance ratio. One automated way of finding a good value for γ is *Bayesian regularization* [McK92].

3.3.5 Tapped-delay neural networks

Tapped delay neural networks (TDNNs) are MLPs with a buffer (tapped delay line) containing the *k* most recent inputs; see Figure 9. A tapped delay line holds *k* delays through which the input signal passes. The output from the tapped delay line is thus a *k*-dimensional vector containing the input (which of course in itself normally is a vector) at the current time, the previous input etc. Since the network only uses *k-1* previous input values, *k* is an important parameter (often called *the window size*) and must be chosen with care. A tapped delay line could be regarded as a simple short-term memory, and TDNNs have therefore often been used for predictive modeling of problems with a temporal aspect; see e.g. [WHR92] or [SR87].

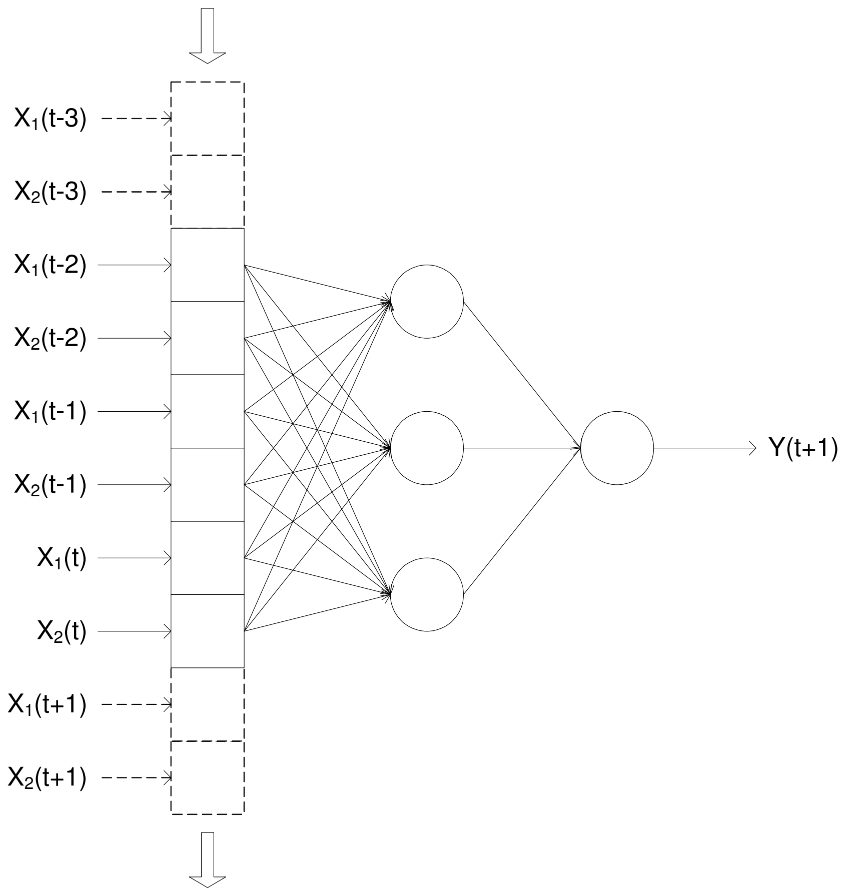


Figure 9: A tapped delay neural network with two tapped inputs.

3.3.6 Recurrent neural networks

A recurrent network is by definition a network where connections in both directions are allowed. A recurrent network in some way feeds outputs or signals from hidden layers back into its own inputs. This means that the activation levels of the network form a dynamic system.

The Elman network [Elm90], often referred to as a *simple recurrent network* (SRN), is normally a three-layer network with feedback from the hidden layer to the input layer; see Figure 10. This recurrent connection allows the SRN to detect and generate time-varying patterns. Put another way, Elman networks augment the input pattern with the condition of the hidden layer at the previous time step. Thus the feedback units establish a context for the current input. Elman networks are therefore, among other things, capable of handling conflicting input patterns since they use the context to discriminate between "identical" patterns occurring at different times.

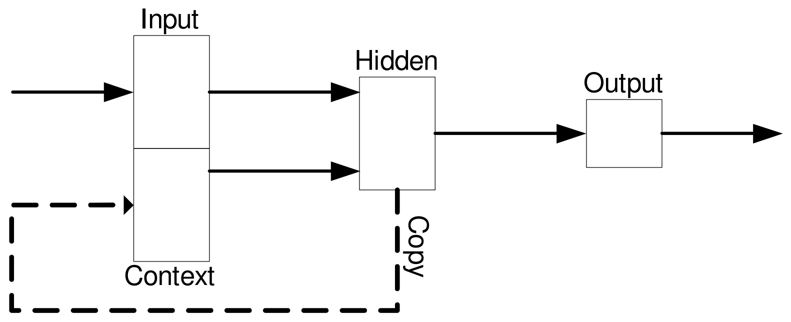


Figure 10: An SRN.

Normally, the SRN has sigmoid units in the hidden layer and linear units in the output layer. The delay from the hidden layer stores information for future use, so the network is capable of learning both temporal and spatial patterns.

Recurrent networks are, for obvious reasons, often used on problems with a temporal aspect. One advantage for recurrent networks compared to TDNNs is the fact that no window size has to be determined, i.e. the network will "choose" the number of previous input values to use.

3.3.7 Decision trees and ANNs – a short comparison

Decision trees and ANNs are often used for data mining. Although both methods have their origin in the field of machine learning, they have different advantages and drawbacks.

Tree models are easy to understand and explain. They also handle continuous and discrete variables with ease. Although most tree algorithms are used for classification, there are methods for constructing regression trees or rule-based linear models; e.g. CART and CUBIST.

ANNs are extremely powerful, normally outperforming tree models. ANNs can be used for both classification and regression tasks and there are architectures capable of temporal processing. Some preprocessing of the original data is normally required when ANNs are used. Two reasons for this are that ANNs internally handle all variables as continuous and that each variable preferably should be normalized to a specific interval.

One, often cited, drawback of ANNs is the inability to explain the found relationships. This is even more pronounced for domains where the ANN is used as part of a decision support system. Data mining obviously falls into this category. It is interesting to note that, although ANNs (without explanation facilities) are standard in most data mining packages, many research papers stress the importance of an explanation facility to obtain full acceptance of ANNs as data mining tools; see e.g. [LSL95] and [CS97a].

3.4 Genetic algorithms

Like ANNs, genetic algorithms[1] (GAs) [Hol75] are based on a biological metaphor; here the underlying theory is that of evolution. When GAs are used for problem solving, the solution has three distinct stages:

1. The individual potential solutions of the problem are encoded into representations that support the necessary variation and selection operations; often these representations, called *chromosomes*, are as simple as bit strings.

2. A fitness function judges which solutions are the "best" life forms, that is, most appropriate for the eventual solution of the problem. These individuals are favored in survival and reproduction, thus shaping the next generation.

3. Mating, (called *crossover*) and *mutation* produce a new generation of individuals by recombining features of their parents.

Eventually a generation of individuals will be interpreted back to the original problem domain and the most fit individual represents the solution.

[1] Genetic Algorithms, Genetic Programming and Evolution Strategies are collectively referred to as Evolutionary Algorithms.

The basic genetic algorithm is given in pseudocode below:

```
Input:
p_c                    // Probability for crossover
p_m                    // Probability for mutation

Output:
P                      // Population of evolved chromosomes

Algorithm (GA):
Create a population P consisting of N chromosomes;
While terminal condition is not met
  Evaluate the fitness of each chromosome in P;
  Create a new, empty, population P_new;
  While number of chromosomes in P_new is less than N
    Select a pair of chromosomes for mating;
    With the probability p_c exchange parts of the two selected
      chromosomes using crossover to create two offspring;
    With a probability p_m randomly change the gene values in the
      two offspring chromosomes;
    Place the resulting chromosomes in P_new;
  Replace P with P_new;
```

The exact way of selecting chromosomes for mating differs between implementations. Often, though, the probability for a certain chromosome to mate is proportional to its fitness. This strategy is called *roulette wheel selection*. Another popular method, called *tournament selection*, has chromosomes compete against each other for an opportunity to mate.

Normally GAs are used for optimization problems. The fitness function reflects the measure that should be optimized and candidate solutions are represented like chromosomes. The theoretical foundation for GAs is the schema theorem; see [Hol75]. In a nutshell it states that parts of chromosomes that contribute positively to the fitness function are more likely to be included in the next generation, meaning that the average fitness of a population is likely to rise between generations. Because of this, the search is really among different schemes (parts of chromosomes) with the effect that short above-average schemata receive exponentially increasing trials in subsequent generations. These building blocks are then combined (using genetic operators) into solutions, which should turn out to be near-optimal.

3.4.1 Genetic operators in GAs

Genetic operators are used to produce new chromosomes that combine components of their parents. The two basic operators are called *crossover* and *mutation*.

Crossover

The crossover operator randomly chooses a crossover point where the two parent chromosomes "break", and then exchanges the chromosomes parts after that point; see Figure 11 below. If a pair of chromosomes do not crossover, *cloning* takes place; i.e. the offspring chromosomes are exact copies of their parents.

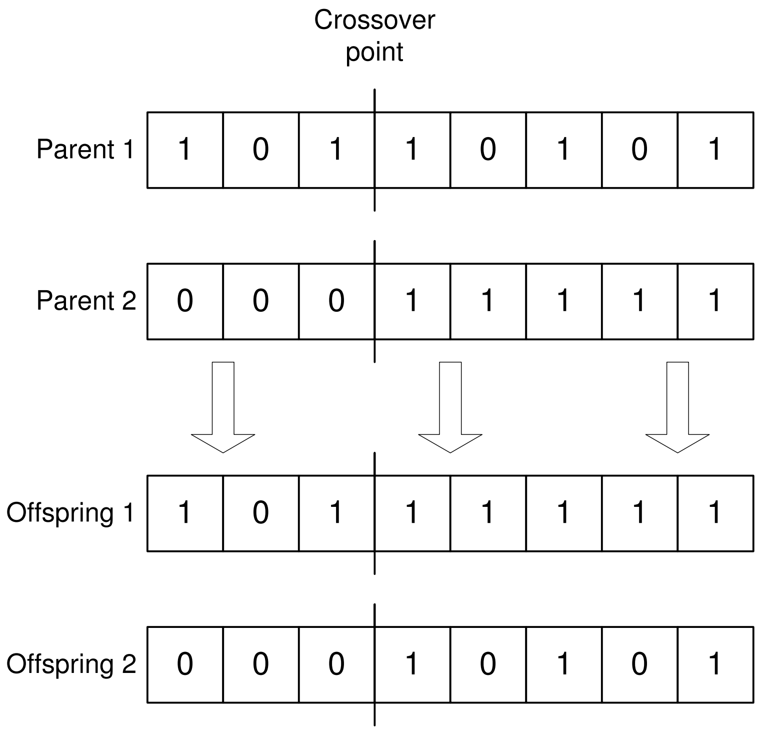


Figure 11: GA (single-point) crossover.

Mutation

Mutation takes a single candidate and randomly changes some aspect of it; i.e. a single gene value is changed. Mutation is important as a tool for avoiding stagnation. Normally mutation is harmful (just as it is in nature), but sometimes it leads to significant improvement in fitness. Technically speaking, mutation represents a random search and its role is to guarantee that the search algorithm is not trapped in a local optimum.

3.4.2 GAs in data mining

GAs are normally not used directly on any standard data mining problem. The most common application of GAs, in the data mining context, is probably as an alternative to standard training algorithms for ANN training. The process is quite simple; the weights of the network are coded as a chromosome, and the fitness is inversely proportional to the error function in use over the training set.

3.5 Genetic programming

The main idea of genetic programming (GP) [Koz92] is very similar to that of GA, with the important difference that the chromosomes to optimize represent a computer program. There are several different possible representations; e.g. linear structures or graphs (for details see [BNK+98]), but the majority of all GP applications use a tree-based representation. More specifically, a *genetic program* is normally a parse tree consisting of *functional nodes* and *terminal leaves*. Figure 12 shows a sample genetic program representing the Boolean expression $(X_1 > 10) \wedge (X_2 < 20)$.

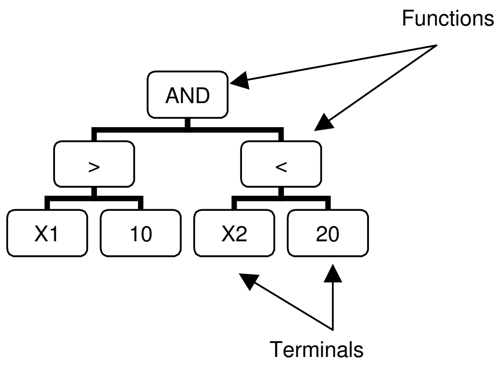


Figure 12: Genetic program representing a Boolean expression.

Internally the genetic programs are often written as S-expressions; see Figure 13.

$$(AND\ (> X_1\ 10)\ (< X_2\ 20))$$

Figure 13: S-expression representing a Boolean expression.

The choice of available functions (the *function set*) and terminals (the *terminal set*) must be made from the problem at hand. Function and terminal sets should be as small as possible to reduce the search space, but must also be able to represent a solution to the problem. Even more important is the choice of fitness function, which obviously must capture the property to be optimized, for the specific problem.

3.5.1 Generating the initial population

GP starts with a population of randomly generated genetic programs. As mentioned above, the available functions are determined from the current problem, but could typically include arithmetical functions, conditional operators, relational operators and Boolean operators. The terminals normally represent variables and constants. The initial trees vary in shape and size.

Since the initial population is created more or less randomly, it will contain a wide variety of programs. Normally, it is ensured that all programs are syntactically correct. Koza [Koz92] describes three basic methods for the actual creation:

- **Full**: The function in the root is first picked randomly from the function set. After that other functions are recursively selected as arguments until the prescribed depth of the tree is reached. At that time only terminals are chosen as leaves. Each and every time a function is selected, it is chosen randomly from the function set. The resulting tree is balanced and all branches are of equal length.
- **Grow**: The process is similar to Full but here each argument to a function could be a function or a terminal. The program is complete when no function misses an argument; i.e. when all leaves are terminals. Obviously, at max depth only terminals are chosen. Grow normally produces short and unbalanced trees having branches of varying length.
- **Ramped half-and-half** is a combination of Full and Grow. Ramped half-and-half creates an equal number of trees having depths from 2 to the max depth. All individual trees are created using either Full or Grow, but it is ensured that at each depth 50% of the trees are generated each way.

If, as an example, the max depth is 6 and the initial population should hold 100 individuals there would be 10 trees created by Grow having depth 2, 10 trees created by Full having depth 2, 10 trees created by Grow having depth 3, and so forth up to depth 6.

3.5.2 Evolving a population of genetic programs

The process of evolving the initial population in GP is extremely similar to that of a GA:

1. Execute each program in the population and calculate its fitness. Designate the best-so-far individual as the current result of the run.
2. Following the assigned probabilities, select a genetic operator to perform cloning, crossover or mutation.

3. If *cloning* is chosen, select one program from the current population and copy it to the new population. If *crossover* is chosen, select a pair of programs, create a pair of offspring and place them in the new population. If *mutation* is chosen, select one program, perform mutation and place the mutant in the new population. All programs are selected with a probability based on their fitness. If the probability for being selected is directly proportional to the fitness the selection process is termed *roulette wheel* selection.

4. Repeat step 3 until the size of the new generation is equal to the size of the current population.

5. Replace the current population with the new one.

6. Go to step 1 and repeat the process until the termination criterion is satisfied.

3.5.3 Genetic operators in GP

When crossover is to take place, two points (one in each parent) are randomly chosen. The sub-trees below those random points are swapped to produce two offspring; see Figure 14 below.

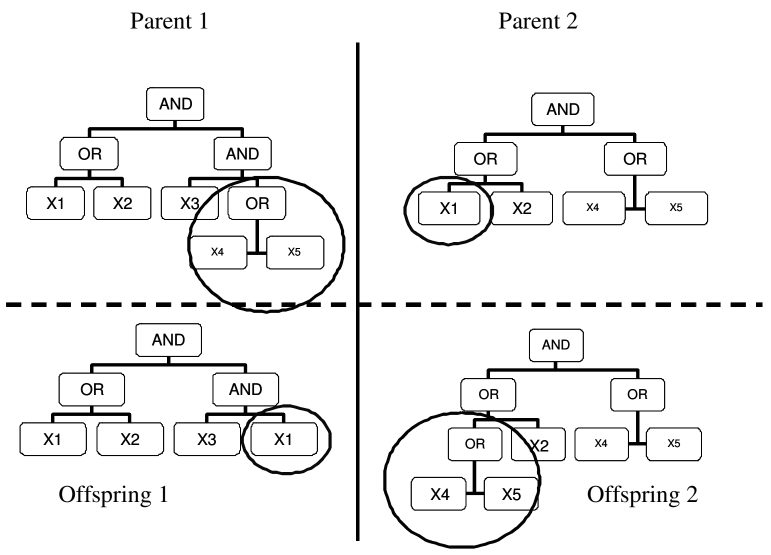


Figure 14: GP crossover.

Mutation is performed on a single tree. First a random point in the tree is chosen. The sub-tree starting at this point is then replaced by a random tree generated according to the process chosen for generating the initial population; see Figure 15 below.

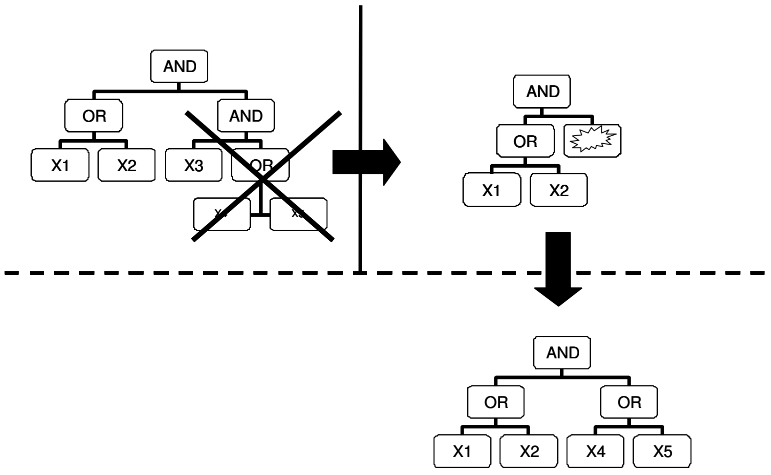


Figure 15: GP mutation.

Crossover is normally very common; around 85% of each generation is created by crossover. Mutation is typically used on less than 2% of each generation.

Chapter 4

Rule extraction

For the data mining domain, the lack of explanation facilities seems to be a serious drawback for techniques based on ANNs, or, for that matter, any technique producing opaque models. Within the field of *symbolic AI*, the term *explanation* refers to an explicit structure which is used internally for reasoning and learning and externally for the explanation of the results to the user. Normally, the explanation facility in symbolic AI includes intermediate steps of the reasoning process, like trace of rules firing and proof structures. Generally speaking, the explanation facilities are capable of answering the "*how*" and "*why*" questions. The answer to a how-question is an explanation of how the result was found. A why-question is supplied by the user during execution of the system and the answer specifies why the system performed a certain operation; e.g. queried the user.

Experience from the field of expert systems has shown that an explanation capability is a vital function provided by symbolic AI systems. In particular, the ability to generate even limited explanations is absolutely crucial for user acceptance of such systems [DBS77]. Since the purpose of most data mining systems is to support decision making, the need for explanation facilities in these systems is apparent. Nevertheless many systems (especially those using ANN techniques, but also ensemble methods like boosting) must be regarded as black boxes; i.e. they are opaque to the user.

In [ADT95] the authors Andrews, Diederich and Tickle highlight this deficiency of ANNs, and argue for *rule extraction*; i.e. to create more transparent representations from trained ANNs:

> *It is becoming increasingly apparent that the absence of an explanation capability in ANN systems limits the realizations of the full potential of such systems, and it is this precise deficiency that the rule extraction process seeks to reduce.* (p. 374)

Andrews, Diederich and Tickle also list five more reasons for the importance of being able to interpret trained ANNs:

- For ANNs to be used in safety-critical problem domains, the explanation facility must be considered mandatory.
- If ANNs are integrated into larger software systems without an explanation facility, the entire systems would be very hard to verify.
- An explanation facility may provide an experienced user with the capability to anticipate or predict a set of circumstances under which the ANN is likely to generalize poorly.
- ANNs may discover previously unknown dependencies, but without an explanation facility these dependencies are incomprehensibly encoded in the model.
- Extracted rules from ANNs could be directly added to the knowledge base of a symbolic AI system.

It should be noted that an explanation facility also offers a way to determine data quality, since it makes it possible to examine and interpret relationships found. If the discovered relationships are deemed doubtful when inspected by a human, they are less probable to actually add value. "Nonsense" relationships found would, if used on a production set, most likely produce poor results. The task for the data miner is thus to identify the complex but general relationships that are likely to carry over to production data and the explanation facility makes this easier.

There are basically two methods that can be used to gain understanding of the relationship found by a trained ANN; *sensitivity analysis* and *rule extraction*. Sensitivity analysis does not produce a new model, but is used to gain some basic understanding of the relationship between input variables and the output. Rule extraction is an activity where the trained ANN is transformed into another, more comprehensible model, representing the same relationship.

4.1 Sensitivity analysis

Sensitivity analysis does not provide explicit rules, but is used to find the relative importance of the inputs to the output of the neural net. There are some small variations in how the analysis is performed, but the overall procedure is to record changes in the output following changes in specific input attributes. Normally, the average value for each input is chosen as the starting point and the changes should vary from small changes all the way up to the extreme values. If the difference in output is small even for large changes in a specific attribute, this attribute is probably not very important; i.e. the network is *insensitive* to that attribute. Other attributes might have a large effect on the output and the network is then said to be *sensitive* to these attributes.

Obviously, there could be combinations of features that are very important for the network, which would require the sensitivity analysis to be performed on two or more attributes at the same time, in order to find these particular patterns.

It is safe to say that sensitivity analysis is a good tool to get some basic understanding of the underlying problem, but also that it normally is unable to produce explanations. The purpose of performing a sensitivity analysis is thus usually not to actually explain the relationship found. Instead sensitivity analysis is normally used either as a tool to find and remove unimportant input attributes or as a starting point for some more powerful explanation technique.

4.2 Rule extraction from trained neural networks

The knowledge acquired by an ANN during training is encoded as the architecture and the weights. The task of extracting explanations from the network is therefore to interpret, in a comprehensible form, the knowledge represented by the architecture and the weights.

Craven and Shavlik [CS97A] coined the term *representation language* for the language used to describe the network's learned model. They also used the expression *extraction strategy* for the process of transforming the trained network into the new representation language.

4.2.1 Taxonomy of rule extraction approaches

In [ADT95] the authors proposed a classification schema for rule extraction approaches. The presentation below intentionally follows the one given in the paper closely. In the paper, the method of classification is based on five dimensions:

- The *expressive power* of the extracted rules; i.e. the chosen representation language.
- The *translucency* of the view taken within the rule extraction technique of the underlying ANN units; i.e. does the technique look inside the trained neural net and utilize knowledge about connections and weights or is the network treated as an oracle.
- The extent to which the underlying ANN incorporates specialized training regimes.
- The quality of the extracted rules; i.e. how well the required explanation is performed.
- The algorithmic complexity; i.e. how efficient the underlying rule extraction algorithm is.

Representation languages typically used include *(if-then) inference rules*, *M-of-N rules*, *fuzzy rules*, *decision trees* and *finite-state automata*. In the translucency dimension there are two fundamentally different approaches; *decompositional* (*open-box* or *white-box*) and *pedagogical* (*black-box*).

Decompositional approaches focus on extracting rules at the level of individual units within the trained ANN; i.e. the view of the underlying ANN is one of *transparency*. According to Andrews, Diederich and Tickle, a basic requirement for this category of rule extraction is that the computed output from each unit must be mapped into a binary outcome, corresponding to a rule consequent. Each unit can be interpreted as a step function, meaning that the problem is reduced to finding a set of incoming links whose summed weights guarantee that the unit's bias is exceeded regardless of other incoming links. When such a combination of links is found, this is readily translated into a rule where the output of that unit is a consequent of the inputs. The rules extracted at the individual unit level are then aggregated to form the composite rule set for the ANN as a whole.

Pedagogical approaches treat the trained ANN as a black box; i.e. the view of the underlying ANN is *opaque*. The core idea in the pedagogical approach is to treat the ANN as an oracle and view the rule extraction as a learning task, where the target concept is the function learnt by the ANN. Hence the rules extracted directly map inputs to outputs. Black-box techniques typically use some symbolic learning algorithm, where the ANN is used to generate the training examples. The easiest way to understand the process is to regard black-box rule extraction as an instance of predictive modeling, where each input-output pattern consists of the original input vector and the corresponding prediction from the opaque model. From this perspective, black-box rule extraction becomes the task of modeling the function from the (original) input variables to the opaque model predictions; see Figure 16.

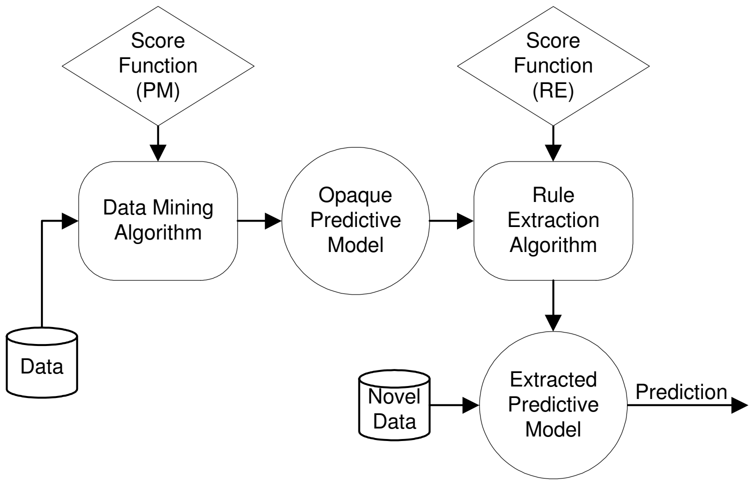


Figure 16: Black-box rule extraction.

The first two dimensions (i.e. expressive power and translucency) are in [ADT95] suggested to be the primary classifiers of rule extraction algorithms.

4.2.2 Evaluation of rule extraction algorithms

There are several criteria used for evaluating rule extraction algorithms. In [CS99] Craven and Shavlik listed five criteria:

- **Comprehensibility**: The extent to which extracted representations are humanly comprehensible.
- **Fidelity**: The extent to which extracted representations accurately model the networks from which they where extracted.
- **Accuracy**: The ability of extracted representations to make accurate predictions on previously unseen cases.
- **Scalability**: The ability of the method to scale to networks with large input spaces and large numbers of weighted connections.

- **Generality**: The extent to which the method requires special training regimes or places restrictions on network architectures.

Most researchers have evaluated their rule extraction methods using the first three criteria but, according to Craven and Shavlik, scalability and generality have often been overlooked. In the paper, scalability is defined in the following way:

> *Scalability refers to how the running time of a rule extraction algorithm and the comprehensibility of its extracted models vary as a function of such factors as network, feature-set and training-set size.* (p. 2)

Craven and Shavlik argue that methods that scale well in terms of running time, but not in terms of comprehensibility will be of little use. The reason is obvious from the fact that the overall purpose of rule extraction always is to produce a comprehensible model available for human interpretation. If this becomes impossible for larger problems it must be considered a serious limitation for a proposed method.

It should be noted that scaling is an inherent problem, regarding both running time and comprehensibility, for decompositional methods. The potential size of a rule for a unit with n inputs each having k possible values is k^n, meaning that a straightforward search for possible rules is normally impossible for larger networks. This, and the problem with continuous inputs, are normally to some extent handled by clustering inputs into disjoint ranges. Craven and Shavlik also highlight that the size of rule sets produced by decompositional algorithms tend to be proportional to the network size.

Craven and Shavlik recommend rule extraction researchers to pursue different lines of research that have not been explored to a large extent, to try to overcome the problem of scalability:

- Methods for controlling the comprehensibility vs. fidelity trade-off; i.e. the possibility to improve the comprehensibility of an extracted rule set by compromising on its fidelity and accuracy.
- Methods for anytime rule extraction; i.e. the ability to interrupt the rule extraction at any time and still get a solution.

Regarding generality, Craven and Shavlik argue that rule extraction algorithms must exhibit a high level of generality to have a large impact. In particular, algorithms requiring specific training regimes or algorithms limited to narrow architectural classes are deemed less interesting. Craven and Shavlik finally say that rule extraction algorithms ideally should be so general that the models they are trying to describe must not even be ANNs. Obviously there is also a need to explain complex models like ensembles or classifiers using boosting, so it is natural to extend the task of rule extraction to operate on these models. A rule extraction algorithm capable of coping with different underlying kinds of models would therefore be of high value.

Yet another important criterion, often overlooked but recognized in [TS93], is *consistency*. A rule extraction algorithm is consistent if it extracts similar rules every time it is applied to a specific data set. According to Towell and Shavlik, consistency is important since it would be very hard to give any significance to a specific rule set if the extracted rules vary significantly between runs.

Craven and Shavlik also pointed out another issue they believe to be a key to the success of rule extraction methods, namely *software availability*; i.e. researchers should make their methods available to potential users and fellow researchers to receive testing and evaluation.

An interesting discussion about the purpose of rule extraction is found in [Zho04], where Zhou argues that rule extraction really should be seen as two very different tasks; rule extraction *using* neural networks and rule extraction *for* neural networks. The first task prioritizes accuracy while the second focuses on fidelity. Rule extraction using neural networks thus is aimed at finding a comprehensible model with higher accuracy than a comprehensible model created directly from the data set using, for instance, a decision tree algorithm. Rule extraction for neural networks, on the other hand, is solely aimed at understanding the inner workings of a trained neural network. The claim made by Zhou is that it is never important to obtain both high fidelity and high accuracy; the goal is always one of them, and, consequently the other should be ignored.

4.3 Related work concerning rule extraction

Since one key contribution of this thesis is a novel method for rule extraction from opaque models, one very important related research area is previously presented rule extraction algorithms. In this section some important rule extraction algorithms are discussed.

Below, three specific algorithms for rule extraction are presented. The motivation for including RX [LSL95] and TREPAN [CS96] is that they are well-known, general techniques, representing very different approaches. The algorithm based on genetic programming [DRS+02] is included since it is similar to the G-REX method presented later in the thesis. TREPAN and RX have also been used for comparison with the G-REX algorithm. Results from this comparison are reported in Chapter 6.

4.3.1 RX

With the RX algorithm [LSL95], Lu, Setino and Liu present a decompositional rule extraction algorithm producing Boolean rules from feed-forward ANNs. The description below is a slight simplification of the presented method; for more details see the original paper. Lu, Setino and Liu use an approach consisting of three steps to create classification rules:

1. Network training: A three-layer ANN is trained using standard techniques; e.g. backpropagation. The coding used for the classes is localist; i.e. there is one output unit per class. To facilitate the following pruning, it is desirable to have many weights with values so small that they can be set to zero. This is accomplished by adding a penalty function to the error function; for details see the original paper.

2. Network pruning: The fully connected network of step 1 is pruned to produce a much smaller net without raising the classification error "too much". More specifically, links with small weights are iteratively removed and the smaller network is retrained until the accuracy on the training set falls below an acceptable level.

3. Rule Extraction: Rules are extracted from the pruned network. The rules generated are of the form *if* $(a_1\ \theta\ v_1)$ *and* $(a_2\ \theta\ v_2)$*… and* $(a_n\ \theta\ v_n)$ *then* C_j where a_i's are the attributes of an input instance, v_i's are constants, C_j is one of the class labels and θ is a relational operator.

The process for the actual rule extraction is given in pseudocode below:

```
Input:
D                     // Training data
N                     // Pruned neural network

Output:
R                     // Extracted rules

Algorithm (RX):
Cluster hidden nodes activation values;
Generate rules that describe the output values
  in terms of the discretized hidden activation values;
Generate rules that describe the discretized hidden output values
  in terms of input values;
Combine the two sets of rules;
```

It should be noted that continuous inputs must first be discretized, here by dividing their range into subintervals. Normally, continuous inputs are then coded using thermometer coding.

The RX algorithm relies heavily on the success of the pruning since, if a node has n input links, there could be as many as 2^n distinct input patterns. Another problem is the fact that the activation value of a hidden node could be anywhere in the range [-1, 1], (assuming a hyperbolic tangent activation function), depending on the input instance. For a large training set this makes the activation function almost continuous. The RX algorithm handles this by discretizing the activation values of hidden nodes into a "manageable" number of discrete values. A small set of discrete activation values makes it possible to determine the dependencies between hidden node and output node values, as well as the dependencies between input and hidden node values. From these dependencies, rules can be generated; in the RX algorithm this is done by using the *X2R rule generator* [Liu95].

Although the RX algorithm is often cited, and even sometimes used as an example of typical rule extraction; see e.g. [Dun03], it is not widely used in practice. The source code is not publicly available, which makes it hard to correctly evaluate the performance. Applying the criteria from section 4.2.2, it is obvious that RX most likely will fail regarding scalability. The case study reported in the original paper uses a fairly small data set (1000 instances with initially 7 attributes each) and the pruning is very successful; only 17 of 386 links remain after the pruning, while the accuracy is still over 90%. Although most criteria regarding rule quality (comprehensibility, accuracy and fidelity) seem to be well met, this is very hard to judge from just one problem.

Regarding generality, RX is tightly bound to feed-forward ANNs. The demands for repeated training during the pruning phase, a tailored error function and the many "tricks" to get inputs, outputs and activation values into suitable formats also make RX a very specialized algorithm. It should, in addition, be noted that RX extracts rules from *one* network only. If, as often is the case, the predictive model consists of an ensemble of networks, RX would have to extract from a less accurate model. This is a disadvantage compared to pedagogical methods, which would operate directly on the actual predictive model.

A rule extraction technique based on RX is CaST (Cluster and See Through) [LJK04]. The main idea of CaST is to apply a clustering technique similar to the one used by RX, but to the activation values of the *input* nodes; i.e. to the input values directly. This alteration in reality makes CaST a black-box rule extraction algorithm because the extracted rules now describe outputs in terms of inputs. Naturally, this makes it possible for CaST to extract rules from any opaque model, not just single feed-forward ANNs. In the study reported in [LJK04], CaST is evaluated against *NeuroRule* (RX) and C5.0 on three data sets. The main result is that CaST and NeuroRule have almost identical accuracy on two problems, but the rules found by CaST are significantly more compact. On the third problem (Soybean) NeuroRule fails to extract rules since the pruning is not effective enough. CaST on the other hand, produces a fairly complex rule set having higher accuracy than C5.0.

4.3.2 TREPAN

TREPAN [CS96] is a pedagogical rule extraction algorithm for classification problems producing decision trees. Each internal node in the extracted tree represents a splitting criterion and each leaf represents a predicted class. TREPAN uses *M-of-N* expressions for its splits. M-of-N expressions are Boolean expressions in disjunctive normal form, with *N* conjuncts and the requirement that at least *M* of these should be true for the entire expression to be true.

TREPAN is a black-box method since it focuses exclusively on the input-output relationship, instead of looking inside the neural net. In a way, TREPAN uses the network as an oracle; i.e. its results are regarded as ground truth. TREPAN grows trees in a best-first fashion, since the node with the greatest potential to increase the fidelity of the extracted tree is expanded first.

The TREPAN algorithm is given in pseudocode below:

```
Input:
D                      // Training data
N                      // Trained neural network

Output:
DT                     // Extracted decision tree

Algorithm (TREPAN):
Initialize the tree as a leaf node;
While stopping criteria not met
  Pick the most promising leaf node to expand
    Draw a sample of instances;
    Use the network to label the instances;
    Select splitting test for the node;
    For each possible outcome of the test make a new leaf node;
```

The task of TREPAN is, according to Craven and Shavlik, to induce the function represented by the trained ANN; i.e. fidelity is the basis of the score function.

Craven and Shavlik [CS99] describe TREPAN as similar to conventional decision tree algorithms such as C4.5 with some basic differences. Below is a summary of the main properties of TREPAN:

- TREPAN uses the ANN to label all instances. This also means that TREPAN can use the ANN to label new instances and thus learn from arbitrarily large samples.
- In order to decide which node to expand next, TREPAN uses an evaluation function to rank all of the leaves in the current tree. The evaluation function used for node *N* is: $f(N) = reach(N) \times (1 - fidelity(N))$ where *reach(N)* is the estimated fraction of instances that reach node N and *fidelity(N)* is the estimated fidelity of the tree to the network for those instances.
- TREPAN gets a sample of instances from two sources to find the logical test with which to partition the instances that reach the node and to determine the class labels for the leaves. First, it uses the ANN's training examples that reach the node. Second, TREPAN constructs a model (using the training examples) of the underlying distribution and uses this model to draw instances. These instances are randomly drawn but are subject to the constraint that they would reach the node being expanded if classified by the tree. In both cases TREPAN queries the ANN to get the class label.
- TREPAN uses information gain as the evaluation measure when selecting splitting tests.

TREPAN is publicly available, the authors report several case studies (see [Cra96]), and has also been extended in different ways; for instance to return fuzzy decision trees [FJK99]. TREPAN performs well, both in reported studies and in the experiments conducted in this thesis. The accuracy is normally higher than that of models generated directly from the data set; e.g. by C5.0. ANN fidelity is naturally high, since this is the purpose of the algorithm. Regarding comprehensibility it can be argued that decision trees automatically produce good explanation, but for more complex and bushy trees this is questionable. TREPAN handles this by extracting the tree in a best-first fashion and allows the user the option to stop the growth at any level; an example of anytime extraction.

Nevertheless there is still a trade-off between accuracy and comprehensibility. Actually for some of the extracted trees reported (see e.g. the tree on page 14 of

[CS97b]), as well as some found during the research for this thesis, the ease of human inspection is questionable. This is arguably partly due to the use of M-of-N rules, which for most people are tricky to read.

TREPAN was, according to Craven and Shavlik, designed with scalability and generality in mind. The scalability criterion naturally favors black-box approaches, since the computational complexity for black-box methods does not depend directly on the network architecture. The node expansion itself is of polynomial computational complexity in the sample size, the number of features and the maximum number of values for a discrete feature. Thus, TREPAN is likely to scale up well to larger problems, at least regarding computational complexity. Since scalability in comprehensibility requires that growth of the tree is stopped early, the accuracy of the extracted tree is obviously very dependent on that the evaluation function used constantly finds the best splits first. It should be noted that TREPAN in itself does not balance accuracy against comprehensibility, but leaves this decision to the user. The user must choose when to stop the growth of the tree, or put in another way, which tree of several, all with different complexity, to actually use on novel data.

Regarding generality, TREPAN does not really require the underlying model to be an ANN. There are no reports, though, of studies where the original TREPAN program is used for rule extraction from anything else than ANNs. In the third-party implementation later used for experimentation in this thesis, it was, however fairly easy to convert TREPAN into a true black-box rule extraction algorithm; see 6.5.4.

4.3.3 Rule extraction using evolutionary algorithms

Dorado et al. in [DRS+02] present a novel approach to rule extraction based on evolutionary algorithms. The approach has many similarities to the G-REX method for rule extraction proposed in this thesis. One should note that the two methods were developed independently[1].

The algorithm suggested by Dorado et al. is a black-box method where the extraction strategy is based on GP. The algorithm can handle different

[1] As a matter of fact the two methods were presented almost simultaneously at different conferences, (with G-REX just two weeks ahead) in September 2002.

representation languages with ease, since the choice of representation language corresponds to the choice of function and terminal sets. In the paper, Dorado et al. give three examples, two where Boolean rules are extracted for classification problems and one where a mathematical function, similar to, but more complex than a Box-Jenkins model, is derived for a time series forecasting problem. The suggested approach is also compared (with good results) to several existing techniques on well-known problems. A key result reported by Dorado et al. is the comparison between the rule extraction algorithm and GP applied directly on the data set. The accuracy of the rule extraction is slightly higher, supporting the claim that the neural net in a sense is a better (more general) representation of the data than the data set itself.

The purpose of the proposed method is to use GP for the search process. More specifically, candidate rules (which could be Boolean rules, decision trees, regression trees etc.) are continuously evaluated according to how well they resemble the ANN. The best rules are kept and combined using genetic operators to raise the fitness (performance) over time. After many iterations (generations) the most fit program (rule) is chosen as the extracted rule.

It should be noted that the fitness function is crucial for determining what to optimize, and that the choice here is to solely optimize *the fidelity of the extracted representation to the neural net*.

Dorado et al. do not state the algorithm explicitly, so the description below is based on an interpretation of their paper.

```
Input:
D                      // Training data
N                      // Trained neural network
F                      // Function set
T                      // Terminal set

Output:
R                      // Extracted representation

Algorithm (Rule extraction using EA):
Initialize a first generation of candidate rules;
While number of generations not reached
  Evaluate all candidate rules using the fitness function (fidelity);
  Choose a number of candidate rules for reproduction;
  Combine chosen candidate rules using genetic
   operators (crossover) to create offspring rules;
  Replace old candidate rules with offspring rules;
```

The reported accuracy for the approach reported by Dorado et al. is high. Since GP is a recognized tool for optimization problems and fidelity to the neural net is the optimization criteria here, it is no surprise that the fidelity is high. The problem for the algorithm of Dorado et al. is clearly comprehensibility. None of the rules reported in the paper (see pages 489, 490 and 492) would enable human inspection and understanding.

Regarding time scalability, the proposed method is likely to perform well. Although GP is rather computationally intensive, this applies even more for the original training of the neural net, making it unlikely that the rule extraction would be the bottle-neck. The computational complexity of a GP approach is dependent on parameters like the number of programs in each generation, and to a lesser degree, the size of each program. This could potentially be a difficulty for very complex problems, where large generations and/or programs are needed. This is obviously an issue that should be looked in to.

How well comprehensibility scales up is a totally different matter. Since Dorado et al. do not try to enforce short rules, complex data sets with many variables will inevitably produce long and complicated rules.

The generality of the proposed approach is very high and is actually the most appealing property of the method. Dorado et al. apply the rule extraction on both feed-forward and recurrent networks and extract rules for both classification and regression. It is also obvious that, even if the authors do not explicitly point this out, the algorithm does not require the underlying model to be a neural net.

Chapter 5

Ensembles

This chapter presents some basic techniques for aggregating multiple models into a composite model; an *ensemble*[1]. *Ensemble learning* refers to a large collection of methods that learn a target function by training a number of individual learners and combining their predictions. Figure 17 below shows how *N* models, built using *N* data sets drawn from the available data, are combined into an ensemble. The ensemble prediction, when applied to a novel instance, is a function of all included base models. It should be noted that the different data sets in practice could be (and often are) overlapping or even identical.

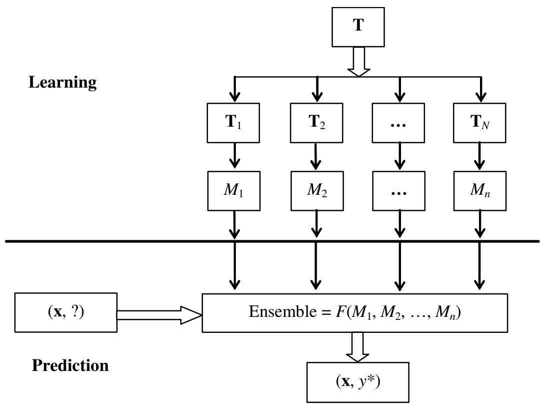


Figure 17: Schematic ensemble.

[1] Ensembles are also referred to as, for instance, *committees*, *committee machines* or *combined models*.

5.1 Motivation for ensembles

As expected, the overall purpose of using ensembles is to obtain higher accuracy, and there are at least two, intuitive, explanations to why ensemble learning should produce more accurate models.

First of all: if several models are combined by averaging, uncorrelated errors of individual classifiers can be eliminated. As an example, adapted from [Die97], if 40 base classifiers are combined using *majority voting*, and each base classifier has an error rate of 0.3, the probability that an instance will be misclassified by the ensemble is as low as 0.002. More generally; the probability that at least r classifiers from N misclassify a specific instance, given an identical error rate p for each base classifier, can easily be found using the binomial distribution; see (50) below.

$$P(error) = \sum_{r=\frac{N}{2}+1}^{N} \binom{N}{r} p^r (1-p)^{N-r} \tag{50}$$

As a matter of fact, for classifiers that are independent in predictions and have accuracy better than chance, it is possible to push accuracy arbitrarily high by adding more classifiers to the ensemble; see [HS90]. The key assumption is, however, that the classifiers are independent in their predictions. This is clearly not a very reasonable assumption for a huge majority of all predictive modeling problems. Still, there are a large number of case studies where ensembles have been very successful, despite the fact that the base models used were correlated.

Second, there is no such thing as a universal "best" learning technique; a fact that is often referred to as the "no free lunch theorem"; see [Wol95]. On the other hand, different algorithms have different inductive biases, making them more or less suitable for specific problems. In addition, most learners are not general function approximators; i.e. they are unable to represent some target functions. Often, however, such target functions could be approximated by averaging several models. One specific example is decision trees where the decision boundaries are hyperplanes parallel to the coordinate axes. By averaging several decision tree models it becomes possible to approximate, for instance, a diagonal decision boundary to an arbitrary precision; see Figure 18 below.

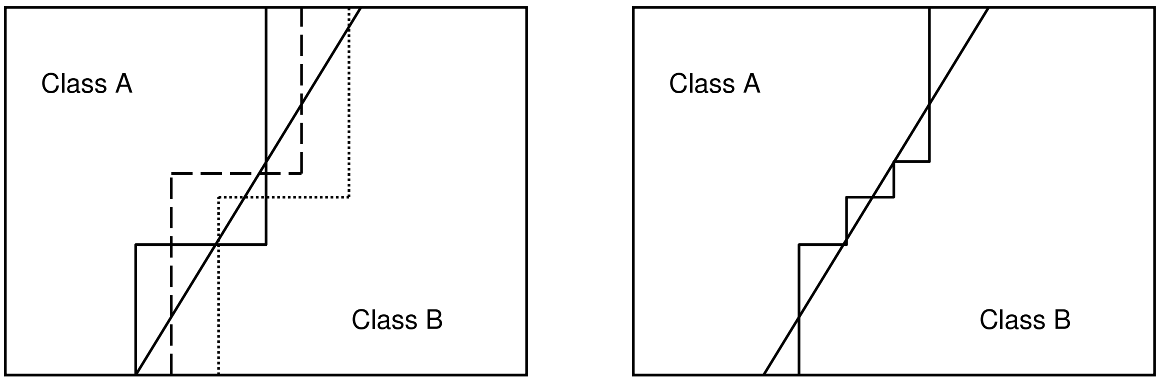


Figure 18: Averaging decision trees to obtain a diagonal decision boundary.

5.2 Ensemble construction

There are two major design choices when creating ensembles; how to generate the base models and how to integrate them. The two main alternatives regarding integration are *combination* and *selection*. As the names imply; combination combines predictions from all base models, while selection uses only a subset of the trained models for prediction. In addition, both combination and selection can be either *static* or *dynamic*; see Table 5 below. The main motivation for a dynamic approach is the assumption that each model is the best in some sub-areas of the whole data set, where its local error is comparatively less than the corresponding errors of the other models.

	Static	**Dynamic**
Combination	All models are combined. If weighting is used it is not based on the current test instance.	All models are combined. If weighting is used it is based on the current test instance.
Selection	A subset of models is selected. The selection is not based on the current test instance.	A subset of models is selected. The selection is based on the current test instance.

Table 5: Integration strategies.

There are many ways to combine individual models, but the most natural for regression and classification are *averaging* and *voting*, respectively. When using averaging, the prediction from the ensemble is, of course, the average of the predictions from the individual models. When an ensemble classifies using (majority) voting, the class predicted by most individual models constitutes the ensemble prediction. Naturally, averaging could also be used for classification if the base models produce numeric values; e.g. probabilities or belief values.

5.2.1 Generation strategies

Ensembles have been created using virtually all machine learning algorithms; e.g. decision trees (ID3, C4.5), instance-based learning (kNN), Bayesian classification (Naïve Bayes) and neural networks (MLPs).

Tan, Steinbach and Kumar [TSK06], describe four main generation strategies:

- **Manipulating the training set.** Multiple models are generated by training on different data sets obtained by resampling a common training set. Two specific techniques using this approach are Bagging (see 5.2.2) and Boosting (see 5.2.3).
- **Manipulating the input features.** Multiple models are generated by training on different representations or different subsets of a common input vector. One specific, rather recent technique using this approach is *Random Forests*; see [Bre01]. The basic idea is to create an accurate ensemble of decision trees by introducing randomness in both instance selection (similar to bagging), and in feature selection. More specifically; all splits are restricted to a random subset of inputs available. Despite being very simple and fast, the Random Forest technique is robust to noise and often obtains accuracy comparable to Boosting.
- **Manipulating the output variables.** A multiclass problem is first decomposed into a number of binary classification problems. After that, a classifier is trained to solve each binary problem and, finally, the ensemble consists of the resulting models. More specifically; the class labels are randomly partitioned into two disjoint subsets and one classifier learns the binary problem of separating these two subsets of classes. This procedure is repeated several times to obtain a number of base classifiers. When predicting a novel instance, each base classifier prediction is in fact a vote for several classes, which is handled as a vote for each class belonging to that specific subset of classes. The ensemble prediction is, of course, the class receiving most votes altogether. A technique using a variation of this generation strategy is *error-correcting output coding* (ECOC); see [DB95]. ECOC is based on a procedure originally used for sending messages across noisy channels. The main idea is to add redundancy to the message in order to make it possible to automatically correct minor errors in the message. ECOC represents each class with a bitstring of length *n*, which is called the *codeword* for that class. During learning, one classifier is trained to predict each bit in the

bitstring, resulting in n binary classifiers. The predicted class of a test instance is given by the codeword closest (using Hamming distances) to the codeword produced by the binary classifiers. Naturally, the design of appropriate codewords is very important, and the guiding principle is that the codewords must have both their row-wise and column-wise distances well separated. A large row-wise separation is needed for the error-correcting ability, while the larger column-wise distance ensures that the binary classifiers are mutually independent.

- **Modifying parameters of the learning algorithm:** A number of classifiers are built with different learning parameters; i.e. number of neighbors in kNN, different initial weights or different topologies in an ANN, etc.

Another, relatively uncommon, approach is to use ensembles consisting of heterogeneous models. One specific dynamic combination technique, typically using heterogeneous models, is *stacking* [Wol92]. When using stacking, the combiner is another learner whose parameters also are trained. In more detail; multiple classifiers are first generated by using different learning algorithms on a single data set. Then, a meta-level classifier is learned to combine the outputs of the base classifiers. To generate the training set for the meta-level classifier, a leave-one-out or a cross validation procedure is applied. The meta-level data set thus consists of instances where the input vector is the predictions of the base classifiers and the target is the correct class of the example at hand. So, stacking is a way of estimating and correcting for the biases of the base classifiers; i.e. the purpose is to teach the combiner how the base classifiers make their errors.

Dzeroski and Zenko in [DZ04] empirically evaluate several state-of-the-art methods for constructing ensembles of heterogeneous classifiers with stacking. The main result is that they perform (at best) comparably to selecting the best classifier from the ensemble by cross validation. In the paper, Dzeroski and Zenko also propose two, slightly technical, extensions of stacking. In the experimentation, these extensions perform better than both previously existing stacking approaches and the straightforward choice of selecting the best classifier by cross validation; for details see the original paper.

5.2.2 Bagging

Bagging, [Bre96], rely on resampling to obtain different training sets for each base model. Every training set (called *bootstrap*) has the same size as the original training set, and is created by repeated sampling (according to a uniform distribution) of training instances. Since sampling is done with replacement, some instances may appear several times, while others are omitted. On average, a bootstrap sample contains approximately 63% of the original training instances. After training one individual model on each bootstrap sample, a test instance is classified using averaging or voting. Bagging is more common for classification, but can also be used for regression. When performing regression, the median is often used instead of the average to increase robustness. When using Bagging, the overall purpose is to reduce the variance part of the bias-variance decomposition; see (6). In practice, Bagging will, at least, hardly ever increase the error. The Bagging algorithm is summarized below.

```
Input:
D                     // Training data having N instances

Output:
P                     // Prediction from ensemble

Algorithm (Bagging):
for each classifier
  Generate bootstrap by sampling N instances, with replacement, from D
  Train a model using the bootstrap

Ensemble prediction is average or majority vote from all models trained
```

5.2.3 Boosting

In Bagging, the base models are generated, at least logically, independently and in parallel. Boosting, introduced by Shapire [Sha90], on the other hand, is a sequential procedure mainly applied to classification, where the performance of a preceding model is used when generating all subsequent models. The main principle is that difficult training instances are assigned a higher weight, making the base models focus on these instances. More specifically; each training instance is initially assigned the same weight but after training one model, the instances incorrectly classified have their weights increased, while those correctly classified have their weights decreased. Naturally, the idea is to focus the base classifiers on portions of data space not previously well predicted. The weights are either used as part of the score function or to prioritize instances with higher weights when bootstrapping. Most often, the ensemble prediction is a weighted majority vote, where each base classifier is weighted according to obtained accuracy on a validation set. Below is a generic description of Boosting.

```
Input:
D                      // Training data having N instances
W                      // A weight vector of size N
M                      // Number of times to boost

Output:
P                      // Prediction from ensemble

Algorithm (Boosting):
set all weights to 1/N
do M times
  train a model on D
  find all instances in D that the model predicts wrong
  increase the weights of those instances
  decrease the weights of the others

Ensemble prediction is weighted vote from all models trained
```

There are several different boosting schemes, typically differing in how weights are updated and how predictions from the base classifiers are combined. The most well-known boosting algorithm is probably *AdaBoost*; for details see [FS96].

In theory, Boosting is aimed at reducing both the variance and bias terms in the decomposed error function (6). In practice, however, boosting is susceptible to noise and may sometimes overfit it.

5.3 Diversity

Krogh and Vedelsby in [KV95] derived an equation, stating that the generalization ability of an ensemble is determined by the average generalization ability and the average *diversity* (*ambiguity*) of the individual models in the ensemble. More specifically; the ensemble error, E, can be found from

$$E = \bar{E} - \bar{A} \tag{51}$$

where $\bar{E}$ is the average error of the base models and $\bar{A}$ is the ensemble diversity, measured as the weighted average of the squared differences in the predictions of the base models and the ensemble. In a regression context and using averaging, this is equivalent to:

$$E = (\hat{Y}_{ens} - Y)^2 = \frac{1}{M}\sum_i (\hat{Y}_i - Y)^2 - \frac{1}{M}\sum_i (\hat{Y}_i - \hat{Y}_{ens})^2 \tag{52}$$

From (52) it is obvious that the error of the ensemble is guaranteed to be less than or equal to the average error of the base models. The first term is the (possibly weighted) average of the individual classifiers and the second is the diversity term; i.e. the amount of variability among ensemble members. Since the second term (which is always positive) is subtracted from the first to obtain ensemble error, this decomposition proves that the ensemble will always have higher accuracy than the average accuracy obtained by the individual classifiers. If there is no diversity, the ensemble will have the mean accuracy of the individual base models. Naturally, this is a very encouraging result for ensemble approaches. But, the overall error depends on both the average error of ensemble members and the average diversity, so increasing diversity will decrease the ensemble error, as long as it does not result in an increase in average error. The problem is, however, that the two terms are highly correlated, making it necessary to balance them rather than to just maximize diversity.

By relating this to the bias-variance decomposition in (6), and assuming that the ensemble is a convex combined ensemble (e.g. using averaging), a bias-variance-covariance decomposition can be obtained for the ensemble MSE; see (53) below.

$$E = (\hat{Y}_{ens} - Y)^2 = \overline{bias}^2 + \frac{1}{M}\overline{\mathrm{var}} + \left(1 - \frac{1}{M}\right)\overline{\mathrm{cov}} \qquad (53)$$

From this it is evident that the error of the ensemble depends critically on the amount of correlation between models, quantified in the covariance[1] term. Ideally, the covariance should be minimized, without causing changes in the bias or variance term.

When discussing ensembles in a classification context, it should be noted that unless classification is handled like an instance of regression (i.e. the outputs are ordinal and typically represent probabilities), the framework described above does not apply. When predictors are only able to output a class label, the outputs have no intrinsic ordinality between them, thus making the concept of covariance undefined. So, with a zero-one loss function, there is no clear analogy to the bias-variance-covariance decomposition. Consequently, the overall goal of obtaining an expression where the classification error is decomposed into error rates of the individual classifiers and a diversity term is currently beyond the state of the art.

Since diversity is not uniquely defined for classification problems, and the relationship between diversity and ensemble accuracy is far from completely understood, there are many different heuristic measures used to quantify

[1] Covariance can, of course, be negative, while bias and variance are always positive.

diversity. Naturally, the goal is to find a diversity measure correlating well with majority vote accuracy, with the underlying assumption that such measure could be utilized in order to design (or search for) an accurate ensemble. Unfortunately, despite extensive experimental work, no such measure has been conclusively established; see e.g. [BWH+05].

In [KW03], Kuncheva and Whitaker studied ten statistics measuring diversity among binary classifier outputs; i.e. correct or incorrect vote for the class label. The main result was that all diversity measures evaluated showed low or very low correlation with test set accuracy, so the study clearly raised some doubts about the usefulness of diversity measures when building classifier ensembles. The experiments conducted, however, used rather artificial settings. More specifically, most experiments did not use "real" data sets or "real" classifiers. The measures used in the study are presented in 5.3.1 below, and they are further evaluated in the experimentation chapter; see 8.5.

5.3.1 Diversity measures

The presentation of the different diversity measures follows [KW03] closely. The output of each classifier D_i is represented as an N-dimensional binary vector $\mathbf{y}_i$, where $y_{j,i}$=1 if D_i correctly recognizes instance z_j and 0 otherwise. When describing the four pairwise measures the notation N^{ab} meaning the number of instances for which $y_{j,i}=a$ and $y_{j,k}=b$, is used. As an example, N^{11} is the number of instances correctly classified by both classifiers.

The first measure, Yule's Q *statistic* [You00] is, for two classifiers, D_i and D_k

$$Q_{i,k} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}} \tag{54}$$

Q varies between -1 and 1. If the classifiers commit their errors on different instances, Q will be negative. For an ensemble consisting of L classifiers, the averaged Q over all pairs of classifiers is

$$Q_{av} = \frac{2}{L(L-1)} \sum_{i=1}^{L-1} \sum_{k=i+1}^{L} Q_{i,k} \tag{55}$$

The *correlation* between y_i and y_k is

$$\rho_{i,k} = \frac{N^{11}N^{00} - N^{01}N^{10}}{\sqrt{(N^{11}+N^{10})(N^{01}+N^{00})(N^{11}+N^{01})(N^{10}+N^{00})}} \tag{56}$$

For any two classifiers, Q and ρ have the same sign. The *disagreement* measure is the ratio between the number of instances on which one classifier is correct and the other incorrect to the total number of instances:

$$Dis_{i,k} = \frac{N^{01} + N^{10}}{N^{11} + N^{10} + N^{01} + N^{00}} \tag{57}$$

The *double-fault* measure is the proportion of instances misclassified by both classifiers

$$DF_{i,k} = \frac{N^{00}}{N^{11} + N^{10} + N^{01} + N^{00}} \tag{58}$$

For all pairwise measures, the averaged value over the diversity matrix is calculated similarly to (55).

Turning to non-pairwise measures, $l(z_j)$ denotes the number of classifiers that correctly recognize z_j. The *entropy* measure E, varying between 0 and 1 (highest possible diversity), is

$$E = \frac{1}{N} \sum_{j=1}^{N} \frac{1}{(L - \lceil L/2 \rceil)} \min\{l(z_j), L - l(z_j)\} \tag{59}$$

The *Kohavi-Wolpert* variance [KW96] can be used to obtain another diversity measure KW, which turns out to differ from the averaged disagreement measure only by a coefficient; for details see [KW03].

$$KW = \frac{1}{NL^2} \sum_{j=1}^{N} l(z_j)(L - l(z_j)) \tag{60}$$

Yet another measure, called *interrater agreement* (κ) and used in [Die00] is

$$\kappa = 1 - \frac{\sum_{j=1}^{N} l(z_j)(L - l(z_j))}{LN(L-1)\overline{p}(1-\overline{p})} \tag{61}$$

where $\overline{p}$ is the averaged individual accuracy.

The *difficulty* measure was used in [HS90] by Hanson and Salomon. Let X be a random variable taking values in $\{0/L, 1/L, \ldots, 1\}$. X is defined as the proportion of classifiers that correctly classify an instance $\boldsymbol{x}$ drawn randomly from the data set. To estimate X, all L classifiers are run on the data set. The difficulty θ is then defined as the variance of X.

The *generalized diversity* measure was proposed by Partridge and Krzanowski in [PK97].

$$GD = 1 - \frac{p(2)}{p(1)} \tag{62}$$

where

$$p(1) = \sum_{i=1}^{L} \frac{i}{L} p_i \; and \; p(2) = \sum_{i=1}^{L} \frac{i}{L} \frac{(i-1)}{L(L-1)} p_i \tag{63}$$

If Y is a random variable expressing the proportion of classifiers that are incorrect on a randomly drawn instance then p_i is the probability that $Y=i/L$ and $p(i)$ is the probability that i randomly chosen classifiers will fail on a randomly chosen instance. GD varies between 0 (minimum diversity) and 1.

The last measure, *coincident failure diversity,* is a modification of GD also presented in [PK97].

$$CFD = \left\{ \begin{array}{ll} 0 & , p_0 = 1.0 \\ \dfrac{1}{1-p_0} \displaystyle\sum_{i=1}^{L} \frac{L-i}{L-1} p_i, & p_0 < 1.0 \end{array} \right\} \tag{64}$$

5.3.2 Taxonomy of methods for diversity creation

In the paper [BWH+05], Brown et al. introduce a taxonomy of methods for creating diversity. The first obvious distinction made is between *explicit* methods, where some metric of diversity is directly optimized and *implicit* methods, where the method is likely to produce diversity without actually targeting it. After that the authors move on to group methods using three categories; *starting point in hypothesis space*, *set of accessible hypotheses* and *traversal of hypothesis space*, which are further described below.

Starting point in hypothesis space

This group, as described by Brown et al., is mostly relevant for ANNs. The most obvious method is to simply randomize the starting weights; something that must be considered a standard procedure for all ANN training. Alternatively, the weights could be placed in different parts of the hypothesis space. Unfortunately, experimentation has found (see e.g. [SNS95]) that ANNs often converge to the same or very similar optima in spite of starting in different parts of the space. Thus, according to Brown et al., varying the initial weights of ANNs does not seem to be an effective stand-alone method for generating diversity.

There are very few explicit methods using different starting points in hypothesis space, but Brown et al. mention two techniques. In the first, called *Fast Committee Learning* [SA98], a single ANN is trained, but *M* snapshots of the weights are taken during training. The ensemble then combines the *M* different ANN snapshots. The reported performance was rather poor, though, so the only advantage is the reduced training time compared to training *M* individual ANNs. The second method discussed by Brown et al. is an approach presented by Maclin and Shavlik [MS95] where competitive learning is used to create ANNs with weights initialized far from the origin. The results reported there were significantly better, compared to standard randomized weight initialization, but they were based on only two data sets.

Set of accessible hypotheses

The two main principles are to either manipulate training data or the architecture. Several methods attempt to produce diversity by supplying each classifier with a slightly different training set. Regarding resampling, the view is that it is more effective to divide training data by feature than by pattern; see [DT00]. All standard resampling techniques are by nature implicit. The standard technique AdaBoost, on the other hand, explicitly alters the distribution of training data fed to each classifier based on how "hard" each pattern is.

Manipulation of architectures, for ANNs, is most often about number of hidden units, which also turns out to be quite unsuccessful. More interesting is that hybrid ensembles where, for instance, MLPs and RBF networks are combined, seem to be a "productive route"; see [PY96]. Regarding hybrid ensembles, Brown et al. [BWH+05] argue that two systems which represent the problem and search the space in very different ways probably will specialize on different parts of the search space. This implies that when using hybrid ensembles, it would most likely be better to select one specific classifier instead of combining outputs.

Traversal of hypothesis space

A common solution is to use a penalty term enforcing diversity in the error function when training ANNs. A specific, and very interesting example, is Negative Correlation learning (NC) [Liu98], where the covariance between networks is explicitly minimized. For regression problems it has been shown that NC directly controls the covariance term in the bias-variance-covariance trade-off; see [Bro04]. For classification problems, however, this still does not apply. Finally, Brown et al. discuss evolutionary methods where diversity is a part of a fitness function and some evolutionary algorithm is used to search for an accurate ensemble. One example is the ADDEMUP method presented in [OM96].

5.4 Related work concerning ensemble creation

In Chapter 8, a novel algorithm for ensemble creation is presented. Since the novel algorithm is based on GP, this section describes some existing approaches using evolutionary algorithms when creating ensembles.

Any algorithm aimed at building ensembles must somehow both train individual models and combine these into the actual ensemble. The standard techniques Bagging and Boosting, as described above, rely on resampling techniques to obtain different training sets for each of the classifiers. Both Bagging and Boosting can be applied to ANNs, although they are more common when using decision trees; see e.g. [OM99]. Another option is to train a number of classifiers independently (most often using common data) and then either combine all classifiers or select a subset to form the actual ensemble.

Several approaches create ensembles by somehow using genetic algorithms to search for optimal ensembles. Zhou et al. in [ZWJ+01] and [ZWT02] proposed an approach named GASEN, where several ANNs are trained before GAs are used to select an optimal subset of individual networks. In GASEN, the optimization is performed on individual ANNs, and each ANN is coded (in the chromosome) as a real number indicating the goodness of including that ANN. The optimization criterion (the fitness) boils down to accuracy on a holdout (validation) set. The number of ANNs in the ensemble can vary, since all ANNs with strength values higher than a specific threshold (which is a pre-set parameter) is included in the ensemble.

Opitz and Shavlik [OM96] suggested a method called ADDEMUP where GAs are used to create new ANNs (on the unit, link and weight level) as parts of an ensemble. The size of the ensemble is predetermined and fixed. ADDEMUP uses a fitness function directly balancing accuracy against diversity, measured on a validation set.

Langdon and Buxton, in a series of papers, suggest evolving ensembles using GP; see e.g. [LB01a], [LB01b] and [LB01c]. In the paper [LBB03], Langdon, Barett and Buxton create ANN ensembles using GP and evaluate them on a drug discovery problem.

In the study, the 75 trained ANNs are presented to the GP as 75 problem specific functions. Each function returns the classification and an associated confidence of the corresponding ANN for the current instance, coded as a single float number. Values below 0.5 are treated as class 0, while those above 0.5 are regarded as predicting class 1. The magnitude of the difference from 0.5 indicates the ANN's confidence. Normally, the output of a neural network is converted into a binary classification by testing to see if the value is greater or less than 0.5. Here, however, the fixed value 0.5 is replaced by a tunable threshold *[0 . . . 1]*. By making the threshold the argument of the function, the choice of suitable operating point is left to the GP. These arguments are treated like any other by the GP, and so can be any valid arithmetic operation, including the base classifiers themselves. GP thus simultaneously adapts the non-linear combination of the base classifiers and their operating points.

The terminals or leaves of the trees being evolved by the GP are either constants or the adjustable threshold. The function set contains the ANNs and the four arithmetic operators, but also functions like *MIN*, *MAX*, *INT* and *FRAQ*. There is also a simple *if-then-else* function. Each individual in the initial population contains five random trees. Each tree within an individual returns a signed real number. The classification of the individual is the sum of the answers given by the five trees. Note that the GP can combine the supplied classifiers in an almost totally arbitrary, non-linear way.

Fitness of each individual is calculated on the training set. The adjustable threshold *T* is set to values 0.1 apart, starting at 0 and increasing to 1. For each setting, the evolved program is used to predict each instance in the training set and true positive (TP) and false positive (FP) rates are calculated. Each TP, FP pair gives a point on an ROC curve. The fitness of the classifier is the area under the convex hull of these (plus the fixed points 0, 0 and 1, 1). Using this fitness function, the GP is not only rewarded for getting answers right, but also (using the threshold parameter) for getting a range of high scores.

Chapter 6

A novel technique for rule extraction

In this chapter a novel technique for rule extraction from opaque models is presented and evaluated. The technique, named G-REX (Genetic Rule EXtraction), is a black-box method where the extraction strategy is based on GP. G-REX was initially presented in [JKN03], but has after that been both extended in several ways and thoroughly evaluated; see e.g. [JSK+03], [JKN04], [JNK04] and [MBG+06]. G-REX is also discussed in a survey of rule extraction algorithms; see [HBV06].

One key property of G-REX is the fact that GP can use a variety of function and terminal sets. When rule extracting, this flexibility makes it possible to choose a suitable representation language for the problem at hand. Another, equally important, feature is the option to directly balance accuracy against comprehensibility by using an appropriate fitness function. The most important difference between G-REX and the method proposed by Dorado et al. in [DRS+02], is the fact that G-REX tries to find accurate and comprehensible rules, while Dorado et al. focus solely on the fidelity towards the ANN. Obviously the approach taken by Dorado et al. will lead to long and complex rules with questionable value for decision makers.

The outline of this chapter is as follows:

- Section 6.1 presents the core G-REX technique and the G-REX implementation.
- Section 6.2 describes how different representation languages can be used by G-REX. This section also includes two introductory experiments,

where G-REX is used on well-know classification problems to extract rules in two different formats.

- Section 6.3 relates G-REX to the criteria presented in 4.2.2. In particular, it is discussed how each criterion could be evaluated through experimentation.
- Section 6.4 introduces and discusses the novel approach of using *oracle data* (i.e. original *test data* inputs together with *test data* predictions from the opaque model) when performing rule extraction.
- Section 6.5 evaluates G-REX against the criteria using five experiments.

6.1 The G-REX technique

An algorithmic description of G-REX is given in pseudocode below. Note that the fitness function is considered an input to G-REX, since it can be varied to enforce different requirements on the extracted representation.

```
Input:
D                      // Training data
O                      // Trained opaque model
FF                     // Fitness function
F                      // Function set
T                      // Terminal set
$p_c$                     // Crossover rate
$p_m$                     // Mutation rate
N                      // Population size
G                      // Number of generations

Output:
R                      // Extracted representation

Algorithm (G-REX):
Initialize a first generation of N candidate rules P;
While number of generations not reached
  Evaluate all candidate rules using the fitness function FF;
  Create a new, empty, population $P_{new}$;
  While number of rules in $P_{new}$ is less than N
    Select a pair of rules for reproduction;
    With the probability $p_c$ use crossover to create two
      offspring rules else perform cloning;
    With a probability $p_m$ perform mutation on offspring rule;
    Place the resulting rules in $P_{new}$;
  Replace P with $P_{new}$;
Set R to the most fit rule over all generations;
Return R;
```

Most of G-REX parameters can easily be varied. For a description of default selections and other options see Table 6 below. It should be noted, however, that G-REX is still under development, so future development might reveal improved default settings.

Parameter	Default	Other options
Population size	2500	Typical: 1000-10000
Generations	100	Typical: 50-1000
Crossover rate	0.8	Typical: 0.5-0.8
Mutation rate	0.01	Typical: 0.001-0.05
Creation depth	8	Typical: 4-10
Selection	Roulette wheel	Tournament
Creation method	Ramped half-and-half	Full, Grow
Elitism	Yes	No
Persistence	Not used	Typical: 10-50

Table 6: GP parameters for G-REX.

Crossover and mutation are performed in a normal fashion; i.e. as described in 3.5.3. Parameters like crossover and mutation rates, number of individuals in a population and number of generations would typically be found from initial experimentation on the specific data set at hand. The default values have, however, proven quite robust over a large number of data sets and experiments, so they represent a very good first setting.

When using G-REX on a specific problem, fitness function, function set and terminal set must be chosen. The function and terminal sets determine the representation language, while the fitness function measures the quality of the extracted representation. Typically, the fitness function would at least include components measuring fidelity and comprehensibility.

6.1.1 Some details regarding the G-REX implementation

G-REX was developed as a stand-alone Java application, with a graphical user interface (GUI). To be able to perform fully automated experimentation, we have also made it possible to interact with G-REX from MatLab[1].

In addition to the standard GP parameters described in Table 6 above, some settings directly related to the rule extraction are also needed. First of all, G-REX requires the data set to have a specific format; more specifically a tab separated text file, where the last column represents the target value. The user has the option to divide the data set in three parts; training set, validation set and test set. This division is based either on a marker column in the data set text file or the *percentages* input in the GUI. If percentages are used, the appropriate number of patterns is always taken from the *beginning* of the data file, using the order

[1] www.mathworks.com

training, validation and test. It should be noted that training and validation sets are used identically during evolution; i.e. the fitness is calculated based on accuracy on both these sets. It is, however, possible to specify different costs for misclassified training and validation instances. The purpose of the test set is solely to evaluate the final extracted model; i.e. it is not used at all during evolution. The intended use of the different sets is thus as follows:

- The training set should include the instances originally used for generating the opaque model.
- The validation set should typically include instances used either as an early stopping validation set or as a holdout set when building the original, opaque model. If a G-REX validation set was used as a holdout set during the construction of the original model, it could be regarded, to some extent, to contain unseen data. If so, it might be a viable option to prioritize accuracy on this holdout set, trying to improve the generalization capability.
- The test set must, of course, consist of instances not used at all for the construction of the opaque model.

Another very important parameter accessible in the GUI is the length penalty used in the fitness function. Several slightly different fitness functions have been tried, but the main property is the ability to balance accuracy against comprehensibility by using a penalty for longer rules. The fitness function, currently used for classification problems, is very simple; see (65) below.

$$fitness = \alpha \cdot CI_T + \beta \cdot CI_V - \gamma \cdot S \qquad (65)$$

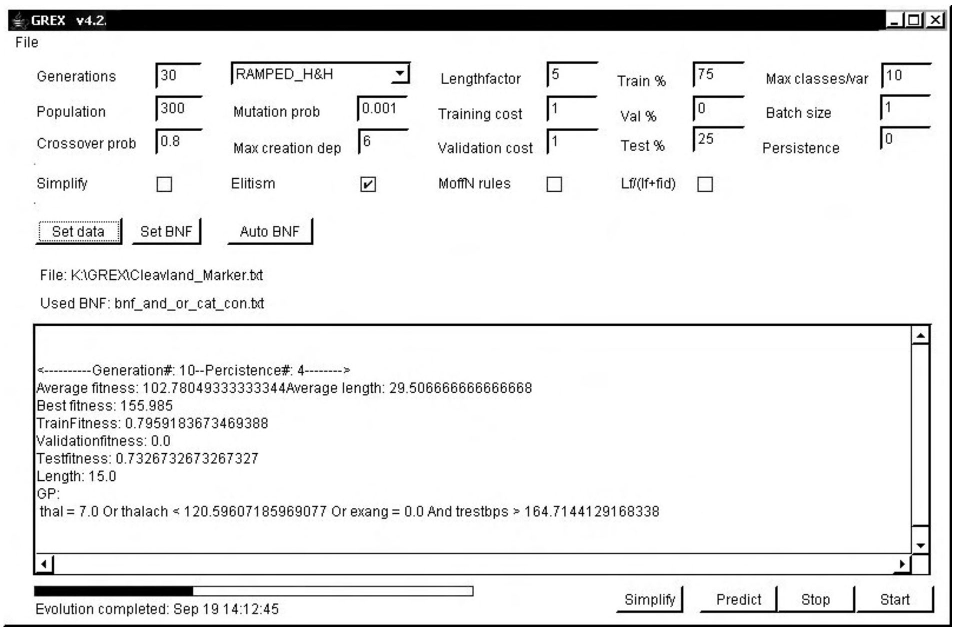
CI_T is the number of training instances classified correctly, CI_V is the number of validation instances correctly classified and S is the size of the rule[1]. α, β, and γ are coefficients determining the relative importance of the three components. Typical values, if a validation set is used, are $\alpha=1$, $\beta=2$ and $\gamma=0.05$. It must be noted that, in the rule extraction context, the term *correctly classified*, for training

[1] It should be noted that the size component in the fitness function is based on the internal S-expression representation used by G-REX. Because of this, the exact definition of size is dependent on the representation language used and, normally not equal to the number of nodes in the extracted tree. When comparing sizes between trees obtained by different techniques, however, measurements like *total number of nodes* or *number of interior nodes*, in the trees are used.

data in fact means classified identically to the opaque model. For validation data, the target value is normally also the output of the opaque model, but the user has the option to use true target values instead.

G-REX also has the ability to use different representation languages; i.e. to extract rules in very different formats. To increase flexibility, the user has the option to choose representation language by loading an appropriate text file containing the relevant BNF grammar[1]. At the moment, G-REX has access to a number of common grammars, but it is also possible for the user to use a novel representation language, just by supplying another BNF file. The only restriction is that all functions used must already exist in G-REX. The issue of representation languages is covered in more detail in 6.2.

Another setting is *batch*, with the effect that G-REX will perform several runs on the data set and return the single extracted representation having the highest fitness from any of these runs. Finally there is an option, called *simplify*, which is used to simplify the extracted rule. The algorithm used for this simplification is described in 6.1.2 below. Figure 19 below shows the G-REX GUI.



Generations	30	RAMPED_H&H		Lengthfactor	5	Train %	75	Max classes/var	10
Population	300	Mutation prob	0.001	Training cost	1	Val %	0	Batch size	1
Crossover prob	0.8	Max creation dep	6	Validation cost	1	Test %	25	Persistence	0
Simplify	☐	Elitism	✓	MoffN rules	☐	Lf/(lf+fid)	☐		

File: K:\GREX\Cleavland_Marker.txt

Used BNF: bnf_and_or_cat_con.txt

```
<----------Generation#: 10--Percistence#: 4-------->
Average fitness: 102.78049333333344Average length: 29.506666666666668
Best fitness: 155.985
TrainFitness: 0.7959183673469388
Validationfitness: 0.0
Testfitness: 0.7326732673267327
Length: 15.0
GP:
 thal = 7.0 Or thalach < 120.59607185969077 Or exang = 0.0 And trestbps > 164.7144129168338
```

Evolution completed: Sep 19 14:12:45

Figure 19: G-REX GUI.

[1] A BNF grammar is a description of a language using the standard Backus-Naur form.

6.1.2 G-REX simplification

Although rules produced by G-REX are often rather compact they still contain unnecessary elements on occasions. These elements can be parts of a rule that is never executed or simply logical nonsense. The term *intron*, used in the GP community, is defined as "all uncalled subtrees or subtrees that are called and whose result is ignored". Introns are normally described as a beneficial aspect of the genetic search that serves at least two important purposes. The first benefit of introns is their ability to protect the program which they are a part of from destructive crossover. The second benefit is to allow the population as a whole to protect and thereby preserve the best building blocks in the population. In some studies there are even efforts to insert extra introns into the population as a search aid; see e.g. [BNK+98].

While introns are necessary in the evolutionary process, they only add meaningless complexity to the final rule. Obviously, it would be possible to (automatically or manually) remove logical nonsense from the extracted rule after evolution has finished. A more subtle issue is, however, the fact that rules often include perfectly logical and semantically correct parts that either are never called or whose value are never needed to determine the prediction. While it is certainly possible to design an algorithm for removal of all introns by inspecting whether individual nodes are used or not, we decided to use an alternative solution in G-REX.

The simplification algorithm used in G-REX was suggested in [KJN06]. The overall idea is to use G-REX also for the simplification. More specifically; when trying to simplify an extracted rule, G-REX starts by replacing the original target value with the corresponding prediction from the extracted rule. This new data set is then used by G-REX to evolve a new population of rules. Here, however, the fitness function is altered. A rule that fails to classify all training instances *exactly* as the initial rule is given a very large penalty, making it extremely unlikely that a faulty rule is going to survive, even to the next generation. Another, much smaller, penalty is used to encourage shorter evolving rules. This fitness function all but guaranties that this second round of G-REX will produce a shorter rule, 100% faithful to the initial rule.

To improve the performance and speed of the simplification process the initial population is created using the extracted rule. In more detail; the population is divided into five equally sized parts, each subjected to different levels of mutation. One part starts with just exact copies of the original rule while the remaining parts start with individuals mutated between one and four times.

It must be noted that using the simplification option should not be seen as an alternative to penalizing longer programs in the fitness function. If there is no penalty for longer rules, it becomes very easy for G-REX to overfit the training data, which almost always leads to poor generalization. Based on experimentation, the best use of the simplification option appears to be after ordinary G-REX rule extraction; i.e. still including a rather large length penalty in the fitness function. With this approach, simplification starts from rather compact rules but is still often able to further reduce complexity; for more details see [KJN06] and the experimentation chapter.

6.2 Representation languages

As described in 3.5, a genetic program is a parse tree consisting of functional nodes and terminal leaves. When using GP for rule extraction, the available functions and terminals constitute the literals of the representation language. Functions will typically be logical or relational operators, while the terminals are either input variables or constants. Naturally more complex functions like *if-then-else*, *1-of-n* or even *m-of-n* could also be used.

The function and terminal sets do not, however, fully specify the representation language, since it is not always possible to freely combine all functions and terminals. Technically, this is called *typed GP*; i.e. some nodes require child nodes to have a specific type. With this in mind, representation languages are presented here not only as function and terminal sets, but also using BNF grammars.

When performing binary classification, the easiest representation language to use is *Boolean trees*. When using Boolean trees one class is considered to be the default class; i.e. if the Boolean tree evaluates to 'true' for a certain instance, that instance is classified as belonging to the default class. Naturally, different operators and conditional expressions could be used in a Boolean tree, but each non-leaf node must be possible to evaluate to a Boolean value. Figure 20 below specifies a sample representation language based on Boolean trees. In this specific example there are *n* (continuous) input variables. Each interior node is either a *Boolean expression* (a conjunction or disjunction of Boolean trees) or a *Relational expression* where the value of a specific input variable is compared to an initially randomized constant. It should be noted that when using this grammar it is not syntactically correct to compare one input variable to another. Whether this operation is useful or not, must be determined from the specific problem at hand.

```
F = {and, or, <, >}
T = {i1, i2, …, in, ℜ}

BoolTree        :-        BoolExp | RelExp
BoolExp         :-        (BoolOp BoolTree BoolTree)
RelExp          :-        (RelOp Input Constant)
BoolOp          :-        and | or
RelOp           :-        < | >
Input           :-        i1 | i2 | … | in
Constant        :-        ℜ
```

Figure 20: Representation language for Boolean trees. (*n* continuous inputs.)

Boolean trees are of course restricted to binary problems; multiclass problems require different representation languages. Although there are several possible choices, one very straightforward is to extract a decision tree; i.e. all paths represent a series of tests eventually leading to a specific classification. This requires the inclusion of an *if-statement* node, typically having three children; a *test* (a Boolean tree) and two *subtrees*. The test determines which of the two subtrees to enter, with the convention that if the test is true, the first subtree is chosen. So, interior function nodes would be either *if statements, Boolean expressions* or *Relational expressions* while interior terminal nodes are either *input variables* or *random constants*. On the other hand, each leaf node must be a terminal representing a particular class. Figure 21 below specify a sample representation language based on decision trees. In this specific example there are *n* (continuous) input variables and *m* classes.

```
F = {if, or, and, <, >}
T = {i1, i2, …, in, c1, c2, …, cm, ℜ}

DTree           :-        (if BoolTree Dtree Dtree) | Class
BoolTree        :-        BoolExp | RelExp
BoolExp         :-        (BoolOp BoolTree BoolTree)
RelExp          :-        (RelOp Input Constant)
BoolOp          :-        and | or
RelOp           :-        < | >
Input           :-        i1 | i2 | … | in
Class           :-        c1 | c2 | … | cm
Constant        :-        ℜ
```

Figure 21: Representation language for decision trees.

Up to this point, G-REX has used many different representation languages; including Boolean trees, decision trees, fuzzy trees and regression trees. As mentioned above, the representation language is internally supplied to G-REX as a text file. Figure 22 below shows a sample BNF file representing the grammar for decision trees given in Figure 21 above.

```
----------BNF---------
function;head;if;
function;if;and,or,greater,less;if,class;if,class;
function;and;and,or,less,greater;and,or,less,greater;
function;or;and,or,less,greater;and,or,less,greater;
terminalfunction;less;var;rnd;
terminalfunction;greater;var;rnd;
terminal;var;var;
terminal;rnd;rnd;
terminal;class;
```

Figure 22: BNF file for decision trees.

Each file starts with the name *head* determining the functions available to the root node. Each row starting with the name *function* represents an interior node function where the parts (separated by semicolons) give the name of the function followed by the functions or terminals allowed in each child node, separated by commas. As an example; the function *if* allows the functions *and*, *or*, *greater* and *less* in the first child node. In the second and third child node the only allowed functions and terminals are another *if* function or a *class* terminal. A *terminalfunction* is a function where each child node must represent a terminal, and, of course, a *terminal* represents exactly a terminal. The constants *rnd* and *var* represents random numbers ($\Re$) and input variables, respectively.

Subsection 6.2.1 below demonstrates how G-REX can use different representation languages when performing rule extraction. The examples were originally presented in [JKN03].

6.2.1 Rule extraction using G-REX – two introductory examples

Problem 1: Wisconsin breast cancer (WBC)

The breast cancer database was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg and has previously been used for several ML studies; see e.g. [Zha92]. The problem is to classify breast cancer as either benign or malignant, using nine attributes, all with integer values between 1 and 10.

The input variables are:

I1: Clump Thickness	I2: Uniformity of Cell Size	I3: Uniformity of Cell Shape
I4: Marginal Adhesion	I5: Single Epithelial Cell Size	I6: Bare Nuclei
I7: Bland Chromatin	I8: Normal Nucleoli	I9: Mitoses

The output classes are:

C1: Benign	C2: Malignant

This data set has 683 samples. In this experiment they were randomly divided into training and test sets with 513 and 170 samples respectively. Part of the training set was used for early stopping. Since this problem is known to be fairly easy (i.e. most techniques will produce models with accuracy around 95%) one single ANN was deemed sufficient. For the actual rule extraction the grammar for Boolean trees, as described in Figure 20 above, was used. With the variables at hand the function and terminal sets thus became:

```
F = { and, or, <, >}
T = { Clump Thickness, Uniformity of Cell Size, Uniformity of Cell
      Shape, Marginal Adhesion, Single Epithelial Cell Size,
      BareNuclei, Bland Chromatin, Normal Nucleoli, Mitoses }
```

The fitness function was based on fidelity with a penalty applied to longer programs. The purpose was to prioritize accuracy while still encouraging smaller and therefore more readable rules. Since no validation set was used the fitness function was very simple; see (66).

$$f = CI_T - 0.02 \cdot S \tag{66}$$

Table 7 shows test set results for the ANN, C5.0 and G-REX.

	ANN	C5.0	G-REX
WBC	98%	94%	96%

Table 7: WBC - percent correct on test set.

The accuracy obtained by G-REX is close to that of the ANN, and slightly higher than C5.0. The main result is, however, that the rather accurate rule extracted is very short, and therefore quite comprehensible; see Figure 23.

```
(OR  (AND (< I6 3) (AND (< I9 3) (< I2 4)))
     (AND (< I3 3) (< I1 7)))
```

Figure 23: Extracted rule for Benign.

The rule has manually simplified constants (for example 3 instead of 2.6) since the inputs are discrete. It is interesting to note that only five of nine inputs are present in the rule. Figure 24 shows a graphical representation of the evolved rule.

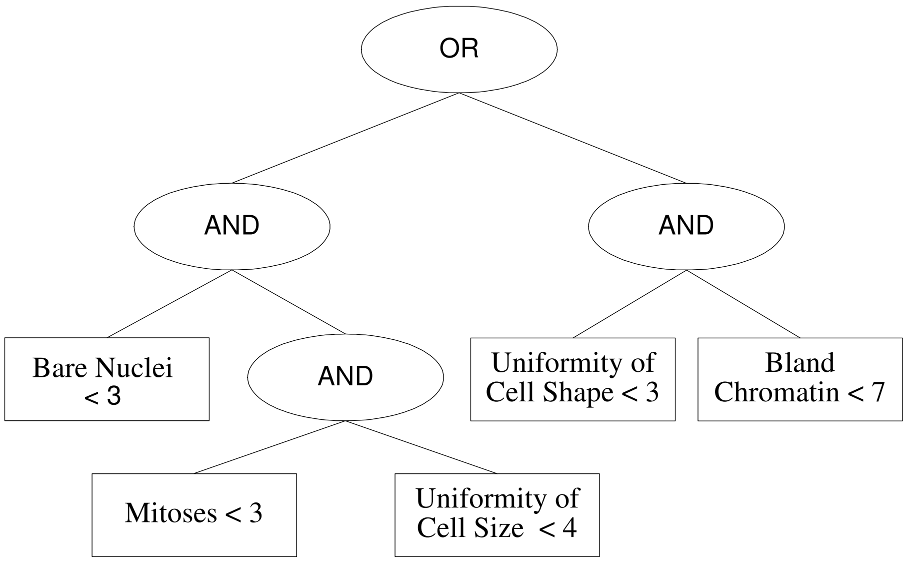


Figure 24: Graphical representation of extracted rule for Benign.

Problem 2: Iris

The Iris data set [Fis36] has been used extensively in ML studies, in particular several reports about rule extraction exist; see e.g. [WL02]. The goal is to classify three types of Iris plants (Setosa, Versicolour and Virginica) based on continuous values representing physical measurements. The input variables are:

I1: Petal width	I2: Petal length	I3: Sepal width	I4: Sepal length

The output classes are:

C1: Setosa	C2: Versicolor	C3: Virginica

One class (Setosa) is linearly separable while the other two (Versicolor and Virginica) are not. The Iris data set holds 150 instances, 50 of each class. In this experiment they were randomly divided into training and test sets with 113 and 37 samples respectively. This problem too is known to be fairly easy, so once again one single ANN was used. Since Iris is a multiclass problem, the grammar described above in Figure 21 was used. With the actual variables, the function and terminal sets became:

```
F = { if, < , >, and, or}
T = { Petal width, Petal length, Sepal width, Sepal length, Setosa,
      Versicolor, Virginica, ℜ }
```

The fitness function was identical to the one used in the WBC experiment.

The results show a few misclassifications, but overall the accuracy was very high; see Table 8.

	ANN	C5.0	G-REX
Iris	98%	92%	97%

Table 8: Percent correct on the test set for Iris.

Figure 25 shows the compact decision tree obtained. The first part establishes a specimen as Virginica if the petal width is over 17 mm. The others are classified as Versicolor if the petal length is over 27 mm and Setosa if not.

```
(If  (> Petal width  17) Virginica
     (If (> Petal length 27) Versicolor Setosa))
```

Figure 25: Extracted decision tree for Iris.

6.3 Evaluation criteria

The purpose of this section is to identify and describe each criterion relevant for the evaluation of rule extraction algorithms.

6.3.1 Accuracy

High accuracy, defined as low error on unseen data, must be considered the primary criterion for all predictive modeling. In the rule extraction domain, this criterion means that an extracted model should have high accuracy when used for prediction. One specific demand is that the extracted model should have higher accuracy than techniques directly creating transparent models from the data set. If this is not the case, there is no point in building an opaque model to extract rules from in the first place. In addition, a novel rule extraction technique must have accuracy comparable to existing techniques. As a matter of fact, if a novel technique is constantly outperformed by existing techniques regarding accuracy, it could be argued that all other criteria become unimportant. Another, more subtle, question is how much accuracy it is acceptable to give up in order to obtain comprehensibility. In practice, this is probably problem dependent, but it is nevertheless important to get a feel for how significant the accuracy vs. comprehensibility trade-off is for different rule extraction techniques. So, the accuracy of the extracted model must also be compared to the accuracy of the original, opaque model.

To evaluate the accuracy criterion, G-REX is consequently compared to both decision tree algorithms and some typical, existing, rule extraction algorithms.

For this comparison, C5.0 and CART were chosen as decision tree algorithms, with the obvious motivation that they are the most widespread techniques, available in most data mining software packages. Choosing existing rule extraction techniques to benchmark against is much harder, since no specific technique is regarded as superior to all others. With this in mind, the rule extraction algorithm RX is studied as a typical example of an open-box method and TREPAN is used as a typical black-box method.

6.3.2 Comprehensibility

In this thesis comprehensibility is always measured as the *size of the extracted representation*. This is arguably a simplification, chosen with the motivation that it makes comparison between techniques fairly straightforward. Nevertheless, it should be noted, that, strictly speaking, there is not a one-to-one relationship between size and comprehensibility. As an example, an extracted model, designed to produce a sequence of questions, where each question always relates to previous answers, would probably be considered quite comprehensible, even if the total size is rather large. Naturally, size, in turn, could be calculated in a number of ways. Some typical choices (for tree structures) include *number of questions*, *total number of tests*, *number of nodes* and *number of interior nodes*.

In practice, comprehensibility is not linear, making all measures based on size slightly dubious. As a matter of fact, it is questionable if comprehensibility is even ordinal, maybe it should be regarded as a binary property; a particular model is either comprehensible or not, given the specific situation. Using this point of view, it is not really important whether a certain extracted model has 1000 nodes or only 500; both must be considered incomprehensible. Similarly, a model with, for instance, ten nodes, should not really be considered "twice as comprehensible" when compared to a model with 20 nodes; they are probably both just "clearly comprehensible". All in all, though, the choice to evaluate comprehensibility using the size of the model must be regarded as the most accepted.

For the actual evaluation, G-REX is again compared to both decision trees (C5.0, CART) and existing rule extraction techniques (RX, TREPAN). It should be noted that although the syntax is not identical for all techniques, most measures mentioned above are applicable to them all. Another important observation is that G-REX is the only technique directly designed for producing compact representations. Nevertheless, both C5.0 and CART have different settings, which in practice will prioritize shorter rules. Especially pruning will, of course, reduce the complexity of the trees. When using RX, the network must be heavily pruned just to enable rule extraction, but there is no additional technique aimed at reducing the extracted rule set. TREPAN, finally, by growing trees in a best-first fashion, implicitly have the ability to stop extraction before the tree becomes too complex. The exact settings for each technique, as well as the exact measurements used when calculating the sizes, are given in the description of each experiment.

6.3.3 Fidelity

Fidelity is a measure that captures how similar the output from the extracted model is to the output from the opaque model. For classification problems this is easiest expressed as *percentage of instances classified identically*. For regression problems *correlation* or a *standard error measure* like MSE could be used. Even though accuracy and comprehensibility are the two most important criteria, the fidelity must also be high. A low fidelity would mean that the rule extraction algorithm does not express the relationship found by the opaque model, but instead finds another relationship. Naturally, the risk for this is increased if the rule extraction algorithm has access to instances not used when building the opaque model. On the other hand, models produced by rule extraction are normally much simpler than the opaque models they are extracted from. The implication is that extracted models are generally not capable of fully representing the complex relationships found by the opaque models. Having said that, extracted models must still have fairly high fidelity and, more importantly, this property must also transfer to a test set; i.e. the extracted model must still perform similarly to the opaque model when applied to novel data.

When evaluating G-REX fidelity, it is not very interesting to directly compare G-REX fidelity to RX and TREPAN fidelity. The reason is that fidelity is not an entity that necessarily should be maximized; it only has to be sufficiently high. Having said that, it is very hard to give a specific level representing "sufficient fidelity", instead this must be determined for each problem. More specifically, fidelity should not be independently evaluated; it must always be measured while at the same time also considering comprehensibility and accuracy.

6.3.4 Scalability

Most often scalability measures how the running time of an algorithm vary depending on the input size. For rule extraction algorithms, the term *input size* is, however, rather complex. Factors to consider are, at least, network size, number of instances, number of attributes and number of classes. In addition, it is very important to also recognize how the comprehensibility of the extracted models varies.

As described in 4.2.2, for open-box methods, the most important parameter when measuring scalability is the size of the network. For G-REX, however, this is not the case, since the actual architecture is not used in any way. The factors determining the running time when using G-REX are instead number of instances and the GP parameters used. Especially the number of generations and the population size, but also creation depth and creation method directly influence the running time.

The use of a length penalty makes it possible for G-REX to always produce short rules. In fact, G-REX can easily be forced to produce arbitrary small rules, no matter the characteristics of the data set. The interesting question is therefore how the accuracy of comprehensible rules varies with certain parameters related to the data set and the opaque model. With this in mind, the first two obvious factors are number of classes and number of attributes; both affecting the GP search space. Whether the attributes are continuous or categorical will also affect the GP search, since different operators are used. In addition, it is not obvious how the level of accuracy obtained by the opaque model will affect the rule extraction.

In the experimentation chapter, there is no experiment explicitly aimed at investigating scalability. Instead, scalability is analyzed and conclusions are drawn, using the results obtained in other experiments. Nevertheless, data sets are selected to guarantee sufficient variation in all properties described above.

6.3.5 Generality

Generality is a rather complex criterion. Naturally, the overall intention is that the technique should be as flexible as possible, but there is no exact definition. Here, generality is therefore treated as a multifaceted issue, requiring evaluation of several aspects. Below is a short description of the most important factors that are recognized in this thesis and used for evaluating G-REX.

Applicability to a variety of opaque models

First of all, a rule extraction technique is very limited if it is only capable of extracting rules from a specific kind of opaque model; typically a single MLP. Ultimately, a technique should be able to disregard all factors like underlying architecture and training regime to extract rules from any kind of opaque model in an identical fashion. Naturally, this criterion is impossible to achieve for open-box approaches, but it should also be noted that not all black-box techniques aim for this kind of generality. Since G-REX only uses the *output* from the opaque model together with original inputs, G-REX is indifferent to the technique used when building the opaque model. This criterion must therefore be considered conclusively met by G-REX. Nevertheless, G-REX ability to extract rules from single ANNs, ANN ensembles and boosted trees are demonstrated in this thesis. In addition, G-REX has also been used to extract rules from heterogeneous ensembles (see [JNK04] and [JKN05]) and Support Vector Machines [MBG+06].

Ability to handle various predictive modeling problems

Ideally, it should be possible to extract accurate rules from all kinds of predictive models. This includes, of course, both regression and classification problems. Furthermore, the technique must manage both binary classification and multiclass problems. Naturally, the rule extraction technique should also be able to handle all kinds of input variables; i.e. continuous, categorical and ordinal.

It should be noted that this criterion also requires the *performance* of the rule extraction technique to be fairly robust over different problems; i.e. it must not constantly perform poorly on a specific group of problems.

G-REX has almost exclusively been used on classification problems, but has also been applied to one regression task in the *Impact of Advertising* study; see 7.5.3. Additionally, G-REX is tested on a number of different problems in the experiments presented in 6.5, where, among other things, the robustness of G-REX is evaluated.

Capability to extract rules in several representation languages

There are essentially two different motivations for the ability to use several representation languages. First of all, for many problems the characteristics of the data set determine the representation language; e.g. categorical variables require different operators than continuous. Another example is that binary classification problems are probably best handled by using a Boolean tree, while multiclass problems require a decision tree. Second, the possibility to choose representation language based on the situation may increase subjective comprehensibility for decision makers.

G-REX ability to extract rules in a multitude of representation languages is demonstrated in the experimentation chapter and the Impact of Advertising case study; see 6.5 and Chapter 7.

6.3.6 Consistency

Consistency tries to measure how much extracted models vary between different runs. The motivation is that if extracted models differ greatly, it becomes very hard to put a lot of faith in one specific model.

Consistency could obviously be measured (or compared) in different ways. The most natural way is probably to somehow compare actual rules. If, as an example, the extracted representation is a decision tree, the first splits could be compared to determine if all instances prioritize the same input variables. The drawback of this approach is that it is very hard to find a general measurement applicable to all possible representations. A more straightforward technique is therefore to ignore the actual rules and instead compare *predictions* from the rules. This approach makes it rather easy to find numerical values, and is also the technique used in, for instance, [TS93]. Another subtlety is the question of whether consistency should be measured between rule extractions from the same opaque model or between entire experiments, just using the same data set. Towell and Shavlik in [TS93] used the latter approach, which is reasonable since their rule extraction approach is deterministic in the sense that it will always produce identical rules when using a specific trained ANN. For G-REX, however, this is not the case, since there is always a certain amount of randomness in all GP search. With this in mind, the most natural way to measure consistency for G-REX is by comparing extracted rules from a common opaque model. In addition, it is also interesting to compare rules extracted from different opaque models, all built for the same problem.

6.4 Using oracle data for rule extraction

The normal result when performing rule extraction is another predictive model (the *extracted model*) which in turn, is used for actual prediction; see Figure 16. With this perspective, accuracy must be considered the prioritized criterion for rule extraction as well. Although this may seem almost obvious, most rule extraction techniques instead focus on the criterion fidelity; i.e. how well the extracted model mimics the opaque. In this thesis, however, fidelity is mainly a way to reach the goal (i.e. high accuracy) and not an end in itself.

At the same time, it is important to realize that the opaque model is normally a very accurate model of the original relationship between input and target variables. Furthermore; access to the opaque model makes it possible to generate predictions for novel instances with unknown target values as they become available. Naturally, these instances could also be used by the rule extraction algorithm. This is a major difference compared to techniques directly building transparent models from the data set, where each training instance must have a known target value. Nevertheless, all rule extraction algorithms that the author is aware of, use only training data (possibly with the addition of artificially generated instances) when extracting the transparent model. Here it is argued that there are situations where a data miner might benefit from also using test data together with predictions from the opaque model when performing rule extraction. Below, test data inputs together with test data predictions from the opaque model is termed *oracle data*, with the motivation that the predictions from the opaque model (the oracle) are regarded as ground truth during rule extraction. Naturally, target values for test data are by definition not available when performing rule extraction, but often input values and predictions from the opaque model could be. The interesting question is if the use of oracle data will increase accuracy when performing rule extraction. The use of oracle data was first suggested in [JLK$^+$06a] and further evaluated in [JLK$^+$06e].

The hypothesis is that maximizing fidelity, which is the measurement we have available for the test set, will in fact also increase accuracy. This is, of course, in sharp contrast to the claim made by Zhou in [Zho04] stating that either accuracy *or* fidelity must be the target; and once this is decided the other criterion becomes irrelevant.

Since the use of oracle data means that the same novel data instances used for actual prediction also are used by the rule extraction algorithm, the problem must be one where predictions are made for sets of instances rather than one instance at the time. This is a description matching most data mining problems, but not all. In some situations, a sufficiently sized oracle data set is not available. One such example is a medical system where diagnosis is based on a predictive model built from historical data. In that situation, test set instances (patients) would probably be handled one at a time. On the other hand, if, as an example, a predictive model is used to determine the recipients of a marketing campaign, the oracle data set could easily contain thousands of instances. With access to a sufficiently sized oracle data set, the rule extraction algorithm could either use *only* oracle data or augment the training data with oracle instances.

6.5 G-REX evaluation using public data sets

Five different experiments are used to evaluate G-REX against the criteria presented earlier. In the first experiment, G-REX is compared to the standard techniques C5.0 and CART regarding accuracy.

The main purpose of the second experiment is to determine if the use of oracle data is beneficial when performing rule extraction. In this experiment, G-REX, using oracle data, is compared to CART and C5.0. The primary criterion is again accuracy.

In the third experiment, the size of trees extracted by G-REX is compared to the size of trees induced by standard decision tree algorithms. The purpose is to evaluate the comprehensibility criterion.

In the fourth experiment, G-REX is compared to existing rule extraction techniques; i.e. TREPAN and RX. Accuracy and comprehensibility are again the most important criteria.

In the fifth experiment, the consistency of G-REX is evaluated. This experiment consists of two parts: First the predictions from several models extracted from a *common* opaque model are compared. Second, predictions obtained by rule extracting from (slightly) *different* opaque models are compared. Naturally, the more similar the predictions are between runs the higher the consistency is.

The data sets used are all publicly available and were gathered from the UCI Repository [BM98]. Below is a very short description of each data set:

- **BUPA liver disorders (BLD)**: The problem is to predict whether or not a male patient has a liver disorder based on blood tests and alcohol consumption.
- **Cleveland heart disease (Cleve)**: Prediction of whether a patient has a heart disease or not from medical measurements.
- **Contraceptive Method Choice (CMC)**: Prediction of contraceptive method used (no use, short-term, long-term), by a woman based on her demographic and socio-economic characteristics.
- **Credit card applications (Crx)**: This is a data set where all attribute names and values have been changed to meaningless symbols to protect confidentiality of the data. The target variable is acceptance or rejection of credit card applications.

- **German**: German credit data. The task is to predict whether a credit card applicant is “good” or “bad” based on financial and social attributes.
- **Glass**: Classification of origin for glass (e.g. building window, vehicle window, head lights etc.) using proportion of elements. The study of classification of types of glass is motivated by criminological investigations.
- **Johns Hopkins Ionosphere (Iono)**: The purpose is to predict "Good" radar returns; i.e. those showing evidence of some type of structure in the ionosphere.
- **Iris**: Prediction of class of iris from measurements on plants.
- **Labor**: The data includes all collective agreements reached in the business and personal services sector for locals with at least 500 members (teachers, nurses, university staff, police, etc) in Canada in 1987 and first quarter of 1988. The attributes describe the contract and the binary target variable is *acceptable* and *unacceptable*.
- **Lymph**: Lymphography data from Ljubljana Oncology Institute. The task is to classify the character of a lymph node based on medical observations.
- **PIMA Indians Diabetes (PID)**: Prediction of whether or not a patient shows signs of diabetes based on personal and medical attributes.
- **Sonar**: The task is to train a model to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock.
- **Teaching assistant evaluation (TAE)**: Prediction of how good evaluation a teacher assistant is likely to receive based on teacher and course data.
- **Tic-Tac-Toe**: This database encodes the complete set of possible board configurations at the end of tic-tac-toe games, where X is assumed to have played first. The target concept is *Win for X*.
- **Wisconsin breast cancer (WBC)**: Prediction of whether breast cancer is malignant or benign from medical observations and measurements.
- **Wine**: Recognition of wine based on quantities of 13 constituents found in each of the three types.

- **Zoo database (Zoo)**: The task is to classify animals into types from their different physical attributes.

For a summary of the characteristics of the data sets, see Table 9 below.

Data sets	Instances	Classes	Continuous inputs	Categorical inputs
BLD	345	2	6	0
Cleve	303	2	6	7
CMC	1473	3	2	7
Crx	690	2	6	9
German	1000	2	7	13
Glass	214	7	9	0
Iono	351	2	34	0
Iris	150	3	4	0
Labor	57	2	8	8
Lymph	148	4	3	15
PID	768	2	8	0
Sonar	208	2	60	0
TAE	151	3	1	4
Tic-Tac-Toe	958	2	0	9
WBC	699	2	9	0
Wine	178	3	13	0
Zoo	100	7	0	16

Table 9: UCI data set characteristics.

6.5.1 Experiment 1 – comparing G-REX to C5.0 and CART

Here, G-REX is thoroughly compared to two standard techniques for creating transparent predictive models directly from the data set.

Method

This experiment uses all the 17 UCI data sets in Table 9. The main experiment compares G-REX to CART [BFO+84] and C5.0 [Qui98], using test set accuracy. CART and C5.0 results were obtained using the commercial data mining tool Clementine[1]. Naturally, both techniques have many different parameters; but Clementine also supply some automated settings for CART and C5.0. More specifically; the CART node has an option to set the mode to *simple*, meaning that the values of all parameters are set automatically, based on data set properties. Similarly, the C5.0 node has two automated modes called *simple mode – favoring*

[1] In Clementine there is no node named CART. There is, however, a node called C&RT. The C&RT algorithm is, according to the manual, "An algorithm for creating a decision tree based on minimization of impurity measures. Also known as CART." With this in mind, the name CART is used exclusively in this thesis.

accuracy and *simple mode – favoring generality*. When accuracy is favored this in fact means training set accuracy, while the generality mode tries to obtain a model more likely to generalize well to test data. In this experiment, both simple modes for C5.0 and the simple mode for CART are used.

For actual experimentation, standard 10-fold cross validation is used to obtain the performance for each technique on every data set. For statistical comparison of the techniques, the Friedman test, as described in 2.7.2, is used. This is the approach recommended by Demšar [Dem06] for comparing more than two algorithms over several data sets.

The opaque models used to rule extract from are ANN ensembles each consisting of 20 independently trained ANNs. All ANNs are fully connected feed-forward networks and a localist representation was used. Of the 20 ANNs, five have no hidden layer, ten have one hidden layer and the remaining five have two hidden layers. The exact number of units in each hidden layer is slightly randomized, but is based on the number of inputs and classes in the current data set. For an ANN with one hidden layer, the number of hidden units is determined from (67) below.

$$h = \left\lfloor 2 \cdot rand \cdot \sqrt{(v \cdot c)} \right\rfloor \tag{67}$$

v is the number of input variables and c is the number of classes. *rand* is a random number in the interval [0, 1]. For ANNs with two hidden layers, the number of units in the first hidden layer is h_1 determined from (68) below and the second hidden layer is h_2 from (69) below. v is again the number of input variables and c is the number of classes.

$$h_1 = \left\lfloor \sqrt{(v \cdot c)} / 2 + 4 \cdot rand \cdot (\sqrt{(v \cdot c)} / c) \right\rfloor \tag{68}$$

$$h_2 = \left\lfloor rand \cdot (\sqrt{(v \cdot c)} / c) + c \right\rfloor \tag{69}$$

Regarding ANN training, no early stopping, based on validation set performance, was used. In addition, no hold out set was used to try to emulate performance on unseen data. Instead all available data was used for the actual ANN training. Each ANN was, however, trained using only 70% (randomized without replacement) of all instances in an attempt to increase ANN diversity.

The representation language used by G-REX is similar to decision trees, but compound queries in the nodes are allowed; see Figure 26 for the exact BNF used.

```
F = {if, or, and, equals, <, >}
T = {i₁, i₂, …, iₙ, c₁, c₂, …, cₘ, ℜ}

DTree      :-    (if BTree Dtree Dtree) | Class
BTree      :-    BExp | RExp
BExp       :-    (BOp BTree BTree)
RExp       :-    (ROp ConInp ConConst) | (equals CatInp CatConst)
BOp        :-    and | or
ROp        :-    < | >
CatInp     :-    Categorical input variable
ConInp     :-    Continuous input variable
Class      :-    Class label
CatConst   :-    Categorical attribute value
ConConst   :-    ℜ
```

Figure 26: BNF used in Experiment 1.

The GP settings used for G-REX in this experiment are given in Table 10 below.

Parameter	**Value**	**Parameter**	**Value**
Crossover rate	0.8	Creation depth	8
Mutation rate	0.001	Creation method	Ramped half-and-half
Population size	2500	Fitness function	Fidelity with small length penalty
Generations	100	Selection	Roulette wheel
Persistence	50	Elitism	Yes

Table 10: GP parameters for Experiment 1.

Here, a fitness function with a rather small penalty for longer rules was used. Because of this, the main component in the fitness function is just fidelity to the opaque model, on the training data. More specifically the fitness function used is:

$$fitness = CI_T - 0.01 \cdot S \qquad (70)$$

where CI_T is the number of training instances classified identically to the opaque model, and S is the size of a G-REX individual.

Results

Table 11 shows the main results from Experiment 1. All results are mean accuracies over the ten folds for each data set.

Data sets	C5.0 Acc	C5.0 Gen	CART	Ensemble	G-REX		
	Test Acc.	Test Acc.	Test Acc.	Test Acc.	Train Fid.	Test Fid.	Test. Acc.
BLD	0.597	0.629	0.629	0.685	0.791	0.753	**0.650**
Cleve	0.740	0.760	0.737	0.800	0.869	0.820	**0.793**
CMC	0.530	0.555	0.547	0.550	0.741	0.770	**0.565**
Crx	**0.871**	0.862	0.852	0.848	0.895	0.923	0.849
German	0.716	0.718	0.719	0.742	0.767	0.815	**0.723**
Glass	0.638	**0.671**	0.652	0.690	0.701	0.729	0.643
Iono	0.891	**0.903**	0.857	0.934	0.944	0.906	**0.903**
Iris	0.953	0.933	**0.967**	0.973	0.979	0.980	**0.967**
Labor	0.783	0.767	0.883	0.917	0.998	0.850	**0.900**
Lymph	0.779	0.771	**0.800**	0.850	0.855	0.793	0.786
PID	0.757	**0.758**	0.739	0.766	0.840	0.836	0.746
Sonar	0.765	0.755	0.700	0.825	0.836	0.775	**0.770**
TAE	0.447	0.393	**0.527**	0.573	0.782	0.727	0.500
Tic-Tac-Toe	0.809	0.822	0.822	0.911	0.847	0.874	**0.835**
WBC	0.941	0.942	0.929	0.967	0.977	0.977	**0.952**
Wine	0.941	0.924	0.900	0.988	0.968	0.947	**0.947**
Zoo	**0.940**	**0.940**	0.930	0.970	0.926	0.940	0.920
#Best	**2**	**4**	**3**	**N/A**	**N/A**	**N/A**	**11**

Table 11: Comparing G-REX, C5.0 and CART accuracies.

First of all it should be noted that the opaque model outperforms the best decision tree on 15 of 17 data sets. This is no surprise, but confirms that the use of decision tree algorithms in order to obtain transparent models, most often will result in a loss of accuracy. When comparing G-REX to CART and C5.0, G-REX actually obtained at least as high accuracy as the best tree algorithm (CART or C5.0) on a majority (11) of the data sets.

The Friedman test uses the test statistic:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2} \tag{71}$$

where

$$\chi_F^2 = \frac{12N}{k(k+1)}\left(\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right) \tag{72}$$

This test statistic is distributed according to the *F-distribution* with *k-1* and *(k-1)(N-1)* degrees of freedom. *N* (the number of data sets) is 17 and *k* (the number of algorithms) is 4. R_j is the average rank for algorithm *j*.

The ranks obtained in this experiment are given in Table 12 below.

Data sets	**C5.0 - Acc**	**C5.0 - Gen**	**CART**	**G-REX**
BLD	4	2	3	1
Cleve	3	2	4	1
CMC	4	2	3	1
Crx	1	2	3	4
German	4	3	2	1
Glass	4	1	2	3
Iono	3	1	4	2
Iris	3	4	1.5	1.5
Labor	3	4	2	1
Lymph	3	4	1	2
PID	2	1	4	3
Sonar	2	3	4	1
TAE	3	4	1	2
Tic-Tac-Toe	4	2	3	1
WBC	3	2	4	1
Wine	2	3	4	1
Zoo	1.5	1.5	3	4
Average rank:	**2.91**	**2.44**	**2.85**	**1.79**

Table 12: Ranks for techniques producing transparent models.

Using the average ranks, F_F=*3.03*. The critical value of *F(3,48)* for α=*0.05* is *2.80*, so the null hypothesis that all four algorithms have performed equivalently is rejected.

Using the post-hoc Bonferroni-Dunn test [Dun61], the performance of G-REX and the decision tree algorithms is significantly different if the corresponding average ranks differ by at least the critical difference given in (73).

$$CD = q_{\alpha}\sqrt{\frac{k(k+1)}{6N}} = 2.394 \cdot \sqrt{\frac{4(4+1)}{6 \cdot 17}} = 1.06 \qquad (73)$$

So, based on the Freiedman test and the Bonferroni-Dunn post-hoc test, G-REX in this experiment was significantly more accurate than C5.0-Acc and CART. To further evaluate G-REX against C5.0-GEN, a pair-wise comparison, using the Wilcoxon signed-ranks test (see 2.7.2), was carried out; see Table 13 below.

Data sets	C5.0 - Gen	G-REX	Difference	Rank
BLD	0.629	0.650	0.021	11
Cleve	0.760	0.793	0.033	14
CMC	0.555	0.565	0.010	3
Crx	0.862	0.849	-0.013	6.5
German	0.718	0.723	0.005	2
Glass	0.671	0.643	-0.028	13
Iono	0.903	0.903	0.000	Ignore
Iris	0.933	0.967	0.034	15
Labor	0.767	0.900	0.133	17
Lymph	0.771	0.786	0.015	8.5
PID	0.758	0.746	-0.012	5
Sonar	0.755	0.770	0.015	8.5
TAE	0.393	0.500	0.107	16
Tic-Tac-Toe	0.822	0.835	0.013	6.5
WBC	0.942	0.952	0.010	4
Wine	0.924	0.947	0.023	12
Zoo	0.940	0.920	-0.020	10

Table 13: Wilcoxon signed-ranks test between G-REX and C5.0-Gen.

From this data, the sums $R^+=117.5$ and $R^-=34.5$. The test statistic $T=min\{R^+, R^-\}=34.5$. The critical value for a two-tailed test using *17* data sets and $\alpha=0.05$ is *35*; see Table 2. So, the difference in accuracy between *G-REX* and *C5.0-Gen* turns out to be just significant, at this confidence level.

Conclusions

In this experiment, the ANN ensemble used outperformed the best decision tree algorithm on all but two data sets. This result clearly indicates that the choice of a decision tree algorithm, in order to obtain a transparent model, most often will result in lower accuracy compared to, for instance, the opaque ANN ensemble. With this in mind, we must recognize the trade-off between accuracy and comprehensibility, when selecting data mining technique. The ability to reduce this trade-off by producing comprehensible yet accurate models is therefore a very strong argument for rule extraction in general.

In addition, the comparison between G-REX and C5.0 and CART showed that G-REX, in this experiment performed significantly better than the well-known decision tree algorithms.

6.5.2 Experiment 2 – the use of oracle data

The overall purpose of this experiment is to evaluate the benefit of including unlabeled, "oracle" data when performing rule extraction.

Method

The main experiment compares test set accuracy when using oracle data to normal rule extraction; i.e. when only training data is used to build the extracted model. More specifically, three slightly different approaches are compared; G-REX using only training data (called *original* below), G-REX using both training data and oracle data (*all*) and G-REX using only oracle data (*oracle*). It should be noted that, in practice, Experiment 1 was performed as a part of this larger experiment. This means that the ANN ensembles used to rule extract from were identical and all G-REX settings were the same – the only difference is the addition of oracle data. For evaluation, standard 10-fold cross validation followed by a Friedman test was again used.

Results

Table 14 shows the main results from the experiment with oracle data. All results are mean accuracy over the ten folds for each data set.

Data sets	**C5.0 Acc**	**C5.0 Gen**	**CART**	**Ensemble**	**G-REX original**			**G-REX all**			**G-REX oracle**	
	Test Acc.	**Test Acc.**	**Test Acc.**	**Test Acc.**	**Train Fid.**	**Test Fid.**	**Test. Acc.**	**Train Fid.**	**Test Fid.**	**Test Acc.**	**Test Fid.**	**Test Acc.**
BLD	0.597	0.629	0.629	0.685	0.791	0.753	0.650	0.773	0.826	**0.659**	0.912	0.656
Cleve	0.740	0.760	0.737	0.800	0.869	0.820	0.793	0.867	0.897	0.797	0.970	**0.810**
CMC	0.530	0.555	0.547	0.550	0.741	0.770	**0.565**	0.741	0.758	0.544	0.799	0.556
Crx	**0.871**	0.862	0.852	0.848	0.895	0.923	0.849	0.890	0.938	0.858	0.974	0.854
German	0.716	0.718	0.719	0.742	0.767	0.815	0.723	0.766	0.837	**0.729**	0.895	0.727
Glass	0.638	0.671	0.652	0.690	0.701	0.729	0.643	0.707	0.757	**0.671**	0.819	0.652
Iono	0.891	0.903	0.857	0.934	0.944	0.906	0.903	0.945	0.954	**0.934**	0.994	**0.934**
Iris	0.953	0.933	0.967	0.973	0.979	0.980	0.967	0.979	0.993	0.967	1.000	**0.973**
Labor	0.783	0.767	0.883	0.917	0.998	0.850	0.900	0.988	0.983	**0.933**	1.000	0.917
Lymph	0.779	0.771	0.800	0.850	0.855	0.793	0.786	0.863	0.893	0.829	0.979	**0.843**
PID	0.757	0.758	0.739	0.766	0.840	0.836	0.746	0.839	0.875	**0.759**	0.914	**0.759**
Sonar	0.765	0.755	0.700	0.825	0.836	0.775	0.770	0.832	0.860	0.785	0.995	**0.830**
TAE	0.447	0.393	0.527	0.573	0.782	0.727	0.500	0.757	0.793	0.513	0.947	**0.560**
Tic-Tac-Toe	0.809	0.822	0.822	0.911	0.847	0.874	**0.835**	0.824	0.876	0.814	0.893	0.816
WBC	0.941	0.942	0.929	0.967	0.977	0.977	0.952	0.975	0.983	0.961	1.000	**0.967**
Wine	0.941	0.924	0.900	0.988	0.968	0.947	0.947	0.958	0.959	0.971	1.000	**0.988**
Zoo	0.940	0.940	0.930	0.970	0.926	0.940	0.920	0.915	0.970	0.940	0.980	**0.950**
#Best	**1**	**0**	**0**	**N/A**	**N/A**	**N/A**	**2**	**N/A**	**N/A**	**6**	**N/A**	**10**

Table 14: Results for experiment with oracle data.

The overall picture is that the use of oracle data was very successful. When comparing all transparent models, the model extracted using oracle data only, has the highest accuracy on 10 of 17 data sets. The second highest number of wins (6) was obtained by *G-REX all*. As a matter of fact, the use of oracle data is successful on all data sets except CMC and Tic-Tac-Toe, where *G-REX original* has the highest accuracy.

On some data sets (Cleve, CMC, German, Labor and Sonar), the extracted model even outperformed the opaque. This may appear to be an anomaly, but one possible explanation is that these rules generalized so well (better than the opaque model) because they only had expressive power to capture the most important relationships. Generally though, it is of course vital for rule extraction that the opaque model is very accurate to start with.

Regarding fidelity, it is interesting to note that although the training fidelity is normally rather high, it is often far from 100%. This is natural since the extracted model is per definition less powerful than the opaque model, especially when forced to look for compact rules. On the other hand, it is reassuring that the fidelity for *G-REX original*, overall, drops only marginally between the training and the test set. It must, of course, be noted that the term *test fidelity* for *G-REX all* and *G-REX oracle* is somewhat misleading, as test set instances were, after all, part of the fitness function.

Since the purpose of the experiment is to compare the techniques extracting or inducing transparent models, the decision tree algorithms are compared to all G-REX variants. Since another Friedman test should be used, the ranks were calculated for all algorithms; see Table 15 below.

Data sets	**C5.0 Acc**	**C5.0 Gen**	**CART**	**G-REX original**	**G-REX all**	**G-REX oracle**
BLD	6	4	5	3	1	2
Cleve	5	4	6	3	2	1
CMC	6	3	4	1	5	2
Crx	1	2	5	6	3	4
German	6	5	4	3	1	2
Glass	6	2	4	5	1	3
Iono	5	3	6	4	1.5	1.5
Iris	5	6	3	3	3	1
Labor	5	6	4	3	1	2
Lymph	5	6	3	4	2	1
PID	4	3	6	5	1.5	1.5
Sonar	4	5	6	3	2	1
TAE	5	6	2	4	3	1
Tic-Tac-Toe	6	2	3	1	5	4
WBC	5	4	6	3	2	1
Wine	4	5	6	3	2	1
Zoo	3	3	5	6	3	1
Average rank:	**4.76**	**4.06**	**4.59**	**3.53**	**2.29**	**1.76**

Table 15: Ranks for techniques producing transparent models.

Using the average ranks, F_F*=12.2*. The critical value of *F(5,80)* for α*=0.05* is *2.33*, so we strongly reject the null hypothesis that all algorithms have identical accuracy.

Using the post-hoc Nemenyi test [Nem63] the performance of two algorithms is significantly different if the corresponding average ranks differ by at least the critical difference given in (74).

$$CD = q_{\alpha}\sqrt{\frac{k(k+1)}{6N}} = 2.850 \cdot \sqrt{\frac{6(6+1)}{6 \cdot 17}} = 1.829 \qquad (74)$$

So, the conclusion is that *G-REX oracle* has significantly higher accuracy than all decision tree approaches. *G-REX all*, at the same time, has significantly higher accuracy than *C5.0-Acc* and *CART*. It could also be noted that *G-REX oracle* is fairly close to be significantly more accurate than even *G-REX original.*

Conclusions

The main contribution of this experiment was to show the benefit of using test set data instances, together with predictions from the opaque model, when performing rule extraction. The technique evaluated means that the same novel data instances used for actual prediction also are used by the rule extraction algorithm. As demonstrated in the experiments, rules extracted using only oracle data were significantly more accurate than both rules extracted by the same rule extraction algorithm (using training data only) and standard decision tree algorithms. The overall implication is that rules extracted in this way will have higher accuracy on the test set; thus explaining the predictions made on the novel data better than rules extracted in the standard way; i.e. using training data only.

6.5.3 Experiment 3 – comprehensibility

In this experiment, the size of the trees extracted by G-REX is compared to the size of trees induced by C5.0 and CART. The underlying assumption is of course that smaller trees are easier to grasp and therefore provide better explanations.

Method

This experiment uses exactly the same trials as Experiment 2. Here, however, the variant *C5.0-Acc* is excluded from the analysis. The reason is that prioritizing training set accuracy normally leads to very large trees.

First of all, it must be noted that it is not entirely trivial to compare the sizes of trees built using different techniques. Although the main characteristics are the same, the exact representation languages differ slightly. As described above, G-REX in this experiment uses a representation language where tests may include nested disjunctions and conjunctions. For categorical variables, the operator *equals* is used exclusively. For continuous variables, the tests use the relational operators; i.e. '<' and '>'. C5.0 and CART tests involving continuous variables also use relational operators. For categorical variables, however, C5.0 and CART can use both simple tests (e.g. *x1=0*) and more complex *1-of-N* tests; for instance *x1 in [1, 2, 3]*.

With the slightly different representation languages in mind, two different measurements of size were used for the comparison. The first measurement, called *simple size* calculates only the *number of questions* (interior nodes) in the tree, thus ignoring the complexity of individual questions. For G-REX this in practice means that only the number of *if-statements* is counted. Consequently, the complexity of each if-statement is disregarded. For C5.0 and CART, multi-splits are only counted as one question. This includes all *1-of-n* questions, but also questions like the one illustrated in bold in Figure 27 below. The *Mode* part of each test branch shows the majority class for that branch.

```
field7 = 1 [ Mode: 3 ]
     field4 <= 3 [ Mode: 2 ] => 2
     field4 > 3 [ Mode: 3 ]
              field8 in [ 1 2 3 ] [ Mode: 2 ] => 2
              field8 in [ 4 ] [ Mode: 3 ]
                       field1 <= 30 [ Mode: 1 ] => 1
                       field1 > 30 [ Mode: 3 ] => 3
field7 = 2 [ Mode: 3 ]
     field5 = 1 [ Mode: 3 ]
              field4 <= 6 [ Mode: 3 ]
                       field1 <= 35 [ Mode: 3 ] => 3
                       field1 > 35 [ Mode: 1 ] => 1
              field4 > 6 [ Mode: 1 ] => 1
     field5 = 0 [ Mode: 1 ] => 1
field7 = 3 [ Mode: 3 ]
     field1 <= 23 [ Mode: 1 ] => 1
     field1 > 23 [ Mode: 3 ] => 3
field7 = 4 [ Mode: 3 ] => 3
```

Figure 27: Illustrating a multi-split test in C5.0.

The reason behind measuring *simple size* is that it constitutes the total number of questions that need to be answered in order to obtain a class label for every possible instance.

The second measurement, called *complex size*, tries to capture the total number of *individual tests* in a tree. For G-REX, this means that every single test is counted, which translates to calculating the total number of operators ('<', '>', '=') in the extracted tree. For C5.0 and CART, each test on continuous variables is counted once. For multi-split questions the principle employed is to calculate the *smallest* number of individual tests needed, in the worst case, to determine the answer to the question. As an example, the question in Figure 28 only counts as 1, since it is practice is equivalent to *if field8 = 4 then class 3 else class 1*.

```
field8 in [ 1 2 3 ] [ Mode: 1 ] => 1
field8 in [ 4 ] [ Mode: 3 ] => 3
```

Figure 28: A 1-of-n test in C5.0 with complex size 1.

Accordingly, the question highlighted in Figure 29, has a complex size of 4, since it is equivalent to: *if (field12 = 0 or field12 = 1) then class 1 else if (field12 = 4 or field12 = 5) then class 2 else …* thus requiring four individual tests to determine the answer to the question.

```
field12 in [ 0 1 ] [ Mode: 1 ] => 1
field12 in [ 2 3 ] [ Mode: 1 ]
     field1 <= 61 [ Mode: 2 ]
              field1 <= 55.500 [ Mode: 1 ] => 1
              field1 > 55.500 [ Mode: 2 ] => 2
     field1 > 61 [ Mode: 1 ] => 1
field12 in [ 4 5 ] [ Mode: 2 ] => 2
```

Figure 29: A 1-of-n test in C5.0 with complex size 4.

Consequently, the multi-split question in Figure 27 above has a *complex size* of 3, since it translates to *if field7 = 1 then … else if field7 = 2 then … else if filed7 = 3 then … else class 3*; i.e. requiring three tests to answer the question.

Results

Table 16 shows the results from the experiment measuring simple size. All results are mean values over the ten folds for each data set.

Data sets	**C5**	**CART**	**G-REX Original**		**G-REX All**		**G-REX Oracle**	
			Raw	**Simplified**	**Raw**	**Simplified**	**Raw**	**Simplified**
BLD	11.0	21.5	5.5	4.6	5.2	5.1	2.0	1.9
Cleve	4.6	17.8	4.2	4.0	5.2	4.7	2.2	2.1
CMC	34.1	18.9	8.8	7.1	9.0	6.6	6.2	4.8
Crx	5.2	13.9	4.6	4.2	4.1	3.4	2.1	2.1
German	15.4	17.4	4.3	4.3	7.0	6.2	4.5	3.8
Glass	8.4	21.0	4.0	4.0	5.7	5.2	2.7	2.7
Iono	6.7	8.0	3.2	3.2	4.9	4.4	1.9	1.9
Iris	2.6	4.9	3.1	3.0	2.8	2.8	2.0	2.0
Labor	1.7	5.1	3.5	3.3	3.6	3.5	1.1	1.1
Lymph	6.5	12.6	3.3	3.2	5.1	4.7	1.6	1.6
PID	10.4	18.6	4.6	4.0	4.3	3.7	2.5	2.5
Sonar	7.8	12.0	5.9	5.4	5.2	5.0	2.3	2.3
TAE	1.8	26.2	9.3	9.0	6.7	6.4	2.9	2.9
Tic-Tac-Toe	16.7	32.8	5.6	5.2	4.9	4.9	2.4	2.4
WBC	5.7	7.2	3.3	3.3	3.6	3.4	1.5	1.5
Wine	3.7	5.9	4.6	4.3	3.7	3.5	2.2	2.1
Zoo	6.1	8.1	6.2	5.7	5.9	5.4	3.1	3.1
Mean:	**8.7**	**14.8**	**4.9**	**4.6**	**5.1**	**4.6**	**2.5**	**2.4**

Table 16: Complexity measured as number of questions.

First of all, it must be noted that while G-REX explicitly targets finding compact rules, this is not the case for C5.0 and CART. Nevertheless, it is very encouraging for the G-REX approach that the extracted trees most often are smaller than corresponding trees induced by C5.0 or CART. It is not surprising that the trees extracted using only oracle data are the most compact, since there are fewer instances to model. On the other hand, it is obvious that *G-REX oracle* is capable of producing extremely small, yet very accurate explanations of the predictions made by the opaque model.

Even though the trees extracted by G-REX are rather compact, the simplification process most often succeeds in further reducing the size. Since simplification is an integral part of G-REX, the most interesting comparison is between C5.0 and *G-REX original simplified* since they use an identical number of instances when building the model. Table 17 below shows the data for a pair-wise comparison, again using the Wilcoxon signed-ranks test (see 2.7.2), between *G-REX original simplified* and C5.0.

Data sets	C5.0	G-REX	Difference	Rank
BLD	11.0	4.6	-6.4	12.5
Cleve	4.6	4.0	-0.6	3.5
CMC	34.1	7.1	-27.0	17
Crx	5.2	4.2	-1.0	5
German	15.4	4.3	-11.1	15
Glass	8.4	4.0	-4.4	11
Iono	6.7	3.2	-3.5	10
Iris	2.6	3.0	0.4	1.5
Labor	1.7	3.3	1.6	6
Lymph	6.5	3.2	-3.3	9
PID	10.4	4.0	-6.4	12.5
Sonar	7.8	5.4	-2.4	7,5
TAE	1.8	9.0	7.2	14
Tic-Tac-Toe	16.7	5.2	-11.5	16
WBC	5.7	3.3	-2.4	7,5
Wine	3.7	4.3	0.6	3.5
Zoo	6.1	5.7	-0.4	1.5

Table 17: Wilcoxon signed-ranks test between G-REX and C5.0. Size.

Here $R^{+}=25$ and $R^{-}=128$. The test statistic $T=min\{R^{+}, R^{-}\}=25$. Comparing this to the critical value 35 (two-tailed test using *17* data sets and $\alpha=0.05$) the difference is statistically significant.

Table 18 below shows the results when measuring size as number of tests.

Data sets	C5	CART	G-REX original		G-REX all		G-REX oracle	
			Raw	Simplified	Raw	Simplified	Raw	Simplified
BLD	11.0	21.5	9.7	7.8	8.3	7.8	3.0	2.9
Cleve	9.0	18.4	7.6	7.3	10.5	9.5	3.2	3.2
CMC	40.9	21.0	14.7	11.1	14.3	10.7	10.4	8.1
Crx	9.4	26.8	7.7	6.6	7.1	5.8	3.6	3.5
German	25.3	28.9	7.3	7.1	12.2	10.5	5.9	5.2
Glass	8.4	21.0	6.4	5.8	8.7	7.4	3.0	3.0
Iono	6.7	8.0	5.9	5.2	6.4	5.7	2.7	2.7
Iris	2.6	4.9	3.4	3.2	3.3	3.1	2.0	2.0
Labor	3.3	5.4	5.7	5.4	5.1	5.0	1.1	1.1
Lymph	8.9	13.9	6.7	6.3	8.1	7.3	2.2	2.2
PID	10.4	18.6	6.2	5.0	6.7	5.7	3.3	3.3
Sonar	7.8	12.0	7.9	7.2	8.5	7.7	2.9	2.9
TAE	11.3	57.3	13.0	12.4	10.7	10.4	4.1	4.1
Tic-Tac-Toe	33.4	32.8	14.6	13.5	13.2	12.4	6.2	6.1
WBC	5.7	7.2	6.9	6.4	7.0	6.5	2.6	2.6
Wine	3.7	5.9	6.3	5.3	5.0	4.4	2.4	2.3
Zoo	6.2	8.1	9.1	7.5	7.1	6.4	3.1	3.1
Mean:	**12.0**	**18.3**	**8.2**	**7.2**	**8.4**	**7.4**	**3.6**	**3.4**

Table 18: Complexity measured as number of tests.

The overall picture is almost identical; G-REX in general produces more compact trees than both C5.0 and CART. For illustration, Figure 30 below shows a G-REX rule from the BLD data set having *complex size* 7, which is close to the average size over all data sets.

```
if X2 > 65.5
 |T: if X6 > 1.99 Or X2 < 68.4 And X6 < 16.1
 |    |T: if X2 < 39.8 Or X2 < 68.4
 |    |    |T: [0] {4/1}
 |    |    |F: [1] {7/2}
 |    |F: if X2 < 76.3
 |    |    |T: [1] {2/1}
 |    |    |F: [0] {4/1}
 |F: [1] {9/4}
```

Figure 30: Sample G-REX tree with complex size 7.

If this data is used to perform another pair-wise comparison between *G-REX original simplified* and C5.0, the sums in the Wilcoxon signed-ranks test become $R^{+}=29.5$ and $R^{-}=123.5$. R^{+} is again smaller than the critical value 35, thus making the difference statistically significant.

Conclusions

The main result of this experiment was that G-REX produces rather compact (and thus comprehensible) rules. It was also shown that the extracted trees were significantly smaller than corresponding trees induced by C5.0. In this experiment, a large majority of all trees extracted by G-REX have *complex sizes* smaller than 10; i.e. the tree contains fewer than 10 tests altogether.

For a data miner, the fact that trees extracted using only oracle data are both extremely compact, and very accurate, should be an interesting observation. With this approach, it is clearly feasible to obtain a both accurate and comprehensible model of production instances. Whether the transparent model should be used for actual prediction or only as an explanation facility, must be determined for each specific project.

6.5.4 Experiment 4 – comparing rule extraction algorithms

The purpose of this experiment is to compare G-REX to the two well-know rule extraction algorithms RX and Trepan.

Method

The Trepan version used in this experiment is based on the publicly available MatLab implementation produced at Centre for Molecular Design, University of Portsmouth; see [HWF+03]. To enable a fair comparison with G-REX, the implementation was slightly modified. More specifically, the modified version makes it possible to extract rules from arbitrary models; e.g. ANN ensembles.

Although the Portsmouth implementation (just like Craven's) store the ANN on a unit and link level, Trepan uses the ANN as a black-box oracle; i.e. the inner workings are irrelevant. Because of this, the only requirement is, in fact, that the opaque model must be implemented as a MatLab function. Using the modified version, it is straightforward to extract rules from, for instance, an ANN ensemble; whenever Trepan needs an oracle prediction, this is produced by the ANN ensemble. Unfortunately, the implementation does not support multiclass problems, so the comparison with G-REX is restricted to binary problems.

The most important Trepan parameters, and the values used in this experiment, are given in Table 19 below. For an exact description of the Trepan implementation from University of Portsmouth, see the documentation accompanying the MatLab scripts[1]. All values used are the default values suggested in the documentation. The parameter *max_tree_size* determines the maximum number of nodes in the tree. *max_mofn_n* is the maximum number of binary splits in an *m-of-n* rule. With the value used here, the *m-of-n* rules are equivalent to logical ANDs and ORs. *min_sample_size* is the minimum number of generated instances that must reach each node. The *beam_width* is the number of steps used in the beam search when searching for optimal splits. Nodes reached by less than *min_objects* training instances are not expanded. *max_splits* determines how many of the possible splits that are considered at each node. More specifically, the candidate splits at each node are ordered using information gain, and only the top *max_splits* are used during beam search. Finally, *mofn_alpha* is the significance level used when comparing *m-of-n* tests during beam search, and *dist_alpha* is the significance level used when comparing distributions to see if they differ.

Parameter	Value
max_tree_size	21
max_mofn_n	2
min_sample_size	1000
beam_width	2
min_objects	20
max_splits	100
mofn_alpha	0.05
dist_alpha	0.1

Table 19: Trepan parameters.

[1] http://www.cmd.port.ac.uk/cmd/software.shtml

The RX algorithm was implemented in MatLab, based on the description given in 4.3.1. Most of the relevant parameters actually concern the training of the ANN and the pruning before RX. It should be noted that ANN training and pruning is carried out simultaneously. More specifically, a number of ANNs are first trained, and the single net having highest accuracy on training and validation data is selected. This net is now pruned by removing links with small weights. After that, units with no remaining links are also removed. It is made sure that the performance of the pruned net is not severely affected by the pruning. Finally, a new group of ANNs, with architecture based on the resulting net, are trained. This procedure is repeated until the resulting ANN is small enough.

Five parameters control how training and pruning are performed. First of all, *maxInput* is the highest acceptable number of remaining inputs after the pruning; i.e. it determines when the pruning is aborted. *lossAcc* determines how much accuracy it is acceptable to sacrifice during each pruning step. If *lossAcc* is set to 0.98, the accuracy after pruning must be no lower than 98% of the accuracy obtained by the current ANN. 1. So, if *lossAcc* is 0.98 and the current ANN has an accuracy of 0.8, the pruned ANN must have an accuracy of at least 0.784. Finally, the three parameters *noOfAnns*, *hidden*, and *hiddenInc* determine the number of ANNs and their number of hidden units. The training-pruning phase is described in more detail below:

1. The initial number of hidden units is determined as $hidden = \sqrt{v \cdot c}$ where *v* is the number of input variables and *c* is the number of classes.

2. A pool of ANNs are trained. The number of ANNs and the exact number of hidden units are determined by *nrOfAnns* and *hiddenInc*. If, as an example, *noOfAnns*=4 and *hiddenInc*=3, the pool would consist of 12 ANNs; four with *hidden+1* hidden units, four with *hidden*+*2* hidden units and four with *hidden*+*3* hidden units.

3. All ANNs in the pool are evaluated on the training and validation data. The single most accurate ANN is selected. This ANN is called the *current ANN*.

4. The current ANN is pruned. The pruning tries to set individual weights to zero without affecting the behavior of the ANN. Inactive links and units having no active links are removed. It should be noted that the ANN is pruned as much as possible, but the resulting accuracy must not be worse than *lossAcc*% of the accuracy obtained by the current ANN.

5. If the number of (active) inputs is not larger than *maxInputs*, pruning is terminated and the resulting pruned net is used by RX. Otherwise, the architecture of the pruned net is used as a basis for the next iteration. More specifically, all pruned inputs are no longer considered, and *hidden* is set to the number of hidden units in the pruned net.

Steps 2-5 are repeated until a sufficiently pruned ANN is returned.

For the actual rule extraction, the RX algorithm, as described in 4.3.1, is used. Here, the final parameter *rxAcc* determines the minimum accuracy required by the ANN using discretized activation values. If, as an example, *rxAcc*=0.99, the resulting accuracy must be at least 99% of the accuracy obtained by the pruned ANN, measured on training and validation data. The default parameter values used for RX in the experimentation are given in Table 20 below. Using these parameter values, RX did produce rule sets from all data sets except Tic-Tac-Toe, where the pruning was unsuccessful. Extensive experimentation, gradually lowering *lossAcc*, finally produced a rule set when *lossAcc* was set to 0.92.

Parameter	Value
max_input	10
noOfAnns	2
hiddenInc	3
lossAcc	0.98
rxAcc	0.99

Table 20: RX parameters.

G-REX, in this experiment, used exactly the same settings as in Experiment 1; i.e. no oracle data was used. G-REX and Trepan extracted rules from a common ensemble, while RX used the technique described above to obtain one ANN to extract from. For the actual evaluation, standard 10-fold cross validation was used. The sizes reported are number of splits for G-REX and Trepan, and number of rules for RX; i.e. the measure previously referred to as *simple size*.

Results

Table 21 below shows the results from the comparison between G-REX and Trepan. Regarding test set accuracy, G-REX outperforms Trepan on nine out of ten data sets. A Wilcoxon signed-ranks test shows that the difference in accuracy is statistically significant. In this experiment, both G-REX and Trepan produced very compact trees. As a matter of fact, all extracted trees must be considered comprehensible.

Data sets	Ensemble		G-REX					Trepan				
	Train Acc.	Test Acc.	Train Fid.	Train Acc.	Test Fid.	Test Acc.	Size	Train Fid.	Train Acc.	Test Fid.	Test Acc.	Size
BLD	0.938	0.706	0.763	0.716	0.756	**0.662**	4.5	0.752	0.706	0.762	0.656	8.6
Cleve	0.999	0.812	0.866	0.866	0.803	**0.787**	6.5	0.830	0.829	0.830	0.760	10.2
Crx	0.986	0.843	0.892	0.879	0.923	**0.859**	5.6	0.882	0.869	0.929	0.848	6.6
German	0.996	0.731	0.749	0.746	0.825	**0.726**	3.4	0.736	0.732	0.839	0.718	9.4
Iono	1.000	0.926	0.940	0.940	0.909	**0.914**	4.3	0.867	0.867	0.860	0.843	9.0
Labor	1.000	0.920	0.992	0.992	0.880	**0.880**	3.1	0.733	0.733	0.840	0.760	2.6
PID	0.920	0.745	0.837	0.770	0.845	0.742	5.5	0.841	0.776	0.868	**0.750**	9.7
Sonar	1.000	0.805	0.806	0.806	0.775	**0.750**	5.0	0.711	0.711	0.675	0.670	4.1
Tic-Tac-Toe	0.993	0.910	0.846	0.842	0.872	**0.834**	5.4	0.832	0.829	0.802	0.794	6.1
WBC	0.999	0.965	0.978	0.977	0.970	**0.955**	5.2	0.959	0.958	0.968	0.954	7.2

Table 21: Results G-REX and Trepan.

Table 22 below shows the results for the comparison between G-REX and RX. The G-REX results reported here are, for binary problems, taken from Table 21. Similarly, G-REX results on the multiclass problems are taken from Table 11.

G-REX obtains higher test set accuracy than RX on 13 of 17 data sets. Another Wilcoxon signed-ranks test shows that the difference is statistically significant. One reason for this is that RX extracts rules from one ANN while G-REX uses an ensemble. As seen in Table 22, the ensemble is most often considerably more accurate than the ANN used by RX.

Data sets	RX ANN		Ensemble		G-REX					RX				
	Train Acc.	Test Acc.	Train Acc.	Test Acc.	Train Fid.	Train Acc.	Test Fid.	Test Acc.	Size	Train Fid.	Train Acc.	Test Fid.	Test Acc.	Size
BLD	0.723	0.653	0.938	0.706	0.763	0.716	0.756	**0.662**	4.5	0.978	0.726	0.982	0.647	6.3
Cleve	0.894	0.820	0.999	0.812	0.866	0.866	0.803	**0.787**	6.5	0.982	0.892	0.953	0.780	8.5
CMC	0.562	0.526	0.717	0.550	0.741	0.561	0.770	**0.565**	7.1	0.934	0.560	0.920	0.526	8.2
Crx	0.868	0.848	0.986	0.843	0.892	0.879	0.923	**0.859**	5.6	0.991	0.867	0.991	0.846	3.1
German	0.768	0.719	0.996	0.731	0.749	0.746	0.825	**0.726**	3.4	0.972	0.763	0.966	0.715	12.1
Glass	0.708	0.629	0.971	0.690	0.701	0.686	0.729	**0.643**	4.0	0.984	0.705	0.967	0.624	10.8
Iono	0.943	0.877	0.926	0.940	0.940	0.940	0.909	**0.914**	4.3	0.987	0.940	0.957	0.863	9.3
Iris	0.973	0.953	1.000	0.973	0.979	0.979	0.980	**0.967**	3.0	0.997	0.972	0.973	0.940	3.1
Labor	0.988	0.840	0.920	0.992	0.992	0.992	0.880	**0.880**	3.1	0.994	0.990	0.940	0.860	3.0
Lymph	0.887	0.786	1.000	0.850	0.855	0.855	0.793	0.786	3.2	0.978	0.883	0.936	**0.793**	5.5
PID	0.787	0.767	0.745	0.837	0.837	0.770	0.845	0.742	5.5	0.982	0.787	0.970	**0.763**	5.4
Sonar	0.918	0.765	0.805	0.806	0.806	0.806	0.775	**0.750**	5.0	0.985	0.913	0.915	**0.750**	13.4
TAE	0.579	0.373	0.790	0.573	0.782	0.616	0.727	**0.500**	9.0	0.981	0.581	0.953	0.387	4.5
TicTacToe	0.866	0.832	0.910	0.846	0.846	0.842	0.872	0.834	5.4	0.968	0.862	0.954	**0.842**	21.4
WBC	0.967	0.952	0.965	0.978	0.978	0.977	0.970	**0.955**	5.2	0.993	0.964	0.984	0.951	4.1
Wine	0.976	0.947	1.000	0.988	0.968	0.968	0.947	**0.947**	4.3	0.992	0.975	0.959	0.929	4.7
Zoo	0.955	0.910	1.000	0.970	0.926	0.926	0.940	**0.920**	5.7	0.974	0.947	0.940	0.900	7.0

Table 22: Results G-REX and RX.

At the same time, it is interesting to observe that RX, in general, has much higher fidelity, on both training and test sets. Regarding comprehensibility, most RX rule sets are quite small. On some individual folds, however, the number of rules can be 30 or more. Sometimes, these rules are also rather complex. Overall, however, the difference in size between G-REX and RX rule sets must be considered unimportant.

Conclusions

The main result from this experiment is that G-REX was significantly more accurate than both Trepan and RX. Another important result was that all three algorithms, with the parameter settings used, produced compact and therefore comprehensible rule sets.

When comparing G-REX and Trepan, both algorithms used a common ensemble to rule extract from, so the difference in performance must be credited to the algorithms used, and possibly the parameter settings. Having said that, it is important to be aware of that the Trepan implementation used was not the original one, and that all parameters were set to their default values.

The comparison between RX and G-REX is slightly harder. From the very high fidelity, it is obvious that RX succeeds remarkably well in explaining the underlying ANN. Despite this, the models extracted by G-REX turned out to be significantly more accurate when applied to a test set. Finally, it should be mentioned that the RX implementation used sometimes had quite severe scaling problems. Individual folds, handled in minutes by G-REX, could take several hours for RX.

6.5.5 Experiment 5 – consistency

The purpose of this experiment is to evaluate the consistency of G-REX. More specifically; the algorithm is run several times to extract a number of trees. The trees are then compared in order to determine whether they are similar or not. Part of this experiment was reported in [JKN07].

In this experiment, similarity is measured only based on the predictions made by different extracted trees. This means that the meaning of the rules is ignored. The actual evaluation is divided in two parts; *intra-model* and *inter-model*. Intra-model compares trees extracted from a common opaque model, while inter-model compares trees extracted from different opaque models.

The measure used is the proportion of data instances classified identically by the two extracted representations. Note that this measure is blind to whether the rules are correct or not; it is only concerned with if the rules make identical predictions.

Method

In this experiment, nine data sets of the available 17 were used. For each data set, the available data was randomly divided in four equal-sized parts. During experimentation, each part was used as test set once; i.e. the effective size of the test set was always 25%, while the remaining 75% was used for training. The ensembles used to rule extract from were created identically to Experiment 1. The parameters for G-REX were also the same as in Experiment 1. No oracle data was used.

When evaluating intra-model consistency, G-REX was run five times, each time extracting from the same model. When evaluating inter-model consistency, a new ensemble was trained every time, before G-REX started to extract rules. In both cases, each of the five rule sets found was then used on the test set; resulting in five different predictions for each test set instance. The five rule sets were then pair-wise compared to determine the proportion of all data points identically classified by the two rule sets. Separate scores were kept for training and test sets. Finally, this was repeated for each of the four folds on every data set.

Results

Table 23 below shows the intra-model results for *one fold* from the Zoo problem. In this fold, the average consistency is *0.910* for the training set and *0.892* for the test set. The interpretation is that a pair of extracted trees, on average, agree on 91.0% of all training set instances, and on 89.2% of all test instances.

TRAIN	R2	R3	R4	R5	**TEST**	R2	R3	R4	R5
R1	0.920	0.920	0.960	0.907	R1	0.880	0.880	0.960	0.880
R2		0.907	0.880	0.920	R2		0.920	0.880	0.880
R3			0.893	0.907	R3			0.880	0.880
R4				0.880	R4				0.880

Table 23: Intra-model consistency. One fold Zoo problem.

Similarly, Table 24 shows the inter-model results for one PID fold. Here the average consistency is *0.910* for the training set and *0.874* for the test set.

TRAIN	R2	R3	R4	R5	**TEST**	R2	R3	R4	R5
R1	0.899	0.891	0.905	0.908	R1	0.885	0.859	0.880	0.901
R2		0.943	0.922	0.915	R2		0.901	0.911	0.870
R3			0.917	0.896	R3			0.854	0.844
R4				0.903	R4				0.833

Table 24: Inter-model consistency. One fold PID problem.

Table 25 below shows the overall results. On average, two extracted rule sets agree on approximately 85% of all instances. Somewhat surprising, this holds for both training and test sets, as well as intra-model and inter-model.

Data sets	**Intra-model**		**Inter-model**	
	TRAIN	**TEST**	**TRAIN**	**TEST**
BLD	0.805	0.751	0.829	0.793
Cleve	0.893	0.845	0.912	0.886
CMC	0.863	0.855	0.853	0.849
Glass	0.822	0.797	0.789	0.755
PID	0.908	0.893	0.904	0.881
TAE	0.809	0.788	0.681	0.653
Tic-Tac-Toe	0.854	0.840	0.876	0.876
WBC	0.986	0.971	0.987	0.967
Zoo	0.897	0.903	0.911	0.858
Mean:	**0.868**	**0.850**	**0.860**	**0.835**

Table 25: Average consistency over all pairs and all folds for each data set.

Conclusions

Consistency is a hard criterion to assess. Although it must be considered important to have at least fairly high consistency, it is not obvious exactly what level this translates to. As comparison; C5.0 and CART will produce identical trees every time, so their consistency would be 100%. With this in mind, the level obtained by G-REX clearly raises some questions. As an example, on some data sets, more than 20% of all instances are, on average, classified differently by two G-REX runs. Is it really possible to put faith in one specific extracted rule when another run might produce a different rule, disagreeing on 20% of all instances? The answer to this question is not obvious. Intuitively, most data miners would probably want *one* accurate and comprehensible model. On the other hand, most individual G-REX runs do produce exactly that; i.e. a compact yet accurate tree. As seen in the previous experiments, *any* single G-REX tree is most often both more compact and more accurate compared to a tree induced by C5.0 or CART. So maybe, the apparently low consistency should not be considered a major problem in practice. In addition, it could be argued that if several trees are extracted by G-REX, they each provide some information; i.e. they describe the relationship between input and output slightly differently. This could potentially be used by a data miner.

Finally, since the difference between intra-rule and inter-rule is rather small it is primarily the evolutionary approach that leads to the rather low consistency. This is also pointed out in [HBV06] as the most serious inherent drawback of rule extraction algorithms based on evolutionary algorithms.

Chapter 7

Impact of advertising case study

The purpose of this chapter is to report several empirical studies conducted, all related to the problem called *Impact of Advertising*. Very briefly, the Impact of Advertising is a predictive modeling problem, where measurable effects of advertising are predicted based on commercial investments in different media. It should be noted that this problem was extensively investigated in several studies during a rather long period of time; i.e. several years. Consequently, some design choices, especially in the earlier studies may, in hindsight, appear rather naïve. The entire project was carried out in co-operation with Nordisk Media Analys AB (NMA). Initially, the sole purpose was to build models as accurate as possible, but after a while the focus shifted to also recognize the importance of comprehensibility. Although the studies presented in this chapter do not directly address the main issues in the thesis statement, work carried out on this case study was the driving force behind initial development of G-REX. In addition, it is likely that some problems encountered along the way are of general interest, and hence identified considerations and proposed solutions could be useful for the data mining and marketing communities.

The organization of the chapter follows the chronological order of the studies, all reported in various papers. More specifically, each section presents one study, with focus on purpose, method, results and conclusions. Important considerations and requirements that were identified, either during the study or afterwards, are described and discussed, at the end of each section. The first paragraph of each section summarizes the section, briefly describing the purpose of the study, the method and the main results.

7.1 High accuracy prediction in the marketing domain

This section is based on the paper "Predicting the Impact of Advertising: a Neural Network Approach" [JN01].

7.1.1 Summary

The purpose of the study was to evaluate if ANNs, by using the temporal structure of the domain, can improve accuracy when predicting the outcome of investments in advertising, compared to the traditional methods used. The focus was to investigate if commercial impact can be predicted from historical outcome and planned future media investments. The domain was car sales in Sweden.

The study was comparative between both different ANN architectures and traditional approaches based on multiple linear regression. The main result was to conclusively show that sequential and recurrent approaches exploit the time series dependencies and yield a performance superior to traditional approaches.

7.1.2 Background

The ability to predict effects of investments in advertising is important for all companies using advertising to attract customers. Ideally, a function mapping invested money to customers' desire to buy a product should exist. Intuitively though, other important factors affect the decision what make of car to actually buy. However, some other effects, typically measuring how advertising is *received* by the public, may be predictable.

In the media analysis domain, the focus has traditionally been to, most often in retrospect, explain effects of marketing investments. The methods used are often based on linear models and have low predictive power. The underlying domain of the time series and, especially the somewhat weak connection between input and output, make predicting hard. It is also obvious that many more factors than money invested affect the impact of advertising. Examples of this include that most people are biased towards the brand that they currently own a car from, and that Swedish and German cars have a very good reputation in Sweden. Another aspect is that a change of marketing strategy (e.g. using a new marketing company or changing the media mix) would significantly affect the outcome, making it very hard to build predictive models from historical data.

However, it is also important to identify differences between expected outcome and actual outcome. In cases where there is a substantial difference, efforts have to be made to identify the cause. This is the reason why it is important to generate models that show good predictive performance on typical data. It is thus assumed that historical data for a brand contains information about its individual situation (e.g., how well its marketing campaigns are perceived) and that this could be used to build a predictive model.

The domain

Every week a number of individuals are interviewed (by NMA) to find out if they have seen and remember adverts in different areas (here the focus is on car adverts). From these interviews the following percentages are produced for each brand:

- Top Of Mind (TOM). The make is the first mentioned by the respondent.
- In Mind (IM). The respondent mentions the make.
- Preference (PREF). The make is the preferred choice; i.e. the respondent's favorite car.
- Possibility (POSS). The make is a possible choice of purchase for the respondent.

This first study focused on TOM and IM. The hypothesis was that TOM and IM are easier to predict since they mainly rely on the short-term effects of advertising, not the long-term brand building effects. Nevertheless, this problem is far from trivial. As an example, Figure 31 shows the relationship between total investments and TOM over 100 weeks for Volvo. Note that the percentages have been scaled to hide the actual values. The relationship is clearly not linear and there is no obvious pattern. The correlation (*r-value*) is as small as 0.22.

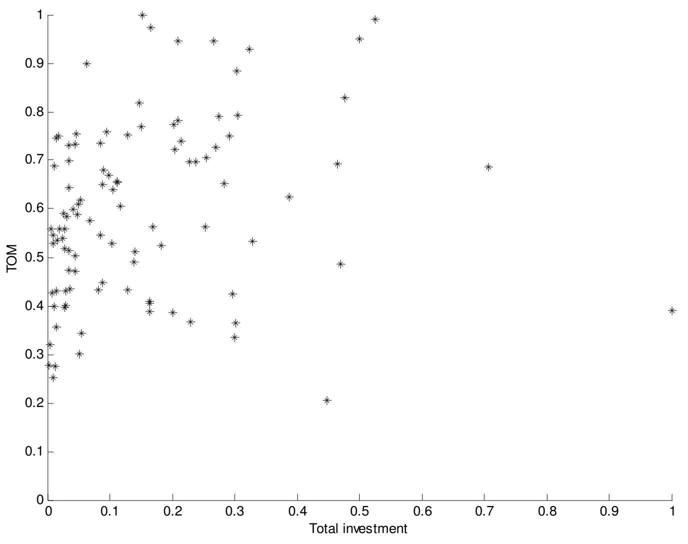


Figure 31: TOM against total investment for Volvo.

In Figure 32 and Figure 33 the same relationship is displayed over a period of 100 weeks. These figures show that a large investment will produce a higher TOM, but that the effect is not immediate (see for instance the large investment in week 12 and its effect in week 15). This suggests that there is a temporal aspect of the problem and, consequently, that previous investments must be included in a predictive model.

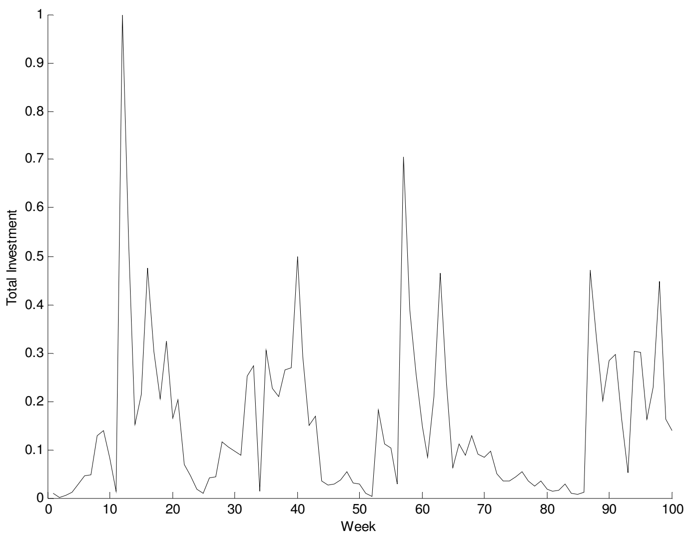


Figure 32: Total investment over 100 weeks.

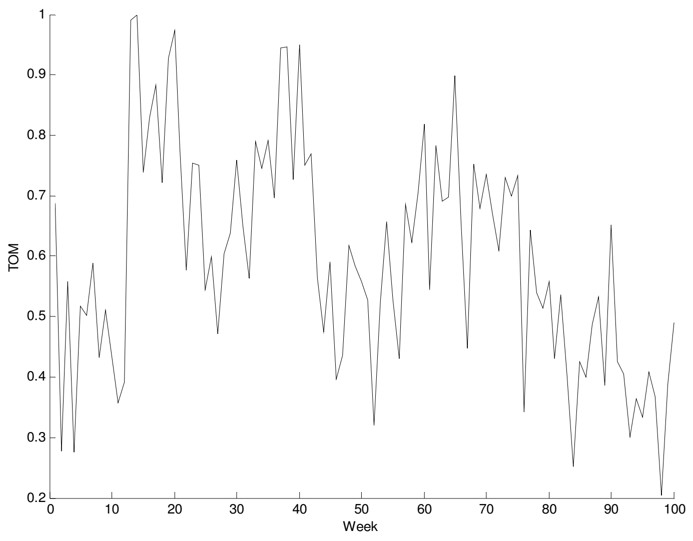


Figure 33: The TOM as a result of the investments.

The overall purpose was to supply companies with useful information about the outcome of their planned media investment strategy. This predictive modeling task is normally divided into two sub-tasks: a monthly prediction (with updates every week) and a longtime forecast covering approximately one year.

7.1.3 The data set

The data set contains 157 weeks of information (i.e. investments and effects) from the car industry on the Swedish market. As a first step, all data was preprocessed using moving averages over periods of four weeks, see equation (75).

$$x_n = \frac{x_n + x_{n-1} + x_{n-2} + x_{n-3}}{4} \tag{75}$$

The reason for using the moving average was to define a smoothing function on the data, since the effect in one week depends not only on the investment in that particular week. Obviously the investment is likely to have a longer duration than one week. The data was also normalized by dividing each value with 3 times the max value (in the training set) for that particular variable. The motivation for this was to allow generalization outside the borders of the training data (i.e. to extrapolate up to an output activation of 1.0), which often is necessary in real-world problems. In the marketing domain the trend is that companies spend more and more money on advertising.

The data set includes altogether 37 makes, covering the entire Swedish market. Some makes had to be removed due to lack of data or the fact that their investments and effects were so small that forecasting was deemed impossible. Of the remaining 17 makes, the following nine makes were randomly chosen for this initial study: Audi, Citroën, Ford, Hyundai, Mitsubishi, Opel, Toyota, Volvo and Volkswagen.

The input consisted of investments in different media categories; *TV*, *radio*, *cinema*, *morning press*, *evening press*, *popular press*, *special interest press* and *outdoor*. A parameter called *share of voice* (SoV) was also used, but only for short-term forecasts. SoV is defined as the total investment for a specific make, divided by the total investment for all 37 makes; see equation (76).

$$SoV_{Make_i} = \frac{Total\ Investment\ Make_i}{\sum_j Total\ Investment\ Make_j} \tag{76}$$

The motivation for including SoV, in short-term forecasts only, is that it is reasonable to assume that a certain company has information about other companies marketing investments, but only on a very short horizon.

7.1.4 Linear models

A multiple linear regression analysis was performed. The results are given in Table 26. R^2 is the multiple coefficient of determination, i.e. the proportion of the variance in the dependent variable that is explained by the combination of the independent variables in the multiple regression model.

Company	**TOM**	**IM**	**POSS**	**PREF**
Audi	0.43	0.30	0.00	0.02
Citroën	0.69	0.52	0.19	0.12
Ford	0.10	0.11	0.01	0.04
Hyundai	0.11	0.08	0.00	0.08
Mitsubishi	0.11	0.16	0.04	0.03
Opel	0.03	0.12	0.04	0.03
Toyota	0.27	0.32	0.08	0.22
Volvo	0.25	0.18	0.05	0.01
Volkswagen	0.25	0.14	0.04	0.06
Mean:	**0.25**	**0.21**	**0.05**	**0.07**

Table 26: R^2 values for multiple linear regression weeks 1-100.

From Table 26, it is obvious that the variability in TOM and IM is not very well explained using linear models. For POSS and PREF this is even more apparent.

The measurement used when comparing predictions from the different methods is also the standard coefficient of determination (R^2), but here between predicted values and actual values. The reason for using this measurement is the fact that it is scale-independent and also often used when determining the performance of the traditional methods. For future comparison the linear models created using data from week 1-100, were also used to produce long-term forecasts of TOM and IM. Table 27 shows the R^2 values between the forecast and the correct values for weeks 101-157.

Company	TOM	IM
Audi	0.18	0.25
Citroën	0.36	0.26
Ford	0.26	0.30
Hyundai	0.09	0.16
Mitsubishi	0.45	0.22
Opel	0.61	0.59
Toyota	0.26	0.61
Volvo	0.26	0.28
Volkswagen	0.03	0.14
Mean:	**0.28**	**0.31**

Table 27: Long-term forecast using multiple linear regression.

Multiple linear regression was also used to produce models for weekly forecasting, where the models are updated every week. Table 28 shows results for predicting week 101-157.

Company	TOM	IM
Audi	0.30	0.44
Citroën	0.32	0.22
Ford	0.34	0.43
Hyundai	0.17	0.14
Mitsubishi	0.52	0.23
Opel	0.76	0.58
Toyota	0.51	0.69
Volvo	0.35	0.40
Volkswagen	0.19	0.33
Mean:	**0.38**	**0.38**

Table 28: Short-term forecast using multiple linear regression.

Many traditional forecasting methods in use are based on multiple linear regression, making comparison between these models and ANNs applied to the same task interesting, see e.g. [UR99].

7.1.5 Architectures

Two kinds of ANNs were used, Simple Recurrent Nets (SRN) [Elm90] and TDNNs. Recurrent architectures with their inherent short-term memory are well suited to prediction tasks; see e.g. [Yas99]. Figure 34 shows the SRN architecture for this problem.

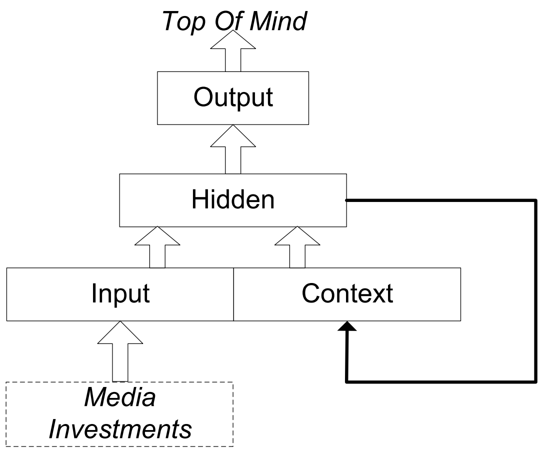


Figure 34: SRN with input and output (in italics) for this problem.

As noted in, for instance [Dor96], the technique with tapped delay lines can be seen as an extension of auto-regressive time series modeling where the output is assumed to be a linear combination of a fixed number of previous series values. ANNs on the other hand, will typically produce non-linear models.

Different sizes of the tapped buffers and number of hidden neurons were tested. For each network configuration, five independent ANNs were trained, all starting from different sets of initial random weights. The actual prediction is the average prediction from these five ANNs. The standard technique to combine several ANNs into an ensemble is very robust, resulting in models likely to generalize well; see e.g. [Bis95].

For long-term forecasts, the ANNs were trained using weeks 1-100 and tested on weeks 101-157. For weekly predictions, the ANNs were trained using weeks *1-n* when predicting week *n+1*. This was repeated for *n* between 100 and 153, and for horizons between 1 and 4 weeks. The training algorithm was backpropagation with momentum and variable learning rate. The standard techniques of early stopping and automated regularization using a Bayesian framework [McK92], were employed to improve generalization.

To establish the fact that the temporal aspect of the problem must be utilized for increased performance, comparison with standard feed-forward networks was also conducted. Table 29 shows the results for feed-forward nets with 10 hidden units.

Company	**TOM**	**IM**
Audi	0.04	0.17
Citroën	0.33	0.17
Ford	0.16	0.24
Hyundai	0.11	0.15
Mitsubishi	0.42	0.19
Opel	0.15	0.01
Toyota	0.32	0.60
Volvo	0.35	0.39
Volkswagen	0.11	0.23
Mean:	**0.22**	**0.24**

Table 29: R^2 values for long-term forecast using a feed-forward net.

As seen in Table 29, for this specific task, a multi-layer net (potentially exploiting any non-linearity) turned out to be a less accurate model than the linear regression model.

7.1.6 Results

Multiple linear regression analysis showed that the variability in TOM and IM is not very well explained by a linear model. This analysis also showed that PREF and POSS are impossible to accurately predict with linear models.

Long-term predictions

The overall picture is that TOM and IM are predictable to some degree, and that ANNs using a temporal structure is a good tool for this prediction. Typical R^2 values for TOM and IM, on the test set, are around 0.4. Figure 35 shows a sample long-term prediction. The test set is week 101-157.

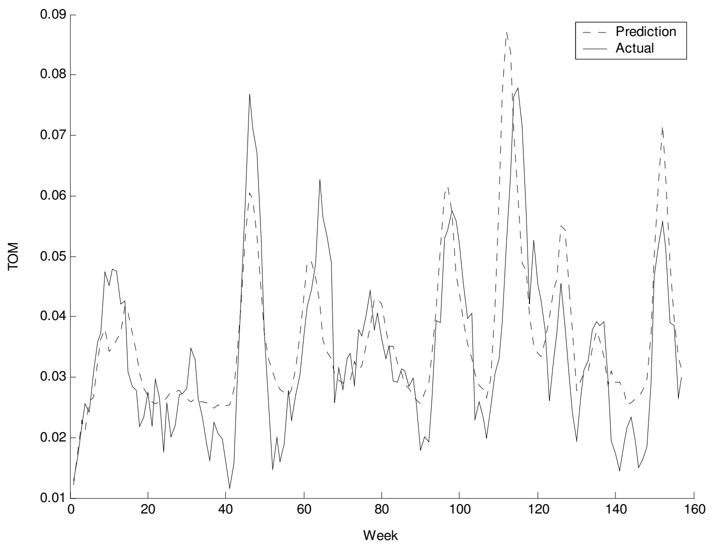


Figure 35: SRN long-term prediction of Ford TOM.

Table 30 shows the results for the test set (week 101-157) for the two different architectures; SRN with 15 hidden units and TDNN with 10 hidden units and 5 tapped inputs. Since the parameter SoV would not be accessible during the test period, it is not part of the model.

Company	**SRN**		**TDNN**	
	TOM	**IM**	**TOM**	**IM**
Audi	0.15	0.22	0.19	0.26
Citroën	0.47	0.27	0.37	0.14
Ford	0.55	0.39	0.68	0.39
Hyundai	0.46	0.36	0.44	0.29
Mitsubishi	0.46	0.28	0.40	0.27
Opel	0.55	0.40	0.38	0.52
Toyota	0.40	0.72	0.53	0.51
Volvo	0.51	0.58	0.51	0.56
Volkswagen	0.11	0.49	0.22	0.36
Mean:	**0.41**	**0.41**	**0.41**	**0.36**

Table 30: R^2 values for long-term forecast.

Corresponding R^2 results for POSS and PREF are typically smaller than 0.1 for long-term forecasting.

Short-term Predictions

For weekly predictions, the R^2 values are naturally much higher. A main result from this first study is that it is possible to rather accurately predict expected TOM and IM in the next month, using previous outcome and an investment plan. Figure 36 shows a sample short-term prediction (one week ahead) for Volvo. Note that the figure shows only the 57 test weeks.

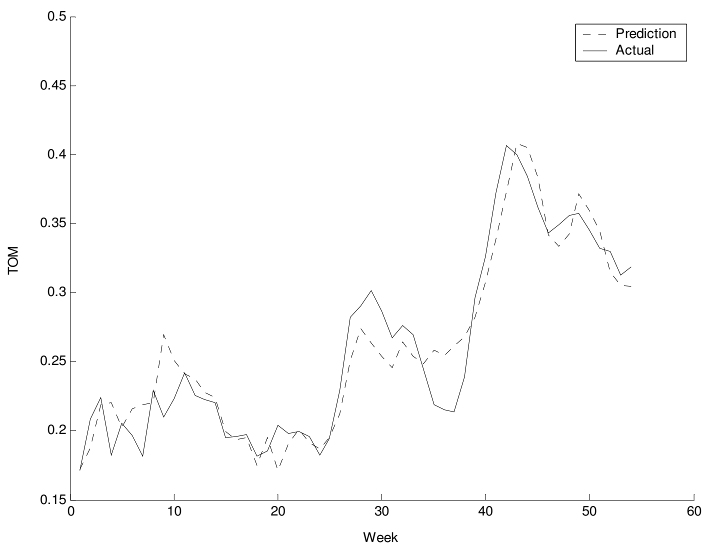


Figure 36: TDNN short-term prediction of Volvo TOM.

Only TDNN results are presented in Table 31 below, since TDNNs for this task clearly outperformed SRNs. The TDNNs used had 15 hidden units and 10 tapped inputs.

Target	**TOM**				**IM**			
Horizon	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
Audi	0.66	0.67	0.67	0.58	0.63	0.68	0.63	0.64
Citroën	0.57	0.66	0.68	0.60	0.57	0.66	0.68	0.60
Ford	0.78	0.76	0.74	0.66	0.78	0.76	0.74	0.66
Hyundai	0.68	0.62	0.56	0.51	0.68	0.62	0.56	0.51
Mitsubishi	0.70	0.70	0.75	0.72	0.70	0.70	0.75	0.72
Opel	0.74	0.77	0.78	0.68	0.74	0.72	0.64	0.57
Toyota	0.64	0.66	0.62	0.59	0.65	0.66	0.64	0.59
Volvo	0.89	0.86	0.82	0.73	0.89	0.86	0.82	0.73
Volkswagen	0.68	0.73	0.71	0.58	0.64	0.59	0.54	0.38
Mean:	**0.70**	**0.71**	**0.70**	**0.63**	**0.70**	**0.69**	**0.67**	**0.60**

Table 31: R^2 values for forecast, one to four weeks ahead, using TDNNs.

7.1.7 Conclusions

The results show that ANNs can exploit the temporal structure of the media investment domain in order to increase predictive accuracy. Here, the higher accuracy, compared to linear models traditionally used, was more dependent on the use of the temporal structure than on the non-linearity in the models.

The long-term predictions were more accurate than the standard methods based on linear regression. Whether or not the accuracy would be deemed high enough to act upon, and exactly how the found model could be used, must be determined separately for each brand.

The results also show that high predictive accuracy can be achieved for short-term predictions (1-4 weeks), and that TDNNs, in this study, performed this task better than SRNs. The high performance, when predicting on a short horizon, is mostly due to the possibility to retrain networks every week, as new data becomes available, but the inclusion of the variable SoV was also important.

7.1.8 Considerations

The main purpose of the first study was to examine the usefulness of ANNs as predictors in the media domain. The performance was deemed good enough to continue the line of work. A couple of issues were identified, however:

- Some understanding or explanation of the model was desirable if a system based on ANNs should be used by decision-makers.
- The measurements used for the study were not necessarily suitable for the domain. More specifically; when predicting a media effect, it is not vital to pin-point the exact week. For media analysts it is much more important to predict total effect of an investment strategy.

It was also noted that the two tasks long-term and short-term predictions probably should be treated separately, although both are interesting for decision-makers. The purpose of a long-term forecast is strategic and a predictive model would probably mostly be used to compare different long-term media strategies. If an accurate, general and comprehensible model was found, that model would also be of great value as an explanation tool for how marketing efforts of a specific company are received. The purpose of a short-term forecast is more tactical; to actually predict an outcome based on an existing marketing strategy, or to alter a strategy slightly, based on recent information and knowledge about how competitors act. The main purpose of such model is therefore to function as a simulation tool to evaluate different media strategies. With this in mind, it may be more natural to represent a short-term forecast as a classification task. If so, predictions should be whether the impact is higher than a certain level or not. Here too, however, a comprehensible model could be used to understand the underlying relationships.

7.2 Increased performance and basic understanding

This section is based on the paper "Increased Performance with Neural Nets - An Example from the Marketing Domain" [JN02a].

7.2.1 Summary

One purpose of the second study was to look into different techniques for preprocessing. The ambition was to both increase predictive accuracy and to establish a fundamental understanding of the models. Another purpose was to try to find measurements closer to what is actually targeted by media analysts.

This study was comparative between different network architectures and different preprocessing techniques. The performance was also evaluated against the results from the first study. The same data sets were used, with the addition of several more car makes. In this study, the focus was solely on long-term forecasts.

One main result was the need to somehow change either the problem or the score function to get better resemblance with approaches taken by media analysts. Here, this problem was handled by using original data for training and then post-processing the prediction and the target. Another key result was the successful use of sensitivity analysis to determine the most important input categories. In addition, the discarding of non-important categories, as identified by the sensitivity analysis, made the model more robust and increased the accuracy. Obviously, the sensitivity analysis also provided some problem specific insights for the marketing analysts.

7.2.2 Background

In order to perform predictions of this nature, it was assumed that much experimentation would be needed. Central questions were for instance, "how fast is the effect of a marketing campaign?", "for how long does the effect of an advertisement last and how does that affect the effect of future advertisements?" and "how do competitors' advertisements interfere with each other?" Answers to questions like these will determine how predictive models (from money invested to impact) should be designed. The assumption here was not only that historical data can be used to predict the outcome of future media investments, but also that different media categories affect the outcome to different degrees and at different speeds. Finally, it was also assumed that data for a particular product reflects how it is affected by the continuous advertising by its competitors.

In the media analysis domain, the focus has traditionally been on explaining the effects of previous investments, i.e. to present an interpretable function. Since the methods are often based on linear models, they have shown low predictive power in the marketing domain; i.e. their performance on novel data is poor. This is actually an instance of the accuracy vs. comprehensibility trade-off. Linear models are used since they are simple and relatively easy to interpret. Accurate predictions from these models are, however, normally not possible. ANNs, on the other hand, would probably produce much more accurate predictions but suffer from the inability to present explanations.

7.2.3 Method

The focus of this study was on TOM and IM only. Several different setups, varying the number of hidden units and the learning algorithms were tried. The performance was not very sensitive to the exact parameters. For the experiments presented below, the SRNs had 25 hidden units and the TDNNs had 10 hidden units and five tapped weeks. The training algorithm was backpropagation with momentum and variable learning rate. For each make, five nets were trained and combined into an ensemble. The ensemble prediction was the average of the predictions from the ANNs. For the forecasting, the nets were trained using weeks 1-100 and tested on weeks 101-157.

In this study, an effort was made to include all makes with sufficient data available. The main reasons for the exclusion of as many as 20 makes were that for these makes, historical data was missing for certain periods, or that the makes had extremely small and spread investments, resulting in effects close to zero. The cumulative TOM for makes included is, in most weeks, well over 99.5%.

The coefficient of determination (R^2) was used to correlate the predicted value and the target value. This is, as mentioned before, not particularly suitable for the problem at hand, but was chosen since it is standard in the media analysis domain. As noted in the first study, the actual goal is rather to predict an average effect over a period than to isolate a certain level of effect to a specific week. This, together with the fact that investments are cumulative, led to the use of various moving averages. Two basic formats of input and output combinations of data were used for training; the *raw* data and the *moving average* (MA1) based on week *n* and its three preceding weeks; as used in the previous study, see equation (75).

Media analysts stress the importance of predicting the "total" outcome of media investments. It is not vital to pin-point the exact week when an effect takes place. As an example, in Figure 37 the overall prediction is pretty good; i.e. the total area predicted is close to the actual. However, the performance when measured as MSE or correlation coefficient is rather poor, since these measures only compare the prediction and the target on a week-by-week basis.

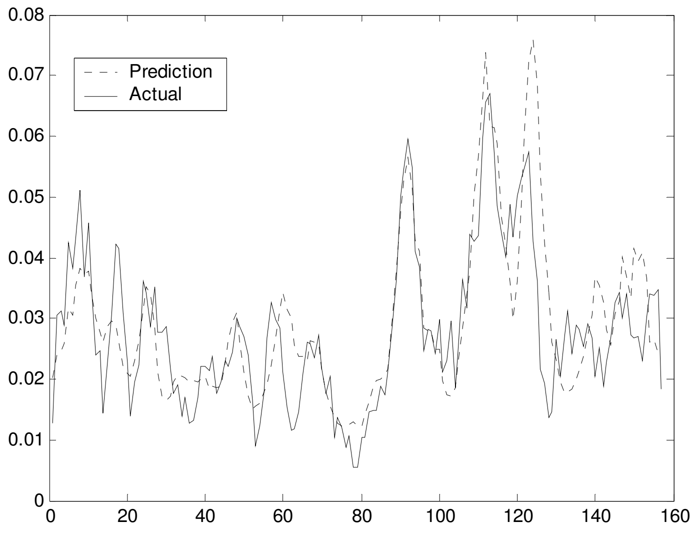


Figure 37: The problem with using R^2 as quality measure.

Since the ambition was to compare results with traditional methods and the previous study, R^2 was kept as measure. The problem with good "area predictions" having low R^2 values therefore had to be handled by a compromise. When the ANNs after training were used for predictions, the results (and the actual values of the targeted effect) were post-processed. More specifically, both the predictions and the desired values were averaged over a period of time. Here the period was five weeks; see equation (77). This average is called MA2. The use of an average like MA2 is also a standard technique applied by media analysts, thus facilitating comparison between methods.

$$Y_n = \frac{y_{n-2} + y_{n-1} + y_n + y_{n+1} + y_{n+2}}{5} \tag{77}$$

In the experiments, the predictions were evaluated both with (MA2) and without (No) the post-processing. It should be noted that it would be misleading to compare results (accuracies) obtained with and without post-processing since averaged values are normally easier to predict. The purpose of introducing MA2 thus was *not* to get higher R^2 values but, as mentioned above, to avoid the problem illustrated in Figure 37.

7.2.4 Results

As seen in Table 32, the use of the post-processing MA2 gives higher R^2 values. This should be expected, since it is based on a five weeks moving average. More importantly, this measurement is actually more informative, since it is closer to the "area" target emphasized by media analysts. The results presented in Table 32 also show that it, in this case, works just as well to use raw input and output, compared to using MA1 input and output, when post-processing with MA2.

Company	**raw-No**		**raw-MA2**		**MA1-No**		**MA1-MA2**	
	TOM	**IM**	**TOM**	**IM**	**TOM**	**IM**	**TOM**	**IM**
Audi	0.00	0.00	0.09	0.19	0.19	0.26	0.33	0.03
BMW	0.04	0.00	0.10	0.24	0.00	0.00	0.01	0.00
Citroen	0.00	0.00	0.00	0.11	0.37	0.14	0.34	0.13
Ford	0.30	0.28	0.73	0.65	0.68	0.39	0.82	0.74
Honda	0.00	0.00	0.00	0.10	0.00	0.00	0.00	0.00
Hyundai	0.13	0.06	0.53	0.29	0.44	0.29	0.55	0.18
Mazda	0.00	0.02	0.00	0.07	0.05	0.42	0.12	0.66
Mercedes	0.05	0.13	0.11	0.27	0.10	0.19	0.12	0.22
Mitsubishi	0.11	0.07	0.20	0.26	0.40	0.27	0.33	0.34
Nissan	0.20	0.18	0.19	0.42	0.14	0.03	0.10	0.05
Opel	0.08	0.23	0.33	0.52	0.38	0.52	0.17	0.37
Peugeot	0.25	0.14	0.51	0.37	0.27	0.19	0.26	0.27
Renault	0.00	0.00	0.01	0.14	0.01	0.20	0.02	0.27
Saab	0.06	0.20	0.23	0.41	0.01	0.16	0.02	0.22
Toyota	0.13	0.15	0.49	0.46	0.53	0.51	0.78	0.61
VW	0.04	0.08	0.31	0.16	0.51	0.56	0.07	0.25
Volvo	0.29	0.50	0.49	0.72	0.22	0.36	0.50	0.73
Mean:	**0.10**	**0.12**	**0.25**	**0.32**	**0.25**	**0.26**	**0.27**	**0.30**

Table 32: R^2 values for the TDNN architecture.

The experiment reported in the third column (MA1-No) has the same parameters that were used in the first study, see Table 30. The reason for the worse mean result is the fact that most of the added car makes have poor results. Table 33 shows the results for the SRN. As can be seen, the results for the SRN are here slightly better than those for the TDNN.

Company	raw-No		raw-MA2		MA1-No		MA1-MA2	
	TOM	IM	TOM	IM	TOM	IM	TOM	IM
Audi	0.06	0.06	0.26	0.32	0.15	0.22	0.31	0.53
BMW	0.11	0.05	0.08	0.21	0.08	0.00	0.11	0.00
Citroen	0.03	0.00	0.28	0.01	0.47	0.27	0.50	0.32
Ford	0.24	0.31	0.62	0.58	0.55	0.39	0.72	0.60
Honda	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Hyundai	0.16	0.09	0.61	0.40	0.46	0.36	0.54	0.43
Mazda	0.00	0.01	0.00	0.03	0.03	0.37	0.05	0.55
Mercedes	0.13	0.06	0.10	0.03	0.15	0.22	0.14	0.19
Mitsubishi	0.25	0.13	0.59	0.40	0.46	0.28	0.47	0.26
Nissan	0.17	0.18	0.31	0.56	0.08	0.09	0.09	0.13
Opel	0.13	0.25	0.59	0.61	0.35	0.40	0.41	0.50
Peugeot	0.36	0.21	0.45	0.45	0.26	0.20	0.21	0.30
Renault	0.00	0.04	0.00	0.32	0.03	0.23	0.04	0.38
Saab	0.11	0.20	0.28	0.48	0.28	0.31	0.29	0.35
Toyota	0.28	0.27	0.43	0.49	0.40	0.72	0.55	0.62
VW	0.04	0.05	0.06	0.11	0.49	0.19	0.05	0.22
Volvo	0.43	0.53	0.62	0.83	0.51	0.58	0.58	0.74
Mean:	**0.15**	**0.14**	**0.31**	**0.34**	**0.28**	**0.28**	**0.30**	**0.36**

Table 33: R^2 values for the SRN architecture.

7.2.5 Sensitivity analysis

A limited sensitivity analysis was performed. The purpose of this analysis was to find out the relative importance of the media categories. Obviously, this knowledge is interesting in itself, but the hypothesis was that it also could be used to increase the predictive power of the model by removing unimportant categories. The sensitivity analysis was conducted in a very simple way that, however, was judged sufficient for the purpose. Here, only IM was studied since the TOM for several makes is so small that single responses could significantly change the result. IM is also, intuitively, the effect that is most strongly correlated with marketing investments.

For every make, one investment in a specific week and a specific category was increased with 20%. The entire input sequence (with the single change) was then fed to the net and compared to the result when the original sequence was used. Each time five different nets were used. This process was then repeated for every different media category and for each week in a period between week 50 and 120. The result was then averaged over all weeks to get a single number for every media category and brand, representing the strength of that category.

These results were then used to remove some media categories from the input of certain brands. The exact method for which categories to remove and which to keep was to a large extent ad hoc, but turned out to be efficient. First of all, only categories with at least 15 entries in the period were kept. After that categories with an averaged result smaller than one third of the category with the highest value were removed.

The sensitive analysis showed that several categories were unimportant for different makes. As a matter of fact, the model found their contribution to be negative. Or, to put it in another way, an increased investment in that category resulted in a decreased effect. Obviously money put into a specific media category can only add to the effect, not reduce it, so this points at a flaw in the model. Hopefully ANNs forced to build models without those categories would find different, more correct models. Overall, the picture was that the most effective media were TV and morning papers. For some brands, evening papers or specialist papers were the most effective. Radio, movie and outdoor advertisements were in general ineffective.

The reduction resulted in that input vectors for different makes now consisted of between one and four media categories. The categories kept are shown in Table 34.

Company	TV	Outdoor	Morning press	Evening press	Popular press	Special press
Audi	X		X	X		
BMW			X			
Citroën	X		X			X
Ford	X					
Honda	X		X			
Hyundai	X					
Mazda			X	X		
Mercedes			X		X	
Mitsubishi			X		X	X
Nissan	X					
Opel	X		X	X		X
Peugeot				X		
Renault		X	X			
Saab	X		X			
Toyota	X					
VW	X					
Volvo	X					

Table 34: Media categories found to be important.

For media analysts, Table 34 turned out to be extremely interesting. Using the interpretation that media categories kept are the categories effective for a specific brand, several remarkable observations were made. As an example, for some brands, the results must be considered good news since they invest mostly in the media categories found effective. On the other hand, for some companies, the opposite was true. As an example, a specific brand had a marketing strategy where 75% of all investments were spent on advertisements in morning press.

For this specific brand, morning press, however, turned out to be a rather ineffective medium. In contrast, when the company did run one of its rather infrequent TV commercials, this had a very large impact.

7.2.6 Experiments with reduced input

This new, reduced, input set was then used for another series of predictions, both with and without the use of moving averages. When categories were removed from the input, according to the principle above, and the ANNs were retrained, the accuracy for most makes improved (see Table 35 below). This is a good sign for the model itself, since a different result would mean that the model had to rely on the ability to assign negative values to certain categories, which, as mentioned before, for this particular application must be wrong.

Company	**IM**			
	Training without MA1		**Training with MA1**	
	Original	**New**	**Original**	**New**
Audi	0.19	0.52	0.03	0.52
BMW	0.24	0.31	0.00	0.13
Citroën	0.11	0.39	0.13	0.36
Ford	0.65	0.58	0.74	0.64
Honda	0.10	0.00	0.00	0.00
Hyundai	0.29	0.60	0.18	0.60
Mazda	0.07	0.25	0.66	0.28
Mercedes	0.27	0.21	0.22	0.03
Mitsubishi	0.26	0.49	0.34	0.37
Nissan	0.42	0.57	0.05	0.59
Opel	0.52	0.60	0.37	0.60
Peugeot	0.37	0.00	0.27	0.00
Renault	0.14	0.36	0.27	0.49
Saab	0.41	0.35	0.22	0.43
Toyota	0.46	0.63	0.61	0.68
VW	0.16	0.23	0.25	0.34
Volvo	0.72	0.70	0.73	0.65
Mean	**0.32**	**0.40**	**0.30**	**0.39**

Table 35: Results for reduced data set using MA2 post-processing.

For the brands where the results were worse than the original, this could be due to several reasons. Honda has poor results both before and after, so for that brand it appears that the relationship between investments and effect is not strong enough. Honda also has a small IM with an average around 1.5%. BMW is a special case, since they changed their marketing strategy in the test period. During the training period BMW never used the TV category, but in the test period they invested heavily in TV. This is impossible for the model to capture, so the performance when predicting effects for BMW should be poor. Mercedes is also special, in the way that during the training period a notorious accident where their new model turned over happened. This introduced an error in the model since Mercedes appeared to get an extremely high impact for their money spent during that period.

For the brands with very high performance (e.g. Ford and Volvo) where some categories with positive contribution were removed, the decrease in performance might come from the fact that the model without these categories is slightly worse. It could also be that the original performance is "too good" and that the model used categories with negative contribution in a way that happens to give good performance in the test period. Peugeot is the only make that has a significantly lower performance after the removal of categories. The reason for this sharp drop in performance has not been identified.

7.2.7 Conclusions

The results reported again support the claim that ANNs can exploit the temporal structure of the media investment domain, in order to increase the predictive capacity. It is, however, very important to design the predictive modeling in a way where the desired entity is in fact targeted. Here, total effect (impact) was the desired target, which to some degree was handled by using the post-processing moving average.

The technique of using sensitivity analysis to determine which input categories to discard made the model more robust and added to the predictive performance. This is also an important contribution to the media domain. Techniques like this might shed some light on which combinations of media categories that should be used for greatest effect, for the individual makes.

It is also interesting to note that for some brands the most important categories (as found by the sensitivity analysis) were the categories with the highest investments. This should be reassuring for those companies. For other brands, the opposite was true; i.e. categories with mostly small investments were found most important. Exactly how this information should be interpreted and used must be left to the companies. Major changes in the media mix would probably change the problem to an extent where the models built would perform very poorly. As always, extrapolating a model is risky, and that would be the case here. Nevertheless the information in itself should be taken seriously by the affected companies.

7.2.8 Considerations

Even though the performance of the predictive models increased, it is still doubtful if strategic decisions could be based on predictions with this level of accuracy. On the other hand, it should be noted that these decisions *have* to be made and that even the models found in the first study have higher performance than the standard methods.

The need to include application of the methods to data from other domains was highlighted. It might be that the car domain (expensive products) has an impact on the results. Analysis on a domain with less expensive products could find a difference in how the advertising actually also effect our attitude towards possibility and preference.

The sensitivity analysis should ultimately also be extended to rule extraction from trained ANNs. Extracted rules could, for example, describe the relative importance between media categories, the length of the delay from investment to effect and the duration of an effect. These rules would most certainly be specific for each make but could still provide an explanation of the relationships found.

7.3 Initial rule extraction in the marketing domain

This section is based on the papers "Neural Networks - from Prediction to Explanation" [JN02b] and "Rule Extraction from Trained Neural Networks Using Genetic Programming" [JKN03].

7.3.1 Summary

The purpose of this study was to evaluate if ANNs, when used in the marketing domain, can be used not only for prediction, but also for explanation. The focus was to investigate if the powerful, but opaque, ANNs can be transformed into a representation comprehensive enough to act upon, while keeping high accuracy.

This is the study where G-REX was introduced, so one main objective was to estimate the performance to expect from G-REX. Important comparisons were made against a technique directly producing transparent models (C5.0) and existing rule extraction algorithms (TREPAN and RX). Naturally, the most important criteria to compare were accuracy and comprehensibility, but fidelity was also measured.

In this study, the focus was on short-term forecasts and the original regression problems were transformed into corresponding classification problems. The reason for this change was the fact that the two existing rule extraction techniques used for comparison with G-REX perform classification only.

The ANNs were first trained to identify weeks with high impact of advertising. The trained networks were then used for "rule extraction" i.e. the knowledge learned by the ANN was transformed into a more comprehensive representation; here decision trees.

The study showed that decision trees generated from the trained network have higher accuracy than decision trees created directly form the data set. In addition, G-REX obtained slightly higher accuracy than TREPAN and RX on most makes.

7.3.2 Background

The ANNs used in the previous studies are clearly capable of building a model where the impact of advertising is a function of sequences of investments in different media categories. The trained networks could be used for both short-term (1-4 weeks) and long-term (1 year) predictions with, what was deemed to be, good results. Typical use of these models would be to evaluate planned investment strategies by comparing the predictions produced.

Often though, something more than a simulation tool is needed since business executives would like to have a comprehensive and transparent model to act upon. Trained ANNs must, however, be regarded as black boxes; i.e. their representation of the underlying problem does not allow human inspection or understanding. As explained in 4.2, this is a serious drawback for ANNs when used for data mining. The process to derive more transparent formulations from trained ANNs called *rule extraction* (or sometimes *knowledge extraction*) is therefore rightly recognized as an important task to further establish ANNs as a tool for data mining.

The process of creating action rules from trained networks may use different extraction strategies and representation languages, but all methods must fulfill certain common demands. Although Craven and Shavlik in [CS99] also discuss scalability and generality, the most important criteria from [ADT95] are accuracy, comprehensibility and fidelity. With this in mind, the overall aim is to extract clear rules with high accuracy. This is obviously a trade-off, since more complex rules can express more complicated relationships. From the constraint that rule extraction must produce comprehensible rules to provide explanation, the focus for this study was to investigate if short and clear rules still could be expected to have acceptable accuracy.

7.3.3 Method

In the study, trained ANNs were used as a starting point for the task of finding a model comprehensible enough to enable decision-making. The focus was on short-term forecasts, so the two variables SoV and previous effect were assumed to be available and therefore included in the model. SoV for a specific make and week is, as mentioned before, the total investment for that make in relation to the total investment for all makes. Previous effect (PE) is the targeted effect in the previous week. Only a subset of all available makes was used in this study. More specifically, seven makes with good performance in previous studies were selected.

The sensitivity analysis in the previous study was performed for long-term forecasts. Now, a similar sensitivity analysis was performed for short-term forecasting. One key result was that the newly introduced variables SoV and PE were found to be very important. Table 36 presents the brands and categories used.

Company	Categories
Volvo	TV, Share-of-voice, Previous effect
Ford	TV, Share-of-voice, Previous effect
Toyota	TV, Share-of-voice, Previous effect
Volkswagen	TV, Share-of-voice, Previous effect
Saab	TV, Share-of-voice, Previous effect
BMW	Morning Press, Share-of-voice, Previous effect
Opel	TV, Morning Press, Share-of-voice, Previous effect

Table 36: Companies and media categories used.

Since TREPAN and RX perform classification only, the original problem had to be reformulated. The regression task at hand was to produce weekly forecasts of the real-valued effect (e.g. IM) on different horizons. This was changed into predicting if the effect for a certain week exceeds a specific limit. The limit chosen (with the motivation that it represents a week with high impact) was the 66-percentile of the training set. For the experiments, as mentioned above, only some brands were used and in addition only IM was targeted. The reason for this is that brands with very low effects are extremely sensitive to small changes in the number of respondents who mention the brand. When the task is altered from a regression task to a classification task, this is even more apparent since many data points just around the limit may cause poor results, even if the original prediction is rather good.

To get compact rules, the number of inputs was kept to a minimum. Just two previous weeks were lagged. For a brand where only one media category (e.g. TV) was used, the total number of inputs was as small as 9. This is not explicitly required by the rule extraction algorithms, but the results from the previous study implied that fewer inputs were not only sufficient, but even advantageous for the performance.

A single TDNN was trained for each make to classify whether the next week will have high impact or not. The reason for not using an ensemble or retraining ANNs every week is that both RX and the TREPAN implementation used here[1] require the actual network representation; i.e. connections and weights.

It should be noted that there are several approaches not based on a sensitivity analysis to find and remove inputs not vital for a neural net. In [CS97b] the results are improved by use of "clearning" [WZN96] before the rule extraction, and most decompositional methods rely on some form of pruning. As comparison, decision trees generated directly from the data set using C5.0 were applied on the test set. Below are some details for RX and G-REX.

RX

When using RX the following steps were performed:

1. Each input was split in eight intervals of equal length. These intervals were used to convert the continuous input to discrete values. With the localist coding used, each original continuous input resulted in eight binary inputs.

2. An ANN was trained for classification using the binary inputs.

3. Repeated pruning and retraining were performed on the resulting network, until no further pruning was possible without significantly reducing the accuracy.

4. The actual rule set was built by merging the rule sets found for hidden layer to output and input to hidden layer.

[1] The original TREPAN implementation found at http://www.cs.cmu.edu/~craven/trepan.sh

G-REX

In this study the function and terminal sets were:

$F = \{\text{and, or, >}\}$
$T = \{\text{Input}_1, \text{Input}_2, \ldots, \text{Input}_n, \Re\}$

The data sets were of course identical to the ones used by the other techniques, but for G-REX the training set was divided into one training set and one validation set. The comparison between techniques is still fair though, since the test set is intact. The training set consisted of 75 patterns, the validation set 25 and the test set 50.

The fitness functions

Obviously there is a direct connection between how the fitness function is formulated and the rule sets evolved. This is a nice property for the task of rule extraction, since the exact choice of what to optimize in the rule set is transferred into the formulation of the fitness function. This function could, for example, include how faithful the rules are to the ANN (fidelity), how compact the rules are (comprehensibility), and how well they perform on a validation set (accuracy). In this study, three different fitness functions were evaluated:

The first fitness function (Fidelity) measured whether the rule set performed similarly to the ANN on the training set; i.e. each instance classified in the same way as the network increased the fitness value by one. The second fitness function (Accuracy) used the training set in the same way, but also considered the validation set; i.e. the set of instances used to stop the training but not to adjust the weights in the network. Each validation pattern classified correctly increased the fitness value by one. The third fitness function (Comprehensibility) was an extension of the second with the addition of a penalty applied to longer programs. The purpose was to encourage smaller and therefore more readable rules.

Syntax

Internally, G-REX rules are represented as S-expressions. Figure 38 shows a sample program for the Impact of Advertising problem. Note that the representation language is Boolean trees and that the default class is *high impact.*

```
(OR (> TV2 10515)
    (OR  (OR (> PE3 10592)(> PE1 12026))
         (AND(> TV0 10036)(> PE3 10292))))
```

Figure 38: A sample rule for Impact of Advertising.

7.3.4 Results

Table 37 shows the results for the different makes and different techniques.

Company	**ANN**	**C5.0**	**TREPAN**	**RX**	**G-REX (Fid.)**	**G-REX (Acc.)**	**G-REX (Comp.)**
Ford	92%	84%	88%	80%	86%	84%	84%
Toyota	92%	76%	86%	74%	84%	86%	86%
VW	94%	80%	82%	76%	88%	90%	88%
Saab	96%	96%	86%	100%	94%	94%	96%
BMW	86%	80%	88%	*	84%	86%	82%
Opel	84%	80%	84%	74%	88%	86%	90%
Volvo	94%	90%	92%	100%	96%	94%	96%
Mean	**91%**	**84%**	**87%**	*	**89%**	**89%**	**89%**

Table 37: Percent correct on the test set for Impact of Advertising.

The ANNs were usually the most accurate. The only exception was for SAAB, which on the other hand is a bit special, since the test period did not contain one single week with high impact. Both TREPAN and G-REX showed higher accuracy than C5.0. G-REX slightly outperformed TREPAN on most makes. RX had perfect accuracy for some makes, but performed poorly on others. RX failed to produce rules for BMW, since the pruning was not able to reduce the net to a size possible to extract rules from. The experiment had to be aborted after several days of execution time.

To evaluate comprehensibility, the complexity of all trees (here measured as number of *interior* nodes) is presented in Table 38 below. To enable this comparison, TREPAN was here forced to use binary tests on single variables in each node.

Company	**C5.0**	**RX**	**TREPAN**	**G-REX (Fid.)**	**G-REX (Acc.)**	**G-REX (Comp.)**
Ford	12	5	28	48	44	9
Toyota	14	600	31	54	50	9
VW	12	62	20	44	44	7
Saab	9	23	20	40	43	7
BMW	15	*	17	26	22	9
Opel	14	409	18	32	50	11
Volvo	8	4	12	33	36	5
Mean	**12**	*	**21**	**40**	**41**	**8**

Table 38: Complexity measured as interior nodes.

It is obvious that G-REX, using the fitness function *Comp.* produced very compact rules. It should also be noted that TREPAN generally created more complex trees than C5.0. RX mixed very compact rule sets with extremely complex. Figure 39 shows a sample rule (for Volkswagen) extracted by G-REX. This could be compared to the M-of-N tree extracted by TREPAN in Figure 40, but note that the normalization is different. The variables are written as category followed by lag; i.e. PE0 stands for IM the previous week and TV0 denotes money invested in TV commercials the present week.

```
(OR  (OR (> TV0 12267) (> PE2 11142))
     (AND(> PE1 9568)  (> PE3 10620)))
```

Figure 39: G-REX rule for *High Impact for Volkswagen.*

```
IF 4 of {SOV2 > -0.0069, TV0 > 0.0697,
         PE0 > -0.0193, PE1 > -0.0193, TV0 > 0.1666,
         PE1 > 0.1488, PE0 > 0.0305} THEN HIGH
ELSE IF PE0 > 0.1169 THEN HIGH
     ELSE LOW
```

Figure 40: TREPAN rule for *High Impact for Volkswagen.*

From the extracted sample rule, it follows that for Volkswagen the most important category is *Previous Effect*. This implies that media investments do not change the effect rapidly, but rather build the effect over a longer time.

7.3.5 Conclusions

The classification problem turned out to be easier than expected. This is due to the fact that the variables previous week and share-of-voice were extremely important. Access to these values when predicting on a shorter horizon therefore changed the problem radically. ANNs are inherently very powerful, so it is not surprising that they outperformed C5.0 in this case study. The interesting part is to choose an adequate tool when transformation to a clearer model is needed.

Both TREPAN and G-REX performed well in this study; extracted trees were more accurate than corresponding trees generated directly from the data by C5.0. The study therefore supports the claim that ANNs is a tool for generalization; i.e. rules extracted from the model realized by the trained network have higher accuracy than rules created directly from the data set.

The possibility to tailor the fitness function directly to the demands on extracted rules is especially appealing, and a key argument for the G-REX method. In [ADT95] the criteria comprehensibility is emphasized:

…the end product is explanation not obfuscation. (p. 13)

In this study, the rules extracted without a penalty for long programs were often quite complex. The usability of such rules is questionable; they could very well fall into the category "obfuscation". For the problems at hand, however, the suggested approach using a fitness function enforcing shorter programs found rather compact rules, without sacrificing much accuracy. This could obviously be problem-dependent, but is still an important result.

In the experiments conducted, the use of a validation set in the fitness function did not improve performance. The most important reason for this is probably that the same validation set was originally used to stop ANN training. Nevertheless, for other problems this could still be an option.

For the classification problem at hand, it should be noted that C5.0 had an accuracy that probably would be regarded as “good enough” to act upon; so the advantage of using ANNs and rule extraction was not evident. Even so, the difference in performance between ANNs and C5.0 was significant. So, based on these results, the task of transforming the accurate but opaque ANN into a transparent model faithful to the net was assigned high priority.

7.3.6 Considerations

TREPAN is one tool for rule extraction and has some appealing properties, especially the simplicity of the chosen representation language. The choice of representation language is vital and decision trees or some kind of if-then rules are arguably the easiest to interpret for a user. Comprehensibility is, however, also closely related to the length of the extracted rules. Shorter rules are likely to be easier to interpret. There is obviously a trade-off between powerful, but complex representation, and smaller but less accurate. Loosely put, the representations should have high accuracy, but still be “clear enough” to make them possible to act upon. If extracted representations fail to be comprehensible, there is no reason *not* to use the original ANN with its documented high performance. Nevertheless, some papers still present rule extraction strategies where the rules are very long and could not in any way be regarded as explanations; see e.g. [DRS$^+$02].

Ideally, an algorithm for rule extraction should in itself enforce both accurate and clear rules. For most problems this would again require a trade-off between accuracy and comprehensibility. TREPAN to a large extent leaves this decision to the user, who may interrupt the extraction process at any time. When interrupted, TREPAN returns the current decision tree, and, since the exploration is performed in a “best-first” fashion, this tree is likely to be rather accurate. Nevertheless the trade-off is not explicitly handled; i.e. there is no mechanical way to try to search for a tree representing an optimal trade-off.

It should also be noted that TREPAN extraction strategy focuses on maximizing the fidelity towards the ANN. Although this is perhaps the most obvious requirement when performing rule extraction, accuracy and comprehensibility are more important for practical use.

G-REX, on the other hand, is designed to automatically handle the accuracy vs. comprehensibility trade-off by incorporating a penalty function for longer rules. In addition, when comparing G-REX to TREPAN, the possibility to easily change representation language is a major advantage for G-REX. This, original version of TREPAN, is also restricted to extracting rules from single ANNs. This is in sharp contrast to G-REX, which, as a true black-box rule extraction technique, can extract rules from any opaque model.

7.4 Rule extraction in another marketing domain

This section is based on the paper "Neural Networks and Rule Extraction for Prediction and Explanation in the Marketing Domain" [JSK+03].

7.4.1 Summary

The purpose of this study was to further evaluate the G-REX method and, at the same time, extend the studies in the marketing domain to another, very different, line of business. In this case study, ANNs were again used for prediction and explanation in the marketing domain. Here the underlying product was package holidays. Initially, ANNs were used for regression and classification to predict the impact of advertising from money invested in different media categories. Rule extraction was then performed on trained ANNs, using the G-REX method. Results show that both the ANNs and the extracted rules outperform the standard tool C5.0. In this study, G-REX accomplished the goal of combining high accuracy with keeping rules short and comprehensible. The study also showed that G-REX obtained high fidelity on both training and test sets.

7.4.2 Background

Typical questions when determining a marketing strategy include how the investments should be spread over time and what media categories to choose to maximize the impact. The overall goal is to have an accurate and transparent model for how investments determine the effect. Such models could be used both as simulation tools and for explanation.

In this case study, the experiences from the previous studies was put to use on a very different product; i.e. companies selling package holidays. From previous studies, several key demands on the prediction tool had been established for the results to be useful to decision-makers. Most importantly, the accuracy must be high and the model comprehensible enough to enable explanation of the relationships found.

The ANN technology is mature with established high performance in many diverse problem domains. Nevertheless, previous studies in the marketing domain showed that the exact formulation of the required performance is important and must be carefully chosen. More specifically, standard measurements like MSE and correlation coefficients turned out to be unsuitable, if applied directly. As an example, predictions with higher MSE could sometimes, by a media analyst, be deemed better than those with lower errors, if they capture the "important part" of the prediction; i.e. the correct area of the targeted effect over time.

Regarding the demand for an explanation facility, this initially seemed to rule out a neural network approach. ANNs are often regarded as black boxes, since the internal representation of the model is not comprehensible for humans. As a matter of fact, many introductory AI books (see e.g. [RN03] and [BL98]), where different techniques are presented and compared, state this as a well-known fact and a serious drawback for ANNs. The implication is that ANNs generally are not considered able to provide any conceptual explanation of the underlying problem.

The overall ambition is that rule extraction (and more specifically G-REX) should bridge this gap, so that ANNs with their high performance could be used more frequently on data mining problems. Using ANNs followed by rule extraction provides both predictive power and an explanation facility.

As a further step in evaluating G-REX, the fidelity of the extracted representations to the original ANNs was investigated. Even though accuracy and comprehensibility are the two most important criteria, fidelity must also be high. A low fidelity would mean that G-REX does not express the relationship found by the ANN, but instead finds another relationship. Typically, that would also lead to lower accuracy on the test set, since one of the main purposes of using an ANN in the first place is to guarantee good generalization.

It should always be remembered that rule extraction produces simpler (and less powerful) models. The implication is that overly simplified models may not be capable of representing the underlying relationship. Fidelity, measured on training and validation sets, together with accuracy on a validation sets are good indicators to avoid this.

Initial considerations

The purpose of exploring a different domain was to demonstrate the robustness of the neural network approach. At the same time, it is apparent that the travel business is very different from the car sales business. One major difference is that in the Swedish travel domain, the four leading companies have most weeks a combined TOM of over 99%. These four companies also dominate the advertising completely. For the car sales domain, many more brands have measurable TOM and IM.

The travel companies also vary their marketing more often and more radically. Furthermore, it could be argued that a specific car (the product) is much more tightly bound to the car make than a specific journey is to the travel company. A potential customer watching a commercial for a travel destination could easily miss the actual company since many companies offer (almost) the same products. As a matter of fact, the majority of the travel companies just recently realized this danger and now most commercials do focus on the brand. This adjustment is one of several radical changes in marketing strategies during the time period covered in this study.

The hope of performing long-term predictions of high quality was therefore rather low. Consequently, the focus was on short-term predictions, in this case just the following week. To maximize performance for short-term forecasts, the ANNs could be retrained every week as new data becomes available. This approach was used in [JN01] with good results, and should probably be the preferred option if the sole purpose is prediction. The drawback of this method is that it does not produce one model, but a different model every week. With the ambition to also use the model for explanation, the approach chosen was to train each network only once. When performing short-term forecasts, the two variables SoV and PE were assumed to be known and therefore included in the model.

The data set and preprocessing

The data set used was from Sweden and the travel business. The total data set includes 150 weeks. The data set contains, as input, weekly investments in different media categories for the four dominating travel companies on the Swedish market. The media categories were TV, radio, cinema, morning press, evening press, popular press, weekly press, special interest press and outdoor posters.

A measure that enables comparison between different methods was needed. For this study, a compromise was again made; the main measure used was still correlation coefficient, but moving averages (identical to the ones in section 7.2.3) were applied to get a smoother relationship. The purpose of using MA1 was to get a more stable model for the networks to train on. It is natural that the duration of impacts resulting from investments could be longer than one week. With this in mind, the purpose of MA2 was to focus more on the total effect from an investment.

Since most companies do not use all media categories, it is apparent that a reduction of the number of input variables could take place. Another reason for this is the ultimate goal of producing short and clear rules from the trained ANNs. Inspection of the data sets and a sensitivity analysis led to a decision to reduce the data sets to 6 inputs. These inputs were TV, Morning Press, Other Press, Other Investments, Share-of-Voice and Previous Effect. Other Press is the sum of all investments in press other than morning press. Other Investments is radio, outdoor posters and movies.

It can be argued that certain results are more interesting to predict than others. Especially, it would be appealing to be able to predict weeks with high effects. With the explanation capability in mind, this is even more attractive. Ultimately a clear model (like if-then rules) should exist for the decision-makers to act upon. Because of this perspective, the study focused on classification (instead of regression) by dividing the targeted effect into separate levels (e.g. high, medium and low) and using these as target values for ANN training.

7.4.3 Method

Several experiments were conducted using data from the four dominating travel companies. Each experiment covered both TOM and IM.

- A long-term prediction, without the variables SoV and PE.
- A short-term prediction, without retraining the networks every week but using SoV and PE.
- Classification where the levels are just high and low (really 'not high'). The high level is the upper third of the training samples. This is to detect weeks with high impact.

- Classification with three levels: high, normal and low. The limits are determined by dividing the sorted training set in three parts, each containing one third of all instances.
- C5.0 was used as a comparison for the classification.
- G-REX extracted rules from the ANNs trained for classification.

ANN architecture and training algorithm were found from initial testing, using a 4-fold cross-validation scheme. The training data (100 weeks) was divided into a training set (75 patterns) and a validation set (25 patterns), before training started. This was repeated for all four combinations of training and validation sets. The architectures with the best average result over the validation sets were chosen. The test sets (the last 50 weeks) were, of course, not part of the training but kept hidden until the evaluation.

Different ways of coding the classes were tested; i.e. localist coding and thermometer coding. Thermometer coding is often used when there is an ordinal relationship between the classes. As an example, in the experiment with three classes a target in the low class was coded as 00, a target in the medium class as 01 and a target in the high class as 11.

For the regression tasks, TDNNs with 5 tapped weeks and a single hidden layer with 10 units were used. The training algorithm was backpropagation with momentum and adaptive learning rate.

For the classification task, TDNNs with 3 tapped weeks were used. The exact architecture varied slightly with different codings, companies and targeted effects. For ANN training, resilient backpropagation [RB93] was used.

As usual, the function and terminal sets used by G-REX depended on the number of classes. The experiments with binary output extracted Boolean trees, while the experiments with three classes extracted decision trees.

The main advantage of G-REX is that the requirements imposed on the extracted representation, i.e. comprehensibility, accuracy and fidelity, are all handled by choosing an appropriate fitness function. In these experiments, the fitness was determined from the number of training patterns classified the same way as the ANN (fidelity), the number of correct classification on the validation set (accuracy) and a penalty for longer rules (comprehensibility).

7.4.4 Results

The results from the long-term prediction experiments were extremely varying (see Table 39). Only Apollo reached a level of accuracy comparable to most car brands in previous studies.

Company	**TOM**	**IM**
Always	0.23	0.11
Apollo	0.50	0.55
Fritidsresor	0.00	0.01
Ving	0.01	0.01
Mean	**0.18**	**0.17**

Table 39: Results for long-term predictions given as R^2 values.

Specifically, the predictions for Fritidsresor and Ving were very poor. Further investigation, however, showed that the companies had changed both their marketing strategy and their media mix radically almost exactly at the point between the training set and the test set (week 100). A picture of this is given in Figure 41, where TOM divided with money spent for Fritidsresor is plotted for all 150 weeks (note the striking difference in levels.)

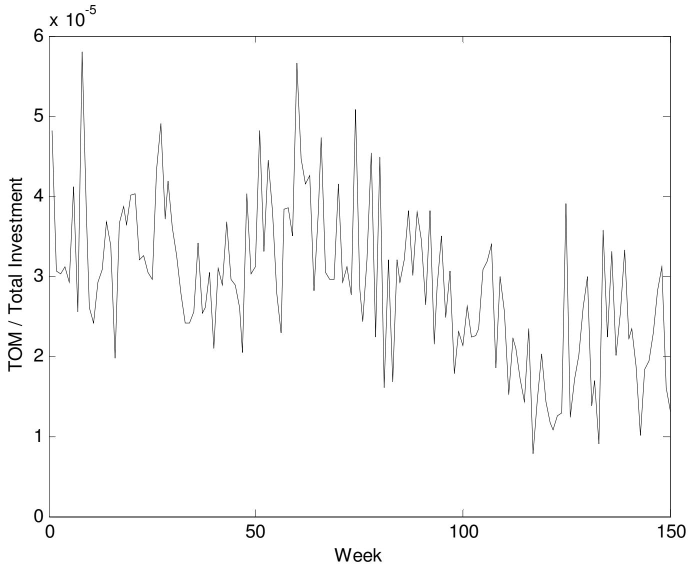


Figure 41: TOM/Total investment for Fritidsresor.

Short-term predictions, using the variables share-of-voice and previous result, were much better (with the exception of Fritidsresor), see Table 40 and a sample prediction in Figure 42.

Company	TOM	IM
Always	0.83	0.83
Apollo	0.76	0.85
Fritidsresor	0.01	0.02
Ving	0.48	0.58
Mean	**0.53**	**0.57**

Table 40: Results for short-term predictions given as R^2 values.

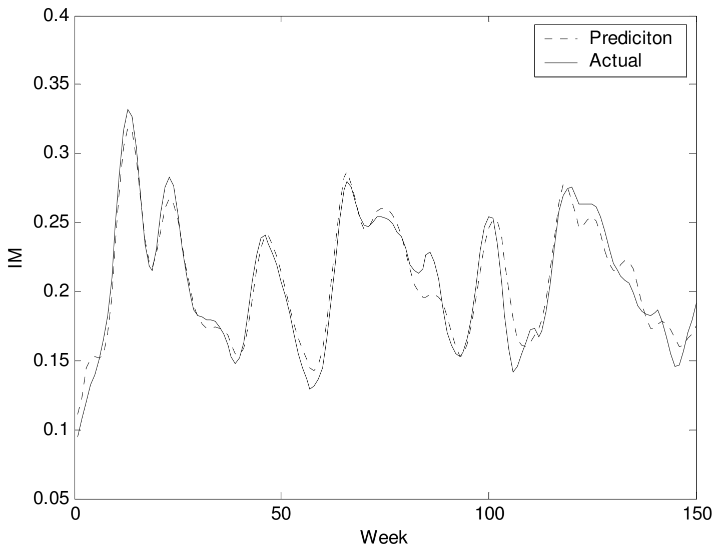


Figure 42: Short-term prediction of IM for Apollo.

The results from the classification experiments show that the ANN almost always has the highest performance; see Table 41 and Table 42. The difference between the two different codings turned out to be very small, hence, only the best (thermometer coding) is reported.

Company	TOM			IM		
	ANN	C5.0	G-REX	ANN	C5.0	G-REX
Always	90%	86%	86%	92%	82%	92%
Apollo	94%	94%	96%	96%	92%	96%
Fritidsresor	100%	100%	100%	100%	78%	100%
Ving	100%	70%	98%	100%	88%	100%
Mean	**96%**	**88%**	**95%**	**97%**	**85%**	**97%**

Table 41: Binary classification. Percent correct on test set.

Company	TOM			IM		
	ANN	C5.0	G-REX	ANN	C5.0	G-REX
Always	78%	76%	70%	88%	86%	84%
Apollo	84%	84%	76%	86%	62%	84%
Fritidsresor	100%	98%	100%	98%	96%	98%
Ving	94%	48%	82%	86%	86%	92%
Mean	**89%**	**77%**	**82%**	**90%**	**83%**	**90%**

Table 42: Classification with three classes. Percent correct on test set.

The reason for some poor results, especially on TOM, is the fact that the intervals turned out to be very small. The *medium* interval could, for some companies, have a range of only 1-2%, with many data points around the limits.

It is very interesting to note that rules extracted with G-REX generally outperform C5.0, even with the restriction that they must be kept short. Figure 43 and Figure 44 show two sample rules.

```
(OR   (AND (> MP2 9644 ) (> PE0 10590 ))
      (AND (> PE1 8731 ) (> Sov0 11495)))
```

Figure 43: Extracted rule for *High TOM for Apollo*.

```
(if   (> Sov0 12273 ) High
      (if (> PE0 9845 ) High
               (if (> PE0 8771) Medium Low)))
```

Figure 44: Extracted tree for *IM for Always* with three classes.

Table 43 and Table 44 show the fidelity accomplished by G-REX.

Company	**TOM**			**IM**		
	Train	**Validation**	**Test**	**Train**	**Validation**	**Test**
Always	98%	98%	96%	96%	96%	100%
Apollo	98%	96%	98%	96%	100%	100%
Fritidsresor	100%	98%	100%	100%	100%	100%
Ving	96%	96%	94%	98%	98%	100%
Mean	**98%**	**97%**	**97%**	**98%**	**99%**	**100%**

Table 43: G-REX fidelity on the binary classification problem.

Company	**TOM**			**IM**		
	Train	**Validation**	**Test**	**Train**	**Validation**	**Test**
Always	96%	94%	90%	92%	96%	94%
Apollo	94%	92%	90%	96%	100%	98%
Fritidsresor	100%	100%	100%	100%	100%	100%
Ving	96%	94%	90%	94%	92%	92%
Mean	97%	95%	93%	96%	97%	96%

Table 44: G-REX fidelity on the classification problem with 3 classes.

The level of fidelity was high and rather constant over training, validation and test sets. This is despite the fact that the class distribution for some companies differs greatly between the sets. Figure 45 to Figure 48 show sample confusion matrixes. Remember that the levels were chosen so that one third of all data points (in the training set) were "high" in the experiments with two classes, and that each class had one third of the data points in experiments with three classes.

		ACTUAL	
		Low	High
PREDICTED	Low	50	0
	High	0	0

Figure 45: Confusion matrix for Ving IM (test set).

		ACTUAL	
		Low	High
PREDICTED	Low	33	0
	High	2	15

Figure 46: Confusion matrix for Apollo TOM (test set).

		ACTUAL		
		Low	Medium	High
PREDICTED	Low	4	1	0
	Medium	1	4	4
	High	0	2	34

Figure 47: Confusion matrix for Always IM (test set).

		ACTUAL		
		Low	Medium	High
PREDICTED	Low	6	7	0
	Medium	1	19	0
	High	0	4	13

Figure 48: Confusion matrix for Apollo TOM (test set).

7.4.5 Conclusions

When working on “real” problems it is vital that the measurements used capture the most relevant aspects. Here, this point was handled in part by the different averages to get the prediction task “closer” to what is considered important, but also by the addition of the classification experiments. Many tasks originally formulated as regression tasks would probably benefit from transformation to classification problems. Often, in the data mining context, the classification task is closer to the problem formulations used by executives.

It is well known that ANNs are good tools for prediction. Often the drawback has been the lack of an explanation capability, something which is inherently present in, for instance, decision trees. When trained ANNs are used for rule extraction, this drawback is overcome. As the study implies, the extracted rules can have both high accuracy and comprehensibility. The experiments also show that G-REX fidelity was very high. Furthermore, the G-REX approach to rule extraction makes it possible to determine the properties that should be optimized through the fitness function.

7.5 Extending G-REX

This section is based on the paper "The Truth is in There - Rule Extraction from Opaque Models Using Genetic Programming" [JKN04].

7.5.1 Summary

The purpose of this study was to evaluate G-REX against the criteria proposed by Craven and Shavlik in [CS99]; i.e. accuracy, comprehensibility, fidelity, scalability and generality. To demonstrate the high generality possessed by G-REX, rule extraction was extended to regression problems. In addition, new representation languages, here regression trees and fuzzy rules, were used. Lastly, G-REX was also applied to another kind of opaque models, here boosted decision trees.

Several experiments, covering both regression and classification tasks, were conducted. Results show that G-REX, in general, was capable of extracting both accurate and comprehensible representations, thus allowing high performance also in domains where comprehensibility is of essence.

When evaluating G-REX against the criteria, it is apparent that G-REX offers a way to control the accuracy vs. comprehensibility trade-off. In addition, G-REX stands out regarding generality. All demands for generality, given by Craven and Shavlik, are well met by G-REX.

7.5.2 Background

There are potentially several different reasons for performing rule extraction from opaque models. The most obvious is the desire to obtain an accurate and comprehensible predictive model. Normally this involves a trade-off, however, since extracted representations must sacrifice some predictive power for increased comprehensibility. If this trade-off is acceptable, the extracted model could be used for the actual predictions; i.e. the rule extraction technique is more or less embedded in the data mining process. Another, slightly different scenario is where a powerful but opaque predictive model exists to start with and even a small loss in accuracy is unacceptable. Nevertheless, an explanation capability might be of value. Although the opaque technique would still be used for the predictions, the extracted representation could be seen as a rough guide to how the predictions are made. This knowledge could potentially lead to a better understanding of the problem and, for some problems, even the ability to create better predictive models. In a way, this is similar to an advanced sensitivity analysis.

Obviously, high accuracy is the overall goal of any predictive data mining technique. The decrease in accuracy from using a comprehensible model should be minimized, although normally some loss is inevitable. In G-REX, this trade-off is explicitly handled by the choice of fitness function. Normally, longer rules are penalized to increase understanding. If this penalty is not present, G-REX will produce quite complex rules. Still these rules are transparent and could, in principle, be interpreted by a human. At least some understanding of the problem (or the behavior of the predictive model) could be gathered from, for instance, inspecting the first splits in an extracted decision tree.

Craven and Shavlik in [CS99] argue that generality is very important. In particular, the ability to handle different underlying models and representation languages is stressed. This study therefore evaluated G-REX using the criteria proposed by Craven and Shavlik, but focusing on generality. Both classification and regression problems were used, forcing G-REX to extract rules using a variety of representation languages.

7.5.3 Method

The overall purpose of this study was to evaluate G-REX on new tasks and using new representation languages. More specifically G-REX was extended to handle:

- Regression problems producing regression trees.
- Classification problems producing fuzzy rules.

In addition, G-REX used not only single ANNs to extract from, but also two other opaque models; i.e. ANN ensembles and boosted decision trees.

The study was comparative since results from G-REX were compared both to the original results (from the opaque model) and to results from standard techniques. The standard techniques were the default selections for the respective problem category in the data mining tool Clementine. For classification tasks this is (boosted) decision trees using the C5.0 algorithm. For regression tasks the technique is CART.

The problems and data used

Two variations of the Impact of Advertising problem were used. In both experiments, TOM and IM were predicted from investments in different media categories. 100 weeks were used for training and 50 for testing. To reduce the number of input variables, only four aggregate variables were used:

- TV: money spent on TV-commercials.
- MP: money spent on advertising in morning press.
- OP: money spent on advertising in other press; i.e. evening press, popular press and special interest press.
- OI: money spent in other media; i.e. radio, outdoor, movie.

The two main experiments were:

- A long-term (one year) regression forecast. This is very similar to the original experiments used in [JN01]. The main difference is the aggregation of input variables.
- A short-term (one month) prediction using classification. This is similar to the experiments in [JSK+03], but here the horizon was one month instead of just one week. This is an important difference since some variables, shown to be very important (e.g. share-of-voice), are not available.

Only four car brands (Volvo, Ford, Hyundai and Toyota) were used in the experiments. Previous studies had produced good results on these data sets.

Long-term regression forecast

The purpose of this experiment was to produce a long-term forecast covering approximately one year. Each input instance consisted of investments during the current week and also from four lagged weeks. The overall problem is thus to predict effects of advertising from sequences of investments. Three approaches were evaluated:

ANNs. The ANNs were standard MLPs with one hidden layer. Initial experimentation, using a validation set, found 8 hidden neurons to be sufficient. For each effect (e.g. TOM for Ford) an ensemble of five independently trained

ANNs were used. To produce the ensemble output, the outputs from the five ANNs were averaged.

CART in Clementine. Here the standard method for producing regression trees in Clementine was used. It should be noted that, as previously mentioned, the technique termed C&R-Trees, according to the documentation, is a "comprehensive implementation of the methods described as CART©" [BFO+84].

G-REX. To enable a fair comparison with CART, G-REX used a functional set consisting only of relational operators and an if-statement. The terminal set consisted of the input variables and random constants in a suitable range.

Figure 49 below shows the BNF for the representation language used when extracting regression trees.

```
F = {if, <, >}
T = {i₁, i₂, …, iₙ, ℜ}

RTree      :-    (if RelExp RTree RTree) | Output
RelExp     :-    (RelOp Input Constant)
RelOp      :-    < | >
Input      :-    i₁ | i₂ | … | iₙ
Output     :-    ℜ
Constant   :-    ℜ
```

Figure 49: Representation language for regression trees.

Using these function and terminal sets, the feasible expressions were exactly the same for G-REX and CART. G-REX used the results of the trained ANN ensemble as fitness cases; i.e. the fitness was based on fidelity. In addition a small penalty term was applied to longer representations, thus enforcing more compact trees.

Short-term prediction using classification

The purpose of this experiment was to produce a short-term prediction using a four week horizon. The original regression problem was transformed into a binary classification problem, where the task is to predict if the effect (TOM or IM) will be higher than the 66-percentile (a week with high impact). In addition to the input variables used in the long-term forecast, the variable previous effect (PE) was used. PE is an important indicator for trends; i.e. detecting when the ratio between investments and effects changes. The task here was to predict an effect four weeks ahead using the investments between now and that week, together with previous effects from between the current week and two weeks back. In this experiment five different approaches were evaluated:

ANNs. The ANNs were standard MLPs with one hidden layer. Initial experimentation using a validation set found 5 hidden units to be sufficient. There was just one output unit and the two classes were coded as –1 and +1. An output over 0 from the ANN represents a predicted class of HIGH. For each effect, eleven ANNs were trained and combined into an ensemble.

C5.0. Both single decision trees and boosted trees created by C5.0 were evaluated. All parameters for C5.0 and the boosting were set to default values.

G-REX extracting Boolean trees from ANNs. The function set consisted of relational operators and logical operators (AND, OR). The terminal set contained input variables and random constants. An extracted representation is thus a Boolean tree. The fitness function was based on fidelity towards the ANN and a small penalty term to enforce short rules.

G-REX extracting Boolean rules from boosted decision trees. The only difference from the previous experiment was that the fitness used fidelity towards the boosted trees.

G-REX extracting fuzzy rules from ANNs. In this experiment, fuzzy rules were extracted. Each input variable was fuzzified, having two possible fuzzy values, labeled *Low* and *High*. Figure 50 shows how the fuzzification was performed. The constants *a* and *b* were, for each variable, chosen as the 20-percentile and the 80-percentile of the training data.

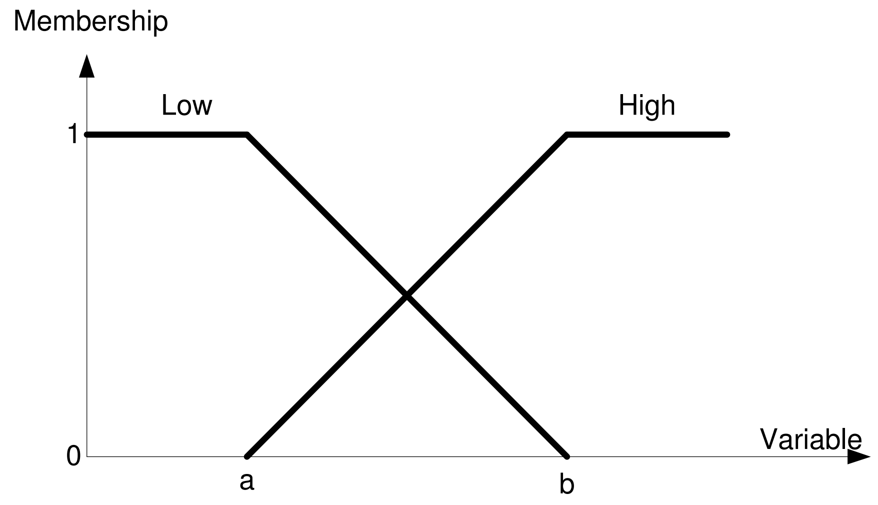


Figure 50: Fuzzification of input variables.

The terminal set contained the input variables and the names of the fuzzy sets. The function set contained logical operators, hedges (very and rather) and the function *is*. If μ_A is the membership mapping function for the fuzzy set A and μ_B

is the membership mapping function for the fuzzy set B, then the logical operators, working on fuzzy variables, are defined like:

$$\mu_{A \text{ AND } B}(x) = \mu_A(x) \wedge \mu_B(x) \quad = \quad \min\{\mu_A(x), \mu_B(x)\}$$

$$\mu_{A \text{ OR } B}(x) = \mu_A(x) \vee \mu_B(x) \quad = \quad \max\{\mu_A(x), \mu_B(x)\}$$

Hedges serve as modifiers of fuzzy values. In this experiment, the two hedges *very* and *rather*, as defined below, were used.

very: $\mu_A'(x) = \mu_A(x)^2$ ***rather:*** $\mu_A'(x) = \sqrt{\mu_A(x)}$

Figure 51 below shows the BNF for the representation language used when extracting fuzzy trees. To produce a prediction, the output from the fuzzy rule was compared to a threshold value, which was also evolved for each candidate rule.

```
F = {is, and, or, rather, very}
T = {i1, i2, …, in, low, high}

FuzzyTree          :-        FuzzyRule | FuzzyExp
FuzzyRule          :-        (FuzzyOp FuzzyTree FuzzyTree)
FuzzyExp           :-        (Input is FuzzyConstant)
FuzzyConstant      :-        FuzzySet | Hedge FuzzySet
FuzzyOp            :-        and | or
Hedge              :-        rather | very
FuzzySet           :-        low | high
Input              :-        i1 | i2 | … | in
```

Figure 51: Representation language for fuzzy trees.

7.5.4 Results

Table 45 shows the results for the regression task. The results are given as coefficient of determination, (R^2), between predicted values and target values on the test set.

Company	TOM			IM		
	ANN	CART	G-REX	ANN	CART	G-REX
Volvo	0.78	0.37	0.60	0.81	0.60	0.80
Ford	0.75	0.48	0.60	0.62	0.44	0.61
Toyota	0.73	0.35	0.58	0.75	0.44	0.61
Hyundai	0.64	0.66	0.65	0.67	0.64	0.67
Mean	**0.73**	**0.47**	**0.61**	**0.71**	**0.53**	**0.67**

Table 45: Results for the regression task.

Figure 52 and Figure 53 show predictions from the ANN and G-REX, plotted against the target values.

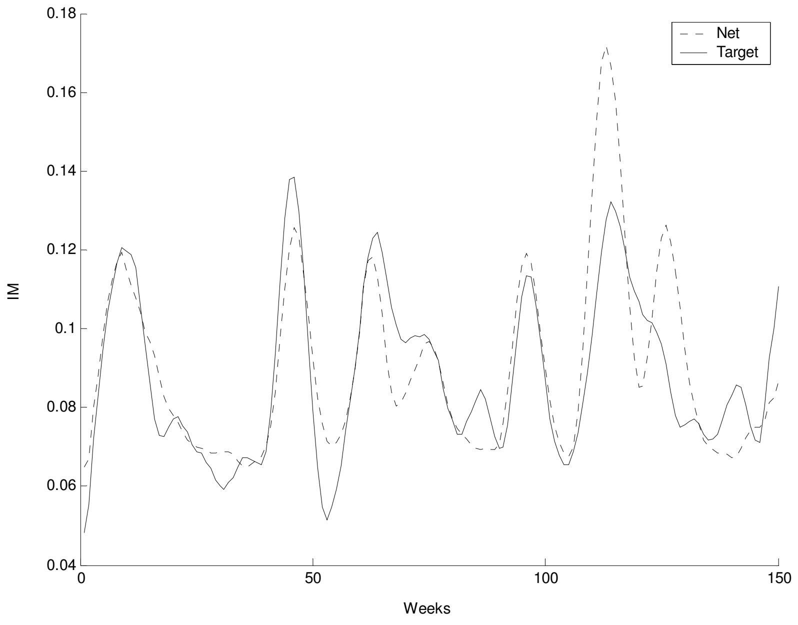


Figure 52: ANN prediction for Ford IM. Training and test set.

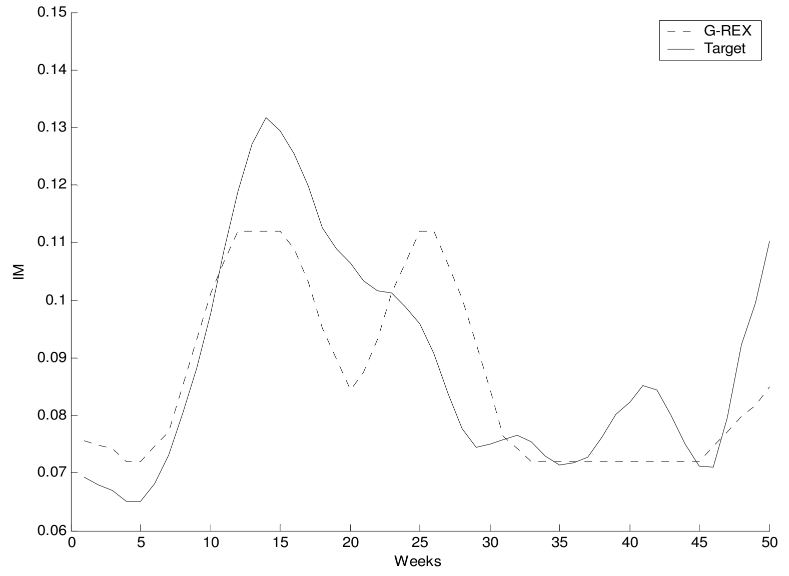


Figure 53: G-REX prediction for Ford IM. Test set only.

A sample evolved S-expression is shown in Figure 54 below:

```
(if  (< TV0 17)
     (if (< TV2 36)
              (if (> OI1 40) 82 72)
              (if (< TV2 97) 83 97))
     (if (> OP1 216)
              (if (< TV2 97) 85 112)
              (if (< TV2 40) 85 112)))
```

Figure 54: Evolved regression tree for Ford IM.

Table 46 and Table 47 show the results from the classification experiments as percent correct on the test set.

Company	**ANN**	**C5.0**	**C5.0 boost**	**G-REX ANN**	**G-REX C5.0 boost**	**G-REX fuzzy**
Volvo	92%	66%	72%	92%	72%	92%
Ford	80%	82%	78%	80%	78%	82%
Toyota	80%	66%	72%	72%	72%	76%
Hyundai	74%	34%	46%	94%	50%	90%
Mean	**82%**	**62%**	**67%**	**85%**	**68%**	**85%**

Table 46: Results for the classification task (TOM).

Company	**ANN**	**C5.0**	**C5.0 boost**	**G-REX ANN**	**G-REX C5.0 boost**	**G-REX fuzzy**
Volvo	90%	74%	74%	90%	72%	88%
Ford	78%	72%	76%	80%	70%	82%
Toyota	84%	72%	82%	80%	78%	82%
Hyundai	84%	62%	74%	84%	72%	84%
Mean	**84%**	**70%**	**77%**	**84%**	**73%**	**84%**

Table 47: Results for the classification task (IM).

Most of the extracted rules were both accurate and very compact. Figure 55 and Figure 56 show sample Boolean and fuzzy rules extracted by G-REX.

```
(AND(OR (> PE0 10558)(> TV0 10596))
    (AND(> TV1 9320 )(> TV0 933  )))
```

Figure 55: Evolved Boolean rule for Toyota High IM.

```
(AND(TV0 is rather high)(PE0 is very high))
```

Figure 56: Evolved fuzzy rule for Ford High IM.

7.5.5 Discussion

In this section, the results obtained in this study are used to evaluate G-REX against the criteria proposed by Craven and Shavlik.

Accuracy: G-REX performed well in this study. Most importantly, the accuracy on test sets was normally almost as good as that of the underlying opaque model. In addition G-REX outperformed the standard tools C5.0 and CART.

Comprehensibility: Craven and Shavlik emphasize "methods for controlling the comprehensibility/fidelity trade-off" as an important part of rule extraction algorithms. The possibility to dictate this trade-off by the choice of fitness function is, consequently, a key property of G-REX. At the same time, the experiments showed that, for the data sets investigated, G-REX is often capable of producing short *and* accurate rules. As a matter of fact, for most problems studied, G-REX performed just as well when forced to look for short rules. Another important aspect of the G-REX algorithm is the possibility to use different representation languages, to facilitate interpretation of the rules. In this study Boolean rules, fuzzy rules and regression trees were created just by changing the function and terminal sets.

Fidelity: Although this is not the main purpose of the G-REX algorithm, the study showed that extracted representations had similar accuracy to the opaque models, both on training and test sets. Obviously G-REX, especially when forced to look for short rules, is not capable of representing all the complexity of, for instance, an ANN ensemble. With this in mind, it is a fair assumption that G-REX is capable of finding the general relationship between input and output, represented by the opaque model.

Scalability: Scalability was not evaluated in this study but it is obvious that black-box approaches in general have an advantage compared to open-box methods. Black-box approaches are obviously independent of the exact architecture of the opaque model, which is in sharp contrast to open-box methods. Thus the size of the input space and the number of data points are the interesting parameters when considering scalability of black-box approaches. Although G-REX has not yet been tested on really large data sets, there is no reason to believe that G-REX will not perform well on larger data sets, but it remains to be verified. GP also inherently has the ability of "anytime rule extraction", since evolution can be aborted at any time to produce the best rule found up to that point.

Generality: G-REX is very general, since it operates on the mapping found by the opaque model, disregarding things like architecture, training regimes etc. As seen in this study, G-REX does not even require the opaque model to be a neural network. G-REX can be used equally well on, for instance, boosted decision trees and ANN ensembles. G-REX in addition proved feasible not only on classification tasks, but also on regression tasks.

7.5.6 Conclusions

The purpose of this study was to evaluate the versatility of the genetic programming rule extraction algorithm G-REX against the criteria identified by Craven and Shavlik. The results showed that G-REX not only exhibits a high degree of accuracy, but also that this accuracy is not necessarily obtained at the expense of comprehensibility.

Regarding generality, G-REX is very versatile since it acts on opaque models in general and not underlying architectures. G-REX can be applied to many different types of models and generate a multitude of representations. This was clearly demonstrated is this study where G-REX produced regression trees and fuzzy rules, in addition to Boolean and decision trees.

Chapter 8

A novel technique for ensemble creation

In this chapter, six studies related to ensemble creation are presented. The overall goal was to suggest a novel technique for ensemble creation, constantly producing accurate ensembles on most data sets. The basic approach used is to first create a number of models (here ANNs) and then use evolutionary algorithms to select and combine these models into an ensemble.

In the first study, which was originally reported in [JLN05], a simple algorithm based on GAs was evaluated against several straightforward ways of combining ANNs into ensembles. The suggested approach used GAs to search among all possible combinations of the available ANNs. The resulting ensemble is therefore just a subset of the available ANNs, here combined using averaging. The fitness function was based on ensemble accuracy on training and/or validation sets.

The second study, originally published in [JLK+06b], introduced a novel technique for ensemble creation. The technique, named GEMS (Genetic Ensemble Member Selection), first trains a large number of ANNs (between 10 and 50) and then uses genetic programming to build the ensemble by combining available ANNs. The use of genetic programming makes it possible for GEMS to not only consider ensembles of very different sizes, but also to use ensembles as intermediate building blocks, which could be further combined into larger ensembles. The fitness function was again ensemble accuracy on training and validation sets. In this study, GEMS was only evaluated on four data sets, and the results were slightly discouraging. The main problem was that GEMS obtained extremely high accuracy on the parts of the data set used during evolution, but sometimes failed to generalize to test data.

The third study, originally presented in [JLN06], looked into the question whether it is beneficial or not to set aside a part of the data set when training ANNs and later use this validation set when selecting ensemble members. Many approaches, for instance GASEN and ADDEMUP, do exactly this; see 5.4. In both cases, the ensemble is in fact optimized based on validation accuracy; i.e. accuracy on instances not used for ANN training. Naturally, the goal is, however, still to obtain high accuracy on data not used at all during model building and selection. This procedure therefore relies on that the correlation between validation accuracy and test accuracy is fairly high. This study, though, clearly showed that this is generally not the case.

The fourth study, originally reported as two separate papers, [JLK+06c] and [JLK+06d], tried to address the problem with GEMS overfitting the training data, by changing either the representation language or the training regime. Here, GEMS was compared to both straightforward ways of combining ANNs into ensembles and the GA approach tried in the first study. A limited comparison with results published for the standard techniques Random Forests and AdaBoost in [Bre01] was also conducted. The most important result was that GEMS now outperformed most other approaches, indicating that the measures taken to reduce overfitting were successful. Regarding the comparison with Random Forests and AdaBoost, the results were, however, inconclusive.

The fifth study, reported in [JLN07], empirically evaluated the diversity measures described in 5.3.1. The main result was that all diversity measures evaluated, show low or very low correlation with test set accuracy. Having said that, two measures; *double fault* and *difficulty* exhibited slightly higher correlations than the other measures. The experiments furthermore showed that the correlation between accuracy measured on training or validation data and test set accuracy again was rather low, thus confirming the results from the third study.

The sixth and final study, introduced a slightly different and more technical version of GEMS. Here, GEMS uses a fitness function explicitly prioritizing ensembles where the ANNs are diverse. The ensemble diversity was calculated by averaging pairwise double fault measures over all ANNs included in the ensemble. In addition, a procedure similar to G-REX use of oracle data was introduced. More specifically, the fitness function rewarded ensembles predicting test instances identically to the ensemble averaging the predictions from all available ANNs. In this study, the results obtained by GEMS were thoroughly compared to results published for the standard techniques Random Forests and AdaBoost in [Bre01]. The main result was that this version of GEMS is significantly more accurate than AdaBoost and Random Forest.

In these studies, altogether 28 publicly available data sets were used. The following 11 data sets were added to the ones used when evaluating G-REX:

- **Ecoli:** A biological data set where the purpose is to predict the localization site of a protein.
- **Hepatitis Domain (Hepatitis):** Prediction of whether a patient will survive or not based on several medical measurements.
- **Horse colic database (Horse)**: Prediction of whether a horse was operated on or not based on medical measurements.
- **Image**: The instances were drawn randomly from a database of 7 outdoor images. The images were handsegmented to create a classification for every pixel. Each instance is a 3x3 region, and the classes are brickface, sky, foliage, cement, window, path and grass.
- **LED Display (Led7):** This problem contains 7 Boolean attributes (a led segment on or off) and 10 classes, representing the set of decimal digits. The problem would be easy if not for the introduction of noise.
- **Satellite Image (Satellite):** The database consists of the multi-spectral values of pixels in 3x3 neighborhoods in a satellite image, and the classification associated with the central pixel in each neighborhood. The aim is to predict this classification, given the multi-spectral values.
- **Sick:** Prediction whether a patient is hyperthyroid or not based on medical measurements.
- **Thyroid:** This data set is very similar to *Sick*, and the target variable is again whether a patient is hyperthyroid or not. The *Sick* data set has more attributes but fewer instances, though.
- **Waveform:** This is an artificial three-class problem, based on three waveforms. Each class consists of a random convex combination of two waveforms sampled at the integers with noise added. A description for generating the data is given in [BFO$^+$84].

- **StatLog vehicle silhouette (Vehicle):** This data set originated from the Turing Institute, Glasgow, Scotland. The problem is to classify a given silhouette as one of four types of vehicle, using a set of features extracted from the silhouette. The four vehicles are double decker bus, Chevrolet van, Saab 9000 and Opel Manta 400.
- **Votes:** The task consists of classifying US Congressmen as democrat or republican based on their voting record on 16 key votes.

For a summary of the characteristics of these data sets, see Table 48 below.

Data sets	Instances	Classes	Continuous inputs	Categorical inputs
Ecoli	336	8	7	0
Hepatitis	155	2	6	13
Horse	368	2	7	14
Image	2310	7	19	0
Led7	3200	10	0	7
Satellite	6435	6	36	0
Sick	2800	2	7	22
Thyroid	3163	2	7	18
Waveform	5000	3	21	0
Vehicle	846	4	18	0
Votes	435	2	0	16

Table 48: UCI data set characteristics.

8.1 Study 1 – Building ensembles using GAs

The overall purpose of this study was to introduce a simple, GA-based, technique for creating ensembles, and compare this to several straightforward alternatives. More specifically, altogether 18 alternatives, categorized in six groups, were evaluated.

8.1.1 Method

When conducting the experiments, each data set was divided in three parts; training, validation and test. The training set was used to train individual ANNs. The validation set was, as usual, not utilized during training, but was intended to give an indication of the generalization capability. In this study, validation sets were used in different ways to rank and select ensembles. Setups not using any kind of selection used all data except the test set for training. Naturally, test sets were only used for the actual evaluation of each ensemble.

10-fold cross validation was employed and accordingly 10% of each fold was always used for testing. When using a validation set, ¼ of the remaining data was used for validation and ¾ for training. Accordingly, a validation set would hold 22.5% and the training set 67.5% of the entire data set. In all experiments ¼ of the training set was also used for early stopping, regardless of whether another validation set was used or not. The validation set was randomized before the training of each network, so each network was trained on slightly different data.

In this study, the output from an ensemble was always the average of the output from all members. For all problems, a localist coding was used, so there was one output unit per class, and the unit with the highest (averaged) output determined the predicted class.

Ensemble setups

The six groups of ensemble setups used in the study were named *3-layered*, *4-layered*, *Mixed*, *Selected*, *GA* and *All*.

In the 3-layered group, all ANNs had exactly one hidden layer. Three different setups were evaluated; a single ANN, five ANNs and ten ANNs. In this group all ANNs in an ensemble had identical topologies. The exact architecture was based on data set characteristics. More specifically the number of hidden units in the first layer was

$$h = \left\lfloor \sqrt{(v \cdot c)} \right\rfloor \tag{78}$$

where v is the number of input variables and c is the number of classes. The 4-layered group was identical to the 3-layered, with the obvious exception that ANNs used here had two hidden layers. The number of hidden units is found below

$$h_1 = \left\lfloor \frac{3}{2} h \right\rfloor \tag{79}$$

$$h_2 = \left\lfloor \frac{3}{2} c \right\rfloor \tag{80}$$

where c again is the number of classes and h is calculated using (78). The first two groups represent frequently used choices when identical networks are used to form the ensemble.

In the mixed group, two different setups were evaluated. Here either five (M5) or ten (M10) ANNs, with randomized topologies, were combined to make up the ensemble. The randomized topology was based on the heuristics described above. Each ANN in the ensemble could have either one or two hidden layers. The number of hidden units in networks with one layer was

$$M_3 h = h + \lfloor rand \cdot h \rfloor \qquad (81)$$

where M3 stands for *Mixed3-layer*, *rand* is a uniform random number in the range [0, 1] and *h* is, as before, calculated using (78). The numbers of hidden nodes in networks with two hidden layers were, for the first layer

$$M_4 h_1 = h + \lfloor rand \cdot (h / c) \rfloor \qquad (82)$$

and for the second layer

$$M_4 h_2 = \lfloor rand \cdot (h / c) \rfloor + c \qquad (83)$$

respectively. This group is potentially interesting since it represents a rather uncommon choice. The extra work needed, compared to the two previous groups is, however, quite marginal.

In the selected group, 50 ANNs (25 with one hidden layer and 25 with two hidden layers) were trained. The exact architecture for each ANN was randomized according to the procedure described in (81)-(83). After training, the ANNs were sorted on validation set accuracy. The selected ensembles consisted of the single best (S1), the best five (S5) and the best ten (S10) ANNs. This setup is clearly more costly, since many more ANNs are trained. Although the approach is very straightforward, it appears to be extremely uncommon.

The last evaluated setup not based on GAs simply used all 50 trained ANNs in the ensemble. For a summary of the evaluated setups not using GAs, see Table 49.

Name	#ANNs	#Hidden layers	Identical topology	Selection
3-one	1	1	-	-
3-five	5	1	Yes	All
3-ten	10	1	Yes	All
4-one	1	2	-	-
4-five	5	2	Yes	All
4-ten	10	2	Yes	All
M5	5	1-2	No	All
M10	10	1-2	No	All
S1	1	1-2	-	Best from 50
S5	5	1-2	No	Best 5 from 50
S10	10	1-2	No	Best 10 from 50
All	50	1-2	No	All

Table 49: Properties for setups not using GAs.

The GA group, which is a novel approach to the creation of ensembles, contained six slightly different setups. Each setup used the same pool of 50 ANNs as the selected group. Here, however, the selection of members for the ensemble was based on GAs. More specifically, a possible combination of ANNs was represented as a chromosome with 50 genes, each gene denoting a specific ANN. Each chromosome was represented as a sequence of zeroes and ones (a *bitstring*) where individual genes correspond to a specific ANN. As expected a "1" would indicate that the specific ANN should be included in the ensemble. All settings, except fitness function, were identical for all five setups. In each experiment *stochastic uniform selection* was used, the crossover and mutation rates were 0.8 and 0.05 respectively. The maximum number of generations was set to 200, but evolution was aborted if there was no change in fitness (for the most fit individual) over 30 consecutive generations. The populations consisted of 1000 individuals.

The first setup, called GA/Val, used accuracy on the validation set as fitness function. The second setup (GA/TrV) used accuracy on the training and validation sets as fitness function.

The third setup (GA/TaV) used accuracy on the training and validation sets as fitness function. During evolution the best individual (ensemble) from each generation is saved and after completion the "generation winner" with highest accuracy on the validation set was returned.

The fourth and fifth setups (GA/Tr3V and GA/Tr6V) are very similar to GA/TrV. All three used the training and validation sets to calculate the fitness. In GA/TrV a correctly classified instance from the validation set was only worth exactly as much as a correct prediction on the training set. But in GA/Tr3V and GA/Tr6V a correctly classified validation set instance was weighted with a factor 3 or 6, respectively. The obvious motivation for this approach was to prioritize accuracy on data not used for training, while still using as much data as possible when calculating the fitness.

The sixth and final GA setup (GA/Test) used accuracy on the test set as fitness function. It must, of course, be noted that target values for a production set are by definition not available during construction of a model. Therefore this setup is not useful as a construction strategy but serves only as a demonstration of what level of accuracy a combination of the available ANNs could achieve. Table 50 summarizes the evaluated setups using GAs.

Name	Fitness based on	Set used for selection
GA/Val	Validation	-
GA/TrV	Train/Validation	-
GA/TaV	Train/Validation	Validation
GA/Tr3V	Train/3*Validation	-
GA/Tr6V	Train/6*Validation	-
GA/Test	Test	-

Table 50: Properties for setups using GAs.

8.1.2 Results

The three tables below show the results from the experiments. Tabulated values represent average accuracy (on the test set) over all ten folds of each data set. Some of the 23 data sets used here were also used by Lim, Loh and Shih in [LLS00]. The column named *LLS* lists the best results obtained by any algorithm in the LLS study, where altogether 33 different algorithms were evaluated. To calculate the normalized value (*Norm.*) the following procedure was used:

- For each single run the accuracy obtained by a specific setup was divided by the result obtained by GA/Test on the same run. This value presents the accuracy obtained as a percentage of ″optimal″ accuracy, represented by GA/Test.
- The calculated percentages from each run were averaged to produce a single, mean, value for each setup and data set.
- The values produced for each data set were again averaged to produce the single, tabulated, value for each setup.

The row *average rank* shows the average rank for the setup over all data sets and the row #*Best* shows the number of data sets where the setup produced the best result of all setups. It should be noted that ranks were calculated using more than the three tabulated decimals.

Data sets	**3-one**	**3-five**	**3-ten**	**4-one**	**4-five**	**4-ten**	**LLS**
BLD	0.703	0.683	0.700	0.661	0.692	0.686	0.72
Cleve	0.755	0.810	0.803	0.768	0.823	0.803	-
CMC	0.527	0.531	0.546	0.436	0.528	0.542	0.57
Crx	0.739	0.843	0.854	0.664	0.824	0.823	-
German	0.691	0.716	0.709	0.654	0.708	0.698	-
Glass	0.555	0.618	0.673	0.418	0.659	0.659	-
Hepatitis	0.777	0.812	0.818	0.794	0.806	0.806	-
Horse	0.745	0.824	0.840	0.734	0.784	0.795	-
Iono	0.854	0.941	0.924	0.724	0.908	0.941	-
Iris	0.920	0.940	0.960	0.947	0.947	0.960	-
Labor	0.800	0.850	0.850	0.775	0.825	0.838	-
Led7	0.639	0.732	0.736	0.612	0.732	0.735	0.73
Lymph	0.612	0.765	0.777	0.647	0.741	0.782	-
PID	0.738	0.749	0.763	0.710	0.748	0.753	0.78
Satellite	0.808	0.839	0.838	0.776	0.851	0.856	0.90
Sonar	0.746	0.814	0.832	0.746	0.786	0.823	-
TAE	0.418	0.494	**0.553**	0.441	0.494	0.529	0.67
Tic-Tac-Toe	0.719	0.744	0.780	0.816	0.797	0.792	-
Waveform	**0.870**	0.869	**0.870**	0.830	0.869	0.869	0.85
WBC	0.900	**0.974**	0.967	0.865	0.965	0.969	0.97
Vehicle	0.672	0.824	0.831	0.644	0.828	**0.842**	0.85
Wine	0.867	0.939	0.978	0.928	0.961	0.978	-
Zoo	0.673	0.909	0.936	0.718	0.927	0.900	-
Norm.	**0.819**	**0.891**	**0.907**	**0.797**	**0.890**	**0.899**	**-**
Average rank	**15.26**	**10.78**	**7.50**	**16.13**	**11.47**	**9.54**	
#Best	**1**	**1**	**2**	**0**	**0**	**1**	**-**

Table 51: Results for uniform ensembles.

The most interesting observations from Table 51 are:

- There was a large difference in accuracy between single ANNs and ensembles.
- An ensemble with only five members seems to be too small when using identical ANNs, although the difference was much less significant between the two ensembles than between the smaller ensemble and the single network.
- The best results for both Waveform and WBC were better than the best results achieved by LLS, while results for TAE were much worse. It should be noted, however, that ANNs in general always perform very poorly on the TAE data set, compared to, for instance decision trees.

Data sets	**M5**	**M10**	**S1**	**S5**	**S10**	**All**	**LLS**
BLD	0.667	0.708	0.700	0.711	0.711	0.706	0.72
Cleve	0.816	0.800	0.781	0.813	0.819	**0.829**	-
CMC	0.539	0.537	0.524	0.546	0.557	0.548	0.57
Crx	0.853	0.851	0.849	0.846	0.853	**0.857**	-
German	0.717	0.709	0.690	0.711	0.714	0.707	-
Glass	0.555	0.641	0.600	0.686	0.686	0.668	-
Hepatitis	0.818	0.812	0.829	**0.853**	0.824	0.818	-
Horse	0.818	0.816	0.816	0.845	0.845	**0.853**	-
Iono	0.916	0.949	0.914	0.938	0.938	0.935	-
Iris	0.940	0.947	0.940	0.947	0.967	0.960	-
Labor	0.800	0.838	0.838	0.863	0.863	0.850	-
Led7	0.726	0.736	0.731	0.733	0.734	0.732	0.73
Lymph	0.729	0.800	0.729	0.788	0.782	**0.806**	-
PID	0.733	0.756	0.752	0.754	0.761	0.758	0.78
Satellite	0.850	0.846	0.865	0.865	0.865	0.848	0.90
Sonar	0.800	0.809	0.791	0.805	0.818	0.827	-
TAE	0.447	0.488	0.465	0.524	0.512	0.471	0.67
Tic-Tac-Toe	0.774	0.772	0.833	0.861	0.862	0.805	-
Waveform	0.866	0.868	0.864	0.868	0.867	0.865	0.85
WBC	0.957	0.964	0.967	0.969	0.972	0.965	0.97
Vehicle	0.806	0.807	0.784	0.841	**0.842**	0.831	0.85
Wine	0.978	**0.983**	0.956	0.978	0.978	**0.983**	-
Zoo	0.855	0.873	0.909	0.918	**0.946**	0.927	-
Norm.	**0.878**	**0.896**	**0.886**	**0.913**	**0.915**	**0.907**	**-**
Average rank	**12.58**	**9.87**	**12.59**	**6.93**	**5.57**	**7.60**	
#Best	**0**	**1**	**0**	**1**	**2**	**5**	**-**

Table 52: Results for mixed and selected ensembles.

There are several interesting results in Table 52:

- Ensembles built from selected networks were very accurate. S5 and S10 overall performed remarkably well, obtaining high accuracy on most data sets.
- In this study the concept of using mixed architectures did not pay off. For example, both M5 and M10 had worse accuracy than 3-ten.
- Using a very large ensemble (all) did produce rather high accuracy, although not quite as high as S5 and S10. The large ensemble, however, achieved the highest accuracy overall on as many data sets as five.
- The results on the LLS data sets were overall rather good, again with the exception of TAE.

Data sets	Val	TrV	TaV	Tr3V	Tr6V	Test	LLS
BLD	0.703	0.686	0.703	0.708	**0.720**	0.811	0.72
Cleve	0.819	0.816	0.819	0.823	0.797	0.926	-
CMC	0.556	0.555	**0.560**	0.558	0.560	0.638	0.57
Crx	0.850	0.853	0.850	0.856	0.843	0.919	-
German	0.713	0.731	0.724	**0.734**	0.727	0.803	-
Glass	0.664	0.668	0.664	0.659	**0.691**	0.845	-
Hepatitis	0.824	0.841	0.847	0.841	0.847	0.888	-
Horse	0.832	0.824	0.821	0.824	0.824	0.895	-
Iono	0.941	0.943	0.943	0.941	**0.951**	0.973	-
Iris	0.967	0.967	0.967	0.960	**0.973**	1.000	-
Labor	**0.888**	0.863	0.875	**0.888**	0.875	0.938	-
Led7	0.736	**0.737**	0.736	**0.737**	0.736	0.762	0.73
Lymph	0.782	0.794	0.788	**0.806**	**0.806**	0.912	-
PID	0.751	**0.765**	0.760	0.756	0.748	0.830	0.78
Satellite	0.866	**0.867**	**0.867**	0.866	0.865	0.881	0.90
Sonar	0.827	0.832	**0.841**	0.836	0.832	0.936	-
TAE	0.524	0.512	0.535	0.518	0.541	0.694	0.67
Tic-Tac-Toe	0.868	**0.879**	**0.879**	0.873	0.878	0.920	-
Waveform	0.867	0.867	0.869	0.867	0.867	0.884	0.85
WBC	0.964	0.968	0.968	0.964	0.961	0.983	0.97
Vehicle	0.833	0.835	0.826	0.838	0.836	0.928	0.85
Wine	**0.983**	0.978	**0.983**	**0.983**	0.972	1.000	-
Zoo	0.927	0.909	**0.946**	0.927	0.936	1.000	-
Norm.	**0.914**	**0.914**	**0.918**	**0.918**	**0.919**	**1.000**	**-**
Average rank	**6.56**	**5.70**	**4.48**	**4.91**	**5.65**		
#Best	**2**	**4**	**6**	**5**	**5**	**-**	**-**

Table 53: Results for setups using GAs.

The main results in Table 53 are:

- Using the Norm. value, all GA setups had, at least, as high accuracy as any other setup evaluated. Looking at average ranks, all GA approaches outperformed all other setups, with the exception of S10, which had a lower average rank than Val.
- Leaving the TAE data set out, the results are comparable to the best results achieved in LLS.

To determine if there are statistically significant differences between the setups evaluated, a Friedman test was performed. As seen in Figure 57 below, this test showed no significant differences between any of the GA approaches, S5, S10, All, 3-ten, 4-ten and M10.

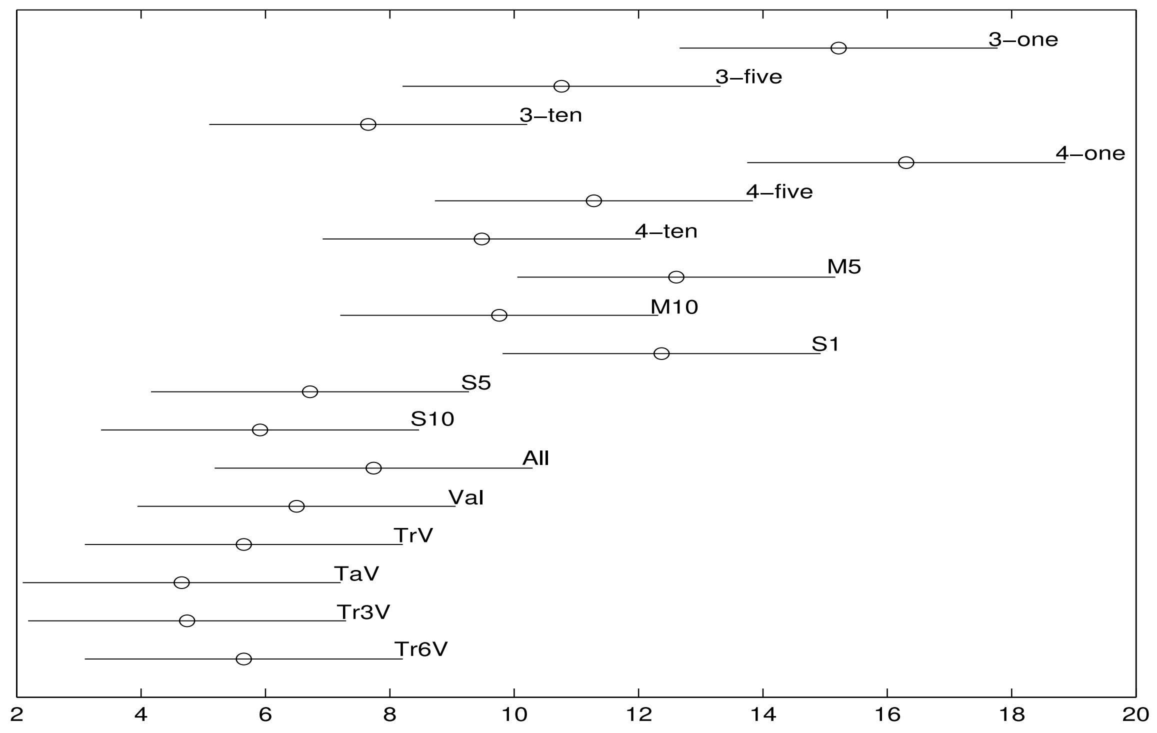


Figure 57: Friedman test Study 1.

8.1.3 Results

The purpose of this study was to compare different ways of creating accurate ANN ensembles. The most important conclusion is that the option to somehow actively select the members of an ensemble appears to be a strong approach. The study showed that the simple procedure of selecting a fixed number of networks, based on validation set accuracy, from a large pool of ANNs, results in increased accuracy. For the practicing data miner this is a basic yet very effective approach.

The GA approaches did produce the most accurate ensembles, although the difference was quite small. On the other hand, the GA setups, especially when emphasizing the validation set, outperformed the non-GA setups on most data sets. It should be noted that the most straightforward GA-approach; i.e. to use only the validation set when calculating the fitness, was not very successful. The reason is probably that the GA is too powerful, leading to overfitting and poor generalization. The idea to actively search for the best members of an ensemble is very appealing. The GA-approach proposed here is easier to grasp and more intuitive than most similar methods, the reason being that the fitness is based directly on accuracy and applied to ensembles instead of single networks. In addition, it is an implicit advantage that ensembles of different sizes are continuously evaluated and compared.

The fact that the statistical test showed few significant differences is partly due to the procedure used; i.e. comparing so many approaches against each other. Having said that, the main picture, looking at the average ranks, is that the GA-approaches, S5, S10 and All were most successful. Somewhat surprising, the very simple method 3-ten also performed pretty well.

A very interesting observation from the experimentation is that the ensemble selected by the GA has much higher accuracy on the "fitness set" than any setup not using GA. Unfortunately, this quite often failed to carry over to the test set.

8.2 Study 2 – Introducing GEMS

The main purpose of this study was to suggest a novel technique for automatic creation of accurate ensembles. The technique proposed, named GEMS, uses genetic programming to build the ensemble by combining available ANNs. As a matter of fact, the GP-framework built for G-REX is also used by GEMS. The only modification needed is the addition of some new functions and terminals, of course also resulting in new grammars. To evaluate performance, GEMS was compared to different ensembles where networks are selected based on individual validation set accuracy.

8.2.1 Method

This section first introduces the novel technique GEMS for creation of ANN ensembles. The second part describes the details regarding the experiments conducted. Since GEMS consists of two steps, each requiring several design choices and parameters, this section starts with a brief description of the main characteristics.

In the first step of GEMS, a number of ANNs are trained and stored in a pool. Since each ANN uses a localist representation, the number of output units is equal to the number of classes. The activation level of the output units for a specific ANN is termed its *result vector*. In the second step, GP is used to create the actual ensemble. When using GP, the ensembles are coded as genetic programs, where each individual represents a possible combination of the available ANNs. More specifically; each ensemble is represented as a tree, where the internal nodes contain operators while the leaves must be either ANNs from the pool or (random) constants. Here, GEMS has only two operators; *FACT* and *AVG*. FACT is used to multiply a result vector with a constant and AVG averages the result vectors from its children.

It should be noted that this in fact means that GEMS builds ensembles using a mix of smaller ensembles and single ANNs as building blocks. Figure 58 shows a GEMS ensemble coded in the tree format described above. This very small, sample, ensemble uses only three ANNs and the result is the average of ANN3 (multiplied with a factor 0.8) and the average of ANN1 and ANN2.

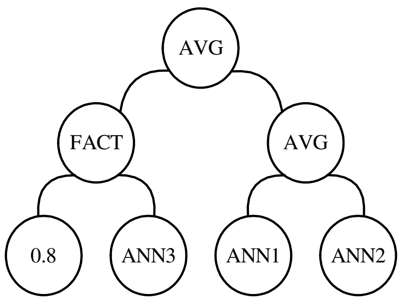


Figure 58: A sample GEMS ensemble.

GEMS settings

This study consisted of two experiments. The number of available ANNs was 50 and 20, respectively. In both experiments, half of the ANNs had one hidden layer, while the other half had two hidden layers. Each ANN was a fully connected MLP, with slightly randomized architecture. For an ANN with only one hidden layer, the number of hidden units was determined from (84) below.

$$h = \sqrt{(v \cdot c)} + \left\lfloor rand \cdot \sqrt{(v \cdot c)} \right\rfloor \tag{84}$$

where *v* is the number of input variables and *c* is the number of classes. *rand* is a random number in the interval [0, 1]. For ANNs with two hidden layers the number of units in each hidden layer is determined from (85) and (86) below.

$$h_2 = \left\lfloor \frac{3}{2}c \right\rfloor \tag{85}$$

$$h_1 = \frac{3}{2}h \tag{86}$$

where *c* again is the number of classes and *h* is calculated using (84). All ANNs were trained with the Levenberg-Marquardt backpropagation algorithm.

When performing GP, the two most important parameters are the representation language used and the fitness function. In this study, both were kept as simple as possible. With this in mind the function and terminal sets were:

```
F = {AVG, FACT}
T = {ANN₁, ANN₂, …, ANNₙ, ℜ}
```

where $\Re$ is a random number in the interval [0, 1]. $\Re$ was used as a scaling factor together with the FACT operator. The representation language used is described in Figure 59 below:

```
ENSEMBLE        :-        AVERAGE | SCALE | NETWORK
AVERAGE         :-        (AVG ENSEMBLE ENSEMBLE)
SCALE           :-        (FACT FACTOR ENSEMBLE)
NETWORK         :-        ANN₀ | ANN₁ | … | ANN₉
FACTOR          :-        ℜ
```

Figure 59: Representation language for GEMS.

The fitness function was based on three components. The first component added a constant value to the fitness for each pattern in the training set that was correctly classified. The second component was identical to the first with the exception that it used the validation set. The third component was a penalty for longer programs that added a negative constant value to the fitness for each part of the program.

The fitness was based on both validation and training accuracy, since that strategy proved to be slightly more successful in the first ensemble study. It is, however, far from trivial exactly how accuracy on validation samples should be balanced against accuracy on training samples. If a penalty for larger ensembles should be used and, if so, the appropriate magnitude, is another tricky question. The motivation for using a length penalty was that prioritizing smaller ensembles might avoid very large ensembles overfitting the training data.

Obviously, the constants used in the fitness function will significantly affect the behavior of the GP. In this initial GEMS study, the constants were set to 1, 3 and 0.01 respectively; resulting in the fitness function given in (87).

$$f = \#correct_{train} + 3 \cdot \#correct_{val} - \frac{1}{100} size \tag{87}$$

Crossover and mutation were performed as usual; with the addition that it is ensured that an offspring always is a correct program. With the representation language chosen, this means that the FACT node must have exactly one (random) constant child node. The other child could be a single ANN (a leaf node) or an AVG node. For an AVG node, both children could be single ANN terminals, another AVG node or a FACT node.

With this representation language, GEMS has the ability to combine the available ANNs in a huge number of ways. During evolution, GEMS is actually using genetic blocks representing ensembles to create new ensembles. This extreme flexibility is a key property of the GEMS technique. Naturally, the GP itself also has several parameters. The most important are given in Table 54 below.

Parameter	**Value**
Crossover rate	0.8
Mutation rate	0.001
Population size	1000
Generations	500
Creation depth	8
Creation method	Ramped half-and-half
Elitism	Yes

Table 54: GP parameters for GEMS.

Experiments

The four data sets used in this study were CMC, TAE, Tic-Tac-Toe and Vehicle. For each data set 40 runs were performed. Before each run, the data set was randomly divided into four parts; a training set (50% of the patterns) used to train the ANNs, a validation set (10%) used for early stopping, another validation set (20%) used to select ensembles and a test set (20%). The test set was of course a holdout set used exclusively to measure performance. For comparison against GEMS, four alternative fixed ensembles were constructed on each run. Each competing ensemble consisted of a fixed number of ANNs, based on validation set accuracy; i.e. an ensemble consisting of five ANNs included the best five individual ANNs, measured on the validation set. The exact number of ANNs in the fixed ensembles is given in Table 55 below.

Ensemble	**#ANNs in first experiment (total 50 ANNs)**	**#ANNs in second experiment (total 20 ANNs)**
Quarter	13	5
Half	25	10
Three-quarter	39	15
All	50	20

Table 55: Number of ANNs in fixed ensembles.

In this study, the output from a fixed ensemble was always the average of the output from all members. Since localist coding was used, the unit (or rather, index in the result vector) with the highest (averaged) output determined the predicted class.

8.2.2 Results

Table 56 below shows the results from the experiment using 50 ANNs in the pool. The values tabulated are mean test set accuracies over all 40 runs.

Data sets	**Quarter**		**Half**		**Three-quarter**		**All**		**GEMS**	
	Val	**Test**	**Val**	**Test**	**Val**	**Test**	**Val**	**Test**	**Val**	**Test**
CMC	0.569	0.551	0.564	0.552	0.556	0.550	0.551	0.550	0.589	0.552
TAE	0.609	0.503	0.580	0.503	0.548	0.508	0.524	0.513	0.724	0.536
Tic-Tac-Toe	0.890	0.864	0.875	0.852	0.860	0.838	0.847	0.833	0.932	0.884
Vehicle	0.863	0.847	0.859	0.847	0.854	0.845	0.846	0.845	0.897	0.846

Table 56: Results using 50 ANNs.

From Table 56 it is very clear that GEMS is capable of achieving high accuracy on the validation set, which of course is a part of the data set used for GP evolution. On all data sets, GEMS clearly outperformed all other ensembles on the validation set. This is despite the fact that the validation set was also used by the other approaches to rank and select ANNs. Unfortunately, this advantage did not transfer to the test set for all data sets. More specifically, GEMS had significantly higher test set accuracy than all other ensembles on TAE and Tic-Tac-Toe, but there was virtually no difference on Vehicle and CMC. Figure 60 below shows test set accuracy vs. validation set accuracy for Tic-Tac-Toe.

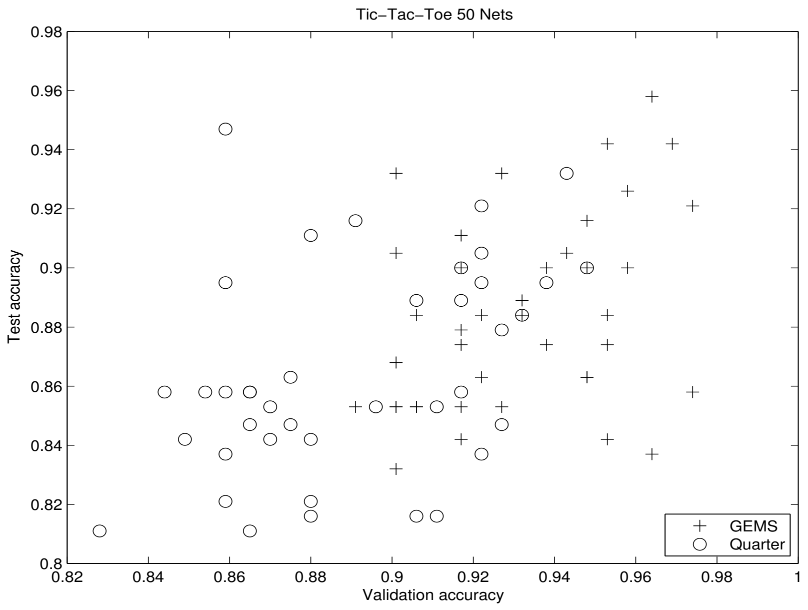


Figure 60: Test accuracy vs. validation accuracy for Tic-Tac-Toe.

The picture is very promising for GEMS; the fact that GEMS markers tend to be further to the top right shows that both validation and test accuracy are higher for GEMS. Figure 61 below, on the other hand gives a totally different picture.

Here, the advantage for GEMS on the validation set does not carry over to the test set.

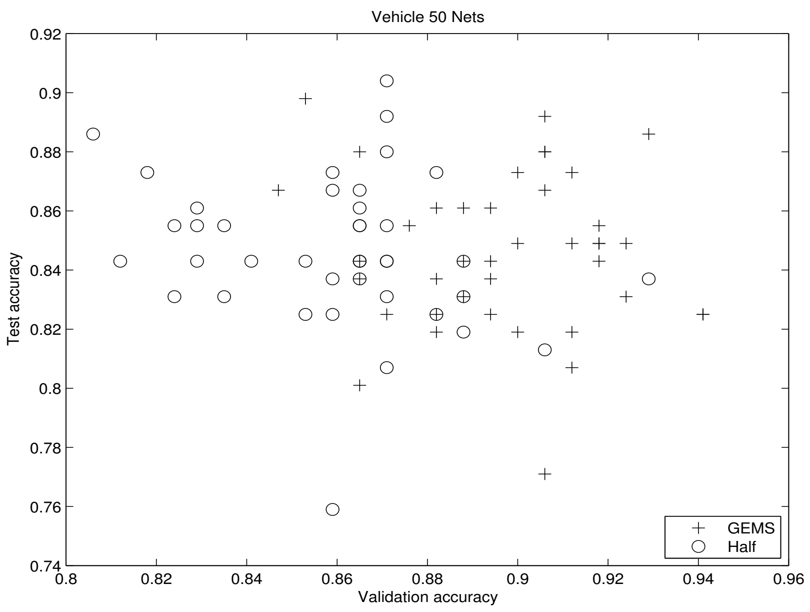


Figure 61: Test accuracy vs. validation accuracy for Vehicle.

The results for the experiment using only 20 ANNs are very similar, see Table 57 below. Again GEMS had significantly higher test set accuracy on Tic-Tac-Toe and TAE, while there were no significant differences on CMC and Vehicle.

Data sets	**Quarter**		**Half**		**Three-quarter**		**All**		**GEMS**	
	Val	**Test**	**Val**	**Test**	**Val**	**Test**	**Val**	**Test**	**Val**	**Test**
CMC	0.574	0.552	0.569	0.555	0.563	0.555	0.553	0.552	0.605	0.554
TAE	0.622	0.519	0.603	0.526	0.582	0.528	0.552	0.529	0.702	0.548
Tic-Tac-Toe	0.868	0.856	0.861	0.850	0.851	0.844	0.839	0.839	0.906	0.865
Vehicle	0.866	0.837	0.864	0.845	0.857	0.845	0.852	0.845	0.892	0.845

Table 57: Results using 20 ANNs.

A comparison of the results from the two experiments shows that there is no significant difference between using 20 or 50 ANNs, and that this holds for both the fixed ensembles and for GEMS. The ensembles constructed by GEMS were extremely varied in both size and shape. The length penalty does, however, put a substantial pressure on the evolution, often resulting in remarkably small ensembles. As a matter of fact, ensembles using less than 5 ANNs are not uncommon. Below are sample ensembles (shown in the internal format used by GEMS) from two different runs. The first ensemble, shown in Figure 62 is rather small and the second (shown in Figure 63) is of about average size.

```
(avg(avg(ann2)(ann3))(avg(ann37)(ann8)))
```

Figure 62: Small GEMS ensemble.

```
(avg(avg(avg(*(0.898)(avg(ann7)(ann18)))(*(0.874)(avg(a
nn15)(ann14))))(avg(*(0.227)(avg(ann15)(ann18)))(*(0.71
7)(avg(ann15)(ann4)))))(*(0.574)(avg(avg(ann6)(ann1))(*
(0.186)(ann13)))))
```

Figure 63: Averaged-size GEMS ensemble.

The possibility of using a specific ANN more than once in an ensemble was often utilized. The ensemble in Figure 63 is one example, since it uses ann15 three times.

8.2.3 Conclusions and discussion

From the results, it is obvious that the GEMS technique should be further evaluated. Arguably the most important ability of GEMS is the possibility to use any combination of the available ANNs; including the option to use specific ANNs several times in one ensemble. Perhaps it is more correct to describe GEMS as a Meta ensemble builder, since its building blocks in fact are ensembles. In addition, the versatile nature of GP makes it very easy to include new operators; e.g. a majority vote node.

When assessing the results obtained by GEMS, it is important to realize that GEMS in this study was only compared to other ensembles. Very often novel ensemble techniques are instead compared to single models; typically "the best ANN available to the ensemble". It must, however, by now be evident that the use of an ensemble is clearly superior to single models, so therefore such comparisons were left out. The most important observation is instead the fact that although GEMS consistently had much higher accuracy on the validation set (compared to the other ensembles) this property was not always preserved in the test set. Even though the fitness function used all available data (i.e. both training and validation data) and a length penalty was used to encourage smaller ensembles, the most probable explanation is that the GP overfitted the validation data. Just to iterate this important point; GEMS had very high accuracy on the part of the data set covered by the fitness function, but this did not necessarily carry over to the test set. How to deal with this problem was identified as the top priority for future studies.

With this in mind, one key question was if the use of a holdout (validation) set for optimization really is beneficial. Since the overall goal is to achieve high accuracy on unseen data, the best possible test seems to be to measure exactly that, accuracy on unseen data. This reasoning, however, has a devious shortcoming; the real issue is how a model chosen from accuracy on a validation set would perform on *yet novel data*. If a validation set is used to somehow choose one model over another, the underlying assumption must be that there is a high correlation between accuracy on that validation set and accuracy on another set of unseen data; i.e. the test set. If this assumption does not hold, there is obviously little to gain from using a validation set as a basis for ensemble construction.

Regarding the GEMS technique, several possible modifications were identified. The first is the very straightforward choice to dispose of the validation set altogether. The best use of the data set might be to use all available data for both ANN training and GP evolution. One micro technique to enforce some diversity among the networks could be to train each ANN using only part of the available training data (e.g. 70%) and randomize the exact patterns for each network.

Another strategy is to change the GP training regime to avoid overspecialization on a specific part of the data. One option is to use something similar to standard boosting and another is to constantly alter the fitness set by randomly adding and removing data patterns between generations. Using either of these regimes should favor more general ensembles, something that hopefully could carry over to the test set.

Finally, the possibility to introduce some locality was identified. If the first levels of the GP-tree could contain splits (similar to standard decision trees) where the questions are expressed in original variables, different ensembles could specialize on different parts of the input space.

8.3 Study 3 – Evaluating the use of a validation set

A very important part of all algorithms building ensembles is how the evaluation of possible ensembles is performed. It is fair to say that a very common procedure is to evaluate the performance of either single base classifiers or ensembles by applying them to a holdout part of the data set not used for training of the individual classifiers; a validation set. As described in the related work section (see 5.4), GASEN and ADDEMUP are two examples where GAs are used to search for ensembles using fitness functions based on validation set accuracy. With the results of the previous study in mind, together with the fact that several existing techniques use validation set accuracy to select ensemble members, the purpose of this study was to look into the importance of validation set accuracy. Naturally, the evaluation boils down to if the correlation between validation set accuracy and test set accuracy is high enough to motivate its use as selection criterion. The overall purpose of this study - to investigate if use of a validation set really is beneficial to a data miner - translates into two questions:

1. Is the correlation between accuracy on one holdout set and another high enough to motivate the use of validation sets for selecting models?

2. Does a model selected on validation accuracy normally outperform a model chosen on training accuracy, when applied to test data?

8.3.1 Method

The main experiments measure the correlation between accuracy on validation sets and test sets for either ANN ensembles or single ANNs. In all four experiments, 11 data sets from the UCI Repository were used. Instead of using ten-fold cross validation, a procedure where, on each data set, the data points are randomly divided into a training set (40%), an early stopping validation set (20%), another validation set (20%) and a test set (20%) before each run was used.

In the first experiment ensembles were evaluated. The following schedule was used: First 20 3-layered ANNs (with slightly randomized number of hidden units) were independently trained, using the training set for weight adjusting and the first validation set for early stopping. The number of hidden nodes was based on the following heuristic, where c is the number of classes and v the number of inputs:

$$h = \left\lfloor \sqrt{(v \cdot c)} \right\rfloor \tag{88}$$

The actual number of hidden nodes for each network was then randomly distributed in the range [h, 2h]. Binary problems used a single output unit, while multiclass problems used the localist representation, having as many output units as classes. After completed training, the five ANNs with lowest accuracy on the validation set were immediately discarded to avoid any extremely poor ANNs being used in the ensembles. From the remaining 15 ANNs all possible ensembles consisting of exactly 10 ANNs were created. The total number of ensembles evaluated in each run thus was 3003. The output from the ensemble was the average output from all ensemble members. In total 25 runs on each data set were performed.

The quantity measured was correlation between accuracy for ensembles on the second validation set and accuracy on the test set. Naturally, a high correlation would indicate that ensembles performing comparatively well on the validation set also perform better on the test set. Or, put in another way, that it is generally beneficial to choose an ensemble with high accuracy on the validation set if the goal is to obtain high accuracy on the test set.

The second experiment used single ANNs instead of ensembles. The data sets were divided in the same way, but here each run used 200 individual ANNs and 20 runs were performed on each data set. In this experiment, it was the correlation between validation set accuracy and test set accuracy for individual ANNs that were measured.

The third and fourth experiments evaluated the performance (accuracy on the test set) for ensembles, based on how available data was used for training and selection. Three different model sets were created and evaluated. In these experiments too, 20% of the total data set was used as a test set. Both experiments were run 20 times each, on every data set.

The first model set (A) used 60% of the data for ANN training (40% training, 20% early stopping) while 20% was used as a validation set. The second model set (B) did not use a validation set but instead utilized all data except the test set for training (¾ training and ¼ early stopping). The final model set (C) also used all available data for training, but without early stopping validation.

In the third experiment, 15 ANNs were trained in each model set. All possible ensembles of size 10 were evaluated and the accuracy on the test set, the validation set and all data but the test set (henceforth called *all available data*) were calculated for each ensemble and each model set.

For the evaluation, the ensembles from each model set were sorted on accuracy on all available data. The ensembles from the (A) model set were also sorted on validation set accuracy. Finally, the mean test set accuracy from the top 5% ensembles from each sorted list was calculated.

The only difference between the third and fourth experiment is that in the fourth experiment 20 ANNs were trained and 10000 random ensembles were evaluated. The number of ANNs in each ensemble varied between 2 and 20.

It should be noted that all experiments used the assumption that all models are powerful enough to learn the specific problem. Accordingly, most individual ANNs had, after training, at least fairly good performance.

8.3.2 Results

Table 58 shows the results for experiment 1; i.e. the correlation between validation set accuracy and test set accuracy for each ensemble per run and data set. Obviously, the main result is that there is virtually no correlation. Consequently, there is no support for the assumption that an ensemble with higher accuracy on the validation set also performs better on novel data.

Data sets	**Mean**	**Std. dev.**
BLD	0.05	0.27
Cleve	-0.01	0.19
CMC	0.09	0.12
Crx	0.02	0.20
German	-0.01	0.16
Glass	0.05	0.19
PID	-0.08	0.19
TAE	0.09	0.31
Tic-Tac-Toe	0.27	0.18
Waveform	0.02	0.14
Vehicle	0.04	0.12

Table 58: Correlation between validation and test accuracy: Ensembles.

To further illustrate the point that ranking of models based on validation set accuracy seems to be futile, Figure 64 shows a plot where the ensembles are sorted on validation set accuracy; i.e. the leftmost ensemble (with number 1) had the lowest accuracy on the test set, while the rightmost (number 3003) had the highest.

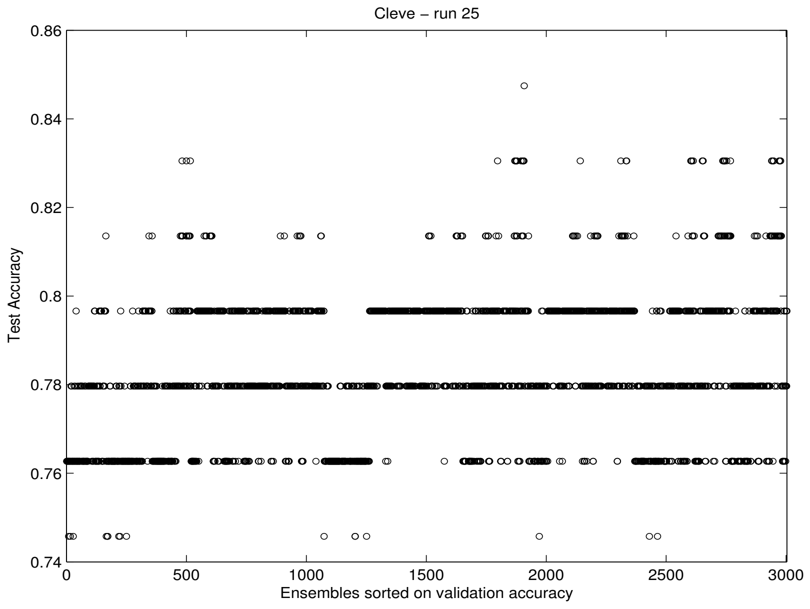


Figure 64: Ensembles sorted on validation accuracy. r=0.24.

The plot shows two important facts. First of all there is very little difference, in general, between ensembles in the right hand part and the left hand part, despite the fact that this is one of the single runs showing relatively high correlation. This again suggests that there is little to gain by choosing ensembles based on validation set accuracy. Second, it must be noted that this does not imply that all ensembles have more or less identical accuracy on the test set. As seen in the plot, the difference between the worst and best ensemble is more than 10%. At the same time it must be noted that this difference corresponds to just 6 instances. The worst ensemble gets 44 of 59 test instances correct and the best 50.

Finally, the ensembles were divided in three parts based on validation set accuracy (for each run) and an ANOVA analysis was performed on test set accuracy. This analysis helped to determine if selecting an ensemble in the best third (measured as validation set accuracy) would result in a statistically significant advantage (on the test set) compared to using an ensemble from one of the lower two-thirds. Table 59 shows the results from this ANOVA analysis. The three numbers in each cell represent the number of runs where the lowest ranked third was significantly better, the number of runs where the highest ranked third was significantly better and the number of runs with no significant difference, respectively. As seen in the table, there is little to gain from selecting an ensemble in the best third. On a couple of data sets it is even more advantageous to constantly select one from the worst third!

Data sets	**1 vs. 2**			**2 vs. 3**			**1 vs. 3**		
	Low	*High*	*No*	*Low*	*High*	*No*	*Low*	*High*	*No*
BLD	6	4	15	6	4	15	6	4	15
Cleve	10	6	9	10	6	9	10	6	9
CMC	6	9	10	6	9	10	6	9	10
Crx	8	9	8	8	9	8	8	9	8
German	2	7	16	2	7	16	2	7	16
Glass	3	14	8	3	14	8	3	14	8
PID	0	10	15	0	10	15	0	10	15
TAE	5	14	6	5	14	6	5	14	6
Tic-Tac-Toe	6	9	10	6	9	10	6	9	10
Waveform	10	6	9	10	6	9	10	6	9
Vehicle	7	11	7	7	11	7	7	11	7
Mean:	**5.7**	**9.0**	**10.3**	**7.8**	**11.5**	**5.6**	**5.5**	**9.1**	**10.5**

Table 59: ANOVA results for ensembles.

Turning to the experiment with single ANNs, Table 60 shows the correlation between validation set and test set accuracy.

Data sets	Mean	Std. dev.
BLD	0.38	0.27
Cleve	0.07	0.15
CMC	0.31	0.14
Crx	0.05	0.16
German	0.07	0.10
Glass	0.44	0.18
PID	0.10	0.21
TAE	0.26	0.20
Tic-Tac-Toe	0.45	0.14
Waveform	0.36	0.42
Vehicle	0.67	0.13

Table 60: Correlation between validation and test accuracy: ANNs.

Again the correlation was very low, although the mean correlation for every data set is here at least positive. On many runs, however, the correlation was extremely small, or even negative. Therefore, even for single ANNs, it seems doubtful if there is a lot to gain from using the validation set to rank and select models. It must also be noted that sometimes the correlation coefficient is rather high because some ANNs perform extremely poorly on both validation set and test set; see one example in Figure 65. This phenomenon did not occur for the more robust ensembles. A more typical plot is shown in Figure 66 below.

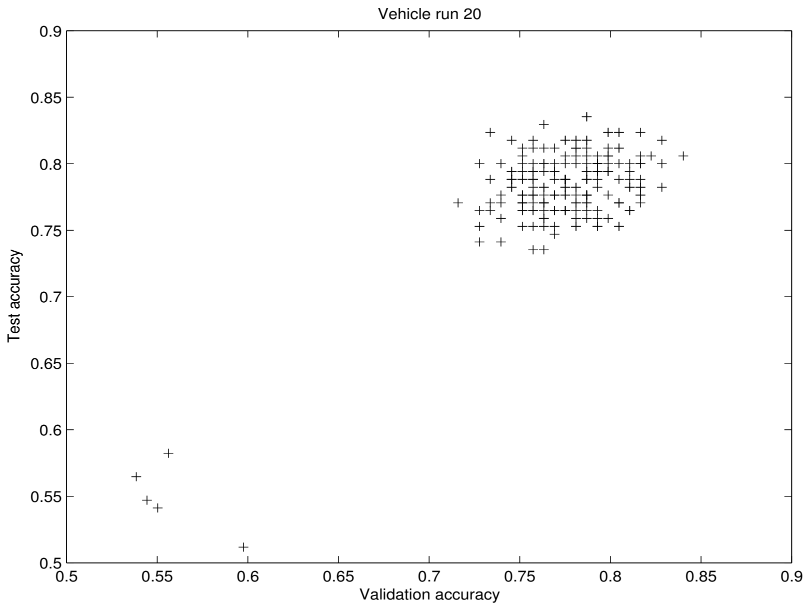


Figure 65: Test accuracy vs. validation accuracy for Vehicle. r=0.77.

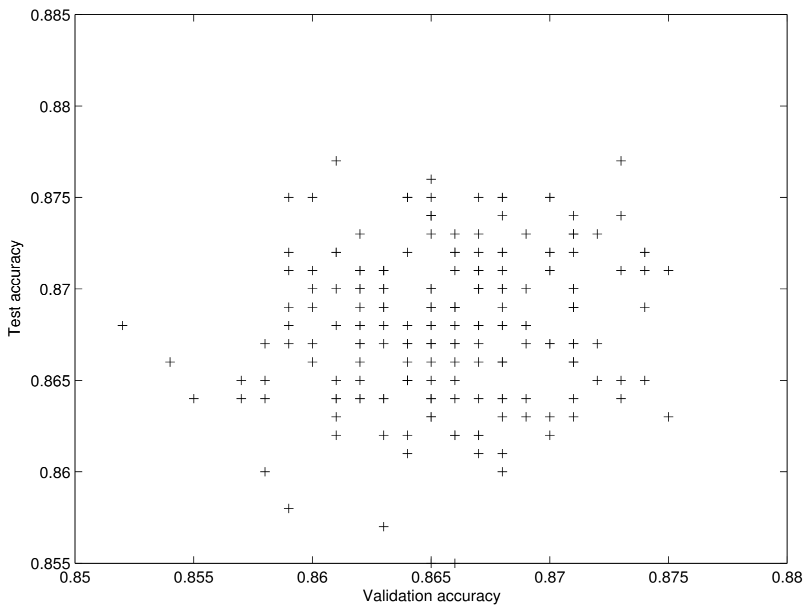


Figure 66: Test accuracy vs. validation accuracy for Waveform. r=0.13.

Similarly to the ensemble analysis, the ANNs from each run were sorted on validation set accuracy and an ANOVA analysis was performed. The result from the ANOVA is shown in Table 61.

Data sets	**1 vs. 2**			**2 vs. 3**			**1 vs. 3**		
	Low	*High*	*No*	*Low*	*High*	*No*	*Low*	*High*	*No*
BLD	0	1	19	0	1	19	0	0	20
Cleve	0	0	20	0	1	19	0	0	20
CMC	0	1	19	1	1	18	0	0	20
Crx	0	1	19	0	2	18	1	0	19
German	0	2	18	0	1	19	0	0	20
Glass	0	0	20	0	2	18	0	0	20
PID	0	0	20	0	0	20	0	0	20
TAE	1	3	16	0	1	19	0	1	19
Tic-Tac-Toe	1	1	18	0	1	19	0	1	19
Waveform	0	0	20	0	0	20	0	0	20
Vehicle	0	3	17	0	2	18	0	0	20
Mean:	**0.2**	**1.1**	**18.7**	**0.1**	**1.1**	**18.8**	**0.1**	**0.2**	**19.7**

Table 61: ANOVA results for ANNs.

In a huge majority of all runs there was no significant advantage in selecting ANNs from any specific part. Figure 67 below, where the ANNs are sorted on validation set accuracy, further highlights why there is no benefit in selecting an ANN based on validation set accuracy.

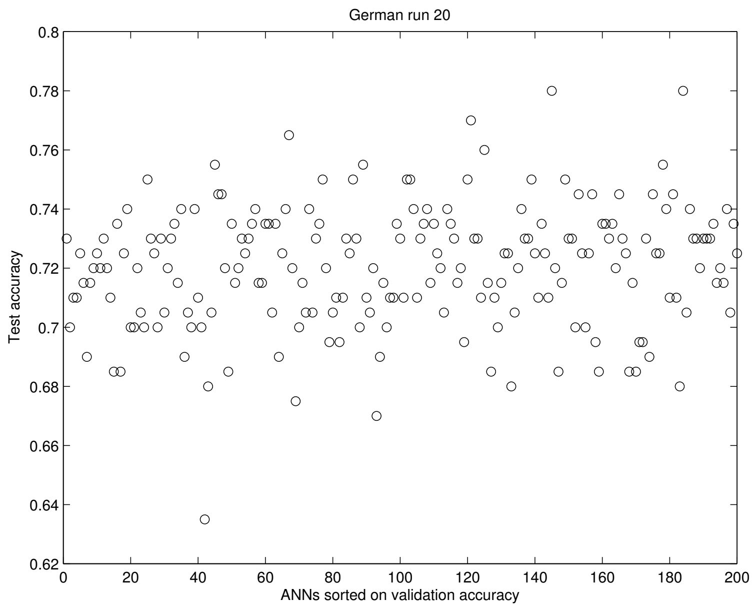


Figure 67: ANNs sorted on validation accuracy. r=0.13.

Table 62 below shows the results for experiment 3. Using only 11 data sets, the differences are far from statistically significant, the critical distance is 1.27. Nevertheless, the results support using all data for actual training instead of setting aside a holdout set for selection.

Model set	**A**	**A**	**B**	**C**
Sets	**train, early stop, val.**	**train, early stop, val.**	**train, early stop**	**train**
Selection	**val.**	**all**	**all**	**all**
BLD	0.701	0.703	0.709	0.723
Cleve	0.831	0.832	0.830	0.823
CMC	0.557	0.557	0.560	0.561
Crx	0.843	0.843	0.845	0.846
German	0.730	0.729	0.728	0.722
Glass	0.682	0.680	0.706	0.703
PID	0.764	0.764	0.760	0.757
TAE	0.525	0.522	0.558	0.575
Tic-Tac-Toe	0.804	0.809	0.820	0.868
Waveform	0.868	0.867	0.866	0.861
Vehicle	0.827	0.829	0.839	0.856
Mean rank	**2.77**	**2.77**	**2.27**	**2.18**

Table 62: Mean test set accuracy for top 5% ensembles, size 10.

Table 63 below shows the results for experiment 4. The overall picture is almost identical; i.e. it seems to be better to use all data for actual training. Using a Friedman test, the difference between *Model A all* and *Model B* turns out to be significant.

Model set	**A**	**A**	**B**	**C**
Sets	**train, early stop, val.**	**train, early stop, val.**	**train, early stop**	**train**
Selection	**val.**	**all**	**all**	**all**
BLD	0.692	0.690	0.704	0.710
Cleve	0.834	0.835	0.839	0.827
CMC	0.548	0.547	0.555	0.553
Crx	0.849	0.848	0.850	0.853
German	0.726	0.726	0.725	0.721
Glass	0.646	0.643	0.661	0.682
PID	0.778	0.777	0.778	0.768
TAE	0.539	0.548	0.591	0.614
Tic-Tac-Toe	0.800	0.803	0.826	0.872
Waveform	0.863	0.862	0.866	0.856
Vehicle	0.820	0.820	0.834	0.852
Mean rank	**2.86**	**3.18**	**1.77**	**2.18**

Table 63: Mean test set accuracy for top 5% ensembles, random size.

Figure 68 plots the mean test accuracy over all 20 runs, for the top 500 ensembles, ranked on either validation set accuracy or accuracy on all available data. The data set is Tic-Tac-Toe. Here, it is quite clear that it is more beneficial to use all available data for training. The almost non-existent difference between ensembles from model set A shown here is quite typical. The difference between selection based on *Validation* and *All* is generally very small. The main point is, however, that Model sets C and B outperform Model set A, something that holds for most data sets.

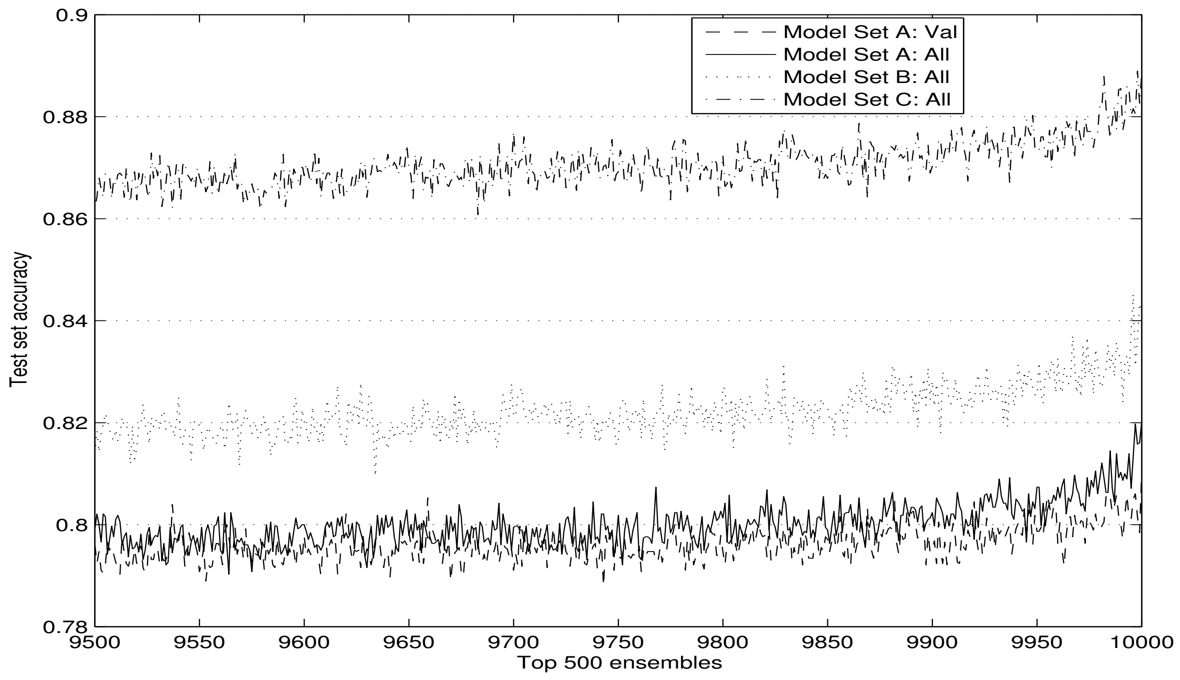


Figure 68: Comparison of model sets.

For this particular data set, it is much better to train without early stopping validation (Model set C), but for several other data sets the opposite is true. No simple rule establishing when use of early stopping is successful or not was established.

The results of experiments 3 and 4, suggest that we in the long run should expect higher accuracy if we always select ensembles based on accuracy on all available data. Nevertheless, how well a selected ensemble will perform on test data still depends, to a large degree, on chance.

8.3.3 Conclusions

The main result in this study was that correlation between validation set accuracy and test set accuracy often is very low, making selection based on validation set accuracy very doubtful. As a matter of fact, in the experiments there was, in general, absolutely nothing to gain from using an ensemble with high validation set accuracy compared to a random ensemble. It should be noted that the fact that most ensembles get very similar results does not weaken this conclusion. So, one obvious conclusion is that methods using validation set accuracy as a key criterion when selecting ensemble members must be considered dubious; the results from this study indicate that such selection procedures do not add value, compared to using a "random" ensemble.

It is slightly harder to draw any general conclusions from the experiment with individual ANNs. The main result is that it is far from certain that a specific ANN outperforming another ANN on the validation set will keep this edge on a test set. Consequently, even for simple ANNs, accuracy measured on one holdout set (here the validation set), is a rather poor predictor for accuracy on another holdout set (the test set).

The main conclusion from experiments 3 and 4 is that the best use of available data most often is to use it all, both for actual training and for ranking of models.

8.4 Study 4 – Two GEMS variants

The main purpose of this study was to further evaluate GEMS. Here, several more data sets were used and GEMS was compared to both the best setups from Study 1 and the straightforward option to combine all available ANNs. The major problem in the first GEMS study was that GEMS sometimes got very high accuracy on training and validation sets but failed to generalize to the test set.

With this in mind, a key target in this study was to somehow try to prevent the effects of overtraining.

8.4.1 Method

In this study, two different extensions to GEMS were tested. The first approach changed the training regime and the second used a different representation language. Both approaches trained the ANNs without using a validation set for early stopping. In addition, no holdout set was used to try to emulate performance on unseen data. Instead all available data was used for actual ANN training. Each ANN was, however, trained using only 70% (randomized without replacement) of all instances, in an attempt to introduce some diversity. It should be noted that this is a major difference from the earlier studies where early stopping validation was used for ANN training, and a holdout set was used when evaluating the ensembles. The results in Study 3 indicate that it might be better for GEMS to use ANNs trained on all data instead of using early stopping and holdout sets, but this remains to be verified through extensive testing. The architecture for each ANNs was again slightly randomized. For ANNs with hidden layers, the architecture was determined using the same equations as in the original GEMS study; see (84) - (86). Both approaches also use the same fitness function; see (89).

$$f = \#correct_{train} - \frac{1}{100}size \tag{89}$$

In this study, 23 data sets were used. For evaluation, GEMS was compared to both the best setups from Study 1, and ensembles consisting of all ANNs available to GEMS. To enable this comparison, the folding used was identical to the one used in Study 1. GEMS results were also compared to results for Random Forests and AdaBoost obtained in [Bre01].

Tombola training

In the first approach, two procedures aimed at decreasing the risk of overtraining were used. First of all, a length penalty was used in the fitness function to prioritize smaller ensembles. Second, a training method called *Tombola training* was introduced and used for evolution. When using Tombola training, subsets are repeatedly drawn (with replacement) from the data set in order to create several partitions, called *training groups*. Each GEMS ensemble was then evaluated on every training group; i.e. every ensemble will have one fitness value per training group. Tombola training has a couple of important parameters, which will affect the progress of the evolution. First of all, the number of training groups and what proportion of the entire data set each group should hold must be determined. Two other design choices are how often a redraw should occur and exactly how the fitness values from each training group should be combined. At least three obvious combination possibilities were identified, and they were named *mean*, *median* and *worst*. When mean combination is used, an ensemble will have the average fitness from the training groups. If median combination is used, the median value is used instead. Finally, if worst combination is used, the ensemble will have a fitness value equal to the worst fitness from the training groups. It should be noted that *worst* is a somewhat misleading word here; *best-worst* is probably more to the point.

The actual fitness (calculated for each ensemble on each training group) was based on accuracy on that training group and a small penalty for every node in the genetic program (ensemble). The settings used for Tombola training will probably have a significant effect on how evolution progresses, but the parameters were not evaluated extensively. The settings used in this study are given in Table 64 below.

Parameter	**Value**
#groups	3
Percent of all data per group	70%
Number of generations between redraws	5
Combination	Worst

Table 64: Tombola training parameters.

The number of ANNs in the pool was 20, five with no hidden layer, ten with one hidden layer and ten with two hidden layers. The GP settings are given in Table 65 below.

Parameter	**Value**
Crossover rate	0.8
Mutation rate	0.01
Population size	1000
Generations	100
Persistence	15
Creation depth	8

Table 65: GP parameters for Tombola training.

Ensemble trees

In the second approach, the representation language was changed. More specifically, the first levels of the GEMS tree could now contain splits (the conditions are expressed in original variables) similar to standard decision trees.

Naturally, the purpose was to make it possible for different (sub)ensembles to specialize on different parts of the input space. Function and terminal sets together with the exact grammar for the representation language are given in Figure 69.

```
F = {if, equals, <, >, avg, *}
T = {i1, i2, …, in, a1, a2, …, ak, C1, C2, …,Cm, ℜ}

DTree    :-      (if RExp Dtree Dtree) | BaseEns
BaseEns  :-      Avg | Ann
Ens      :-      Avg | Fact | Ann
Avg      :-      (avg Ens Ens)
Fact     :-      (* ConConst Ens)
Ann      :-      ANN result vector
RExp     :-      (ROp Conlnp ConConst) | ( equals CatInp CatConst)
ROp      :-      < | >
Catlnp   :-      Categorical input variable
Conlnp   :-      Continuous input variable
CatConst :-      Categorical attribute value
ConConst :-      ℜ
```

Figure 69: Grammar for GEMS ensemble trees.

Using this grammar, GEMS becomes extremely flexible. Now, it is possible to mix tests partitioning the data with ensembles built using either individual ANNs or other ensembles. Figure 70 below shows a sample program in the syntax used internally by GEMS.

```
(if (equals X2 1)
    (avg ann7 ann5)
    (avg (* 0.383 ann8) (avg ann2 ann6)))
```

Figure 70: Sample ensemble tree in GEMS syntax.

Most GP parameters were identical to the ones used for the *Tombola training* experiment; see Table 65. Here, however, the evolution started with less complex programs since the creation depth was six instead of eight. In addition, the ANN pool now contained 10 ANNs instead of 20.

8.4.2 Results

Table 66 below shows the main results. The results for *3-ten*, *S10*, *All_50* and *Tr6V* were taken from Study 1. *All_20* is the ensemble consisting of all 20 ANNs available to GEMS when using Tombola training. Similarly, *All_10* is the ensemble averaging all 10 ANNs available to *GEMS ensemble tree*.

Data sets	**Study 1**				**Tombola**		**Ensemble tree**		**LLS**
	3-ten	**S10**	**All_50**	**Tr6V**	**All_20**	**GEMS**	**All_10**	**GEMS**	
BLD	0.700	0.711	0.706	0.720	0.717	0.729	**0.737**	**0.737**	0.72
Cleve	0.803	0.819	**0.829**	0.797	0.818	0.826	0.809	0.809	-
CMC	0.546	0.557	0.548	**0.560**	0.541	0.553	0.543	0.544	0.57
Crx	0.854	0.853	**0.857**	0.843	0.844	0.848	0.848	0.849	-
German	0.709	0.714	0.707	0.727	0.724	0.731	0.729	**0.732**	-
Glass	0.673	0.686	0.668	0.691	0.660	0.696	0.664	**0.708**	-
Hepatitis	0.818	0.824	0.818	0.847	0.827	**0.875**	0.836	0.845	-
Horse	0.840	0.845	**0.853**	0.824	0.826	0.830	0.831	0.834	-
Iono	0.924	0.938	0.935	**0.951**	0.939	0.944	0.926	0.937	-
Iris	0.960	0.967	0.960	**0.973**	0.960	**0.973**	0.960	**0.973**	-
Labor	0.850	0.863	0.850	0.875	0.933	0.967	0.960	**0.980**	-
Led7	0.736	0.734	0.732	0.736	0.735	0.735	**0.738**	0.734	0.73
Lymph	0.777	0.782	0.806	0.806	0.831	0.853	**0.869**	**0.869**	-
PID	0.763	0.761	0.758	0.748	0.764	0.765	0.765	**0.767**	0.78
Satellite	0.838	0.865	0.848	0.865	0.907	**0.908**	0.897	0.905	0.90
Sonar	0.832	0.818	0.827	0.832	0.811	0.843	0.811	**0.853**	-
TAE	0.553	0.512	0.471	0.541	0.544	**0.606**	0.531	0.581	0.67
Tic-Tac-Toe	0.780	0.862	0.805	0.878	0.882	0.892	0.888	**0.915**	-
Waveform	**0.870**	0.867	0.865	0.867	0.860	0.857	0.866	0.855	0.85
WBC	0.967	**0.972**	0.965	0.961	0.964	0.961	0.967	0.961	0.97
Vehicle	0.831	0.842	0.831	0.836	0.841	0.845	0.841	**0.851**	0.85
Wine	0.978	0.978	0.983	0.972	0.988	**0.994**	**0.994**	**0.994**	-
Zoo	0.936	0.946	0.927	0.936	**0.973**	**0.973**	0.964	0.964	-
Average rank	**5.59**	**4.70**	**5.76**	**4.83**	**4.93**	**2.86**	**4.22**	**3.11**	
#Best	**1**	**1**	**3**	**3**	**1**	**6**	**4**	**11**	**-**

Table 66: Results for Study 4.

The overall results for the two GEMS approaches were very good. It is interesting to note that although the ensemble tree wins almost half of the data sets, the Tombola training approach still has the best average rank. Using a standard Friedman test, the CD = 2.42; i.e. the only statistically significant difference (at α= 0.05) is that both GEMS approaches perform better than *3-ten* and *All_50*; see Figure 71.

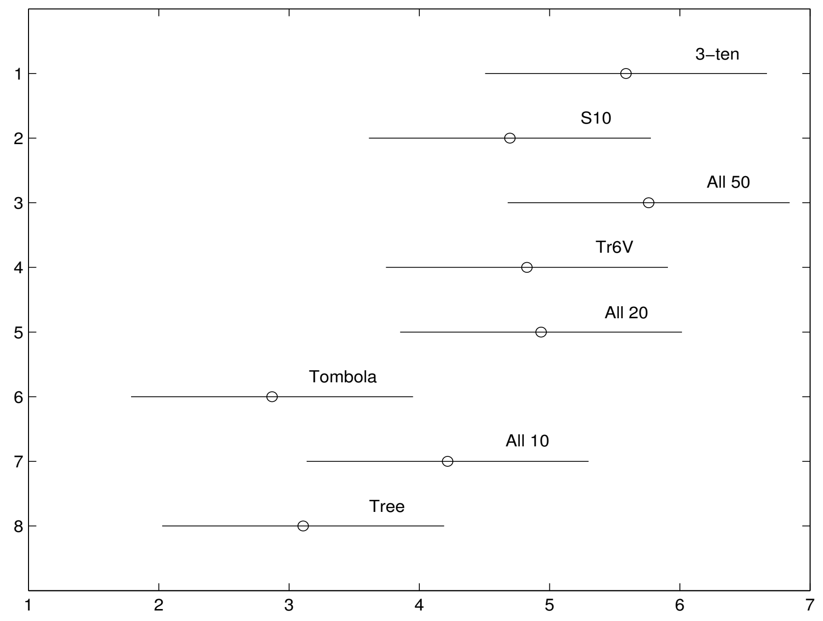


Figure 71: Friedman test Study 4.

If, however, a Bonferroni-Dunn test is used instead, comparing GEMS using Tombola training to all others, the CD instead becomes *1.94*. So, using that test, GEMS with Tombola training has significantly higher accuracy than *3-ten*, *All_50*, *Tr6V* and *All_20*. Also, the difference to *S10* is close to being significant at α= 0.05 and is significant at α= 0.1.

Although it is slightly problematic to directly compare results obtained independently, even if the data sets are identical, it is still interesting to at least compare the level of performance. With this in mind, Table 67 below compare results for GEMS and the two standard techniques Random Forest and AdaBoost. The results for Random Forest and AdaBoost were taken from Table 3 in [Bre01], and were in most cases also obtained using 10-fold cross-validation. For the exact details, as well as all parameters, see the original paper. As seen in the table, 10 of the 19 data sets Breiman used in his study were also used in this GEMS study.

Data sets	GEMS Tombola	GEMS Ensemble Tree	RF	AdaBoost
BLD	0.729	**0.737**	0.727	0.693
German	0.731	0.732	**0.772**	0.765
Glass	0.696	0.708	0.756	**0.780**
Iono	0.944	0.937	**0.945**	0.936
PID	0.765	0.767	**0.770**	0.734
Satellite	0.908	0.905	0.909	**0.912**
Sonar	0.843	0.853	**0.864**	0.844
Waveform	**0.857**	0.855	0.840	0.822
WBC	0.961	0.961	**0.969**	0.968
Vehicle	0.845	**0.851**	0.769	0.768

Table 67: Result summary for GEMS, AdaBoost and Random Forest.

When comparing the best result obtained by GEMS to Random Forest, Random Forest was clearly more accurate on two data sets (German and Glass) and slightly more accurate on another two data sets (Sonar and WBC). GEMS, on the other hand, clearly outperformed Random Forest on BLD, Vehicle and Waveform. On the last three data sets (Iono, PID and Satellite) the difference in accuracy was very small.

Comparing GEMS to AdaBoost, the picture is a little different. AdaBoost was much more accurate on Glass and German, and slightly more accurate on WBC. Here, however, GEMS was clearly more accurate on not only BLD, Waveform and Vehicle but also on PID. In addition, GEMS slightly outperformed AdaBoost on Iono and Sonar. These comparisons are summarized in Table 68 below. The interpretation is: GEMS clearly more accurate (++), GEMS slightly more accurate (+), Very small difference (0), GEMS slightly less accurate (-) and GEMS clearly less accurate (--).

Data sets	GEMS vs. RF	GEMS vs. AdaBoost
BLD	++	++
German	--	--
Glass	--	--
Iono	0	+
PID	0	++
Satellite	0	0
Sonar	-	+
Waveform	++	++
WBC	-	-
Vehicle	++	++

Table 68: Comparison with AdaBoost and Random Forest.

8.4.3 Conclusions

As seen in the experiments, both variants of GEMS are quite successful; obtaining significantly higher accuracy than several approaches evaluated in previous experiments. It is important to recognize, however, that this is partly due to the fact that the ANNs in the pool now are more accurate to start with. This, in turn, is probably caused by the fact that all data is used to train the ANNs instead of holding out data to emulate performance on novel data. At the same time, both setups used appear to have reduced the risk of overfitting.

Regarding the comparison with AdaBoost and RF, the main result is that GEMS overall obtains accuracy comparable to the two well-known techniques. Nevertheless, on a couple of data sets, GEMS is clearly outperformed, so the picture is a bit mixed.

8.5 Study 5 – Evaluating diversity measures

When designing ensembles, the prevailing opinion is that base classifier diversity is vital for how well the ensemble will generalize to new observations. Unfortunately, there is no clear definition of the key term diversity, leading to several diversity measures and many, more or less ad hoc, methods for diversity creation in ensembles. In addition, no specific diversity measure has shown high correlation with test set accuracy. The purpose of this study, was to empirically evaluate ten different diversity measures, using ANN ensembles.

8.5.1 Method

The most important purpose of this study was to evaluate the ten diversity measures, described in 5.3.1, that were initially used in [KW03]. Here, however, ANN ensembles and publicly available real data sets were used. The empirical study is divided in two main experiments, each consisting of two parts. In the first experiment, 15 ANNs were trained and all 3003 possible ensembles consisting of exactly 10 ANNs were formed. In the second experiment, a total of 50 ANNs were trained and 10000 randomized ensembles were formed. Both experiments were divided in two parts, the difference being whether an extra validation set was used or not. If no validation set was used, diversity was measured on the training set, and results were correlations between diversity measures obtained using training data and test set accuracy. When a validation set was used, this set was not used at all during ANN training. Now, however, all diversity measures were obtained using the validation set, and consequently, the results are correlations between those measures and test set accuracy. For actual experimentation, 4-fold cross-validation was used. The experiments are summarized in Table 69 below.

Experiment	#Ensembles	Train	Val.	Test	Evaluation
1a	3003	75%	-	25%	Train vs. Test
1b	3003	50%	25%	25%	Val. vs. Test
2a	10000	75%	-	25%	Train vs. Test
2b	10000	50%	25%	25%	Val. vs. Test

Table 69: Experiments in ensemble Study 5.

All ANNs used in the experiments were fully connected MLPs. In each experiment, ¼ of all ANNs had no hidden layer, ½ had one hidden layer and the remaining ¼ had two hidden layers. The exact number of units in each hidden layer was slightly randomized, but was based on the number of inputs and classes in the current data set. For an ANN with one hidden layer the number of hidden units was determined from (90) below.

$$h = \left\lfloor 2 \cdot rand \cdot \sqrt{v \cdot c} \right\rfloor \tag{90}$$

Here, *v* is the number of input variables and *c* is the number of classes. *rand* is a random number in the interval [0, 1]. For ANNs with two hidden layers, the number of units in the first and second hidden layers was h_1 and h_2, respectively.

$$h_1 = \left\lfloor \sqrt{v \cdot c}/2 + 4 \cdot rand \cdot \sqrt{v \cdot c}/c \right\rfloor \tag{91}$$

$$h_2 = \left\lfloor rand \cdot \sqrt{v \cdot c}/c + c \right\rfloor \tag{92}$$

Diversity was accomplished by using ANNs with different architectures and by training each network on slightly different data. More specifically, each ANN used a training set obtained by randomly picking instances with replacement from the available data until the number of training instances was equal to the size of the available training set. This results in that approximately 63% of available training instances are actually used. Furthermore, only 70% of the features (randomly selected) are used during training of each net. Majority voting is used to obtain the ensemble classification.

In the first experiment, all possible ensembles of size 10 out of 15 were evaluated. In the second experiment, 10000 randomly selected ensembles were evaluated. It was ensured that no duplicates and no ensembles with less than 10 ANNs were evaluated. Consequently, the ensembles in the second experiment consisted of between 10 and 50 ANNs. Both experiments used 11 data sets.

Table 70 below summarizes the ten measurements used in [KW03]. The arrow specifies whether diversity is greater if the measure is lower (↓) or higher (↑). For a description of each measure, see 5.3.1. In addition to the ten diversity measures, the correlations between training and validation accuracy and test accuracy were also measured.

Name	Abbreviation	↑/↓
Q-statistic	Q	(↓)
Correlation coefficient	P	(↓)
Disagreement	D	(↑)
Double fault	DF	(↓)
Kohavi-Wolpert	KW	(↑)
Interrater agreement	κ	(↓)
Entropy	E	(↑)
Difficulty	θ	(↓)
Generalized diversity	GD	(↑)
Coincident failure diversity	CFD	(↑)

Table 70: Diversity measures.

8.5.2 Results

The main results from the first experiment are shown in Table 71 and Table 72 below. The values tabulated in the first 11 columns are the correlation between the specific measure and test set accuracy. *Acc* is accuracy on either training or validation data. It must be noted that, in order to make the tables more readable, all correlations reported are (even for measures where lower values indicate higher diversity) between high diversity and high accuracy. The final three columns show the minimum, maximum and mean test set accuracies obtained by any ensemble.

Data sets	Acc	Q	P	DIS	DF	E	KW	κ	θ	GD	CFD	Min	Max	Avg
Crx	.08	-.10	-.10	-.15	.07	-.14	-.15	-.13	-.10	-.09	.00	.80	.89	.85
German	-.09	.55	.60	.57	.17	.59	.57	.57	.57	.55	.17	.70	.78	.74
Image	.61	-.03	.13	-.22	.58	-.24	-.22	-.03	.61	.07	-.02	.94	.97	.96
Led7	.19	-.02	.02	.01	.26	-.01	.01	.02	.03	.06	.11	.71	.75	.73
PID	-.11	.22	.26	.23	.26	.24	.23	.25	.27	.27	.24	.73	.81	.77
Satellite	.75	-.28	.11	-.38	.92	-.55	-.38	.20	.89	.36	.69	.88	.90	.89
Sick	.64	.33	.37	.36	.23	.36	.36	.39	.36	.39	.29	.94	.97	.96
Thyroid	.79	.36	.58	.40	.35	.38	.40	.53	.65	.55	.44	.97	.99	.98
Tic-Tac-Toe	.79	.78	.84	.67	.90	.66	.67	.84	.88	.90	.81	.72	.85	.79
WBC	.17	.18	.19	.12	-.04	.12	.12	.10	.13	.07	-.08	.95	.98	.96
Vehicle	.26	.23	.29	.11	.44	.14	.11	.25	.37	.37	.21	.76	.84	.80
Mean:	**.37**	**.20**	**.30**	**.16**	**.38**	**.14**	**.16**	**.27**	**.42**	**.32**	**.26**	**.83**	**.88**	**.86**

Table 71: Measures on training set. Enumerated ensembles.

Data sets	**Acc**	**Q**	**P**	**DIS**	**DF**	**E**	**KW**	**κ**	**θ**	**GD**	**CFD**	**Min**	**Max**	**Avg**
Crx	.23	-.11	-.13	-.13	.21	-.17	-.13	-.10	-.07	-.07	.22	.79	.88	.84
German	.12	-.27	-.28	-.31	.12	-.32	-.31	-.27	-.24	-.20	.18	.68	.77	.73
Image	.36	.09	.18	.02	.17	.02	.02	.07	.19	.09	.02	.93	.96	.95
Led7	.14	.00	.04	.03	.01	.00	.03	.03	.04	.04	.10	.72	.75	.74
PID	-.08	-.14	-.17	-.16	-.12	-.16	-.16	-.17	-.18	-.17	-.17	.73	.80	.77
Satellite	.51	-.07	.20	-.08	.66	-.26	-.08	.24	.65	.34	.48	.87	.90	.89
Sick	.62	-.07	-.02	-.11	.66	-.12	-.11	.06	.39	.31	.34	.91	.96	.95
Thyroid	.19	.11	.21	.11	-.05	.11	.11	.12	.24	.05	-.03	.97	.98	.98
Tic-Tac-Toe	.45	.31	.38	.29	.56	.31	.29	.38	.42	.47	.51	.68	.79	.74
WBC	.06	-.13	-.17	-.18	.25	-.18	-.18	-.16	-.01	-.15	-.03	.95	.98	.96
Vehicle	.09	.20	.18	.14	.19	.11	.14	.18	.21	.22	.24	.73	.82	.77
Mean:	**.24**	**-.01**	**.04**	**-.03**	**.24**	**-.06**	**-.03**	**.03**	**.15**	**.08**	**.17**	**.82**	**.87**	**.85**

Table 72: Measures on validation set. Enumerated ensembles.

In the first experiment, the mean correlations were remarkably low. In addition, no measure showed a positive correlation with test set accuracy on all data sets. With this in mind, it is very hard to recommend a specific measure based on the results. Another observation is that correlations between accuracy on training or validation and test set accuracy were not very high either. This supports the results found in Study 3, and of course makes it doubtful to optimize an ensemble based on training or test set accuracy. The results from the experiment with randomized ensembles are shown in Table 73 and Table 74 below.

Data sets	**Acc**	**Q**	**P**	**DIS**	**DF**	**E**	**KW**	**κ**	**θ**	**GD**	**CFD**	**Min**	**Max**	**Avg**
Crx	.28	-.09	-.07	-.11	.33	-.12	.00	-.03	.24	.12	.12	.75	.90	.86
German	-.08	.52	.59	.57	-.02	.50	.59	.55	.55	.49	-.27	.71	.82	.77
Image	.64	.01	.26	-.12	.68	-.12	-.09	.19	.71	.29	.21	.92	.97	.95
Led7	.52	.00	.04	.02	.20	.05	.13	.04	.20	.08	.22	.65	.74	.73
PID	-.08	.09	.16	.12	.15	.13	.15	.13	.18	.15	.07	.72	.81	.77
Satellite	.86	-.10	.19	-.14	.88	-.24	-.09	.13	.84	.24	.22	.83	.90	.89
Sick	.65	.21	.35	.22	.13	.21	.24	.30	.47	.36	.04	.92	.97	.96
Thyroid	.55	.21	.32	.21	.04	.20	.23	.29	.46	.32	.06	.94	.99	.98
Tic-Tac-Toe	.82	.72	.82	.71	.85	.57	.64	.83	.78	.87	.71	.68	.87	.77
WBC	.04	.04	.10	.04	.02	.02	.05	.06	.15	.09	-.02	.92	.98	.96
Vehicle	.41	.24	.33	.15	.41	.20	.26	.29	.51	.41	.31	.75	.87	.81
Mean:	**.42**	**.17**	**.28**	**.15**	**.33**	**.13**	**.19**	**.25**	**.46**	**.31**	**.15**	**.80**	**.89**	**.86**

Table 73: Measures on training set. Randomized ensembles.

Data sets	Acc	Q	P	DIS	DF	E	KW	κ	θ	GD	CFD	Min	Max	Avg
Crx	.42	-.15	-.16	-.20	.39	-.23	-.11	-.12	.12	.05	.20	.73	.90	.86
German	.17	-.05	-.05	-.07	.08	-.01	.06	-.05	.13	-.02	.16	.68	.80	.75
Image	.59	.20	.38	.05	.48	.04	.10	.33	.63	.42	.36	.92	.97	.95
Led7	.40	.00	.02	.00	.15	.04	.13	.02	.20	.07	.20	.66	.74	.72
PID	.10	-.02	-.04	-.05	.07	-.03	.01	-.03	.05	-.01	.12	.68	.77	.72
Satellite	.80	.08	.45	.08	.80	-.03	.17	.43	.83	.51	.43	.85	.90	.89
Sick	.67	.14	.23	.12	.27	.12	.13	.21	.35	.27	.05	.89	.97	.95
Thyroid	.43	.20	.25	.19	-.05	.19	.20	.23	.27	.21	-.03	.93	.99	.98
Tic-Tac-Toe	.62	.48	.55	.46	.64	.37	.46	.54	.56	.61	.33	.65	.83	.73
WBC	.38	.08	.10	.05	.07	.02	.06	.11	.24	.17	.02	.77	.99	.97
Vehicle	.18	-.08	-.05	-.08	.12	-.02	.04	-.05	.13	.01	.14	.70	.83	.78
Mean:	**.43**	**.08**	**.15**	**.05**	**.28**	**.04**	**.11**	**.15**	**.32**	**.21**	**.18**	**.77**	**.88**	**.85**

Table 74: Measures on validation set. Randomized ensembles.

The overall results are quite similar to the first experiment. Here though, the correlation between accuracy on the training and validation data and test set accuracy was slightly higher. The major reason for this is the fact that some ensembles in this experiment turned out to have very low accuracy. In the first experiment, all ensembles had acceptable accuracy.

Regarding the different diversity measures, most correlations were still very low. It could, however, be noted that *Double fault* and *Difficulty* overall showed slightly higher correlations than the rest. On individual folds, the correlation between a diversity measure and test set accuracy was sometimes rather high. Figure 72 below shows an example where more diverse ensembles also were more accurate. The diversity measure used is *double fault*.

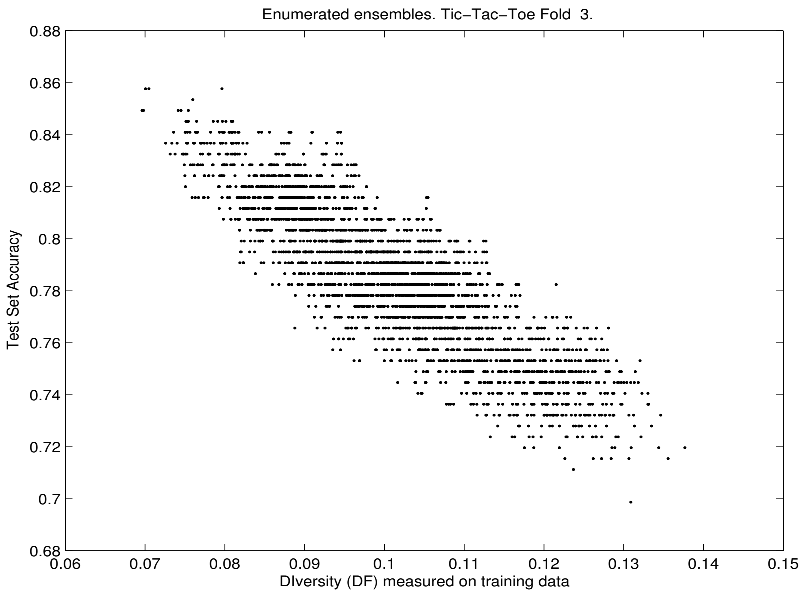


Figure 72: A sample fold with very strong correlation (-0.85).

Unfortunately, the picture is completely different on most folds. In Figure 73 below the correlation between *double fault* and test set accuracy is a more typical -0.27.

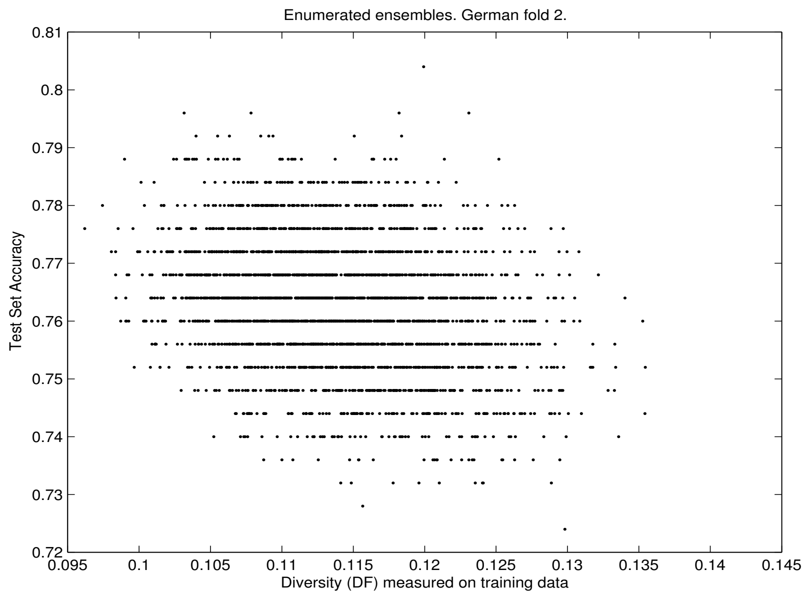


Figure 73: A sample fold with typical correlation (-0.27).

It could be argued that correlation over the entire data set is not really the most important criterion. After all, we would eventually have to choose one ensemble to apply on unseen data. So, an interesting question is whether it is beneficial or not to select an ensemble with higher diversity on the training data, compared to picking a random ensemble. To emulate this, the mean test set accuracy was calculated for the 1% most diverse ensembles; i.e. 30 ensembles in the enumerated experiment and 100 ensembles in the randomized experiment. Diversity was measured on the training set using *double fault* and *difficulty*. Test set accuracies are presented in Table 75 below.

Data sets	Enumerated ensembles			Randomized ensembles		
	All	DF	θ	All	DF	θ
Crx	**0.848**	**0.848**	**0.848**	**0.855**	0.845	0.853
German	0.737	**0.738**	0.732	**0.770**	0.738	0.761
Image	0.957	0.962	**0.964**	0.952	0.956	**0.958**
Led7	0.732	**0.734**	0.732	**0.726**	0.724	0.720
PID	**0.773**	0.766	0.762	**0.774**	0.754	0.763
Satellite	0.891	**0.897**	**0.897**	0.885	0.890	**0.892**
Sick	0.956	**0.960**	**0.960**	**0.957**	0.950	0.955
Thyroid	0.978	0.982	**0.983**	**0.978**	0.974	0.977
Tic-Tac-Toe	0.788	**0.831**	0.830	0.768	0.810	**0.819**
WBC	0.964	**0.966**	0.964	**0.962**	0.954	**0.962**
Vehicle	0.800	**0.807**	0.805	0.810	0.815	**0.818**
Mean:	**0.857**	**0.863**	**0.861**	**0.858**	**0.855**	**0.862**

Table 75: Comparing the top 1% diverse ensembles with all.

In the experiment with enumerated ensembles, where all ensembles consist of exactly 10 ANNs, it is clearly beneficial to select an ensemble with high diversity. When using randomized ensembles, it is, on the other hand, better on most data sets to pick a random ensemble instead of a more diverse. It should be noted, however, that especially the *double fault* measure is sensitive to the number of classifiers in the ensemble.

8.5.3 Conclusions

The results in the study support the claim that there is no diversity measure strongly correlated with test set accuracy. The study also showed that the correlation between accuracy measured on training or validation data and test set accuracy is rather low. These results clearly challenge ensemble design techniques where diversity is explicitly maximized or where ensemble accuracy on a holdout set is used for optimization.

Based on this, it appears to be quite a challenge to come up with an algorithm for ensemble design that would constantly outperform straightforward design choices. A typical example would be the basic technique used in this study; i.e. simply combining a sufficient number of ANNs; each trained using varied architectures, instances and features in order to obtain implicit diversity.

8.6 Study 6 – A unified GEMS

The main purpose of this study was to increase GEMS accuracy and compare the performance to AdaBoost and Random Forest. To achieve higher accuracy, three modifications were introduced:

- Different recoding schemes were evaluated and used on both continuous and categorical input variables, in order to increase ANN accuracy.
- Predictions from fixed ensembles on test set instances were used to guide the GEMS search. This is similar to G-REX using oracle data.
- A diversity measure (double fault) was used as a part of the fitness function.

In addition, all data sets used were now stratified. This is in contrast to previous G-REX and GEMS studies, where the folding used was completely randomized; i.e. without stratification.

8.6.1 Method

To enable a fair comparison with AdaBoost and Random Forest, the experimentation included 13 of the 14 UCI data sets used in [Bre01]. Many of these UCI data sets were also used by Bing, Hsu and Ma in [BHM98], who applied the shuffling utility in C4.5 to stratify the data sets. Since the stratified data sets are available at the research group web page[1], these pre-stratified versions were used in this study. For the data sets where stratified versions were not available, the built-in functionality to stratify data sets in the data mining workbench Weka[2] was used to obtain stratified data sets.

Initial experimentation was performed on all data sets to determine whether it would be beneficial to recode the inputs. More specifically, categorical variables without an ordering were recoded using a localist representation; i.e. one input unit per value. Both continuous variables and ordered categorical variables were recoded using thermometer coding and by dividing each input in ten equally sized intervals; i.e. equal width discretization. All in all, five different recoding schemes were evaluated for each data set:

- **Org:** No recoding is used.
- **Cat:** Only categorical variables are recoded. The recoding uses either the localist coding or thermometer coding depending on whether the variable is ordered or not.

[1] http://www.comp.nus.edu.sg/~dm2/p_download.html

[2] www.cs.waikato.ac.nz/ml/weka

- **Con:** Only continuous variables are recoded using thermometer coding.
- **CatCon:** Both categorical and continuous variables are recoded as described above.
- **All:** All input variables are recoded but the original variables are also kept.

Table 76 below shows the coding found to be most successful for the data sets used in this experiment.

Data sets	Coding
BLD	Org.
Ecoli	Org.
German	All
Glass	All
Image	Org.
Iono	All
PID	Org.
Satellite	All
Sonar	All
Waveform	Org.
WBC	All
Vehicle	Org.
Votes	Org.

Table 76: Codings used in Study 6.

In this experiment, the pool consisted of 20 ANNs. Five ANNs had no hidden layer, 10 had one hidden layer and five had two hidden layers. The number of hidden units was randomized using the heuristics described in Study 5. ANN training, including the use of available training data was also performed identically to Study 5.

Since previous experimentation has shown fixed ensembles to be very robust, almost always producing high accuracy, a technique similar to G-REX using oracle data was tried. More specifically, a term was introduced in the fitness function increasing GEMS fitness for all test instances predicted identically to the *All* ensemble; i.e. the ensemble averaging the predictions from all 20 ANNs in the pool. In addition, another term was added to the fitness function, increasing fitness for higher diversity; measured using the *double fault* measure. The exact fitness function used is shown in Figure 74 below. The coefficients were found from very limited initial experimentation, so they should not be considered optimized.

$$f = \#correct_{train} + 0.5 \cdot \#\,predicted\ identically\ to\ All_{test} - 0.1 \cdot DF \qquad (93)$$

Figure 74: Fitness function used in Study 6.

With the exception of the fitness function, GEMS used standard settings; see Table 54. For evaluation, 10-fold cross validation was used, except for Satellite and Waveform, where supplied training and test sets were used.

8.6.2 Results

Table 77 below shows the main results. *All* is the ensemble consisting of all 20 ANNs. *Top* is an ensemble created in the following way: first all ANNs are sorted on training accuracy, then 19 different ensembles are created by adding one ANN at a time, starting with the most accurate ANN. So, the first ensemble consists of the two most accurate ANNs, the second ensemble of the three most accurate ANNs, and so on until the last ensemble consisting of all 20 ANNs. Finally, after all 19 ensembles are created the most accurate of them is selected as the *Top* ensemble.

Data sets	**Top**		**All**		**GEMS**		**AdaBoost**		**RF**	
	Acc.	**Rank**	**Acc.**	**Rank**	**Acc.**	**Rank**	**Acc.**	**Rank**	**Acc.**	**Rank**
BLD	0.688	5	0.729	2	**0.735**	1	0.693	4	0.727	3
Ecoli	0.858	4	**0.873**	1.5	**0.873**	1.5	0.852	5	0.871	3
German	0.771	4	0.777	2	**0.779**	1	0.765	5	0.772	3
Glass	**0.810**	2	**0.810**	2	**0.810**	2	0.780	4	0.756	5
Image	0.979	3	0.977	4	0.980	2	**0.984**	1.5	**0.984**	1.5
Iono	0.937	4.5	0.937	4.5	0.943	2	0.936	3	**0.945**	1
PID	0.771	3	0.783	2	**0.786**	1	0.734	5	0.770	4
Satellite	0.904	3.5	0.905	2	0.904	3.5	**0.912**	1	0.903	5
Sonar	0.865	2	0.850	4	**0.870**	1	0.844	5	0.864	3
Waveform	0.852	3	**0.867**	1	0.862	2	0.822	5	0.840	4
WBC	0.968	4	0.968	4	**0.970**	1	0.968	4	0.969	2
Vehicle	0.839	2	0.836	3	**0.846**	1	0.768	5	0.769	4
Votes	0.968	2	0.965	3	**0.972**	1	0.952	5	0.959	4
Mean:	**0.862**	**3.23**	**0.867**	**2.69**	**0.872**	**1.54**	**0.847**	**4.04**	**0.856**	**3.27**

Table 77: Results Study 6.

Using a Friedman test followed by a Bonferroni-Dunn test to compare GEMS against the other algorithms, the CD (for α=0.05) is 1.55. So, the result is that GEMS was significantly more accurate than all other algorithms except *All*. At the same time, it should be noted that GEMS outperformed *All* on 9 of 13 data sets.

8.6.3 Conclusions

First of all, recoding of input variables, together with the fact that the data sets were stratified, clearly raised ANN accuracy. Second, the use of *oracle data*, here the predictions from the *All* ensemble, appears to have limited GEMS overtraining. Finally, the inclusion of a diversity measure in the fitness function seems to improve GEMS ability to generalize. In particular, it is interesting to note that GEMS on a large majority of all single folds outperforms the *All* approach. Typically, the difference in accuracy is very small, but GEMS almost always has the edge. Additional experiments, not presented in detail, showed that without the diversity measure in the fitness function, GEMS very often had identical accuracy to *All*. All in all, the most important result is that, taken together, these measures were sufficient for obtaining accuracy significantly higher than both AdaBoost and Random Forest.

Chapter 9

Conclusions and future work

In this chapter, the main conclusions are presented together with suggestions for future work.

9.1 Conclusions

The conclusion section consists of four separate subsections. The first subsection presents the main conclusions regarding ensembles and ensemble creation. This subsection thus relates to questions 1.1-1.3 in the thesis problem statement.

The second subsection presents the main conclusions concerning rule extraction and the accuracy vs. comprehensibility trade-off. This subsection thus relates to questions 2.1-2.5.

The third and fourth subsections present conclusions from the evaluation of the GEMS and G-REX algorithms, respectively.

9.1.1 Ensembles and ensemble creation

The first question addressed in this thesis was (question 1.1 from the problem statement):

What is the relationship between diversity and ensemble accuracy on novel data? More specifically; is it beneficial to use a diversity measure as part of the optimization function when generating or searching for an ensemble?

Below are the main conclusions related to this question:

- As described in 5.3, it is very hard to use the theoretical results about diversity to design an algorithm for ensemble creation. As a matter of fact, when using a zero-one loss function, there is no clear analogy to the bias-variance-covariance decomposition used for regression problems. So, decomposing the classification error into error rates of the individual classifiers and a diversity term is currently not possible.

- Because of this, there is no clear definition of the key term diversity, leading to several diversity measures which are often used in more or less ad hoc methods for diversity creation. Most importantly, as discussed in 5.3, no specific diversity measure has shown high correlation with test set accuracy. In the study reported in 8.5, ten different diversity measures were empirically evaluated, using ANN ensembles. The main result was that all diversity measures, in this study too, showed low or very low correlation with test set accuracy. Nevertheless, two measures; *double fault* and *difficulty* had slightly higher correlations compared to the other measures.
- Despite the discouraging result in 8.5, the inclusion of the measure *double fault* in the GEMS optimization function turned out to be beneficial; see 8.6.
- Most algorithms for ensemble creation do not explicitly target diversity. Instead different techniques are used to introduce implicit diversity. As described in 5.2, implicit methods typically manipulate either the training data or some learning parameter. For ANN ensembles, some diversity is introduced just by normal randomization of the weights. In addition, ANN diversity is often produced by supplying each ANN with a slightly different training set, or by using different architectures for each ANN.

The second question addressed in this thesis was (question 1.2 from the problem statement):

How strong are straightforward approaches, like averaging a fixed number of neural networks, and how important is implicit diversity for such approaches?

The main conclusion related to this question is:

- Even if diversity is not explicitly targeted, implicit diversity as a result of slightly different architectures or training using different parts of the data set, is clearly beneficial. As seen in 8.4, even straightforward techniques, like averaging a fixed number of ANNs, can result in very high accuracy. The requirement is, as demonstrated in 8.1, that the individual ANNs should be accurate and at least slightly diverse. Although this is not a very surprising result, it should be reassuring for practitioning data miners. As a matter of fact, even if more complex techniques like GEMS do obtain significantly higher accuracy, the difference measured in, for instance, extra instances correctly classified, is actually quite small.

The third question addressed in this thesis was (question 1.3 from the problem statement):

How should available data be used when constructing ensembles? In particular; is it advantageous to use ensemble accuracy on a part of the data set not used when creating the base models for ranking of ensembles, or as part of an optimization function?

The main conclusions related to this question are:

- Several techniques set aside a part of the available data and use accuracy on this holdout set for optimization. As clearly shown in 8.3 and 8.5, the correlation between accuracy on such a holdout set and another fresh set of data (a test set) is most often very low. The experiments also show that there is, in general, very little to gain in test set accuracy from choosing an ensemble with high validation accuracy. Because of these results, techniques using accuracy on a validation set to rank either ensembles or ensemble members must be considered doubtful.
- The results in 8.3 show that it is most often beneficial to use all available data for ANN training. This does not mean that every ANN must train on all data, but rather that no data should be wasted on emulating performance on unseen data.
- As seen in 8.6, the input coding used is very important for ANN accuracy. Unfortunately, there is no simple heuristic based on, for instance, data set characteristics determining when to use a specific coding. Instead, initial experimentation is needed.

9.1.2 Rule extraction

The fourth question addressed in this thesis was (question 2.1 from the problem statement):

How significant is the difference in accuracy between opaque and transparent models; i.e. how severe is the accuracy vs. comprehensibility trade-off?

The main conclusion related to this question is:

- ANNs, and especially ANN ensembles, obtain very high accuracy in all studies reported. This holds both for experiments using a large number of publicly available data sets and the Impact of Advertising case study. Specifically, ANN ensembles clearly outperform standard methods producing transparent models like multiple linear regression and decision trees. This result is perhaps most obvious in the study presented in 6.5.1, where the opaque ANN ensemble model outperforms the best decision tree on 15 of the 17 data sets used. These observations support the claim that the accuracy vs. comprehensibility trade-off exist; i.e. that the use of techniques producing transparent models all but guarantees a loss of accuracy.

The fifth question addressed in this thesis was (question 2.2 from the problem statement):

What are the relevant criteria for evaluating rule extraction algorithms?

Below are the main conclusions related to this question:

- The six identified criteria normally used to evaluate rule extraction algorithms are accuracy, comprehensibility, fidelity, scalability, generality and consistency. These criteria are discussed in 4.2.2 and 6.3.
- In the experiment reported in 6.5.3, the size of the extracted model was used to measure comprehensibility. More specifically, two different measures; number of questions and number of tests were used. This is a common simplification which is used to produce numeric measures. In practice, however, comprehensibility is a much more complex criterion; see 6.3.2.
- In this thesis, it is argued that fidelity very rarely should be maximized; see 6.3.3. An extracted model with extremely high fidelity could very well indicate overfitting. This is obviously the case in the experiment where RX is compared to G-REX; see 6.5.4. Despite the fact that RX almost always has significantly higher fidelity, G-REX is more accurate on a large majority of the data sets. So, based on this, fidelity should always be evaluated together with accuracy and comprehensibility.
- Generality is a complex criterion, lacking a clear definition. In 6.3.5, it is suggested that a rule extraction algorithm showing high generality should be applicable to a variety of opaque models, should be able to

handle various predictive modeling problems and should have the ability to extract rules in several representation languages.

- The subtle question how consistency should be measured is discussed in 6.3.6. The approach opted for in the experimentation was to compare predictions from extracted models. An important distinction between the two measures *intra-rule* and *inter-rule* consistency was introduced in 6.5.5. Intra-rule consistency measures the similarity between models extracted from a common opaque model, while inter-rule consistency measures the similarity of models extracted from different opaque models built for the same problem. Naturally, intra-rule consistency only makes sense for non-deterministic rule extraction techniques.
- It is very questionable if consistency, as currently defined, should be recognized as a criterion for evaluating rule extraction algorithms. As argued in 6.5.5, the relevant demand regarding consistency is instead that all models produced should have high and similar accuracy. If this is possible, and the extracted models are still different, the implication is that there are several, equally accurate, ways of describing the relationship found. Clearly, it should not be considered a drawback for a rule extraction algorithm to be able to find more than one accurate model.

The sixth question addressed in this thesis was (question 2.3 from the problem statement):

How well do existing techniques meet the criteria?

The main conclusions related to this question are:

- As discussed in section 4.2.2, all open-box methods for rule extraction will have severe problems regarding scalability and generality. The problems with scalability come from the fact that open-box methods must consider an exponential number (measured as incoming arcs) of possible combinations for each node. The problem with generality is even more apparent, since open-box techniques typically are specialized on a specific architecture (normally MLPs) and restricted to one representation language. From this it follows that all open-box method must fail to meet at least the scalability and the generality criteria. In this thesis, the rule extraction algorithm RX is studied as a typical example of an open-box method; see 4.3.1. Regarding scalability, RX relies heavily on successful pruning. If pruning is unsuccessful, RX will not terminate in a reasonable time. If a more severe pruning is forced, this will often lead to significantly worse accuracy; see 6.5.4. Regarding generality; RX extracts rules exclusively from single MLPs, which must be considered a critical limitation, since actual predictive models rarely consist of a single MLP. Finally, RX does not prioritize compact representations, which inevitably will lead to rather large rule sets and therefore poor comprehensibility.

- In this thesis, TREPAN is used as a typical example of a black-box method (for a discussion see 7.3.3). In the Impact of Advertising study, the original TREPAN implementation produces trees with generally higher accuracy than C5.0. The decision trees extracted by TREPAN are, however, rather complex; i.e. comprehensibility is limited. In the experiment where the second, third-party, implementation of TREPAN is used, the picture is slightly different; see 6.5.4. Here, TREPAN is often outperformed by either C5.0 or CART. This could arguably be caused by poor choices regarding parameter settings, but is nevertheless worth noting. The generality, at least for the original publicly available source code, is questionable. Not only is original TREPAN restricted to classification problems, but also requires the underlying model to be a single MLP. The modification of the third-party reimplementation of TREPAN used is, on the other hand, a true black-box approach; i.e. it supports rule extraction from any opaque model. Unfortunately it is currently restricted to binary classification problems. Finally, it should also be noted that the trade-off between accuracy and comprehensibility is not handled explicitly by TREPAN, but to a large degree left to the user. TREPAN and RX are further evaluated and compared to G-REX in 9.1.4 below.

The seventh question addressed in this thesis was (question 2.4 from the problem statement):

Is it possible to constantly obtain higher accuracy and comprehensibility by using rule extraction, compared to techniques directly inducing decision trees from the data set?

The main conclusion related to this question is:

- A very important experimental result, clearly shown in 6.5.1 and 6.5.2, is that rules extracted by G-REX using opaque models almost always have higher accuracy than techniques producing transparent models directly from the data set; i.e. decision trees. In the experiments, using 23 data sets, G-REX has significantly higher accuracy than both C5.0 and CART.

An equally important result from the same experiments is that the rules extracted by G-REX also are significantly more compact than C5.0 and CART rules. Based on these two results, it follows that there is no reason to choose a less accurate technique just to obtain transparent models. The conclusion is that a model created by a high accuracy technique followed by G-REX rule extraction normally will have higher performance (measured as accuracy *and* comprehensibility) than transparent models induced directly from the data set.

The eighth and final question addressed in this thesis was (question 2.5 from the problem statement):

Can unlabeled data instances be used to increase accuracy when performing rule extraction? More specifically; would rules extracted using opaque model predictions provide better explanations of the predictions made?

The main conclusion related to this question is:

- As shown in 6.5.2, the use of test set data instances together with predictions from the opaque model ("oracle data") when performing rule extraction, was very successful. The addition of oracle data increased the accuracy on 15 of 17 data sets. The technique suggested means that the same novel data instances used for actual prediction also are used by the rule extraction algorithm. Naturally, this requires the problem to be one where predictions are made for sets of instances rather than one instance at the time. The overall implication of the very high accuracy obtained using oracle data is that rules extracted in this way will explain the predictions made on the novel data better than rules extracted using training data only.

9.1.3 GEMS evaluation

- The results obtained in the studies reported in 8.4 and 8.6 make the GEMS concept; i.e. to use GP to construct ensembles by combining ANNs, interesting. Since GEMS is really using smaller ensembles as building material, the most correct description is that GEMS combine smaller ensembles into larger. The experiments nevertheless show that it is very important for GEMS to have accurate ANNs to start with.

- As seen in 8.4 and 8.6, the accuracy for GEMS is most often higher than all straightforward techniques like averaging all available ANNs or averaging the *k* best.

- The main problem for GEMS is overfitting. As demonstrated in 8.4, this problem can be reduced by using Tombola training or using trees with splits.
- Another problem for GEMS is consistency. Two runs may obtain similar training accuracy but very different test accuracy. One way of increasing robustness is to use the predictions from individual base models as a guide. This technique, which is similar to G-REX using oracle data, obtained very good results in 8.6.
- With the last approach, called *unified GEMS*, the obtained accuracy is significantly higher than the results reported for AdaBoost and Random Forest by Breiman; see 8.6.

9.1.4 G-REX evaluation

From the evaluation of G-REX against the criteria proposed by Craven and Shavlik, it is obvious that all demands except consistency are met to a reasonable degree (see 6.5 but also 7.5.5).

- The *accuracy* of G-REX is, especially when oracle data is used, normally only slightly worse than that of the underlying opaque model. Specifically, G-REX obtains higher accuracy than both techniques directly inducing transparent models from the data set and other rule extraction algorithms.
- Regarding *comprehensibility*, G-REX often finds very compact representations. The extracted rules are, as shown in the experiments, significantly more compact than rules induced by CART and C5.0. Even more importantly, it is extremely rare that a specific extracted rule is too complex to allow human inspection and understanding.
- *Fidelity* is not the prime target for G-REX, but as seen in 6.5.1 and 7.4.4, the fidelity is still rather high. More importantly though, fidelity is on the same level for both training and test sets.
- The *scalability* of G-REX, as a black-box method, is only dependent of the size of the data set, not factors concerning the underlying representation. It should be noted, that scalability has not been explicitly targeted during experimentation. Nevertheless, regarding time scalability, no specific G-REX run in this thesis did take more than minutes. As a matter of fact, the ANN training, especially when using ensembles, normally takes longer

time than the actual rule extraction. When it comes to how well G-REX performance scales up to data sets with several classes and many input variables, the picture is a bit mixed. The impression is, though, that G-REX normally performs better when the input variables are continuous rather than categorical. Using the length punishment, G-REX also has some problems when there are many classes. Despite this, the results obtained by G-REX, together with the fact that the data sets used are varying in size and difficulty, show that G-REX has an acceptable scalability.

- Concerning *generality*, G-REX is very versatile since it acts on data sets and ignores the actual underlying architectures. G-REX can be applied to many different types of models and generate a multitude of representations, something clearly demonstrated in all case studies.
- Finally, *consistency* appears to be the major drawback for G-REX. As seen in 6.5.5, both intra-model and inter-model consistency is normally less than 90%. It is, however, also important to realize that it is not entirely obvious that perfect consistency is something to strive for. The author's opinion is that consistency as an isolated entity could very well be deceptive. Why should it be considered better to always obtain exactly one rule, if it is in fact possible to obtain several others, every one more accurate? This issue is discussed further in 6.5.5 and in the future work section below; see 9.2.1.

To summarize, several key results have been identified during the construction and evaluation of G-REX. The key arguments for G-REX are:

- The possibility to tailor the fitness function directly based on demands on the extracted rules; i.e. in terms of accuracy, fidelity and comprehensibility, making it possible to explicitly control the trade-off between accuracy and comprehensibility.
- The option to extract rules from arbitrary opaque models, including ensembles.
- The many different representation languages available, something achieved by just changing the function and terminal sets.
- The fact that G-REX can handle both classification and regression problems.

G-REX compared to RX and TREPAN

An outright quantitative comparison between the three methods has been performed in one study using several UCI data sets (see 6.5.4) and one experiment in the Impact of Advertising case study; see 7.4.4. From results obtained, and the basic construction of the algorithms, it is possible to draw some general conclusions.

- Regarding accuracy, G-REX was significantly more accurate than both TREPAN and RX.
- All techniques supply transparent models, but RX, lacking the ability to prioritize compact representations, generally found longer, less comprehensible rules. Although TREPAN produced significantly less compact rules than G-REX, this difference is less interesting in practice. The overall conclusion is instead that both TREPAN and G-REX normally will obtain quite comprehensible rules.
- RX will most often have the highest fidelity. This is no surprise since RX explicitly targets only fidelity. The difference between TREPAN and G-REX is often small, but overall G-REX shows slightly higher fidelity. This is somewhat surprising since G-REX balances fidelity against rule size. Whether this would hold in a larger study or is, at least partly, due to the parameter settings chosen for TREPAN remains to be verified. At the same time, it must be noted, that fidelity should be regarded mainly as a tool, for both G-REX and TREPAN, to obtain high accuracy.
- Scalability is obviously the Achilles heel of open-box methods, including RX. If the network is not extensively pruned, rule extraction is not even possible, as seen in the experimentation. TREPAN and G-REX do not have this problem. Regarding computational efficiency, RX is clearly outperformed by the black-box methods. With the experimentation setup used, RX required at least several minutes (often hours and sometimes days) for a problem handled within seconds by G-REX or TREPAN. Although both G-REX and TREPAN are quite computationally intensive, it is fair to say that the training of the opaque model, especially if it is an ANN ensemble, most often will take much longer time than the actual rule extraction. Having said that, no really large data set has been used for experimentation in this thesis. In addition, exactly how well G-REX will scale up to really large input spaces, possibly having hundreds of features, remains to be verified.

- Regarding generality, only G-REX is capable of handling both classification and regression problems. In addition, both RX and original TREPAN operate on MLPs only, while G-REX can extract from any opaque model. G-REX ability to use a variety of representation languages also increases the generality. With the modified third-party implementation of TREPAN, generality is vastly improved, but TREPAN is still much less general than G-REX, especially when it comes to representation languages.
- Consistency, finally, is the only criterion where G-REX is clearly inferior to TREPAN and RX. Both RX and TREPAN will always have perfect consistency since the algorithms are totally deterministic. If, on the other hand, G-REX extracts rules several times from the same opaque model, this will typically result in a number of quite different rule sets; see 6.5.5.

Table 78 summarizes the advantages and drawbacks of each algorithm. The scale used is: Very good (++), Good (+), Acceptable (0), Poor (-) and Very Poor (--).

Criteria	**RX**	**TREPAN**	**G-REX**
Accuracy	+	+	++
Comprehensibility	0	+	++
Fidelity	++	+	+
Scalability	--	0	0
Generality	--	0	++
Consistency	++	++	-

Table 78: Comparison of the different algorithms for rule extraction.

The most interesting observations are:

- G-REX meets all criteria but fidelity and consistency at least as well as both RX and TREPAN.
- G-REX clearly meets the criteria accuracy, comprehensibility and generality better than both RX and TREPAN.

9.2 Future work

In this section, future work that has been identified but not considered during the course of writing this thesis is presented.

9.2.1 Future work regarding rule extraction and G-REX

This subsection is divided in several parts, each focusing on a specific aspect.

Representation languages

In the study [JKN04] (see 7.5), G-REX was used on regression problems. The representation language was, however, chosen with the purpose of comparing G-REX to CART. For regression, there are several other, potentially more expressive, representation languages that should be evaluated. One example is model trees; i.e. regression trees with linear expressions in the leaves.

Regarding representation languages for classification, several functions have been implemented in G-REX, but without any extensive evaluation; e.g. *1-of-N rules*, *M-of-N rules* and *fuzzy rules*. A related question is which kind of tests that should be allowed. Normally, G-REX uses only tests where an input variable is compared to a constant value. A straightforward option is, however, to also allow comparisons between input variables; i.e. tests like *(< X1 X2)*. For most data sets, such tests make little sense, but for some problems, typically where the input variables are related, it could be very powerful. For an artificial example, see Figure 75 below, where the G-REX test *(< X1 X2)* separates the two classes perfectly. Here, it would be very hard to express the decision line without the possibility to compare X1 against X2.

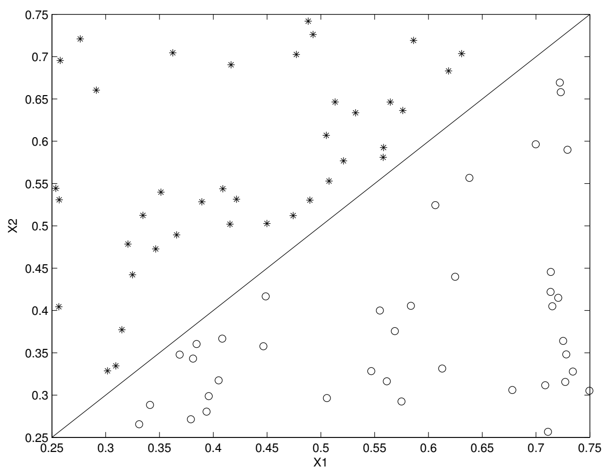


Figure 75: A decision line requiring tests between input variables.

Although comparisons between inputs could, in theory, solve the previous problem, it would fail on the similar problem given in Figure 76 below. Here the decision line requires the use of arithmetic operators. The simplest way of expressing the decision line using G-REX syntax is *(< (+ X1 X2) 1.00)*.

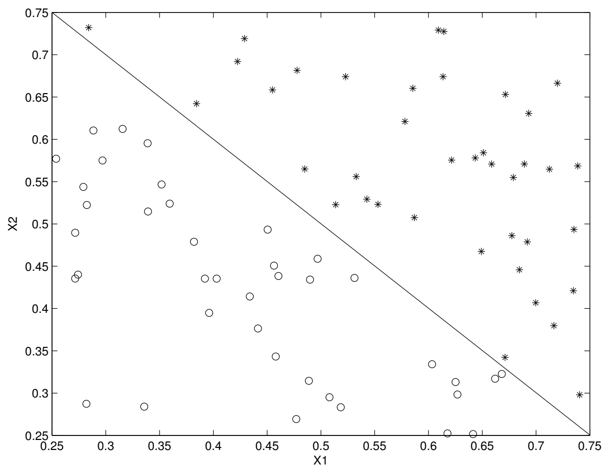


Figure 76: A decision line requiring arithmetic operators.

At the same time, it should be noted that the inclusion of more functions makes the task for G-REX much harder, since the search space increases. In addition, the quality of the rules obtained would be difficult to assess. A rule with high accuracy could be very hard to interpret if the tests compare seemingly unrelated variables. Most such rules are probably the result of nonsense relations in the training data, and would consequently generalize poorly to novel data[1].

From the observations above, it is obvious that the choice of representation language for a specific problem is very important. Consequently, it is interesting to not only evaluate several more representation languages, but also to find a strategy for which representation language to use given a specific problem. Finally, the relationship between choice of representation language and experienced clarity should be investigated.

The G-REX algorithm and the use of oracle data

From the experiments using publicly available data sets, the use of oracle data seems to be very successful; the extracted rules were both very accurate and very compact. Nevertheless, this result should be further verified, preferably by using the technique on several more problems from different domains. Future projects would again compare the accuracy obtained by G-REX, with and without the use of oracle data, to standard techniques like C5.0 and CART.

[1] The rule could of course also represent a valuable but previously unknown relationship.

Although G-REX has been used in many studies, the most basic elements of the algorithm have not been thoroughly investigated. This includes parameters like *number of generations, length penalty* and *population size,* but also exactly how selection, crossover and mutation are performed. From the results obtained in the studies, it is obvious that G-REX performance is rather good, but this does not mean that there is no room for improvement.

The fact that G-REX can extract rules from ensembles could potentially be exploited further. One specific possibility identified is to somehow use the fact that the ensemble members *each* make one prediction. Currently this is not utilized at all by G-REX; just the overall ensemble prediction is used. One possible study is therefore to determine whether or not the use of predictions from individual classifiers in an ensemble could be used to increase G-REX accuracy.

The consequences of inconsistency

Consistency is probably the least investigated criteria for rule extraction algorithms. With this in mind, one obvious task is to establish a standard way of measuring consistency. In this thesis, consistency was measured by comparing the predictions from several extracted models. A more natural procedure is probably to somehow directly compare actual rules. For rules represented as trees, the comparison would most likely prioritize splits close to the root. Even harder is, of course, to suggest a procedure for comparison of rules in different representation languages, especially if the comparison still should be based on syntax or semantics.

An even more interesting question is if the fact that G-REX has rather poor consistency also could be regarded as an asset. More specifically, at least two questions should be studied:

- Is it possible to obtain an even more accurate and/or comprehensible G-REX tree by somehow fusing several extracted trees? It must be noted that although it most likely would be possible to obtain higher accuracy just by combining the predictions from the extracted trees, the result would be an opaque ensemble model. Naturally, that is not acceptable, since the demand for a comprehensible model was the motivation for using rule extraction in the first place. So clearly, the most obvious result of the fusion must be *one* comprehensible model. The most straightforward approach is therefore probably to start with one extracted tree and in some way incorporate additional information from the other trees. Another, very different approach is to somehow try to visualize all models simultaneously, in order to get a feeling for where the different models agree and disagree.

- Access to several extracted rules makes it possible to express more or less belief in a certain prediction. Typically an instance where most extracted trees agree would result in a stronger belief in that prediction. One way of using this is to let G-REX signal that certain predictions are less certain than others. G-REX could even have the option to refuse to classify instances where the belief is very low. It should be noted that a similar belief measure is possible to obtain from a single tree just by analyzing the distribution of instances in the leaf node. Therefore, a first study could be to investigate if the belief in a certain prediction for one instance is better estimated using several trees instead of just one.

Rule extraction as a tool for concept description

One data mining task in which compact and accurate models are especially important is concept description, where the aim is to generate understandable descriptions of concepts or classes [Cri00]. With this in mind, a technique suitable for concept description must produce accurate and comprehensible models. Clearly this makes rule extraction a possible tool for concept description, which we also suggested in [SJ07]. The reasons for choosing rule extraction instead of techniques directly producing comprehensible models are the same as always, the extracted models should be more accurate and/or more comprehensible. One should note that when performing concept description, accuracy is important not as a measure of the model's predictive power, but rather as a measure of how correct the model's explanations are.

In [JSN04], we applied predictive modeling techniques to concept description in the marketing domain, on a problem concerning product testing. More specifically, the aim was to explain a subjective product score in terms of objective testing data. The results were quite encouraging, with both decision trees and G-REX producing accurate and comprehensible models. Later, the same approach was used on a very different problem where the purpose was to automatically find and describe successful poker strategies. In [JSN06], G-REX and decision tree techniques were used to mine data gathered from online poker in order to explain what signifies successful play. The overall result was that the rules induced were rather compact and had very high accuracy, thus providing good explanations of successful play. It is of course quite hard to assess the quality of the rules; i.e. if they provide something novel and non-trivial. The main picture was, however, that obtained rules were consistent with established poker theory. Because of this, the suggested techniques could, in future studies, where substantially more data is available, produce clear and accurate descriptions of what constitutes the difference between winning and losing in poker.

Because of promising initial results, this line of research should be continued:

- Rule extraction should be applied to several more suitable problems, in order to get rule extraction accepted as a tool for concept description. From results obtained, it appears that rule extraction producing accurate and comprehensible models is tailor-made for concept description. Nevertheless, a lot more experimentation, preferably in a variety of domains, remains. A natural first study would be to compare the performance of G-REX rule extraction to standard techniques used for concept description. Naturally, this study should use several typical concept description problems.

Software availability

Currently G-REX is a stand-alone Java application with a GUI. In addition, we have made it possible to call G-REX from both MatLab and Weka. Currently, however, the MatLab scripts and the Weka implementation would be very hard to use for someone unfamiliar to G-REX. Unfortunately, this also holds, to a lesser degree, for the Java application. In addition, all parts of G-REX are still under development, so it would not be correct to say that the current version is final. One prioritized project is therefore to settle for a stable version of G-REX and make it publicly available. Optimally, the Java application should include wizards and other tools making it easier for users. In addition, the MatLab scripts and the Weka implementation should be rewritten with the same purpose; i.e. making it possible for someone else to use them. Furthermore, some specific details should be observed. Currently, it is not possible for a user to use a function set including a function not already implemented in G-REX. Naturally it would add to the flexibility if a user could specify a completely new representation language, even using functions not currently implemented in G-REX. When this functionality is realized, a user simply has to supply a new grammar-file together with the necessary function implementations. Regarding G-REX in Weka, it should be noted that G-REX can be used both as a rule inducer and a rule extractor. Currently, however, it is only capable of extracting rules from another Weka model. This is a drawback since especially the ANNs in Weka are rather primitive. Having said that, one ambitious project is to make it possible to create MatLab ANNs from within Weka, and then let G-REX rule extract from them.

9.2.2 Future work regarding ensemble creation and GEMS

There are several more or less straightforward enhancements that should be evaluated for GEMS.

- If several different parallel populations are used during evolution, each population could specialize on part of the problem. Prediction could use averaging, majority vote or some more complex combination strategy.
- GEMS could use much less powerful ANNs and apply something similar to boosting during evolution. More specifically, the fitness function should change on-line and instances misclassified should be given a higher weight.
- GP could use several more operators during evolution; e.g. arithmetic operators. This would be similar to the approach taken by Langdon et al.; see e.g. [LB01a].
- The last study showed that explicitly targeting diversity by including a diversity measure in the fitness function could be beneficial. This should be further evaluated while testing several more diversity measures.
- Heterogeneous GEMS ensembles should be tested by, for instance, including RBF neural networks.
- Different measures like precision, recall and the area under a ROC-curve should be evaluated as part of a fitness function.
- The ensemble tree approach could be further developed. As an example, GEMS could use only full trees and let the *k* first levels contain splits. The purpose of the first *k* levels is just to determine which ensemble that should do the prediction. This would, however, require a more restricted crossover operation.

References

[ADT95] R. Andrews, J. Diederich and A. B. Tickle, A survey and critique of techniques for extracting rules from trained artificial neural networks, *Knowledge-Based Systems*, 8(6), 1995.

[Alp99] E. Alpaydın. Combined 5 x 2 F test for comparing supervised classification learning algorithms. *Neural Computation*, 11:1885-1892, 1999.

[BFO+84] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees*, Wadsworth International Group, 1984.

[BH89] E. Baum and W. D. Smith, What size net gives valid generalization?, *Neural Computation*, 1(1):151-160, 1989.

[BHM98] L. Bing, W. Hsu and Y. Ma, Integrating Classification and Association Rule Mining, *4th International Conference on Knowledge Discovery and Data Mining (KDD-98)*, New York, NY, 1998.

[Bis95] C. Bishop. *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[BJ76] G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*, Oakland, CA: Holden Day, 1970.

[BL97] M. J. A. Berry and G. Linoff, *Data Mining Techniques: For Marketing, Sales and Customer Support*, Wiley, 1997.

[BL98] D. S. Broomhead and D. Lowe, Multivariate functional interpolation and adaptive networks, *Complex Systems*, 2:321-355, 1998.

[BL00] M. J. A. Berry and G. Linoff, *Mastering Data Mining – The Art and Science of Customer Relationship Management*, Wiley, 2000.

[BM98] C. L. Blake and C. J. Merz, *UCI Repository of machine learning databases*, Department of Information and Computer Science, University of California, 1998, www.ics.uci.edu/~mlearn/MLRepository.html.

[BNK+98] W. Banzhaf, P. Nordin, R. E. Keller and F. D. Francone, *Genetic Programming an Introduction – On the Automatic Evolutions of Computer Programs and Its Applications*, Morgan Kaufmann, 1998.

[Bre96] L. Breiman, Bagging predictors, *Machine Learning*, 24(2):123-140, 1996.

[Bre01] L. Breiman, Random forests, *Machine Learning*, 45(1):5-32, 2001.

[Bro04] G. Brown, *Diversity in neural network ensembles*, PhD thesis, University of Birmingham, 2004.

[BVS91] D. Biggs, B. de Ville and E. Suen, A method for choosing multiway partitions for classification and decision trees, *Journal of Applied Statistics*, 18:49-62, 1991.

[BWH$^+$05] G. Brown, J. Wyatt, R. Harris and X. Yao, Diversity Creation Methods: A survey and Categorisation, *Journal of Information Fusion*, 6(1):5-20, 2005.

[Cra96] M. Craven, *Extracting Comprehensive Models from Trained Neural Networks*, Ph.D. Thesis, University of Wisconsin-Madison, 1996.

[Cri00] The CRISP-DM Consortium, CRISP-DM 1.0, www.crisp-dm.org, 2000.

[CS96] M. Craven and J. Shavlik, Extracting Tree-Structured Representations of Trained Networks, *Advances in Neural Information Processing Systems*, 8:24-30, 1996.

[CS97a] M. Craven and J. Shavlik, Using Neural Networks for Data Mining. *Future Generation Computer Systems: Special Issue on Data Mining*, pp. 211-229, 1997.

[CS97b] M. Craven and J. Shavlik, Understanding time-series networks: A case study in rule extraction, *International Journal of Neural Systems*, 8(4):373-384, 1997.

[CS99] M. Craven and J. Shavlik, Rule Extraction: Where Do We Go from Here?, *University of Wisconsin Machine Learning Research Group working Paper 99-1*, 1999.

[Cyb88] G. Cybenko, Continuous valued neural networks with two hidden layers are sufficient, *Technical report, Department of Computer Science, Tufts University, Medford, Massachusetts*, 1988.

[Cyb89] G. Cybenko, Approximation by superposition of a sigmoidal function, *Mathematics of Controls, Signals and Systems*, 2:303-314, 1989.

[DB95] T. G. Dietterich and G. Bakiri, Solving Multiclass Learning Problems via Error-Correcting Output Codes, *Journal of Artificial Intelligence Research* 2:253-286, 1995.

[DBS77] R. Davis, B. G. Buchanan and E. Shortliffe, Production rules as a representation for a knowledge-based consultation program, *Artificial Intelligence*, Vol. 8. No. 1:15-45, 1977.

[Dem06] J. Demšar, Statistical Comparisons of Classifiers over Multiple Data Sets, *Journal of Machine Learning Research*, 7:1–30, 2006.

[Die97] T. G. Dietterich. Machine learning research: four current directions. *The AI Magazine* 18:97-136, 1997.

[Die98] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation* 10: 1895-1924, 1998.

[Die00] T. G. Dietterich, An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting and randomization, *Machine Learning*, 40(2):139-157, 2000.

[Dor96] G. Dorffner, Neural Networks for Time Series Processing, *Neural Network World*, 4:447-468, 1996.

[DRS$^+$02] J. Dorado, J. R. Rabunãl, A. Santos, A. Pazos and D. Rivero, Automatic Recurrent and Feed-Forward ANN Rule and Expression Extraction with Genetic Programming, *Proceedings 7th International Conference on Parallel Problem Solving from Nature*, Granada, Spain, 2002.

[DT00] R. P. W. Duin and D. M. J. Tax, Experiments with classifier combining rules, *International Workshop on Multiple Classifier Systems (LNCS 1857)*, Cagliari, Italy, Springer-Verlag, pp. 30-44, 2000.

[Dun03] M. Dunham, *Data Mining – Introductory and Advanced Topics*, Prentice Hall, 2003.

[DZ04] S. Dzeroski and B. Zenko, Is Combining Classifiers with Stacking Better than Selecting the Best One?, *Machine Learning*, 54:255–273, 2004.

[Elm90] J. L. Elman, Finding Structure in Time, *Cognitive science*, 14:179-211, 1990.

[Fis36] R. A. Fisher, The use of multiple measurements in taxonomic problems, *Annals of Eugenics*, 7(2):179-188, 1936.

[FJK99] M. Faifer, C. Janikow, and K. Krawiec, Extracting Fuzzy Symbolic Representation from Artificial Neural Networks, *Proceedings 18th International Conference of the North American Fuzzy Information Processing Society*, New York, NY, pp. 600-604, 1999.

[Fri37] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, *Journal of American Statistical Association*, 32:675-701, 1937.

[FS96] Y. Freund and R. Shapire, Experiments with a New Boosting Algorithm, *Proceedings 13th International conference on Machine Learning*, Bari, Italy, pp. 148-156, 1996.

[Har75] J. A. Hartigan, *Clustering Algorithms*, John Wiley & sons, 1975.

[HBV06] J. Huysmans, B. Baesens and J. Vanthienen, Using rule extraction to improve the comprehensibility of predictive models, *FETEW Research Report KBI_0612, K. U. Leuven*, 2006.

[Hol75] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[HS90] L. K. Hansen and P. Salamon, Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993-1001, 1990.

[HWF+03]B. Hudson, D Whitley, M. Ford and A. Browne, Biological Data Mining: A comparison of Neural Network and Symbolic Techniques, *Technical report*, 2003.

[IM80] R. L. Iman and J. M. Davenport, Approximations of the critical of the Friedman statistics, *Communications in Statistics*, pp. 571-595, 1980.

[JKN03] U. Johansson, R. König and L. Niklasson, Rule Extraction from Trained Neural Networks using Genetic Programming, *13th International Conference on Artificial Neural Networks,* Istanbul, Turkey, supplementary proceedings pp. 13-16, 2003.

[JKN04] U. Johansson, R. König and L. Niklasson, The Truth is in There - Rule Extraction from Opaque Models Using Genetic Programming, *17th Florida Artificial Intelligence Research Society Conference (FLAIRS) 04*, Miami, FL, AAAI Press, pp. 658-662, 2004.

[JKN07] U. Johansson, R. König and L. Niklasson, Inconsistency - Friend or Foe, *The International Joint Conference on Neural Networks*, IEEE Press, Orlando, FL, 2007, To appear.

[JLK+06a] U. Johansson, T. Löfström, R. König and L. Niklasson, Why Not Use an Oracle When You Got One?, *Neural Information Processing - Letters and Reviews*, Vol. 10, No. 8-9: 227-236, 2006.

[JLK+06b] U. Johansson, T. Löfström, R. König and L. Niklasson, Introducing GEMS – a Novel Technique for Ensemble Creation, *19th Florida Artificial Intelligence Research Society Conference (FLAIRS) 06*, Melbourne Beach, FL, AAAI Press, pp. 700-705, 2006.

[JLK+06c] U. Johansson, T. Löfström, R. König and L. Niklasson, Genetically Evolved Trees Representing Ensembles, *8th International Conference on Artificial Intelligence and Soft Computing*, Zakopane, Poland, Lecture Notes in Artificial Intelligence, Springer-Verlag, pp. 613-622, 2006

[JLK+06d] U. Johansson, T. Löfström, R. König and L. Niklasson, Building Neural Network Ensembles using Genetic Programming, *The International Joint Conference on Neural Networks,* IEEE Press, Vancouver, Canada, pp. 2239-2244, 2006.

[JLK+06e] U. Johansson, T. Löfström, R. König, C. Sönströd and L. Niklasson, Rule Extraction from Opaque Models – A Slightly Different Perspective, *6th International Conference on Machine Learning and Applications*, Orlando, FL, IEEE press, pp. 22-27, 2006.

[JLN05] U. Johansson, T. Löfström and L. Niklasson, Obtaining Accurate Neural Network Ensembles, *International Conference on Computational Intelligence for Modelling Control and Automation - CIMCA*, Vienna, Austria, IEEE Computer Society, Vol. 2:103-108, 2005.

[JLN06] U. Johansson, T. Löfström and L. Niklasson, Accuracy on a Hold-out Set: The Red Herring of Data Mining, *23rd Annual Workshop of the Swedish Artificial Intelligence Society*, Umeå, Sweden, pp. 137-146, 2006.

[JLN07] U. Johansson, T., Löfström and L. Niklasson, The Importance of Diversity in Neural Network Ensembles - An Empirical Investigation, *The International Joint Conference on Neural Networks*, Orlando, FL, IEEE Press, 2007, To appear.

[JN01] U. Johansson and L. Niklasson, Predicting the Impact of Advertising - a Neural Network Approach, *The International Joint Conference on Neural Networks*, Washington D.C., IEEE Press, pp. 1799-1804, 2001.

[JN02a] U. Johansson and L. Niklasson, Increased Performance with Neural Nets - An Example from the Marketing Domain, *The International Joint Conference on Neural Networks*, Honolulu, HI, IEEE Press, pp. 1684-1689, 2002.

[JN02b] U. Johansson and L. Niklasson, Neural Networks - from Prediction to Explanation, *IASTED International Conference Artificial Intelligence and Applications,* Malaga, Spain, IASTED, pp. 93-98, 2002.

[JNK04] U. Johansson, L. Niklasson and R. König, Accuracy vs. Comprehensibility in Data Mining Models, *7th International Conference on Information Fusion*, Stockholm, Sweden, pp. 295-300, 2004.

[JSK$^+$03] U. Johansson, C. Sönströd, R. König and L. Niklasson, Neural Networks and Rule Extraction for Prediction and Explanation in the Marketing Domain, *The International Joint Conference on Neural Networks,* Portland, OR, IEEE Press, pp. 2866-2871, 2003.

[JSN04] U. Johansson, C. Sönströd and L. Niklasson, Why Rule Extraction Matters, *8th IASTED International Conference on Software Engineering and Applications*, Cambridge, MA, pp. 47-52, 2004.

[JSN06] U. Johansson, C. Sönströd and L. Niklasson, Explaining Winning Poker – A Data Mining Approach, *6th International Conference on Machine Learning and Applications*, Orlando, FL, IEEE press, pp. 129-134, 2006.

[Jud90] J. S. Judd, *Neural Network Design and the Complexity of Learning*, MIT Press, 1990.

[KJN06] R. König, U. Johansson and L. Niklasson, Increasing rule extraction comprehensibility, *International Journal of Information Technology and Intelligent Computing*, Vol. 1, No. 2:303-314, 2006

[KLR$^+$98] R. L. Kennedy, Y. Lee, B. Van Roy, C. D. Reed and R. P. Lippman, *Solving Data Mining Problems Through Pattern Recognition*, Prentice Hall, 1998.

[KNS99] E. Keedwell, A. Narayanan and D. A. Savic, Using Genetic Algorithms to Extract Rules from Trained Neural Networks, *Proceedings Genetic and Evolutionary Computing Conference*, Orlando, FL, Vol. 1:793-800, 1999.

[Koh82] T. Kohonen, Self-organized formation of topologically correct feature maps, *Biological Cybernetics*, 43:59-69, 1982.

[Koz92] J. Koza, *Genetic Programming - On the Programming of Computers by Natural Selection*, MIT Press, 1992.

[KV95] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning, *Advances in Neural Information Processing Systems*, Vol. 2:650-659, San Mateo, CA, Morgan Kaufmann, 1995.

[KW96] R. Kohavi and D. Wolpert, Bias plus variance decomposition for zero-one loss functions, *13th International Conference on Machine Learning*, pp. 275-283, Morgan Kaufmann, 1996.

[KW03] L. I. Kuncheva and C. J. Whitaker, Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy, *Machine Learning*, (51):181-207, 2003.

[LB01a] W. B. Langdon and B. F. Buxton, Genetic programming for combining classifiers, *Genetic and Evolutionary Computation Conference*, San Francisco, CA, pp. 66–73, 2001.

[LB01b] W. B. Langdon and B. F. Buxton, Genetic programming for improved receiver operating characteristics, *2nd International Conference on Multiple Classifier System*, Cambridge, UK ,pp. 68–77, LNCS 2096, Springer Verlag, 2001.

[LB01c] W. B. Langdon and B. F. Buxton, Evolving receiver operating characteristics for data fusion, *Genetic Programming, Proceedings of EuroGP'2001*, Lake Como, Italy, pp. 87–96, LNCS 2038, Springer Verlag, 2001.

[LBB03] W. B. Langdon, S. J. Bartett and B. F. Buxton, Comparison of AdaBoost and Genetic Programming for Combining Neural Networks for Drug Discovery, *Applications of Evolutionary Computing*, LNCS 2611:87-98, Springer Verlag, 2003.

[Lev44] K. Levenberg, A method for the solution of certain non-linear problems in least squares. *Quarterly Journal of Applied Mathematics II*, (2):164-168, 1944.

[Liu95] H. Liu, X2R: A fast rule generator, *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, Vancouver, 1995.

[Liu98] Y. Liu, *Negative correlation learning and evolutionary neural network ensembles*, PhD thesis, University of New South Wales, Australian Defence Force Academy, Canberra, Australia, 1998.

[LJB$^+$89] Y. LeCun, L. Jackel, L. Boser and J. Denker, Handwritten digit recognition: Applications of neural network chips and automatic learning, *IEEE Communications Magazine*, 27(11):41-46, 1989.

[LJK04] T. Löfström, U. Johansson, and L. Niklasson, Rule Extraction by Seeing Through the Model, *11th International Conference on Neural Information Processing*, Calcutta, India, pp. 555-560, 2004.

[LLS00] T.-S. Lim, W.-Y. Loh and Y.-S. Shih, A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms, *Machine Learning*, 40:203-229, 2000.

[LSL95] H. Lu, R. Setino and H. Liu, Neurorule: A connectionist approach to data mining, *Proceedings of the International Very Large Databases Conference*, pp. 478-489, 1995.

[Mar63] D. W. Marquardt, An algorithm for least-squares estimation of non-linear parameters, *Journal of the Society of Industrial and Applied Mathematics*, 11(2):431-441, 1963.

[MBG$^+$06] D. Martens, B. Baesens, T. Van Gestel, J. Vanthienen, Comprehensible credit scoring models using rule extraction from support vector machines, *European Journal of Operational Research*, 2006, In press.

[McK92] D. J. C. MacKay, Bayesian interpolation, *Neural Computation*, 4(3):415-447, 1992.

[McQ67] J. McQueen, Some methods for classification and analysis of multivariate observations, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281-197, 1967.

[MN89] M. Mézard and J. – P. Nadal, Learning in feed-forward layered networks: The tiling algorithm, *Journal of Physics*, 22:2191-2204, 1989.

[MS95] R. Maclin and J. Shavlik, Combining the predictions of multiple classifiers: Using competitive learning to initialize neural networks, *International Joint Conference on Artificial Intelligence*, Montreal, Canada, pp. 524-530, 1995.

[NB95] P. Nordin and W. Banzhaf, Evolving Turing-complete programs for a register machine with self-modifying code, *Genetic Algorithms: Proceedings of the 6th International Conference*, Pittsburgh, PA, pp. 318-325, 1995.

[Nem63] P. B. Nemenyi. *Distribution-free multiple comparisons*, PhD thesis, Princeton University, 1963.

[OM96] D. Opitz and J. Shavlik. Actively searching for an effective neural-network ensemble, *Connection Science*, 8(3/4):337-353, 1996.

[OM99] D. Opitz and R. Maclin. Popular ensemble methods: an empirical study, *Journal of Artificial Intelligence Research*, 11:169-198, 1999.

[PK97] D. Partridge and W. J. Krzanowski, Software diversity: practical statistics for its measurements and exploitation, *Information & Software Technology*, 39:707-717, 1997.

[PY96] D. Partridge and W. B. Yates, Engineering multivision neural-net systems, *Neural Computation*, 8(4):869-893, 1996.

[Qui86] J. R. Quinlan, Induction of decision trees, *Machine Learning*, 11(1):81-106, 1986.

[Qui92] J. R. Quinlan, Learning with Continuous Classes, *5th Australian Joint Conference of Artificial Intelligence*, pp. 343-348, 1992.

[Qui93] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[Qui98] J. R. Quinlan, *See5 version 1.16*, http://www.rulequest.com, 1998.

[RB93] M. Riedmiller and H. Braun, A direct adaptive method for faster backpropagation learning: The RPROP algorithm, *Proceedings IEEE International Conference on Neural Networks*, San Francisco, CA, pp. 586-591, 1993.

[RHW86] D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning internal representations by error propagation, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume: Foundations, MIT Press, 1986.

[RL91] M. D. Richard and R. P. Lippman, Neural Network Classifiers Estimate Bayesian *a Posteriori* Probabilities, *Neural Computation* 3:461-483, 1991.

[RN03] S. Russel and P. Norvig, *Artificial Intelligence - A Modern Approach*, 2nd edition, Prentice Hall, 2003.

[SA98] A. Swann and N. Allinson, Fast committee learning: Preliminary results, *Electronics Letters*, 34(14):1408-1410, 1998.

[Sha90] R. Shapire. The strength of weak learnability, *Machine Learning*, 5(2):197-227, 1990.

[SJ07] C. Sönströd and U. Johansson, Concept Description - A Fresh Look, *The International Joint Conference on Neural Networks*, Orlando, Fl, IEEE Press, 2007, To appear.

[SMT91] J. Shavlik, R. Mooney and G. Towell, Symbolic and neural net learning algorithms: An empirical comparison, *Machine Learning*, 6:111-143, 1991.

[SNS95] N. Sharkey, J. Neary and A. Sharkey, Searching Weight Space for Backpropagation Solution Types, *Current trends in Connectionism: Proceedings of the 1995 Swedish Conference on Connectionism*, pp. 103-120, 1995.

[SR87] T. J. Sejnowski and C. R. Rosenberg, Parallel Networks that Learn to Pronounce English Text, *Complex Systems*, 1:145-168, 1987.

[TS93] G. Towell and J. Shavlik, The extraction of refined rules from knowledge based neural networks, *Machine Learning*, 13(1):71-101, 1993.

[TSK06] P. – N. Tang, M. Stenbach and V. Kumar, *Introduction to Data Mining*, Addison-Wesley, 2006.

[UR99] M. Uysal and S. El Roubi, Artificial Neural Networks vs. Multiple Regression in Tourism Demand Analysis, *Journal of Travel Research*, 38(2):111-118, 1999.

[Vap95] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer Verlag, New York, 1995.

[Wer74] P. J. Werbos, *Beyond regression: new tools for prediction and analysis in the behavioral sciences*, PhD thesis, Harvard University, Boston, MA, 1974.

[WF05] I. H. Witten and E. Frank, *Data Mining – Practical Machine Learning Tools and Techniques*, Elsevier, 2005.

[WHR92] A. S. Weigend, B. A. Huberman and D. E. Rumelhart, Predicting Sunspots and Exchange Rates with Connectionist Networks, *Proceedings of Nonlinear Modeling and Forecasting, SFI Studies in the Sciences of Complexity*, Vol. XII:395-432, 1992.

[WL02] W. Wettayaprasit and C. Lursinsap, Neural Rule Extraction Based on Activation Projection with Certainty Factor Refinement, *The International Joint Conference on Neural Networks*, Honolulu, HI, IEEE Press, pp. 1730-1735, 2002.

[Wol92] D. H. Wolpert, Stacked Generalizaion, *Neural Networks,* 5:197-227, 1992.

[Wol95] D. H. Wolpert, The Relationship between PAC, the Statistical Physics Framework, the Bayesian Framework and the VC Framework, *The Mathematics of Generalization*, Reading, MA: Addison-Wesley, pp. 117-124, 1995.

[WZN96] A. S. Weigend, H. G. Zimmermann and R. Neuneier, Clearning, *Neural Networks in Financial Engineering,* World Scientific, Singapore, 1996.

[Yas99] R.Yasdi, Prediction of Road Traffic using a Neural Network Approach, *Neural computation & Application*, 8:135-142, 1999.

[You00] G. Youle, On the association of attributes in statistics, *Phil. Trans.*, 194:257-319, 1900.

[Zha92] J. Zhang, Selecting typical instances in instance-based learning, *Proceedings 9th International Machine Learning Conference,* Aberdeen, Scotland, pp. 470-479, 1992.

[Zho04] Z.-H. Zhou, Rule Extraction: Using Neural Networks or For Neural Networks?, *Journal of Computer Science & Technology*, 19(2):249-253, 2004.

[ZWJ+01] Z.-H. Zhou, J.-X. Wu, Y. Jiang and S.-F. Chen. Genetic algorithm based selective neural network ensemble, 17th *International Joint Conference of Artificial Intelligence*, Vol. 2:797-802, Seattle, WA, 2001.

[ZWT02] Z.-H. Zhou, J.-X. Wu and W. Tang. Ensembling Neural Networks: Many Could Be Better Than All, *Artificial Intelligence*, Vol. 137, No. 1-2:239-263, Elsevier, 2002.

Department of Computer and Information Science
Linköpings universitet

Dissertations

Linköping Studies in Science and Technology

No 14 **Anders Haraldsson:** A Program Manipulation System Based on Partial Evaluation, 1977, ISBN 91-7372-144-1.

No 17 **Bengt Magnhagen:** Probability Based Verification of Time Margins in Digital Designs, 1977, ISBN 91-7372-157-3.

No 18 **Mats Cedwall:** Semantisk analys av process-beskrivningar i naturligt språk, 1977, ISBN 91-7372-168-9.

No 22 **Jaak Urmi:** A Machine Independent LISP Compiler and its Implications for Ideal Hardware, 1978, ISBN 91-7372-188-3.

No 33 **Tore Risch:** Compilation of Multiple File Queries in a Meta-Database System 1978, ISBN 91-7372-232-4.

No 51 **Erland Jungert:** Synthesizing Database Structures from a User Oriented Data Model, 1980, ISBN 91-7372-387-8.

No 54 **Sture Hägglund:** Contributions to the Development of Methods and Tools for Interactive Design of Applications Software, 1980, ISBN 91-7372-404-1.

No 55 **Pär Emanuelson:** Performance Enhancement in a Well-Structured Pattern Matcher through Partial Evaluation, 1980, ISBN 91-7372-403-3.

No 58 **Bengt Johnsson, Bertil Andersson:** The Human-Computer Interface in Commercial Systems, 1981, ISBN 91-7372-414-9.

No 69 **H. Jan Komorowski:** A Specification of an Abstract Prolog Machine and its Application to Partial Evaluation, 1981, ISBN 91-7372-479-3.

No 71 **René Reboh:** Knowledge Engineering Techniques and Tools for Expert Systems, 1981, ISBN 91-7372-489-0.

No 77 **Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982, ISBN 91-7372-527-7.

No 94 **Hans Lunell:** Code Generator Writing Systems, 1983, ISBN 91-7372-652-4.

No 97 **Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983, ISBN 91-7372-660-5.

No 109 **Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation,1984, ISBN 91-7372-801-2.

No 111 **Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984, ISBN 91-7372-805-5.

No 155 **Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons, 1987, ISBN 91-7870-133-3.

No 165 **James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning, 1987, ISBN 91-7870-183-X.

No 170 **Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems, 1987, ISBN 91-7870-225-9.

No 174 **Johan Fagerström:** A Paradigm and System for Design of Distributed Systems, 1988, ISBN 91-7870-301-8.

No 192 **Dimiter Driankov***:* Towards a Many Valued Logic of Quantified Belief, 1988, ISBN 91-7870-374-3.

No 213 **Lin Padgham:** Non-Monotonic Inheritance for an Object Oriented Knowledge Base, 1989, ISBN 91-7870-485-5.

No 214 **Tony Larsson:** A Formal Hardware Description and Verification Method, 1989, ISBN 91-7870-517-7.

No 221 **Michael Reinfrank:** Fundamentals and Logical Foundations of Truth Maintenance, 1989, ISBN 91-7870-546-0.

No 239 **Jonas Löwgren:** Knowledge-Based Design Support and Discourse Management in User Interface Management Systems, 1991, ISBN 91-7870-720-X.

No 244 **Henrik Eriksson:** Meta-Tool Support for Knowledge Acquisition, 1991, ISBN 91-7870-746-3.

No 252 **Peter Eklund:** An Epistemic Approach to Interactive Design in Multiple Inheritance Hierarchies,1991, ISBN 91-7870-784-6.

No 258 **Patrick Doherty:** NML3 - A Non-Monotonic Formalism with Explicit Defaults, 1991, ISBN 91-7870-816-8.

No 260 **Nahid Shahmehri:** Generalized Algorithmic Debugging, 1991, ISBN 91-7870-828-1.

No 264 **Nils Dahlbäck:** Representation of Discourse-Cognitive and Computational Aspects, 1992, ISBN 91-7870-850-8.

No 265 **Ulf Nilsson:** Abstract Interpretations and Abstract Machines: Contributions to a Methodology for the Implementation of Logic Programs, 1992, ISBN 91-7870-858-3.

No 270 **Ralph Rönnquist:** Theory and Practice of Tense-bound Object References, 1992, ISBN 91-7870-873-7.

No 273 **Björn Fjellborg:** Pipeline Extraction for VLSI Data Path Synthesis, 1992, ISBN 91-7870-880-X.

No 276 **Staffan Bonnier:** A Formal Basis for Horn Clause Logic with External Polymorphic Functions, 1992, ISBN 91-7870-896-6.

No 277 **Kristian Sandahl:** Developing Knowledge Management Systems with an Active Expert Methodology, 1992, ISBN 91-7870-897-4.

No 281 **Christer Bäckström:** Computational Complexity of Reasoning about Plans, 1992, ISBN 91-7870-979-2.

No 292 **Mats Wirén:** Studies in Incremental Natural Language Analysis, 1992, ISBN 91-7871-027-8.

No 297 **Mariam Kamkar:** Interprocedural Dynamic Slicing with Applications to Debugging and Testing, 1993, ISBN 91-7871-065-0.

No 302 **Tingting Zhang:** A Study in Diagnosis Using Classification and Defaults, 1993, ISBN 91-7871-078-2.

No 312 **Arne Jönsson:** Dialogue Management for Natural Language Interfaces - An Empirical Approach, 1993, ISBN 91-7871-110-X.

No 338 **Simin Nadjm-Tehrani**: Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification, 1994, ISBN 91-7871-237-8.

No 371 **Bengt Savén:** Business Models for Decision Support and Learning. A Study of Discrete-Event Manufacturing Simulation at Asea/ABB 1968-1993, 1995, ISBN 91-7871-494-X.

No 375 **Ulf Söderman:** Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.

No 383 **Andreas Kågedal:** Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.

No 396 **George Fodor:** Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.

No 413 **Mikael Pettersson:** Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.

No 414 **Xinli Gu:** RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.

No 416 **Hua Shu:** Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.

No 429 **Jaime Villegas:** Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.

No 431 **Peter Jonsson:** Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.

No 437 **Johan Boye:** Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.

No 439 **Cecilia Sjöberg:** Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.

No 448 **Patrick Lambrix:** Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.

No 452 **Kjell Orsborn:** On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.

No 459 **Olof Johansson:** Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.

No 461 **Lena Strömbäck:** User-Defined Constructions in Unification-Based Formalisms,1997, ISBN 91-7871-857-0.

No 462 **Lars Degerstedt:** Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.

No 475 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.

No 480 **Mikael Lindvall:** An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.

No 485 **Göran Forslund**: Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.

No 494 **Martin Sköld**: Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.

No 495 **Hans Olsén**: Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.

No 498 **Thomas Drakengren:** Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.

No 502 **Jakob Axelsson:** Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.

No 503 **Johan Ringström:** Compiler Generation for Data-Parallel Programming Langugaes from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.

No 512 **Anna Moberg:** Närhet och distans - Studier av kommunikationsmmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.

No 520 **Mikael Ronström:** Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.

No 522 **Niclas Ohlsson:** Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.

No 526 **Joachim Karlsson:** A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.

No 530 **Henrik Nilsson:** Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-x.

No 555 **Jonas Hallberg:** Timing Issues in High-Level Synthesis,1998, ISBN 91-7219-369-7.

No 561 **Ling Lin:** Management of 1-D Sequence Data - From Discrete to Continuous, 1999, ISBN 91-7219-402-2.

No 563 **Eva L Ragnemalm:** Student Modelling based on Collaborative Dialogue with a Learning Companion, 1999, ISBN 91-7219-412-X.

No 567 **Jörgen Lindström:** Does Distance matter? On geographical dispersion in organisations, 1999, ISBN 91-7219-439-1.

No 582 **Vanja Josifovski:** Design, Implementation and

Evaluation of a Distributed Mediator System for Data Integration, 1999, ISBN 91-7219-482-0.

No 589 **Rita Kovordányi**: Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999, ISBN 91-7219-506-1.

No 592 **Mikael Ericsson:** Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999, ISBN 91-7219-532-0.

No 593 **Lars Karlsson:** Actions, Interactions and Narratives, 1999, ISBN 91-7219-534-7.

No 594 **C. G. Mikael Johansson:** Social and Organizational Aspects of Requirements Engineering Methods - A practice-oriented approach, 1999, ISBN 91-7219-541-X.

No 595 **Jörgen Hansson:** Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999, ISBN 91-7219-542-8.

No 596 **Niklas Hallberg:** Incorporating User Values in the Design of Information Systems and Services in the Public Sector: A Methods Approach, 1999, ISBN 91-7219-543-6.

No 597 **Vivian Vimarlund:** An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999, ISBN 91-7219-544-4.

No 598 **Johan Jenvald:** Methods and Tools in Computer-Supported Taskforce Training, 1999, ISBN 91-7219-547-9.

No 607 **Magnus Merkel:** Understanding and enhancing translation by parallel text processing, 1999, ISBN 91-7219-614-9.

No 611 **Silvia Coradeschi:** Anchoring symbols to sensory data, 1999, ISBN 91-7219-623-8.

No 613 **Man Lin:** Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective, 1999, ISBN 91-7219-630-0.

No 618 **Jimmy Tjäder:** Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999, ISBN 91-7219-657-2.

No 627 **Vadim Engelson:** Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing, 2000, ISBN 91-7219-709-9.

No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.

No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.

No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.

No 688 **Marcus Bjäreland:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.

No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.

No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN-91-7373-126-9.

No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91 7373 207 9.

No 725 **Tim Heyer**: Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91 7373 208 7.

No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91 7373 212 5.

No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91 7373 258 3.

No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.

No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.

No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.

No 747 **Anneli Hagdahl:** Development of IT-suppor-ted Inter-organisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.

No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-
318-0.

No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.

No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.

No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.

No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002,
ISBN 91-7373-349-0.

No 774 **Choong-ho Yi:** Modelling Object-Oriented
Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.

No 779 **Mathias Broxvall:** A Study in the
Computational Complexity of Temporal
Reasoning, 2002, ISBN 91-7373-440-3.

No 793 **Asmus Pandikow:** A Generic Principle for
Enabling Interoperability of Structured and
Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.

No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.

No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems´ Development, 2003, ISBN 91-7373-604-X

No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informationsystem, 2003, ISBN 91-7373-618-X.

No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.

No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.

No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.

No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.

No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing - An Emperical Study in Software Engineering, 2003, ISBN 91-7373-779-8.

No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.

No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.

No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.

No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.

No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.

No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.

No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.

No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5

No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004. ISBN 91-7373-966-9.

No 887 **Anders Lindström:** English and other Foreign Linquistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.

No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modellling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.

No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.

No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.

No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.

No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.

No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.

No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.

No 933 **Mikael Cäker:** Management Accounting as Constructing and Opposing Customer Focus: Three Case Studies on Management Accounting and Customer Relations, 2005, ISBN 91-85297-64-X.

No 937 **Jonas Kvarnström:** TALplanner and Other Extensions to Temporal Action Logic, 2005, ISBN 91-85297-75-5.

No 938 **Bourhane Kadmiry:** Fuzzy Gain-Scheduled Visual Servoing for Unmanned Helicopter, 2005, ISBN 91-85297-76-3.

No 945 **Gert Jervan:** Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems, 2005, ISBN: 91-85297-97-6.

No 946 **Anders Arpteg:** Intelligent Semi-Structured Information Extraction, 2005, ISBN 91-85297-98-4.

No 947 **Ola Angelsmark:** Constructing Algorithms for Constraint Satisfaction and Related Problems - Methods and Applications, 2005, ISBN 91-85297-99-2.

No 963 **Calin Curescu:** Utility-based Optimisation of Resource Allocation for Wireless Networks, 2005. ISBN 91-85457-07-8.

No 972 **Björn Johansson:** Joint Control in Dynamic Situations, 2005, ISBN 91-85457-31-0.

No 974 **Dan Lawesson:** An Approach to Diagnosability Analysis for Interacting Finite State Systems, 2005, ISBN 91-85457-39-6.

No 979 **Claudiu Duma:** Security and Trust Mechanisms for Groups in Distributed Services, 2005, ISBN 91-85457-54-X.

No 983 **Sorin Manolache:** Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour, 2005, ISBN 91-85457-60-4.

No 986 **Yuxiao Zhao:** Standards-Based Application Integration for Business-to-Business Communications, 2005, ISBN 91-85457-66-3.

No 1004 **Patrik Haslum:** Admissible Heuristics for Automated Planning, 2006, ISBN 91-85497-28-2.

No 1005 **Aleksandra Tešanovic:** Developing Reusable and Reconfigurable Real-Time Software using Aspects and Components, 2006, ISBN 91-85497-29-0.

No 1008 **David Dinka:** Role, Identity and Work: Extending the design and development agenda, 2006, ISBN 91-85497-42-8.

No 1009 **Iakov Nakhimovski:** Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, 2006, ISBN 91-85497-43-X.

No 1013 **Wilhelm Dahllöf:** Exact Algorithms for Exact Satisfiability Problems, 2006, ISBN 91-85523-97-6.

No 1016 **Levon Saldamli:** PDEModelica - A High-Level Language for Modeling with Partial Differential Equations, 2006, ISBN 91-85523-84-4.

No 1017 **Daniel Karlsson:** Verification of Component-based Embedded System Designs, 2006, ISBN 91-85523-79-8.