Enhancing Genetic Programming for Predictive Modeling

Dedication
To my amazing wife and daugther, your are my sunshine.

Örebro Studies in Computer Sience 58

RIKARD KÖNIG

Enhancing Genetic Programming for Predictive Modeling

Title: Enhancing Genetic Programming for Predicitve Modeling

Publisher: Örebro University 2014
www.oru.se/publikationer-avhandlingar

Print: Örebro University, Repro 02/2014

ISSN 1650-8580
ISBN 978-91-7529-001-0

Abstract

Rikard König (2014): Enhancing Genetic Programming for Predictive Modeling Örebro Studies in Computer Sience 58.

Traditional predictive modeling techniques are most often optimized using a local greedy strategy and are tied to specific representations and optimization criteria, leading to potentially suboptimal solutions with regard to both predictive performance and comprehensibility. However, using genetic programming (GP), both the model representation of the solution and the optimization criteria can be designed specifically for the problem at hand and the model is optimized globally. On the other hand, GP has disadvantages that traditional techniques do not have, i.e., it is inherently inconsistent, struggles with large search spaces, and produces programs that often contain introns. Furthermore, the advantages are rarely fully exploited in the few predictive modeling frameworks that incorporate GP.

Hence, this thesis aims to enhance GP for predictive modeling, by exploiting the advantages and counteracting the deficiencies. A set of criteria for frameworks of predictive modeling based on GP are suggested and several novel techniques that improve the accuracy or comprehensibility of predictive models produced using GP are presented.

With regard to comprehensibility, three novel techniques are proposed, of which the first produces accurate and comprehensible models through rule extraction, the second removes introns for arbitrary representation, and the third generates alternative solutions and helps select an accurate solution that conforms to domain knowledge.

A point made in this thesis is that inconsistency should be considered an advantage and not a disadvantage. In fact, the technique of generating and selecting among alternative models is, for example, only possible due to inconsistency. Furthermore, this thesis also presents two additional novel techniques which enhance probability estimates of the produced programs and create accurate kNN-ensembles by exploiting inconsistency.

A step towards handling large search spaces is taken by applying local search in two novel techniques. The first injects induced mutated decision trees, when a reasonable performance is not achieved for classification tasks, while the second uses the least squares method to increase the predictive performance on regression problems.

Finally, a GP framework for predictive modeling, i.e., G-REX, has been implemented. G-REX fulfills most of the suggested criteria, realizes many of the presented techniques, and is available to the public on www.grex.se.

Keywords: Genetic Programming, Predictive Modeling, Data Mining, Machine Learning, Rule Extraction, Classification, Regression, Ensembles, Comprehensibility.

Rikard König, Department
Örebro University, SE-701 82 Örebro, Sweden, rikard@konigs.org

Table of Contents

List of Publications

Licentiate Thesis

König, R., 2009. Predictive Techniques and Methods for Decision Support in Situations with Poor Data Quality. *Studies from the School of Science and Technology at University of Örebro;5*,
http://urn.kb.se/resolve?urn=urn:nbn:se:his:diva-3208.

Relevant Journal publications

*Johansson, U., **König, R.** & Niklasson, L., 2010. Genetically evolved kNN ensembles. *Data Mining, Annals of Information Systems*, Springer, 8, p.299–313.
König, R., Johansson, U. & Niklasson, L.. Increasing rule extraction comprehensibility, *International Journal of Information Technology and Intelligent Computing,* 1**,** pp. 303-314, 2006.

Relevant International Conference Publication

König, R., Johansson, U., Löfström, T. & Niklasson, L., 2010. Improving GP classification performance by injection of decision trees. In *IEEE Congress on Computational Intelligence*, IEEE CEC'10, IEEE, pp. 1–8.

König, R., Johansson, U. & Niklasson, L., 2010. Finding the Tree in the Forest. In *IADIS International Conference Applied Computing,* IADIS'10, IADIS Press, pp. 135-142.

Johansson, U., **König, R.** & Niklasson, L., 2010. Genetic rule extraction optimizing brier score. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO'10, ACM, pp. 1007–1014.

Johansson, U, **König, R.,** Löfström, T. & Niklasson, L. 2010. Using Imaginary Ensembles to Select GP Classifiers. In *European Conference on Genetic Programming*, EUROGP'10 Springer, pp. 278–288.

---

*König and Johansson are equal contributors

*Johansson, U., **König, R.,** & Niklasson, L., Evolving a Locally Optimized Instance Based Learner, *International Conference on Data mining*, DMIN'08, Las Vegas, Nevada, USA, pp. 124-129, 2008.

König, R., Johansson, U. & Niklasson, L., Using Genetic Programming to Increase Rule Quality, *International Florida Artificial Intelligence Research Society Conference,* FLAIRS'08, Coconut Grove, Florida, USA, pp. 288-293, 2008.

König, R., Johansson, U. & Niklasson, L., G-REX: A Versatile Framework for Evolutionary Data Mining, *IEEE International Conference on Data Mining Workshops,* ICDMW'08, Pisa, Italy, pp. 971-974, 2008.

König, R., Johansson, U. & Niklasson, L., Genetic Programming – a Tool for Flexible Rule Extraction, *IEEE Congress on Evolutionary Computation,* IEEE CEC'07, Singapore, pp. 1304-1310, 2007.

Johansson, U., **König, R.** & Niklasson, L., Inconsistency – Friend or Foe, *International Joint Conference on Neural Networks*, IJCNN'07, Orlando, Florida, USA, pp. 1383-1388, 2007.

Johansson, U., **König, R.** & Niklasson, L., Automatically balancing accuracy and comprehensibility in PM, *International Conference on Information Fusion*, FUSION'05, Philadelphia, PA, USA, 2005.

Johansson, U., **König, R.** & Niklasson, L., The truth is in there - Rule extraction from opaque models using genetic programming, *International Florida Artificial Intelligence Research Society Conference*, FLAIRS'04, Miami Beach, Florida, USA, pp. 658-663, 2004.

Johansson, U., **König, R.** & Niklasson, L., Rule Extraction from Trained Neural Networks using Genetic Programming, *International Conference on Artificial Neural Networks*, ICANN'03, Istanbul, Turkey, pp. 13-16, 2003.

---

* König and Johansson are equal contributors

Other Journal publications

Johansson, U., Löfström, T., **König, R.** & Niklasson, L., Why Not Use an Oracle When You Got One?, *Neural Information Processing - Letters and Reviews*, pp. 227-236, 2006.

Johansson, U., **König, R.**, Löfström, T., Sönströd, C. & Niklasson, L., 2009. Post-processing Evolved Decision Trees. *Foundations of Computational Intelligence Volume 4: Bio-Inspired Data Mining*, pp.149-164.

Other International Conference Publication

Johansson, U., Sönströd, C. , Löfström, T. & **König, R.** 2009. Using Genetic Programming to Obtain Implicit Diversity. In IEEE Congress on Computational Intelligence, IEEE CEC'09. Trondheim, Norway, pp. 2454 - 2459.

Sönströd, C., Johansson, U., **König, R.** & Niklasson, L., Genetic Decision Lists for Concept Description, *International Conference on Data Mining*, DMIN'08, Lasvegas, Nevada, USA, pp. 450-457, 2008.

Johansson, U., Boström, H. & **König, R.**, Extending Nearest Neighbor Classification with Spheres of Confidence, *International Florida Artificial Intelligence Research Society Conference*, FLAIRS'08, Coconut Grove, Florida, USA, pp. 282-287, 2008.

Johansson, U., **König, R.**, Löfström, T. & Niklasson, L., Increasing Rule Extraction Accuracy by Post-processing GP Trees, *IEEE Congress on Evolutionary Computation*, IEEE CEC'08, Hong Kong, China, pp. 3010-3015, 2008.

König, R., Johansson, U. & Niklasson, L., Instance ranking using ensemble spread, *International Conference on Data Mining*, DMIN'07, Las Vegas, Nevada, USA, pp. 73-78, 2007.

Sönströd, C., Johansson, U. & **König, R.**, Towards a Unified View on Concept Description, *International conference on data mining*, DMIN'07, Las Vegas, Nevada, USA, pp. 59-65, 2007.

Johansson, U., Löfström, T., **König, R.** & Niklasson, L., Building Neural Network Ensembles using Genetic Programming, *International Joint Conference on Neural Networks*, IJCNN'06, Vancouver, Canada, pp. 1260-1265, 2006.

Johansson, U., Löfström, T., **König, R.**, Sönströd, C. & Niklasson, L., Rule Extraction from Opaque Models - A Slightly Different Perspective, *International Conference on Machine Learning and Applications*, ICMLA'06, Orlando, Florida, USA, pp. 22-27, 2006.

Johansson, U., Löfström, T., **König, R.** & Niklasson, L.. Genetically evolved trees representing ensembles, *International Conference on Artificial Intelligence and Soft Computing*, ICAISC'06, 613-622, 2006.

Johansson, U., Löfström, T., **König, R.** & Niklasson, L., Introducing GEMS - A novel technique for ensemble creation, *International Florida Artificial Intelligence Research Society Conference,* FLAIRS'06, pp. 700-705, 2006.

Löfström, T., **König, R.**, Johansson, U., Niklasson, L., Strand, M. & Ziemke, T., Benefits of relating the retail domain and information fusion, *International Conference on Information Fusion*, FUSION'06, Florence, Italy, pp. 129-134, 2006.

Johansson, U., Niklasson, L. & **König, R.**, Accuracy vs. comprehensibility in data mining models, *International Conference on Information Fusion*, FUSION'05, Stockholm, Sweden, pp. 295-300, 2004.

Johansson, U., Sönströd, C., **König, R.** & Niklasson, L., Neural networks and rule extraction for prediction and explanation in the marketing domain, *International Joint Conference on Neural Networks*, IJCNN'03, Portland, Oregan, USA, pp. 2866 - 2871, 2003.

Johansson, U., Sönströd, C. & **König, R.**, Cheating by Sharing Information – The Doom of Online Poker?, *International Conference on Application and Development of Computer Games in the 21st Century*, Hong Kong, China, pp. 16-22, 2003.

Aknowledgement

Writing a PhD thesis is a lot like climbing a mountain. You start out with a dream about reaching the summit but without a clue of how to get there. Sometimes you are completely lost, sometimes you think you can almost see the top and in the next moment it feels like you are carrying the mountain on your shoulders. To succeed, you need a team of outstanding people that can provide support, guidance and encouragement all the way to the end. I have been very lucky with my team and I want to thank all of you who made this thesis possible.

First of all I want to thank the leader of this expedition, my main supervisor Professor Lars Niklasson, for guiding me with ageless wisdom and bottomless optimism. Your advice and encouragement always kept me on track and got me out of that sticky bog of thesis writing, where I would otherwise still be, rewriting the same sentences over and over again.

Next, I want to give my sincere and warm thanks to my day to day Sherpa and assistant supervisor, Associate Professor Ulf "Hawkeye" Johansson, who motivated me to start my PhD studies and then guided my daily work with keen eyes. Without you I would surely still be wandering, forever lost, on the endless plains of rejected papers.

Every expedition needs responsible sponsors and naturally I'm also grateful to the University of Borås, University of Skövde and the Knowledge Foundation, for funding my PhD studies. Especially, Rolf Appelqvist and Lars Niklasson who made it happen.

I also want to thank all the great people in my research group CSL@ABS, Ulf Johansson, Tuve Löfström, Cecilia Sönströd, Håkan Sundell, Anders Gidenstam, Henrik Boström, Patrick Gabrielsson, Henrik Linusson, Karl Jansson and Shirin Tavara. Thanks for numerous fruitful discussions and great company, you kept my mood up, during this long and sometimes rainy trek to the top. Special thanks to Tuve who has been in the same boat as me from the start and shared all ups and downs. It is always easier to not be the only über-frustrated PhD-student in the corridor.

I sincerely want to thank Johan Carlstedt for being a great friend and support during the whole expedition. I really appreciate the beer, therapeutic talks and insights from the "real world". Thanks to Andréas Stråhle (who helped giving birth to G-REX in a dark and dusty computer room a long long time ago) and to Marcus Hedblom (who showed me that this was possible), I am still waiting for you to finish pre-processing the data for our first joint article, the birds need us!

Naturally, I want to thank my beloved family; first and foremost my parents for supporting me in all ways possible all these years. My brother and sister, you keep

my life stable and I know that you are always there for me if I need you and I will always be there for you.

Last but definitely not least, tanks to my dear wife Lorena and my amazing daughter Madeleine. Without you I would have given up long time ago. Lorena, you support me in everything and show me by example, again and again, that all is possible if you are prepared to do the work. I am so proud of you! Madeleine, you are a ray of light that can pierce any thundercloud and that is so valuable and precious for a family and for a PhD-student under a lot of stress. You were the sunshine that gave me the energy and motivation for those last steep steps to the top! Keep true to yourself and the world will surely be yours!

Thank you all for being on my team during this expedition on the steep and dangerous mountain of dissertation!

1 Introduction

Knowledge Discovery (KDD) is a field within computer science that focuses on extracting hidden knowledge from observations of a phenomenon. In the cross industry standard process for data mining (CRISP-DM), defined by Chapman et al. (1999), the KDD process is divided into six phases: *business understanding*, *data understanding*, *data preparation*, *modeling*, *evaluation,* and *deployment*. *Data mining,* which is the core of the modeling phase, is often defined as the exploration (automatic or semi-automatic) and analysis of large quantities of data, in order to discover meaningful patterns and rules in large quantities of data, e.g., see (Berry & Linoff 2004) or (Witten & Frank 2005). It is through the use of data mining techniques that new knowledge may be discovered.

Predictive Modeling (PM) is an important branch within data mining, where the goal is to make predictions about some phenomenon, based on previous observations. Normally, the creation of a predictive model is data driven. A dataset comprises a set of observations, or *instances*, which consist of values for *independent* variables that describe the phenomenon and *dependent* variables (most often only one) which define the phenomenon.

In PM, depicted in Figure 1, the extracted knowledge takes the form of a model that catches the relationship between the independent variables and the dependent variable.

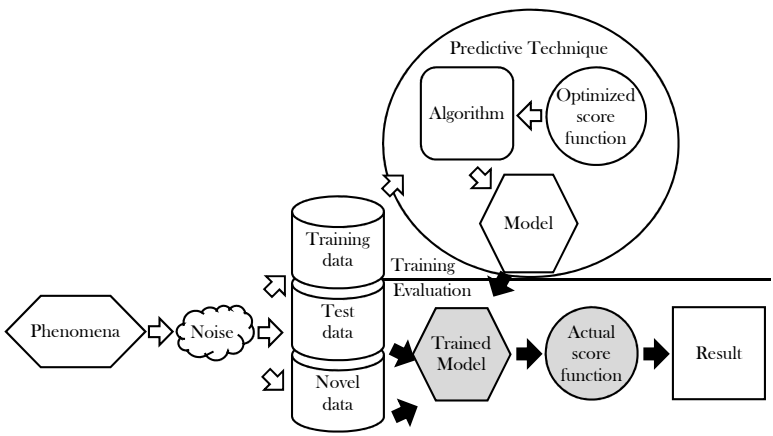

Figure 1 - Overview of Predictive Modeling

In essence, a *model* is nothing other than a set of *functions* and *parameters* connected in some *structure*. The structure, with its functions and parameters, of a certain model type is also called a model *representation*. In data driven approaches,

a model is created, or *trained*, using an algorithm, which is applied to a *training set* that contains instances where the value of the dependent variable is known. When trained, a model can be used to predict the value of *novel* instances for which the values of the dependent variable are unknown.

The training aims to make the model as *accurate* as possible, i.e., to minimize or maximize a *score function* that describes how well the model solves the predictive problem. A score function may concern classification, estimation, ranking, or some of the numerous other properties that a predictive problem can require of the sought model.

Considering that so many score functions exist and that no score function is optimal for all predictive problems, it is surprising that most predictive techniques used today are restricted to optimization of a single, predefined score function. Hence, a predictive technique's *optimized score function* sometimes differs from the *actual score function* that is defined by the problem. Naturally, it would always be preferable to use a technique which optimizes a score function that is identical to the actual score function.

Normally, a phenomenon is observed indirectly, since the independent variables must be measured and are thus exposed to measurement- and human errors. Independent variables usually also contain some kind of randomness in how it affects the dependent variable. Together, measurement errors and randomness of the independent variables (most often called noise) obscure the underlying patterns and make them harder to find.

The presence of noise is always a challenge for PM techniques, since it is the underlying relationships and not the noise that the model should learn. Models that are *overfitted*, i.e., they are too accurate because they have not only learned the underlying relationship but also some of the noise, will perform worse on new novel data, since the learned noise per definition is random. Therefore, models are normally evaluated on a separate *test set* that has not been part of the training. Consequently, the term accuracy refers to the result measured using the score function on the test set, since this should be a better approximation of accuracy on novel data (Freitas 2002).

Today, there are numerous PM techniques which all produce predictive models using different algorithms. The models can differ in their structure and functions, but they all aim to create a mapping between the independent and dependent variables.

Freitas (2002) argues that three general properties should be fulfilled by a predictive model; namely, it should be *accurate*, *comprehensible*, and *interesting*. Accuracy has been defined above and comprehensibility means that the reasons behind a certain prediction should be understandable. The last property,

interestingness, is a very subjective quality which can be hard to achieve. Normally, simple qualities, for example, that the discovered knowledge should capture relationships in the unknown data and or fulfill some user-defined constraints, are used to evaluate whether a model is interesting or not. Freitas also points out that even if interestingness obviously is a very important property, very few techniques are designed to find interesting knowledge. For anomaly detection applications, such as intrusion detection, it is, for example, only the intrusions that are interesting, even if they only make up an extremely small portion of the recorded observations.

A downside of comprehensible models is that in general they are less accurate than *opaque* models, i.e., models that are not easily grasped due to their complexity. Freitas does, however, like Craven and Shavlik (1996), argue in favor of comprehensible models, since models that are hard to understand are often met with skepticism. Another advantage, noted by Andrews, Diederich, and Tickle (1995), is that comprehensible models can be verified, which can be important for safety- or legal reasons. Finally, Goodwin (2002) points out that only predictions made by comprehensible models can be adjusted in a rational way, should a drastic change in the environment suddenly occur.

There is no single or true way to measure comprehensibility, since it is a subjective quality; what is comprehensible for one person could be incomprehensible for another. Factors, such as which and how many functions are used, the number of parameters the model contains, and even the structure, will affect how a model is perceived. Considering only comprehensibility would hence be optimal if a decision maker could choose these elements to his own liking.

Another important property for user acceptance is, according to Turney (1995) and Craven & Shavlik (1999), that the technique is consistent, i.e., it produces repeatable results. Turney argues for consistency, based on studies showing that decision makers tend to lose faith in a technique if it produces different models for different batches of data.

According to Dietterich (1996), the most fundamental source of inconsistency is that the hypothesis space is too large. If an algorithm searches a very large hypothesis space and outputs a single hypothesis, then in the absence of huge amounts of training data, the algorithm will need to make many more or less arbitrary decisions, decisions which might be different if the training set were only slightly modified. This is called informational instability; instability caused by the lack of information, which is naturally a problem for all PM techniques. Informational instability is, for example, often experienced in the medical domain, where datasets often contain a small number of instances (sometimes 100 or less) but still a relatively large number of features. In such cases (large spaces with sparse samples), it is quite likely that different logical expressions may accidently classify

some data well, thus many data mining systems may find solutions which are precise but not meaningful, according to experts; see e.g., Grąbczewski & Duch (2002).

The choice of using a more accurate opaque model or a less accurate comprehensible model is a common dilemma within PM and often called the accuracy versus comprehensibility tradeoff, e.g., see (Domingos 1998) or (Blanco, Hernandez & Ramirez 2004). Naturally, comprehensible models are preferable for most decision support applications, since they can be used to motivate a particular decision.

Rule extraction (RE) is an approach for the accuracy versus comprehensibility tradeoff that tries to transform an accurate opaque model into a comprehensible model while retaining high accuracy. The basic assumption is that an opaque model can remove noise from the original dataset and thus simplify the prediction problem. Normally, the extracted model is used to explain the prediction of the opaque model, but it can, of course, also be used as a normal standalone predictive model.

Based on the discussion above, it is surprising that most predictive techniques are restricted to a predefined model type which is optimized using one or two predefined score functions. Of course, there are many techniques and a decision maker could, in theory, choose a technique that uses an optimization criteria and a model that fit the problem, but traditional data mining techniques tend to optimize the same score functions which limit the choice in practice. Furthermore, choosing among all the techniques would require a vast knowledge of PM, since every predictive technique has different design parameters that need to be tuned to achieve maximum performance. In practice, it is very rare that decision makers and modelers have this knowledge.

Decision trees, which is one of the most popular techniques in the data mining community, can, for example, only be used to induce tree structured models using a predefined score function such as the *gini* or *entropy* measure. Decision tree techniques are still very popular, since they are generated relatively quickly and they produce comprehensible models. Greedy top-down construction, i.e., building a model stepwise while optimizing each step locally, is the most commonly used method for tree induction. Even if greedy splitting heuristics are efficient and adequate for most applications, they are essentially suboptimal (Murthy 1998). More specifically, decision tree algorithms are suboptimal, since they optimize each split locally without considering the global model. Furthermore, since finding the smallest decision tree that is consistent with a specific training set is a NP- complete problem (Hyafil & Rivest 1976), machine learning algorithms for constructing decision trees tend to be non-backtracking and greedy in nature (Jeroen

Eggermont, Kok & Kosters 2004). Hence, due to the local non-backtracking search, decision trees may become stuck in local minima.

Recently, researchers such as Espejo, Ventura, & Herrera, (2010); Freitas, (2007) and Eggermont, (2005) have shown an increased interest in using a technique suggested by Koza (1989) called genetic programming (GP) for PM. GP basically performs a search, based on Darwin's (1859) theory of natural selection, among all possible computer programs. The method performs a global search that makes use of hyperplane search and is hence less likely to become stuck in the local optimum (Jabeen & Baig 2010).

A GP search starts with a randomly generated population of programs, e.g., a predictive model, which are then assigned a *fitness value* based on their performance on a training set. The fitness value is calculated using a *fitness function* which, in essence, is the score function of the technique. Next, a new generation of programs is created by selecting individuals with a probability based on their fitness and, as in nature, applying genetic operations, i.e., *sexual crossover*, *mutation*, and *reproduction*. Crossover creates two new programs by swapping randomly selected substructures from two selected programs. Mutation randomly changes a substructure of a program and reproduction simply copies a program into the new generation unchanged. When the new generation is the same size as the previous population, the process restarts and the new programs are evaluated using the fitness function. The process is repeated until a sufficient performance or a preset number of generations are achieved.

Koza (1992) showed that GP could be used successfully in a wide range of domains including optimal control, planning, sequence induction, discovering game playing strategies, and PM; i.e., induction of decision trees, symbolic regression, and forecasting. Furthermore, O'Neill et al. (2010) list nine (of many) applications where GP has outperformed traditional predictive techniques. Espejo, Ventura and Herrera (2010) note that the main advantage of GP is that it is a very flexible technique which can be adapted to the need of a particular problem. There are three key features that make GP successful in such a wide variety of problems.

- An arbitrary fitness function can be optimized, since the only requirement is that stronger individuals are given a higher fitness.
- The search is independent of the representation, since it is only the output of a program that is evaluated.
- GP performs a global optimization, since a program is evaluated as a whole.

The same features make GP suitable for PM; a fitness function can be designed directly from the score function used for evaluation, the representation could be tailored to the preferences of a decision maker, and a global optimization should ensure a high predictive performance. Finally, Koza also showed how a *parsimony pressure* can be added to a fitness function, by adding a penalty related to the size of a program. The fitness could, for example, be calculated as the number of correct classified training instances made by a program minus the number of conditions in the program. Since a smaller program then receives a smaller punishment, it will be favored during selection. Parsimony pressure is thus a simple way of, to some extent, handling the accuracy versus comprehensibility tradeoff.

On the other hand, GP also has some disadvantages when used for PM. Koza does, for example, point out that GP rarely produces the minimal structure for performing the task at hand. Instead, the resulting programs often contain *introns,* i.e., unused or unnecessary substructures, which of course lower the comprehensibility of the program.

Another disadvantage sometimes mentioned in the literature is that GP is inherently inconsistent due to its probabilistic nature; see, e.g., (Martens et al. 2007). While this is true, it is mainly the underlying informational instability that causes GP to produce different programs every run. Hence, it could instead be argued that GP's inconsistency, if handled with care, could be considered an advantage, since it may shed light on the informational instability of the data and provide alternative solutions.

Furthermore, Eggermont, Kok and Kosters (2004) call attention to the fact that GP searches among all possible programs, which can become a problem, since the search space tends to become extremely large and could, in theory, even be infinite. The search space becomes large because GP creates programs of variable size and the number of possible programs grows exponentially with the maximum allowed size of a program (Poli, Langdon & McPhee 2008). Even if the GP search is powerful, it is also computationally expensive and the necessary time may not be available to solve the problem at hand. When the sizes of the search space become extremely large, it becomes difficult for the standard GP algorithm to achieve a decent classification performance (Jeroen Eggermont, Kok & Kosters 2004).

Finally, O'Neill et al. (2010) argue that there is a need for software that is easier to use and more intuitive. It must be easy for a practitioner to tune parameters, select fitness function, and choose function- and terminal sets.

1.1 Key Observations

The previous description leads to the following, general key observations about PM:

- The nature of the optimized score function has a direct impact on the performance of a predictive model.
- Predictive models should be accurate, comprehensible, and interesting for a decision maker.
- Comprehensible models are more easily accepted by users, can be verified, facilitate rationale manual adjustments, as well as used to explain the underlying relationship.
- Since comprehensibility is a subjective quality, a decision maker should be able to decide the representation of the predictive model, according to his own preference.
- Rule extraction is an approach that may transform opaque models into comprehensible models while retaining high accuracy.
- The accuracy versus comprehensibility tradeoff is an important dilemma for PM.
- Informational instability is a problem for all PM techniques, since most decision makers would prefer to only be presented with a single model for a predictive problem.

The following strengths of using GP for PM can also be identified:

- GP optimizes a model as a whole and hence performs a global optimization in contrast to the greedy search schemes used by most traditional predictive techniques.
- GP facilitates a match between the optimized- and the actual score function, since arbitrary score functions can be optimized using GP.
- GP can optimize models with arbitrary representation.
- To some extent, GP can handle the accuracy versus comprehensibility tradeoff by applying parsimony pressure during evolution.
- If handled with care, the inconsistency of GP could be considered an advantage, since it may shed light on the informational instability of the data and provide alternative solutions.

However, there are also challenges with using GP for PM:

- Since the number of possible programs grows exponentially with the allowed program size, it can sometimes be impractical to use GP if a large program is needed to solve a problem.
- The comprehensibility of GP programs is often reduced by the presence of introns.
- GP is not explicitly designed to produce a program that captures interesting relationships in the data.
- GP software needs to be intuitive and simple when used for PM.
- GP is, in general, very computationally intensive and therefore considered to be much slower than traditional data mining techniques.

1.2 Research question

Based on the key observations above, it is clear that GP has both strengths and weaknesses when used for PM. Hence, there is obviously room for improvement, but the question is how to best exploit the strengths and handle the weaknesses. The mentioned deficiencies can be categorized by how they affect the predictive performance and the comprehensibility of the produced predictive model. Of the deficiencies mentioned above, a large search space naturally affects the predictive performance, while the presence of introns and the inherent inconsistency affect the comprehensibility of the model. Hence, this thesis focuses on developing techniques that enhance the predictive performance and comprehensibility, by counteracting these deficiencies and utilizing the inherent strengths of GP. Another challenge of using GP for PM is, of course, the computational performance, i.e., the time it takes to create the model. Computational performance is, however, not in the scope of this thesis, since most enhancements in the field regard parallelization and are mostly independent of the underlying GP algorithm. With this said, the developed techniques must nonetheless be practical for most non real-time dependent problems to be of interest. The research question of this thesis could therefore be phrased as:

Can the deficiencies and strengths of Genetic Programming, when used for Predictive Modeling, be reduced and exploited by novel techniques, to enhance the accuracy and comprehensibility of the generated models?

1.3 Thesis Objectives

With the research question and the key observations in mind, the thesis objective is to develop, implement, and evaluate novel techniques for improving predictive models created using genetic programming. The focus is more specifically to:

1. Identify criteria for a GP framework for PM, from literature.
2. Identify or develop a GP framework, according to the suggested criteria.
3. Explore the inherent advantages and challenges of using GP for PM.
4. Develop and evaluate techniques that enhance the accuracy of predictive models created using GP.
5. Develop and evaluate techniques that enhance the comprehensibility of predictive models created using GP.

1.4 Main Contribution

This thesis aims to suggest novel techniques that enhance predictive performance and comprehensibility when GP is used for PM. A further aim is to create an intuitive framework for PM using GP that implements the suggested techniques. To create a stable high-performing base for this framework, the first part of the thesis explores the practical implications of the advantages and deficiencies of using GP for PM, identified in the introduction. The result is a set of general recommendations, for GP predictive frameworks regarding how best to exploit the advantages of GP, and a set of areas, related to accuracy and comprehensibility, where the traditional GP algorithm needs adaptation when used for PM. The main contribution of this thesis is a set of techniques that enhances the accuracy and comprehensibility of predictive models optimized using GP.

With regard to comprehensibility, three novel techniques are proposed:

- A rule extraction technique that produces accurate and comprehensible models by exploiting the inherent GP strengths.
- A representation free GP technique to increase the comprehensibility of predictive models by the removal of introns and the simplification of expressions.
- A technique to generate and guide a decision maker among numerous alternative models, thus enhancing the fit between the final model and current domain knowledge.

Furthermore, five techniques that are directly designed to improve predictive performance are presented. Of these, the last three exploit GP's inconsistency, which is often considered a disadvantage, to improve predictive performance:

- A technique that employs a local-based search using least square to evolve accurate regression model trees.
- A novel hybrid approach where the GP search is aided by the injection of decision trees and the automatic adjustment of the parsimony pressure.
- A technique to improve the probability estimates of the evolved programs.
- A technique to select the best model among a set of accurate models.
- A technique to create accurate kNN-ensembles.

Finally, a GP-based predictive framework, called G-REX, is implemented. G-REX fulfills most of the suggested recommendations, i.e., exploits the advantages and handles the disadvantages by realizing most of the suggested techniques.

1.5 Thesis Outline

The theoretical background is presented first in Chapter 2. Sections 2.1-2.4 regard PM and the evaluation of predictive models, while section 2.5 describes GP in the scope of PM. The criteria for GP frameworks for PM are presented in Chapter 3 together with an evaluation of thirteen existing GP frameworks. Chapter 4 presents the design of the GP engine of G-REX, the GP framework used in this thesis. More details about G-REX as a GP framework for PM are provided in the appendix section 11.1.

Chapter 5, the first research chapter, explores the advantages and disadvantages of GP empirically. The main contributions, in the form of enhancements of GP for predictive modeling, are presented in the two following chapters, with enhancements related to accuracy in Chapter 6 and comprehensibility in Chapter7. Finally, conclusions are drawn in Chapter 8, future work is provided in Chapter 9, followed by references in Chapter 10 and the appendix in Chapter 0.

2 Background

2.1 Knowledge Discovery

Roiger and Geatz (2003) define knowledge discovery in databases, (another name for Knowledge Discovery), as an interactive, iterative procedure that attempts to extract implicit, previously unknown, useful knowledge from data. As mentioned in the introduction, CRISP-DM divides this process into a cycle of six phases.

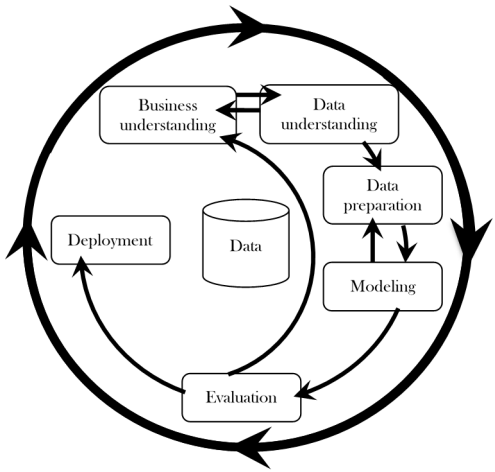

Figure 2 - Phases of CRISP-DM

- *Business understanding.* This initial phase focuses on understanding the project objectives and requirements from a business perspective, then converting this knowledge into a definition of the data mining problem and a preliminary plan designed to achieve the objectives.
- *Data understanding.* The data understanding phase starts with an initial data collection and proceeds with activities to become familiar with the data. Here the aim is to identify data quality problems, gain first insights into the data or detect interesting data subsets, in order to form hypotheses for hidden information.
- *Data preparation.* The data preparation phase covers all activities to construct the final dataset. It is likely that data preparation tasks are performed multiple times and not in any prescribed order. Tasks include

instance and attribute selection, as well as the transformation and cleaning of data for modeling tools.

- *Modeling.* In this phase, various data mining techniques are selected and applied, while their parameters are calibrated to optimal values. Some techniques have specific requirements regarding the form of the data. Therefore, returning to the data preparation phase is often necessary.
- *Evaluation.* An analysis of the developed model to ensure that it achieves the business objectives. At the end of this phase, a decision on the use of the data mining result should be reached.
- *Deployment.* The deployment phase can be as simple as generating a report or as complex as implementing a repeatable data mining process across the enterprise.

Even if the CRISP-DM document is still at the 1.0 version that was created 1996, CRISP-DM I remains the most widely used methodology for data mining, according to a poll conducted by Piatetsky-Shapiro (2007). The second most used methodology, according to the same poll, is SAS Instruments (2011) SEMMA, which stands for Sample, Explore, Modify and Asses. SEMMA is similar to CRISP-DM, but focuses more on the technical aspects of data mining.

Data mining is often defined as the automatic or semi-automatic process of finding meaningful patterns in large quantities of data. The process needs to be automatic, due to the very large amounts of available data, and the patterns found needs to be meaningful.

Dunham (2003) makes a clear distinction between KDD and data mining, where KDD is used to refer to a process consisting of many steps, while data mining is performed within one of these steps, i.e., the modeling step. However, even if data mining is the core of the modeling phase, it is also strongly connected to the preparation and evaluation phases.

Data mining problems are usually broken down into a number of tasks that can be used to group techniques and application areas. Even if the number of tasks and the names of the tasks vary slightly, they all cover the same concept; Berry and Linoff (2004) have devised the following definition:

- *Classification*: The task of training some sort of model capable of assigning a set of predefined classes to a set of unlabeled instances. The classification task is characterized by well-defined classes and a training set consisting of pre-classified examples.

- *Estimation*: Similar to classification, but here the model needs to be able to estimate a continuous value instead of just choosing a class.
- *Prediction*: This task is the same as classification and estimation, except that the instances are classified according to some predicted future behavior or estimated future value. *Time series prediction,* also called *forecasting*, is a special case of prediction based on the assumption that values are dependent on previous values in the time series. Time series methods are either *univariate* (one variable is forecasted based on its past realizations) or *multivariate* (when several variables are used to predict the dependent variable).
- *Affinity Grouping* or *Association Rules*: The task of affinity grouping is to determine which things go together. The output is rules describing the most frequent combinations of the objects in the data.
- *Clustering*: The task of segmenting a heterogeneous population into a number of more homogeneous subgroups or *clusters.*
- *Profiling or Description:* The task of explaining the relationships represented in a complex database.

Berry and Linoff (2004) further divide these tasks into the two major categories of *predictive* and *descriptive* tasks. Classification, estimation, and prediction are all predictive in their nature, while affinity grouping, clustering, and profiling can be regarded as descriptive.

- *Predictive tasks:* The objective of predictive tasks is to predict the value of a particular attribute, based on values of other attributes. The attribute to be predicted is commonly known as the *target* or the *dependent variable*, while the attributes used for prediction are known as *explanatory* or *independent variables*.
- *Descriptive tasks:* Here the objective is to derive patterns (correlations, trends, clusters, trajectories, and anomalies) that summarize the underlying relationships in the data. Descriptive data mining tasks are often exploratory in nature and frequently require post-processing techniques to validate and explain results.

This thesis aims to suggest techniques that increase the accuracy and comprehensibility of predictive models.

2.2 PM techniques

Numerous machine learning techniques for PM have been proposed during the years and all have their own advantages and disadvantages. When discussing predictive techniques, it is important to remember that there are no "free lunches" in PM, i.e., there is no technique that will always outperform all others if evaluated over a large set of problems (Wolpert 1995). Hence, it is important to always compare new techniques against other established methods over a large dataset. The following sections present the predictive techniques that have been used as benchmarks or complement, when evaluating GP as a PM technique. The techniques have mainly been selected on the basis of their popularity and benchmark suitability for the respective study.

2.2.1 Linear Regression

Linear regression is a simple technique suitable for numeric prediction frequently used in statistical applications. The idea is to find the amount of how much each of the attributes $a_1 a_2 ... a_k$ in a dataset contributes to the target value x. Each attribute is assigned a factor w_i and one extra factor is used to constitute the base level of the predicted attribute.

$$x = w_0 + w_1 a_1 + w_2 a_2 + \cdots + w_k a_k \quad (1)$$

The aim of linear regression is to find the optimal weights for the training instances, by minimizing the error between the real and the predicted values. As long as the dataset contains more instances than attributes, this is easily done using the least square method (Witten & Frank 2005). Dunham (2003) points out that linear regression is quite intuitive and easily understood but its downside is that it handles non-numerical attributes poorly and it cannot handle more complex non-linear problems.

2.2.2 Decision Trees

Decision trees (DT) are machine learning models most suited for classification tasks, but they are also applicable to regression problems. In the data mining community, decision trees have become very popular, because they are relatively fast to train and produce transparent models. A decision tree can be seen as a series of questions, arranged in a tree structure, leading to a set of predefined classes. Most often a decision tree consists of nodes containing either a prediction or a Boolean condition (the question) with two corresponding child nodes.

The conditions are chosen to split the dataset used for training into a smaller purer set of records, i.e., sets of instances which are dominated by a single target

class. When it is impossible to find a condition in a node that will make the resulting sets purer, it is marked as a leaf node and labeled, in order to predict the majority class of the instances reaching the node.

When used for classification, the root node is first evaluated, followed by the left or the right node, depending on the outcome of the condition. This process is repeated for each resulting node until a leaf node with a prediction is reached. A path from the root to a leaf can be seen as a rule consisting of simple if else conditions. Figure 3 below shows a simple decision tree for the diabetes dataset from the UCI Repositroy (Blake & Merz 1998). The tree has two conditions which, depending on a patient's plasma and insulin level, predict whether the patient has diabetes or not, using three leaf nodes.

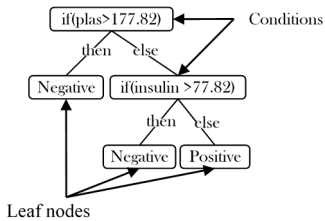

Figure 3 - Simple Decision Tree for the diabetes dataset

Creation

A decision tree is built to minimize the classification error on a training set. The creation of the tree is achieved recursively by splitting the dataset on the independent variables. Each possible split is evaluated by calculating the *purity gain* it would result in if it were used to divide the dataset D into the new subsets $S = \{D_1, D_2, \dots D_s\}$. The *purity gain* is the difference in purity between the original dataset and the subsets, as defined in equation 2, where $P(D_i)$ is the proportion of D that is placed in D_i. The split resulting in the highest purity gain is selected and the procedure is then repeated recursively for each subset in this split.

$$gain(D,S) = purity(D) - \sum_{i=1}^{s} P(D_i) * purity(D_i) \qquad (2)$$

There are several different DT algorithms, such as Kass' (1980) CHAID, Breiman's (1984) CART, and Quinlan's (1986; 1993) ID3 and C4.5, which all use slightly different purity functions.

ID3 uses information gained as purity function which is based on the entropy metric that measures the amount of uncertainty in a set of data. Information gain is calculated for all possible conditions for all independent variables and the entropy E is defined in equation *3*, where p_c is the probability of randomly choosing an instance of class c (of C classes) in the dataset S.

$$E(S) = \sum_{c=1}^{C} \left(p_c log \left(\frac{1}{p_c} \right) \right) \quad (3)$$

Entropy ranges between 0 and 1 and reaches a maximum when all class probabilities are equal.

CART, uses the *gini diversity index* (GDI) that measures the class impurity in the node t, given the estimated class probabilities $p^2 * (j|t), j = 1, \dots, J$ (for J classes). GDI is given by equation 4.

$$GDI(S) = 1 - \sum_{c=1}^{C} {p_c}^2 \quad (4)$$

Pruning

When a DT is fully grown, it is optimized on the training set, which often leads to an over fitting of the model and thereby a high generalization error. Pruning is a method that analyzes a decision tree to make it more general, by removing weak branches. Removing parts of a tree will, of course, result in a decrease in training accuracy, but the idea is that it will make the tree more general and perform better on new unseen data.

There are two approaches to pruning; *prepruning* that tries to stop the growth of the tree before weak branches occur and *postpruning* where a fully grown tree is first created and then pruned. Witten and Frank (2005) note that postpruning has some favorable advantages, for example, a series of conditions can be powerful together, even when they all are weak by themselves.

In general, postpruning algorithms generate candidate subtrees which are evaluated on the training data or a new validation set containing previously unseen instances. Exactly how these candidate subtrees are created differs between algorithms, but they all apply *subtree replacement* and/or *subtree raising* in some fashion. Subtree replacement starts in the leaves of the tree and replaces the selected subtrees with single leaves. Subtree raising moves a subtree to a higher position in its branch, deleting intermediate nodes. During pruning, a large number

of candidate subtrees can be created and the tree with the best performance on the validation data is selected as the final tree.

Regression Trees

Only a small change of the original algorithm is required to use decision trees for regression. A constant numerical value should be predicted instead of a class and another type of purity measure is required. CART and REPTree do, for example, use the variance as purity measure and hence predict the value resulting in the lowest variance, see equation 5. REPTree is a decision tree technique implemented in the WEKA workbench that is optimized for speed. It builds trees by reducing the variance of the resulting subsets, T_1 and T_2 of each split (VarR) according to the equation below.

$$VarR = Var(T) - \sum_i \frac{T_i}{T} * Var(T_i) \tag{5}$$

By default, 2/3 of the training data are used to build the tree and 1/3 is used for *reduced error pruning*, a post-processing technique which prunes a tree to the subtree with the lowest squared error (SE) on the pruning data. Leaf constants are calculated as the mean value.

M5P is another decision tree technique that can be used to create regression trees based on Quinlan's (1992) M5, implemented in the WEKA framework. M5 first grows an optimal decision tree using all the training data by minimizing the standard deviation and then prunes it to minimize the expected RMSE of the tree. Since the tree is optimized for the training data, it will underestimate the true error of the tree. To compensate for this, the expected error is calculated by adjusting the RMSE in each leaf according to equation 6, where n is the number of instances reaching that leaf and v is the number of parameters of the model.

$$e = e\frac{(n+v)}{(n-v)} \tag{6}$$

Hence, the expected error will be calculated with larger pruning factors for large trees, since they will naturally contain leaves with fewer instances. The final pruned tree is the subtree with the lowest expected error. When used to create regression trees, the leaves predict the mean value of the training instance reaching the leaf.

Model trees

Regression trees with constants in the leaves are easy to create and interpret, but they are not always very accurate. To increase the predictive performance, Quinlan (1992) introduced a new type of decision trees called model trees. A model tree is similar to a normal decision tree with the exception that models are used as leaves instead of constants. The term model tree can be used for both classification- and regression trees. Quinlan proposed a new model tree algorithm for regression problems, called M5, which used multiple linear regressions as leaf nodes. A M5 model can be seen as a piecewise linear regression. A M5 tree is created by selecting each split in a way that minimizes the standard deviation of the subtrees. When the tree has fully grown, linear regressions are created using standard regression techniques for each node in the tree. However, the choice of variables is limited to the ones used in the tests or the models of the node's subtree. Next, each model is simplified by considering the estimated error at each node in the same way as for regression trees.

If a model consisting of a subset of the parameters used in the original model has a lower estimated error according to equation 6, it takes the place of the original model. Finally, each non-terminal node is compared to its subtrees in the same way. If the estimated error of the node is lower than its subtree, the subtree is replaced by the model. M5 can also use smoothing, where the predicted value is adjusted to reflect the predictions of the nodes along the path from the root to the node. Finally, Quinlan (1992) concludes that model trees are both more accurate and more compact than regression trees. Another notable difference is that model trees, like M5 trees, can extrapolate outside the range of the training instances.

Probability Estimation

Although the normal operation of a decision tree is to predict a class label based on an input vector, decision trees can also be used to produce class membership probabilities; in which case they are referred to as *probability estimation trees* (PETs). The easiest way to obtain a class probability is to use the proportion of training instances corresponding to a specific class in each leaf. In Figure 4, 6 training instances reach the lower right leaf, and 4 of those belong to class *Positive*. The assigned class probability for class *Positive*, in that leaf, would become *4/6=.67*. Consequently, a future test instance classified by that leaf would be classified as class *Positive*, with the probability estimator *.67*.

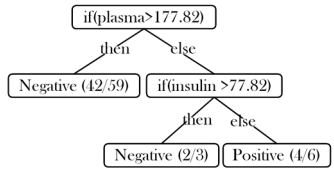

Figure 4 - Correct classifications / instances reaching each node

Normally, the relative frequencies are not used directly for probability estimations, since they do not consider the number of training instances supporting a classification. According to Margineantu and Dietterich (2003), the *Laplace estimate* is instead commonly used to produce calibrated probability estimates based on the support. Equation 7 shows how the probability estimate p is calculated when using Laplace. N is the total number of instances, C is the number of classes and k is the number of training instances supporting the predicted class c. Finally, n is the number of instances reaching the leaf.

$$p_c = \frac{k+1}{n+C} \tag{7}$$

In Figure 4 above, the probability estimate for the lower right node would be calculated as *4/6=.67* without Laplace and *((4+1)/(6+2))=.63* using Laplace. It should be noted that the Laplace estimator introduces a prior uniform probability for each class; i.e., before any instances have reached a leaf (k=n=0), the probability for each class is 1/C.

2.2.3 Artificial Neural Networks

Artificial neural networks (ANNs) are ML techniques loosely based on the function of the human brain. According to Kecman (2001), ANNs are extremely powerful in the sense that they are universal functional approximators, i.e., they can approximate any function to any desired accuracy. Multilayer perceptrons (MLP), which is one of the most common types of ANNs, have been used to solve a wide variety of problems and are frequently used for both classification and regression tasks, due to their inherent capability of arbitrary input output mapping (G. Zhang, Patuwo & Hu 1998).

Basics of a Neural Network

An MLP consists of a set of *units* (neurons) arranged in *layers* and connected by *weighted links* which pass signals between the units. The units in the first layer, the *input layer,* are connected to the input variables and receive input signals accordingly. When a signal is sent through a link, it is multiplied with the weight of the link, which thus decides the impact on the receiving unit. All incoming signals are summarized and an *activation function* is applied to calculate the responding output. Units in the input layer are connected to units in a *hidden layer* which can be connected to another hidden layer or to the *output layer*, see Figure 5.

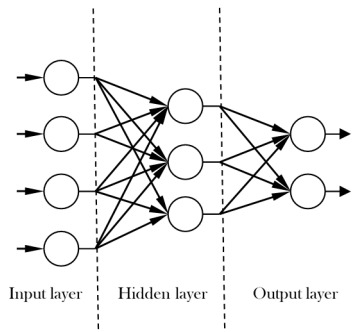

Figure 5 - A simple MLP

Hidden Layers and Hidden Units

The number of hidden layers and hidden nodes in an ANN is a crucial design choice as they allow the network to detect the features and capture the pattern in the data by performing complicated nonlinear mappings between input and output variables. Hornik (1991) shows that one hidden layer is sufficient for ANNs to approximate any continuous function with the desired degree of accuracy. According to Zhang, Patuwo, and Hu, (1998) however, two hidden layers can sometimes be preferable, since it can achieve the same result with less hidden units and approximate discontinuous functions. A network with fewer units will also be faster to train and should, in principle, generalize better.

The most common way to determine the number of hidden nodes is via initial experiments. Several rules of thumb have been proposed, but none of these works well for all problems.

Output Layer and Output Units

Selecting the number of output units is another important factor for the performance of the network. For regression problems, a single output unit is sufficient, but classification tasks are most often designed with a *one of C coding,* where each class is represented by an output unit. Classification is done according to the associated class for the unit with the highest output signal.

Activation Functions

The activation function determines the relationship between the inputs and outputs of a node and a network. In general, the activation function introduces a degree of nonlinearity that is valuable for most ANN applications. The most common activation functions are the squashing sigmoidal functions: the *unipolar logistic function* (8) and the *bipolar sigmoidal function* (9), i.e., see (Kecman 2001).

$$f(x) = \frac{1}{1 + e^{-x}} \tag{8}$$

$$f(x) = \frac{2}{1 + e^{-x}} - 1 \tag{9}$$

Different activation functions can be used in a network, but usually only one type is used in each layer. Zhang, Patuwo, and Hu (1998) recommend a simple *linear activation function,* (10) when the dependent variable is continuous.

$$f(x) = x \tag{10}$$

Training of a Neural Network

A trained ANN is basically a mapping between input and output variables of the data. Training of the ANN is an optimization process of the connection weights. The most commonly used training algorithm, *back propagation,* is a gradient steepest descent method. A step size often called *learning rate* and a momentum parameter must be set before training can begin. These parameters play a critical role in the performance of the final network, but there is no way to calculate the optimal learning rate and momentum for a given problem. Several experiments with different setting are often performed before selecting the final values.

According to Zhang, Patuwo, and Hu (1998), Broyden-Fletcher-Goldfarb-Shanno (BFGS) and Levenberg-Marquardt are two alternatives to the back propagation algorithm, which are more robust and have achieved significant

improvements in training time and accuracy. Another advantage of these methods is that the parameters used by the algorithms can be set automatically.

2.2.4 Instance-based learners

Instance-based learners or *lazy learners* differ from typical predictive techniques in that they do not require any training. Instead, the training instances are used to directly calculate a prediction for a novel instance. The most common technique is Fix and Hodges's (1952) *k-Nearest Neighbors* (kNN) which calculate a prediction based on the k closest (according to some distance measure) training instances. Majority voting among the closest neighbors is normally used for classification tasks, while the mean value is typically used for regression tasks. Hence, kNN bases the prediction on local information instead of using a global model that covers the whole input space, such as ANN or decision trees. In theory, this use of local information facilitates decision boundaries of arbitrary shape, which should be more powerful than the rectilinear decision boundaries that decision trees and rule learners produce.

Often, kNN is used as a benchmark for more powerful techniques like ANN and ensembles, since it, in general, performs well in spite of its simplicity see, e.g., (Bishop 1995). Even if kNN only has one parameter, i.e. k, its value must be set with care, since a too low value will make the technique sensitive to noise, while a too high value reduces the locality aspect which is the core of the technique. Naturally, different problems will require a different k and hence k is normally optimized using some kind of cross-validation on the training data. However, even for a single dataset, it is not certain that the same k is optimal for the whole input space and a single global k is, in essence, suboptimal. Zavrel (1997) did, however, suggest an approach to make kNN less sensitive for the value of k. The main idea was to apply weighted voting, i.e., each vote of a training instance is weighted according to its distance to the novel instance. Nevertheless, even if the problem is reduced, the technique is still restricted to using a single k-value for the entire dataset. An extension to the weighted voting approach is to employ axes scaling (or feature weighting), by somehow allowing more important features to have greater impact when measuring the distance and hence affecting the neighbor selection process.

2.2.5 Ensembles

An ensemble is a combination of the predictions made by several different classification or regression models. It is well known that a combination of several models can improve the accuracy of the prediction on unseen data, i.e., decrease the generalization error. At the same time, an ensemble is by definition a set of

models, making it very hard to express the overall relationship found in original variables. The option to use an ensemble is thus a choice of a black-box model prioritizing accuracy. Models can be combined by simply using majority voting (for classification) and by averaging (for regression) or weighting in some fashion.

The most intuitive explanation of why ensembles work is that combining several models using averaging will eliminate uncorrelated errors of the ensemble models; see, e.g., (Dietterich 1997). Naturally, there is nothing to gain by combining identical models, so the reasoning requires that base classifiers commit their errors on different instances. Informally, the key term diversity therefore means that the base classifiers make their mistakes on different instances.

Krogh and Vedelsby (1995) lay down the theoretical groundwork for ensembles presented below. Diversity (is here named ambiguity, $\bar{A}$) of the ensemble member models, $m_1 \dots m_n$ for the instances $x_1 \dots x_n$, can be calculated using equation 11, where ω_m is the weight for model m and $\bar{V}$ is the weighted ensemble average.

$$\bar{A} = \sum_{i=1}^{n} \left(\sum_{m} \omega_m \left(V^m(x_i) - \bar{V}(x_i) \right)^2 \right) \tag{11}$$

If $\bar{E}$ is the weighted average of the generalization error ($\bar{E} = \sum_m \omega_m E^m$), the generalization error of the ensemble E follows equation 12.

$$E = \bar{E} - \bar{A} \tag{12}$$

This means that the accuracy of the ensemble will increase if the ambiguities can be increased without increasing the generalization error. Note that the ambiguity is measured as the weighted average of the squared differences in the predictions of the models and the ensemble. Since diversity is always positive, this decomposition proves that the ensemble will always have higher accuracy than the average accuracy obtained by the individual classifiers. It must be noted, however, that there is no clear analogy to the bias-variance-covariance decomposition for classification. 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. Nevertheless, several studies have shown that sufficiently diverse classification ensembles, in practice, will almost always outperform even the strongest single model used in the ensemble. Brown et al. (2005) divide methods for creating diversity into explicit and implicit techniques. Implicit techniques optimize some measure of diversity directly while implicit techniques create diversity without actually targeting it. Typical implicit techniques

create diversity by dividing the training data, either by features or by instances, into slightly different training sets for each model.

Two of the most common implicit techniques, which can be used for both classification and regression tasks, are bagging (Breiman 1996) and boosting (Schapire 1990).

Bagging

In bagging, the ensemble members are created from different training sets formed by making *bootstrap* replicas of the original set. The bootstrap replicas are constructed by randomly selecting instances (with reselection) from the original training set. All replicas have the same number of instances as the original dataset, but may contain duplicates of some instances. To make a prediction for a regression problem the ensemble members are simply averaged.

Boosting

Boosting also creates different dataset replicas for each ensemble member, but the creation of the replicas is not independent. All replicas are copies of the original training set, but have different weights for each instance. An ensemble is created sequentially by adding one new member at a time. The weights are calculated depending on the performance of the current ensemble, by giving instances wrongly classified a high weight and thereby a high importance for the new member, while the correctly classified instances receive a low weight. When added to the ensemble, a classifier is usually weighted in some way that relates to its accuracy. Finally, Dunham (2003) points out that Boosting may not always help, especially if the training of the training data contains a lot of noise.

2.2.6 Rule extraction

One approach to overcome the accuracy versus comprehensibility tradeoff, mentioned in the introduction, is rule extraction; i.e., the process of transforming an accurate opaque model to a transparent model while retaining high accuracy, see Figure 6.

A totally different motivation for rule extraction could be that an opaque model exists and an explanation for its behavior is needed.

A decision maker can choose whether he wants to keep the prediction from the opaque model and only use the extracted model as an explanation or whether he wants to make new predictions with the extracted model. In general, the opaque model will still have a slightly higher accuracy, but using its prediction will result in a less transparent solution, as the extracted model will only explain the prediction to a certain degree. For this reason, the extracted model is also most often used to

make the predictions, as the extracted model should have retained a high accuracy and the rule extraction was motivated by a need of comprehensibility in the first case.

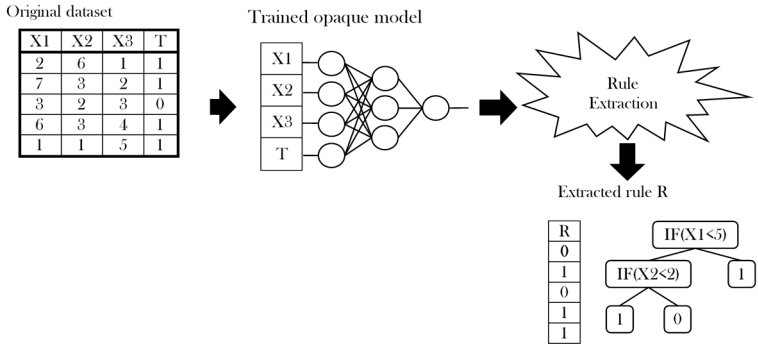

Figure 6 - Rule extraction

Rule extraction is quite a mature field in which much research has been conducted and several different methods have been proposed. To facilitate the comparison of rule extraction techniques and to set guidelines for further research, Craven and Shavlik (1999) suggested five criteria that should be fulfilled by good rule extraction algorithms listed below. These criteria were designed for rule extraction from ANNs, but only need small adjustments to be applicable to arbitrary opaque models.

- *Comprehensibility:* The extent to which extracted representations are comprehensible to humans.
- *Fidelity:* The extent to which extracted representations accurately model the networks from which they were extracted.
- *Accuracy:* The ability of extracted representations to make accurate predictions on previously unseen cases.
- *Scalability:* The ability of the model to scale, e.g., ANNs with large input spaces and large numbers of units and weighted connections.
- *Generality:* The extent to which the method requires special training regimes or restrictions on network architecture.

Some authors, like Towell and Shavlik (1993), add the criterion of consistency. A rule extraction algorithm is consistent if rules extracted for a specific problem are similar between runs; i.e., consistency measures how much extracted rule sets vary

between different runs. The motivation is that if extracted rule sets differ greatly, it becomes hard to put a lot of faith in one specific rule set.

Andrews, Diederich and Tickle (1995) divide rule extraction techniques into two basic categories, *decompositional* and *pedagogical*, and a third, *eclectic,* that combines the two basic approaches. Decompositional techniques try to extract rules by analyzing the internal structure of the opaque model (usually the hidden and output units of an ANN). An example of a decompositional technique is RULEX, for details see (Andrews & Geva 1995). Pedagogical techniques treat the model as a black box and are thus applicable for all kinds of models. Pedagogical techniques convert the rule extraction problem to a learning task from the original inputs to the model outputs, i.e., the original target variable is replaced by the predicted value. Two examples of pedagogical techniques are VIA (Thrun 1993) and TREPAN (Craven & Shavlik 1996). Most pedagogical techniques could, of course, be applied directly to the original dataset as a normal classifier. However, the underlying idea for this type of rule extraction is the assumption that the opaque model contributes in the rule creation process by cleaning the original dataset of noise. A dataset with less noise should be a better foundation for rule induction.

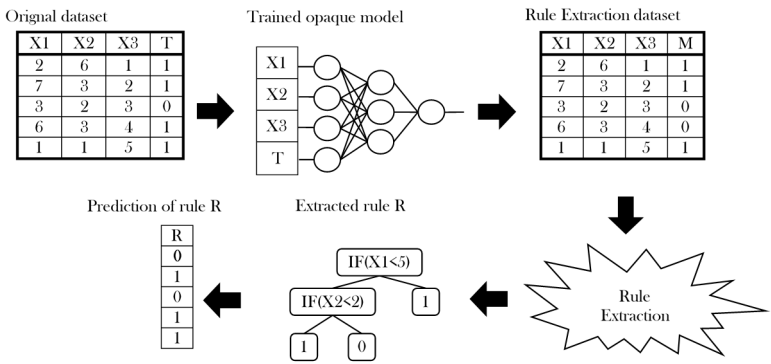

Figure 7 - Pedagogical rule extraction

When rule extracting from ensembles, only pedagogical rule extraction meets the generality criteria; i.e., it would be impossible to create a decompositional rule extraction technique general enough to work with an arbitrary ensemble. As ensembles, in general, are more accurate and robust than single techniques, only pedagogical rule extraction techniques are considered in this research.

2.3 Performance Measures for PM

Performance measures are mathematical functions that calculate the error or difference between the predicted value p and the actual value a. In PM both the score function and the actual score function are normally defined by a performance measure.

2.3.1 Performance Measures for Classification

Accuracy is by far the most common performance measure used for classification. However, Provost et al. (1998) argue that using accuracy for evaluating classifiers has two serious flaws; it is presumed that the true class distribution is known and that the misclassification cost is equal for all classes. This is regarded as a problem, as these assumptions are rarely true for real world problems. Instead, other measures such as Receiver Operating Characteristic (ROC) curve or Brier score are suggested as more appropriate measures for comparing classification performance. The following sections therefore describes these measures in detail. Other common metrics not considered in this thesis are *Precision*, *Recall*, *F-measure* and *Specificity;* e.g, *see,* (Makhoul, et al. 1999), (Bojarczuk, Lopes & Freitas 1999).

Accuracy (ACC)

Accuracy is simply the percentage of all n instances j that are classified correctly.

$$ACC = \frac{1}{n}\sum_{i}^{n}(j_i) \quad where\ j_i = \begin{Bmatrix} 1\ if\ p_i = a_i \\ 0\ if\ p_i \neq a_i \end{Bmatrix} \tag{13}$$

As mentioned above, accuracy assumes that the class distribution is known for the target environment. If this distribution is not known, accuracy is a poor performance measure, as it is not possible to claim that the evaluated model is indeed maximizing accuracy for the problem from which the dataset was drawn (Provost, Fawcett & Kohavi 1998).

Area Under the ROC-curve (AUC)

ROC curves measure the relationship between hit rate and false alarm rate and are based on estimations of the probability that an instance belongs to a certain class. ROC curves have an especially attractive property in that they are insensitive to changes in class distributions. ROC graphs are commonly used in medicine, radiology, psychology, credit scoring, and bioinformatic and have, in recent years,

been used increasingly in machine learning and data mining research; see, e.g., (Fawcett 2006).

In a ROC-curve diagram, the *y*-axis expresses the number of true positives and the false positives are shown on the *x*-axis. Both axes show the percentage of all true and false positives to give the unit square area the sum of 1. To draw a ROC-curve the instances are first ranked according to the probability that they belong to the positive class. Next, the curve is drawn by moving one step up for each instance that is a true positive and taking one step right if it is a false positive.

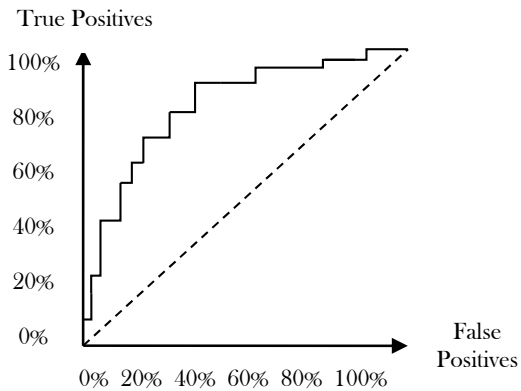

Figure 8 - Sample ROC curve

For a detailed description of how ROC curves are constructed see (Fawcett 2006). To compare classifiers using ROC curves, the curve has to be converted into a single scalar value, normally, the area under the ROC curve (AUC). Since ROC curves are projected onto the unit square, AUC will always be bounded by 0 and 1. An AUC of 1 means the predictive model is perfect, while random guessing corresponds to an AUC of .5 (the dotted line in Figure 8).

Like accuracy, AUC is not without deficiencies. Hand (2009) points out that the AUC can be misleading if ROC curves cross and can also assign different misclassification costs to different models. However, most researchers agree on that AUC is a better performance measure than accuracy.

AUC can be interpreted as the probability that a model will rank two randomly selected instances, one positive and one negative, correctly. Naturally, ranking is important if a subset of the instances is supposed to be selected for some action. A typical example could be a mail order campaign where the company wants to send *n* catalogs to the customers that are most likely to respond. In this case, ACC would not be sufficient, since it does not measure the quality of the probability estimates.

Brier Score (BRI)

A prediction made by a certain model can often be related to a probability estimation. Stefanova and Krishnamurti (2010) point out that for a single instance a probability estimation is neither correct nor wrong; probability estimations are verified by analyzing the joint (statistical) distribution of estimations and observations. While AUC only evaluates how good the produced probability estimates are for ranking the instances, the Brier score suggested by Brier (1950) evaluates how good these estimations really are. This is an important property when a reward or punishment is associated with a prediction. An example could be betting on a horse race where the selection of which horse to play on must be made on the probability estimate and the odds.

Equation 14 shows the definition of the Brier score (BRI), where n is the number of instances, r is the number of possible outcomes (classes), f_{ij} gives the probability estimate that the instance i belongs to class j. E takes the value 1 if the event has occurred and 0 if it has not.

$$BRI = \frac{1}{n}\sum_{j=1}^{r}\sum_{i=1}^{n}\left(f_{ij} - E_{ij}\right)^2 \tag{14}$$

A lower score is better and the lowest possible score is 0, which occurs when all classifications are done correctly with 100% certainty. Random guessing from a binary problem with an equal class distribution will result in a score of .5.

Brevity (BRE)

Sönströd, Johansson, and König, (2007) suggested that a comprehensible program should have high *brevity*, i.e., it should classify as many instances as possible with few conditions. The overall principle was that a typical instance should be classified using a few simple tests, while more complex rules should only be used for more atypical patterns. Brevity is clearly a desirable property, especially if costs are associated with the process of resolving the conditions, which could be the case with medical diagnosis or some chemical analysis

In a later study, König, Johansson, and Niklasson, (2008a) suggested a simple metric that quantified brevity (BRE) as the average number of conditions needed to classify all training instances, see equation 15. Note that *brevity* in this definition does not in any way regard the predictive performance of the model.

$$BRE_p = \frac{1}{n}\sum_{i=1}^{n} \#Conditions_{p_i} \tag{15}$$

2.3.2 Performance Measures for Estimation and Time Series Prediction

There are numerous performance measures for estimation tasks. Armstrong (2001) evaluates the most common and list their strengths and deficiencies. The following section will present some of the more interesting measure discussed by Armstrong.

Mean absolute Error (MAE)

The most straightforward estimate of the error of a regression model is the *mean absolute error* (MAE). It is easy to calculate and easily understood by managers and decision makers. However, it is scale dependent which, for instance, makes comparison over several datasets cumbersome. The MAE for a model m can be calculate using the equation below, where m_i is the prediction made by the model for the instance i and a_i is the actual value for the same instance.

$$MAE_m = \frac{\sum_{i=1}^{n} |m_i - a_i|}{n} \tag{16}$$

Root Mean Square Error (RMSE)

The *mean square error* (MSE) and *root mean square error* (RMSE) are two of the most frequently used metrics today, especially by statisticians. One motivation for the use of RMSE is that it has several desirable mathematical properties, but also that it emphasizes eliminating large errors. Armstrong and Collopy (1992) do, however, point out several deficiencies with RMSE; it is scale dependent, just like MAE, and very sensitive to outliers.

$$RMSE_m = \sqrt{\frac{\sum_{i=1}^{n} (m_i - a_i)^2}{n}} \tag{17}$$

In some cases it could, of course, be appropriate to use RMSE, but this decision must be taken after discussing the specific problem with domain experts, i.e., RMSE should not be used routinely just because the data miner selected a modeling technique implicitly optimizing RMSE.

Pearson Correlation (*r*)

The Pearson correlation coefficient, often denoted as *r*, is yet another frequently used measure. The Pearson correlation, of course, measures the strength of the linear dependencies between the predicted and target value. Often the coefficient of determination, i.e., the square of *r,* is used instead; r^2 is the proportion of the variability in the data that is captured by the evaluated model. r^2 is bounded between *0* and *1,* where *1* is a perfect fit, 100% variability can be explained, and *0* signifies that none of the variability can be explained. Although both uses of the Pearson correlation coefficient convey important information of the model's predictive performance, it must be noted that they ignore the sizes of all errors (Armstrong 2001). The Pearson correlation should therefore rarely be used as the sole criterion.

$$r_m = \frac{\sum_{i=1}^{n}\left((m_i - \overline{m_i})(a_i - \overline{a_i})\right)}{\sqrt{\sum_{i=1}^{n}(m_i - \overline{m_i})^2 * \sum_{i=1}^{n}(a_i - \overline{a_i})^2}} \tag{18}$$

Mean Absolute Percentage Error (MAPE)

Another popular metric is the *mean absolute percentage error* (MAPE), whose strongest advantages are that it is easy to interpret and it is scale free. A problem with this measure is, however, that it is biased towards low predictions for problems which do not include negative target values. Problems with only positive target values are, of course, very common, e.g., when predicting demand or actual sales. In these cases, even a prediction of 0 would result in an error of no more than 100%, while a too high prediction could potentially result in an enormous error. Another problem is that this measure is undefined when the target value is zero. To handle these drawbacks, a cap for the maximum error (Max_e) of a single error is often used, see equation 19. Nevertheless, for these reasons, MAPE is not a good metric for problems where the target values may be close to zero.

$$MAPE_m = \frac{\sum_{i=1}^{n} min\left(\left|\frac{m_i - a_i}{a_i}\right|, Max_e\right)}{n} \tag{19}$$

Mean Unbiased Absolute Percentage Error (MUAPE)

Due to the deficiencies identified for MAPE, as described above, Makridakis (1993) instead argues for the use of the *Mean Unbiased Absolute Percentage Error* (MUAPE), since it is unbiased and less likely to have problems when the targets'

values are close to zero. According to Makridakis, MUAPE is still comprehensible to decision makers, but this has been heavily disputed.

Even if MUAPE is more robust for predictions around zero, it still needs to be capped, similar to MAPE.

$$MUAPE_m = \frac{\sum_{i=1}^{n} min\left(\left|\frac{m_i - a_i}{(m_i + a_i)/2}\right|, Max_e\right)}{n} \tag{20}$$

Geometric Mean of the Relative Absolute Error (GMRAE)

Armstrong and Collopy (1992) evaluated several error measures with regard to reliability, construct validity, sensitivity to small changes, protection against outliers, and their relationship to decision making. Geometric Mean of the Relative Absolute Error (GMRAE) was presented as one of the strongest measures for regression tasks. GMRAE is based on Relative Absolute Value (RAE), which is defined in equation 21 where $p_{s,m}$ is the prediction made by model m on series s. RAE is relative to the random walk (rw) with zero drift, i.e., the most recent known actual value. For series s, rw predicts a_{s-1}

$$RAE_i = \frac{p_{s,m} - a_s}{p_{s,rw} - a_s} \tag{21}$$

Random walk is often used as a reference point since it is straightforward and easily interpreted. In addition, random walk most often outperforms the mean prediction for time series. To handle extreme values, the RAE is trimmed using Winsorizing according to equation 22.

$$WRAE = \begin{Bmatrix} .01 & if\ RAE_s < .01 \\ RAE_s & if\ .01 \leq RAE_s \leq 10 \\ 10 & if\ RAE_s > 10 \end{Bmatrix} \tag{22}$$

GMRAE summarizes the WRAE using the geometric mean. The arithmetic mean is not suitable, as any arithmetic average will be dominated by its large terms. In other words, good small errors should be counted but would essentially be ignored. Li and Zhao (2001) argue that a large error should be balanced by a sufficiently small error, which is the case for the geometric average, but not for the arithmetic averages.

$$GMRAE = \sqrt[n]{\Pi_{s=1}^{n} RAE_{m,s}} \tag{23}$$

2.3.3 Observations regarding the use of performance measures

When comparing different estimation models over several datasets, it is important to use an appropriate performance measure. Armstrong (2001) argues that if mean models are going to be compared over several datasets the performance measure has to be relative. This is true for both estimation tasks (Li & Z. Zhao 2005) and for time series prediction. A relative measure is a measure which is relative to some reference. Usually some simple straightforward model like the mean value of the target attribute is used for reference. A relative measure is also important since the values of the target variable can differ greatly in magnitude between datasets, which, of course, will affect the magnitude of the error. The reference also accounts for the difficulty of the dataset, which is also of great concern in a comparison. It is not always worse to achieve a higher error on a difficult task than a slightly lower error on a simple task.

In practice, relative values have less importance since they are hard to interpret. A normal measure which gives its error in a quantity that is natural to the problem domain or as a percentage is much more useful for an organization. Relative measures are not as usual for classification tasks, since these measures most often return a percentage or a bounded value. Hence, there is no problem with different error magnitudes, even if the difficulty of a dataset can still cause problems if results are averaged over several datasets.

An even better approach for evaluating different models is to use an appropriate statistical test. There are numerous statistical tests which all have different properties, but the next section presents some important measures which can all account for difference in error magnitude and difficulty and are hence very suitable for PM.

2.4 Evaluation of Predictive Models

In PM, a model is trained to minimize the error on a training set. However, error measures on the training set are often a poor estimation of future performance (Witten & Frank 2005). Each prediction is given by some probability function created from a training set which, in practice, rarely contains all possible instances. Therefore, the probability function will not be 100% representative and thus the model cannot be 100% representative. For this reason, the training error will always be optimistic with regard to the actual error for the whole population. A better

approach is to use a *holdout sample* or *test set* to estimate the error on unseen data and compare classifiers. However, the test set will not be representative, but it will be unbiased and can therefore be used to estimate the true error.

2.4.1 Hold-out Samples

A hold-out sample is created prior to training, by randomly removing a subset of the training instances and placing them in the test set. Typically, two thirds are used for training and one third for testing. A downside of this technique is that there is less data available to train the model, which can result in an inferior model. On the other hand, if too little data is used for the test set, it will contain a large variance making the estimated error less reliable. Another problem is that the class distribution can differ between the datasets and thus affect both the model and the estimation negatively.

An extension of the holdout sample technique is called *random subsampling,* which simply creates holdout samples several times. The overall error is calculated as the average error for all models. A problem with this technique is that there is no control of how many times an instance is used for testing and training.

Cross-validation is a more systematic approach to random subsampling. Instead of choosing the holdout set randomly several times, each instance is randomly assigned to one of k equal sized subsets or *folds* before training. Next, k models are trained, each using one of the folds as test set and the other $k-1$ folds as training, so that each fold is used as a test set once. In this way, cross-validation utilizes all instances and each instance is used once for testing and $k-1$ times for training. However, the folds can still be unrepresentative, as they are chosen randomly. One time-consuming approach to this problem is a special case of cross-validation called *leave one out,* where each fold contains a single instance. *Leave one out* uses the maximum amount of data for training the models and each test is mutually exclusive, but the method is rather impractical for most problems. A better approach is to ensure that all folds have a representative class distribution when they are created. Stratification is a simple technique that does this by selecting instances (in a random manner) in a way that ensures that each fold will receive an equal number of instances of each class. For cases where the number of folds and instances of a certain class does not add up, a copy of an already selected instance can be used, or the fold can be left as it is. Most often, 10-fold cross-validation with stratification is used to evaluate classifiers.

2.4.2 Comprehensibility of Predictive Models

A transparent model is one that can be read by a human and understood in its parts. A comprehensible model, on the other hand, needs to be understandable as

a whole, which is another matter. Figure 9 shows a Boolean rule written in prefix notation that is transparent but not comprehensible.

```
( Or( And( Or( Or( Less( Triceps )( 827,609 ) )( Less( Serum )(
342,516 ) ) )( Or( And( And( Greater( Triceps )( 186,974 ) )(
Greater( Plasma )( 133,614 ) ) )( And( Less( Plasma )( 302,766
) )( Less( TPregnant )( 267,67 ) ) ) )( Or( And( Less( Pressure
)( 155,735 ) )( Less( BodyMass )( 547,947 ) ) )( Or( Or( Greater(
Pressure )( 505,055 ) )( Less( Triceps )( 288,807 ) ) )( Greater(
TPregnant )( 574,294 ) ) ) ) ) )( And( Or( Or( Or( Greater( Age
)( 480,969 ) )( Greater( Plasma )( 121,843 ) ) )( Greater(
Triceps )( 692,08 ) ) )( And( Or( Less( Serum )( 219,791 ) )(
Less( Triceps )( 373,068 ) ) )( And( Greater( BodyMass )( 229,002
) )( Greater( Plasma )( 289,4 ) ) ) ) )( Or( And( And( Less(
Triceps )( 131,931 ) )( Greater( BodyMass )( 78,237 ) ) )( Or(
Or( Less( Age )( 304,78 ) )( Or( Less( Serum )( 141,117 ) )(
Less( Serum )( 1,927 ) ) ) )( Less( Triceps )( 613,874 ) ) ) )(
Or( Or( Greater( Plasma )( 82,681 ) )( Less( PedigreeF )( 459,841
) ) )( Or( Greater( PedigreeF )( 753,176 ) )( Less( TPregnant
)( 541,046 ) ) ) ) ) )( Or( Or( Or( Or( Greater( Age )( 204,502
) )( Greater( PedigreeF )( 342,593 ) ) )( Or( Greater( Age )(
539,65 ) )( Greater( Pressure )( 810,805 ) ) ) )( And( And( And(
Greater( BodyMass )( 722,018 ) )( Greater( Age )( 771,832 ) )
)( And( Less( PedigreeF )( 676,094 ) )( Greater( BodyMass )(
548,964 ) ) ) )( And( And( Greater( Plasma )( 496,494 ) )(
Greater( TPregnant )( 626,094 ) ) )( Or( Less( BodyMass )(
202,682 ) )( Greater( PedigreeF )( 541,959 ) ) ) ) ) )( Or( And(
And( And( Less( Plasma )( 586,935 ) )( Greater( Age )( 467,127
) ) )( Or( Greater( PedigreeF )( 129,058 ) )( Greater( PedigreeF
)( 39,673 ) ) ) )( Or( And( Less( Serum )( 536,914 ) )( Less(
TPregnant )( 94,637 ) ) )( Or( Less( PedigreeF )( 606,103 ) )(
Less( BodyMass )( 626,094 ) ) ) ) )( And( And( And( Less( Serum
)( 598,499 ) )( Greater( BodyMass )( 510,324 ) ) )( And( Greater(
Plasma )( 486,2 ) )( Less( TPregnant )( 722,018 ) ) ) )( And(
And( Less( BodyMass )( 626,158 ) )( Greater( Serum )( 804,593 )
) )( Or( Less( Plasma )( 266,305 ) )( Less( PedigreeF )( 845,997
) ) ) ) ) ) ) )
```

Figure 9 - Transparent but not comprehensible model

It does not help that each condition can be understood, when there are too many conditions. There is no hard line deciding whether a rule is comprehensible or not, but when comparing two rules with the same representation, the simpler (smaller) one is most often considered more comprehensible. This is also in accordance with the old theorem called *Occam's razor* (other things being equal smaller theories are preferable to larger ones) and the minimum description principle (MDL) (Rissanen 1978).

Two rules created with the same representation can be compared in a straightforward way by simply counting the number of elements each rule contains. If both rules have the same accuracy, the one with fewer elements should be chosen.

2.4.3 Comparing Models

Sadly, there are no free lunches in PM, i.e., no technique can outperform all others over the space of all possible learning tasks (Schaffer 1994). Hence, when performing data mining, it is normal to create several models using different techniques and settings. However, this means that the models need to be compared in a structured manner, to ensure that the best model is actually selected. There has been much research on the topic of the best way of comparing classifiers and numerous tests are available. Demšar (2006) has examined the current practice, both theoretically and empirically, and recommends two non-parametric tests: the *Wilcoxon* signed rank test is recommended for the comparison of two classifiers and the *Friedman test* with the corresponding post-hoc tests for the comparison of several classifiers over multiple datasets.

Wilcoxon Signed Rank Test

A Wilcoxon signed rank test (Wilcoxon 1945) is used for the comparison of two models over n datasets and based on the ranked difference between the models. First, the absolute value of the difference in performance d_i on each dataset is ranked from the lowest to the highest. In case of equal performance, the ranks of these datasets are split evenly, if there is an odd number, one is ignored. Next, the rankings are summed for the cases which give a positive difference R^+ and a negative difference R^- according to equation 24 below.

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i)$$
$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \quad (24)$$

If the smallest of the sums $T = \min(R^+, R^-)$ is less or equal to the corresponding critical value *(CD)* in Table 1, the hypothesis that the models are equal in performance can be rejected.

n	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
CD	0	2	4	6	8	11	14	17	21	25	30	35	40	46	52	59	66	73	81	89

Table 1 - Critical values of the Wilcoxon test at .05 significance level

For a comparison containing more than 25 datasets, equation 25 can be used instead of Table 1. Here, the hypothesis can be rejected at a significance level of .05 if z is less than -1.96.

$$z = \frac{T - \frac{n(n+1)}{4}}{\sqrt{\frac{n(n+1)(2n+1)}{24}}} \tag{25}$$

Friedman Test

A *Friedman* test (Friedman 1937) is a nonparametric test for comparing m models over n datasets using their rank. Ranking is done by giving the best a rank of 1, the second best 2, and so on. In case of a tie, the average rank of the tied models is assigned. When a model is ranked after a tie, it is assigned a rank according to the number of models that has been ranked so far plus 1. The null-hypothesis for the *Friedman* test is that all models are equal in their performance and hence also their ranks R_j. The Friedman statistic is distributed according to χ_F^2 with *m-1* degrees of freedom when $n>10$ and $m>5$. In cases with fewer models and fewer datasets, exact critical values can be found in (Sheskin 2004).

$$\chi_F^2 = \frac{12n}{m(m+1)}\left[\sum_j R_j^2 - \frac{m(m+1)^2}{4}\right] \tag{26}$$

Iman and Davenport (1979), however, found that the Friedman test is unnecessarily restrictive and suggested a modification according to equation 27.

$$F_F = \frac{(n-1)\chi_F^2}{n(m-1) - \chi_F^2} \tag{27}$$

Some sample critical values of the F-distribution are presented in Table 2. The significance level is .05 and the numerator *(m-1)* is given by the column and the denominator *(m-1) (n-1)* by the row.

DF	1	2	3	4	5	7	10	15	20
1	161.45	199.50	215.71	224.58	23.16	236.77	241.88	245.95	248.01
2	18.513	19.000	19.164	19.247	19.296	19.353	19.396	19.429	19.446
3	1.128	9.5522	9.2766	9.1172	9.0135	8.8867	8.7855	8.7028	8.6602
4	7.7086	6.9443	6.5915	6.3882	6.2560	6.0942	5.9644	5.8579	5.8026
5	6.6078	5.7862	5.4095	5.1922	5.0504	4.8759	4.7351	4.6187	4.5582
7	5.5914	4.7375	4.3469	4.1202	3.9715	3.7871	3.6366	3.5108	3.4445
10	4.9645	4.1028	3.7082	3.4780	3.3259	3.1354	2.9782	2.8450	2.7741
15	4.5431	3.6823	3.2874	3.0556	2.9013	2.7066	2.5437	2.4035	2.3275
20	4.3512	3.4928	3.0983	2.8660	2.7109	2.5140	2.3479	2.2032	2.1241

Table 2 - F-distribution Critical values for P=.05

If only two models are compared, the *ANOVA* test has more power, if the requirements of an *ANOVA* test are fulfilled. On the other hand, Demšar (2006) points out that the ANOVA requirements are not always met and that the ANOVA and the Friedman tests, in practice, most often give the same results.

Nemenyi Test

If the null-hypothesis of the *Friedman* test is rejected, Demšar (2006) recommends the use of the Nemenyi post-hoc test (Nemenyi 1963) to analyze which of the models differ significantly from each other. When all m models are compared to each other over n datasets, a difference in average rank greater than the critical difference CD is significant.

$$CD = q_\alpha \sqrt{\frac{m(m+1)}{6n}} \tag{28}$$

Table 3 gives the value of q_α depending on the significance level α and the number of models.

m	2	3	4	5	6	7	8	9	10
$q_{.05}$	1.960	2.343	2.569	2.728	2.850	2.949	3.031	3.102	3.164
$q_{.10}$	1.645	2.052	2.291	2.459	2.589	2.693	2.780	2.855	2.920

Table 3 - q_α for the Nemenyi test

If only a single model needs to be compared to a set of models, Demšar suggests a more powerful post-hoc test, such as the *Bonferroni-Dunn* test (Dunn 1961). The only difference from the *Nemenyi* test is that the values of q_α in equation 28 should be replaced with values from Table 4. As seen in the table, the difference between the two tests becomes greater with the number of classifiers.

m	2	3	4	5	6	7	8	9	10
$q_{.05}$	1.960	2.241	2.392	2.498	2.576	2.638	2.690	2.724	2.773
$q_{.10}$	1.645	1.960	2.128	2.241	2.326	2.394	2.450	2.498	2.539

Table 4 - q_α values for the Bonferroni-Dunn test

2.5 Genetic Programming for PM

Genetic Programming (GP), introduced by Koza (1992), is a general optimization technique based on Darwin's (1859) theories of natural selection. In essence, Darwin claims that in a population of competing *individuals* (animals), the *fitter, i.e.,* better adapted to the environment, will have a greater chance of survival and reproduce at a higher rate. Since fitter individuals tend to produce stronger offspring, each new generation will become fitter. In nature, reproduction is normally a sexual *recombination* of two individuals (*parents)* which produce one or more offspring, i.e., *children*. Recombination is the driving factor in evolution, and *mutation,* i.e., a random change in the genes of an individual, has a much smaller part in evolution. Based on Darwin's findings, Koza (1992) points out that there are four conditions that must be satisfied for an evolutionary process to occur in nature.

- An individual has the ability to reproduce.
- There is a population of such individuals.
- There is some diversity among the individuals.
- Some difference in ability to survive in the environment is associated with the diversity.

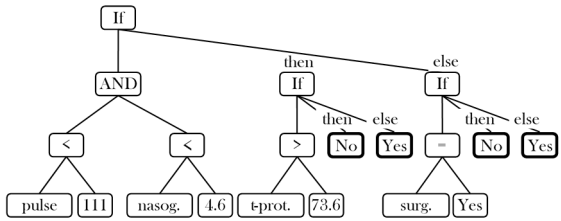

Figure 10 – A program represented using a tree structure

In genetic programming, an individual is a computer *program* represented using a tree structure containing predefined *primitives*, i.e., *functions* and *terminals*. The structure of a program is defined by the arity of its functions, since each argument of a function becomes a tree branch leading to a terminal or a function. A wide variety of functions can be used including:

- *Arithmetic* operators such as (+,-,*,/,...).
- *Mathematical* function like (sin, cos, exp, sqrt...).
- *Boolean* operators such as (and, or, not, <, >, =,...).
- *Conditional* functions (If- then-else).
- Other domain-specific functions.

Terminals are the leaves of the tree and may consist of constants, independent variables, or functions without arguments. It is important that the function and terminals are chosen with the problem in mind, since GP can only find solutions consisting of these primitives. Selecting the wrong functions and terminals may result in a search space that does not include the optimal solution, while superfluous primitives will result in an unnecessarily large search space. For PM problems, the terminal set consists of independent variables and randomly initialized constants. Selecting terminals for a GP process is, in this case, similar to selecting independent variables for a predictive technique.

The GP environment consists of program constraints and a set of instances that describes the problem at hand. To ensure diversity and an approximate average performance of the population as a whole, the initial programs are created randomly. Functions are selected at random and constants are initialized with random or predefined values which then remain fixed.

Next, a fitness value is calculated for each program using a *fitness function* which is defined to match the problem. A lower value usually corresponds to a fitter individual, e.g., the number of misclassified instances could be used as a fitness function for classification problems.

After a fitness value has been calculated for each individual, it is time for the natural selection to play its part. There are several strategies for how to implement natural selection in GP, but all somehow select programs in a probabilistic manner based on fitness. *Tournament* selection, which is the most common selection technique according to Poli, Langdon, & McPhee (2008), selects the winner of *k* randomly selected individuals.

A new *generation*, i.e., population, is normally created in a so-called *steady state* fashion where genetic operations are applied to selected individuals until the new generation has the same size as the current population. There are three main genetic operations; sexual recombination here called genetic *crossover, reproduction,* and mutation. The standard crossover operation creates two new children from two parent programs by switching two randomly selected subtrees.

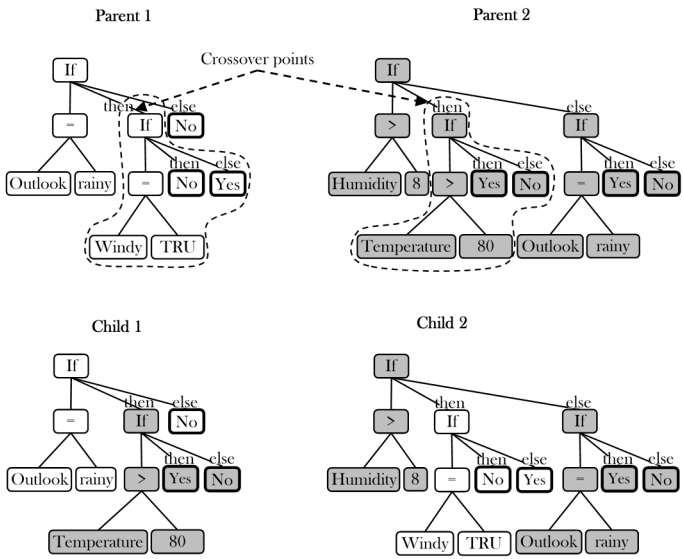

Figure 11 - Crossover

As in nature, crossover (sexual recombination) is the driving force in GP. Typically, 90% of the individuals in the new generation are created through crossover. Reproduction, the second most influential operation, applied to approximately 9% of the programs, simply copies an individual unchanged into the new generation.

Mutation is a secondary operation that is normally only performed on approximately 1% of the programs. In fact, Koza (1992) did not use mutation at all.

Today, however, it is common practice to include a small amount of mutation. In the standard mutation operation, a randomly selected subtree is replaced with a new randomly created subtree, e.g., see Figure 12.

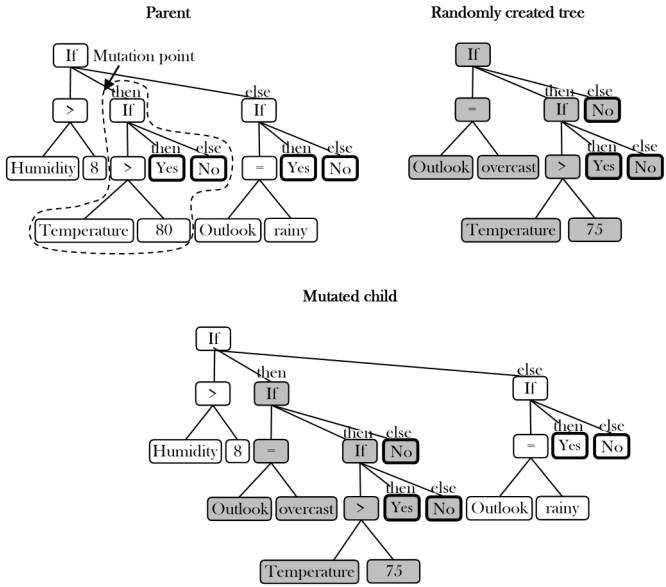

Figure 12 - Mutation

New generations continuously evolve in this manner until a certain number of generations have been reached or until a satisfactory solution has been found. The basic GP steps can be summarized by Figure 13, where P_c, P_r and P_m are the probabilities for crossover, reproduction, and mutation. In the following section, each part of the GP process is discussed in detail.

2.5.1 GP Representations

When defining the terminal set and function set, Koza (1992) points out that the primitives must be selected so that they obtain *sufficiency* and *closure*. The sets are sufficient if the sought after solutions can be expressed by the selected functions and terminals. For PM, it is for example crucial that all relevant independent variables are available. A function set obtains closure, if each function can accept values from all primitives in the function set and the terminals' set.

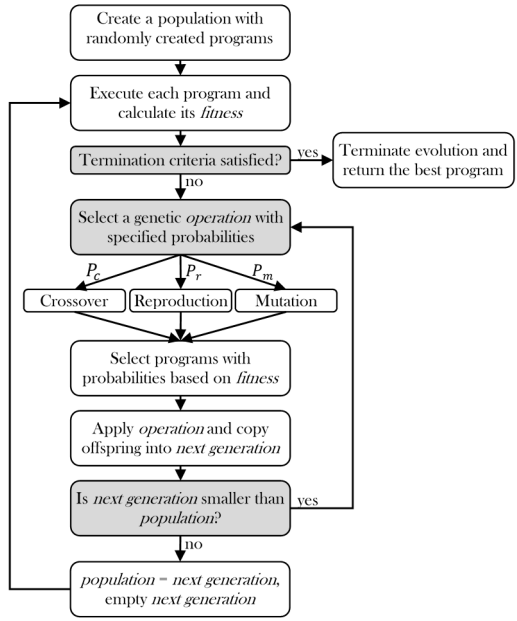

Figure 13 – Overview of the GP process

Closure

Closure is important, since crossover and mutation rearrange functions and terminals randomly which, without closure, could lead to illegal programs. Poli et al. (2008) extend Koza's definition by dividing closure into *evaluation safety* and *type consistency.* A function is evaluation safe if it returns legal values for all possible values. Many arithmetic operators and mathematical functions are not evaluation safe by default. Division is not defined when the denominator is zero and functions like *sqrt* and *log* cannot handle negative values. Normally, these types of issues are handled by creating protected versions of unsafe operators and functions. Division can be defined to return *1* if the denominator is zero and functions that cannot handle negative values can replace negative values with the absolute value.

Type consistency could be summarized so that all primitives should use and return values of the same type. A typical problem is when Boolean operators are needed for conditional functions, but the standard Boolean *true*/*false* values are not accepted by other functions. One simple straightforward solution is to use numerical logic instead. In numerical logic, the Boolean comparative operators,

such as <,>, =return *0* or *1* instead of *true* or *false*. *OR* is normally defined to return the greater of its two arguments, while *AND* returns the smaller value. The conditional function must, of course, also be modified to work with numerical values. A conditional function *If(a,b,c)* could, for example, be defined to execute *b* if *a* is greater than one and otherwise *c*.

Another approach is to create conditional functions that include the Boolean function and instead takes four arguments instead of three, e.g., *ifGreater*(*a,b,c,d*) could compare *a* and *b* and execute *c* or *d* depending on whether *a* is greater than *d*. In this way, no Boolean values are introduced but the program is limited to the predefined Boolean expressions that are included in the conditional function. Numerical logic does not have this disadvantage but can instead produce programs that are harder to comprehend. The program in Figure 14 is, for example, a perfectly legal and reasonable program, if numerical logic is considered. The program can be interpreted as that it is a bad day for golf if the temperature is greater than *79* or if the humidity is greater than *80* (*79/79=1* and *80*.125=1*).

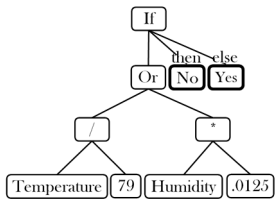

Figure 14 – Legal program considering numerical logic

These program types are not a problem, if some kind of mathematical function or engine control system is evolved and the programs do not need to be easily interpreted. In PM, however, comprehensibility is a crucial factor and, thus, numerical logic is not the best solution. Redefining the conditional function is not a good option either, since it is impossible to know which and how complex a Boolean expression is needed. Instead, either *constrained syntactic structures* or *strongly typed* GP are often used.

Constrained Syntactic Structures

Even if unconstrained GP can solve many problems as long as the function set fulfills the closure property, some problems may require a constrained syntax, (Koza 1992). Constrained syntactic structures (CSS) define a program syntax that exactly defines how different primitives can be combined. Figure 15 below shows syntax for an arbitrary complex Boolean expression that can be used for the weather dataset from the UCI Repository.

program := if *bOp* then *program* | yes | no else *program* | yes | no
bOp := *conOp* | *catOp* | *bOp* AND *bOp* | *bOp* OR *bOp*
conOp := *conV* < *C* | *conV* >= *C*
catOp := Outlook = [sunny | overcast | rain] | Windy = [TRUE | FALSE]
conV := Temperature | Humidity
C := A randomly initialized constant

Figure 15 – CSS GP Syntax for complex Boolean conditions

Constrained syntax overcomes both disadvantages of the previously presented techniques. However, constrained syntactic structures require that the initial population is created according to the constraints and that crossover and mutation only produce syntactically legal offspring. Obviously, this is more computationally expensive, but the constrained syntax effectively limits the search space and thus simplifies the GP search. Hence, the total amount of computational power needed is usually less. Another advantage over unconstrained GP is that the structure of the program can be adapted to the problem domain and thus makes it easier to comprehend.

Strongly Typed GP

A problem with the CSS approach is that the syntax needs to be defined for each primitive in the function set and terminal set. This can be cumbersome, since two problems rarely use the exact same sets of primitives, e.g., *catOp* in Figure 15 is specifically defined for the weather dataset and would not apply to many other datasets.

In strongly typed GP suggested by Montana (1995), each primitive has a predefined type. Function arguments and return values are also specified by type rather than as allowed primitives. A ST GP syntax is hence more general, than a CSS syntax and can be applied to a larger variety of problems. However, creating the initial population, crossover, and mutation still need to be done in a controlled fashion to ensure that each function argument will have the right type. Figure 16 shows a ST GP syntax that can be used for classification.

Since a ST GP syntax is more general, it is naturally less specific. It would, for example, not be possible to specify that a comparison between variables is forbidden, i.e., if both constants and variables are continuous, there is no way to separate them. Hence, since ST GP syntax is more general, it is also harder to tailor to a specific problem domain than the CSS approach.

definitions

Continuous := *conV*, C_{conV}
Categorical := *catV*, C_{depV}, C_{catV}, *if*
Boolean := *bOp*, *condition*

Syntax

program := if *bOp* then *program* | C_{depV} else *program* | C_{depV}
bOp := *Boolean* AND *Boolean*| *Boolean* OR *Boolean*
condition := *Continuous* < *Continuous* | *Categorical* = *Categorical*
C_{depV} := Constant initialized to a random value of the dependent variable
C_{conV} := Constant initialized to a random value of *conV*
C_{catV} := Constant initialized to a random value of *catV*
conV := Independent continuous variable
catV := Independent categorical variable

Figure 16 – ST GP Syntax for complex Boolean conditions

Representations for classification tasks

Espejo, Ventura and Herrera (2010) group representation used for GP classification into three major categories, i.e., *decision trees, classification rules,* and *discriminant functions*, as described.

The simplest form of decision trees, which is identical to the representation used by most decision tree techniques, only performs splits based on a single attribute often called *axis-parallel,* since the splits can be seen as an axis-parallel hyperplane in feature space.

Trees that are allowed to use splits based on linear combinations of attribute are called *oblique*, e.g., see (Bot and Langdon 2000). Finally, *nonlinear multivariate trees,* like the fuzzy trees evolved by Marmelstein and Lamont (1998), are trees that can use nonlinear functions to combine attribute in a split.

There are two main approaches for how GP programs can be used to create classification rules; either each program represents a complete rule set, similar to a decision tree, or each program is seen as a single rule and the final solution is constructed by combining the best rules in some way. In the field of evolutionary algorithms, the first method is called the Michigan approach and the second the Pittsburg approach after the universities where they were first suggested. Normally, when using the Pittsburg approach, a separate run is performed for each class resulting in a rule for each class, e.g., see (Falco, I. De, Della Cioppa, A. & Tarantino 2002). The Pittsburg approach is more suitable for binary problems, since multiclass problems require several rules to be evolved and combined using some function. Often this results in less comprehensible solutions, compared to decision trees, since combining or selecting a specific rule requires that all rules are

evaluated, which is not the case for decision trees. When performing binary classification, however, only rules classifying one of the classes is needed, since all instances not covered can automatically be classified as the other class.

A third way to represent classification rules is to use discriminant functions in the form of mathematical expressions based on mathematical operators and the attribute of the dataset. The output of the discriminant function is numerical values which must be compared to a threshold to perform a classification. For binary problems, a single threshold is sufficient, while multiclass problems must be approached using *n-1* thresholds, or by evolving a function for each class according to the Pittsburg approach.

Representations for regression tasks

GP is very fit to handle symbolic regression, since it simultaneously evolves both the structure and the parameters of a program. There are three main types of symbolic representations that are normally used in GP: polynomial, regression trees, and model trees. All three normally use what Koza calls ephemeral random constants (ERC), i.e., constants that are initialized randomly and then remain constant.

A very common representation for regression problems is to define a program as a polynomial expression. The function set then becomes the allowed mathematical functions and a terminal set the independent variables and ERCs.

Regression trees are very similar to classification trees, with the exception that the predictions are done using ERC instead of a certain class. Regression trees are popular, since they are often easier to interpret than polynomial expressions, but as is typical for the accuracy versus comprehensibility tradeoff, they often have a lower predictive performance.

Model trees are, as discussed in section 2.2.2, decision trees with models in the leaves. Depending on the type of model used in the leaves, model trees can be used for either classification or symbolic regression. The model can be evolved together with the tree or created using another technique. Quinlan's (1992) M5 technique (described in section 2.2.2) does, for example, create a decision tree which uses linear regression in the leaves.

2.5.2 Creation of the initial population

If no prior knowledge about the characteristics of the sought after solution exists, the initial population is created randomly. A randomly created population ensures that the populations as a whole will have an approximately average performance. The randomness also naturally introduces diversity, which is crucial for a successful evolution. In a diverse population, the individuals will be evenly distributed among

the possible solutions, which will result in the GP-process starting its search on a broad front, increasing the chance of finding an optimal solution. Koza (1992) suggested an initialization technique called *ramped half-and-half,* which is still very commonly used today. Ramped half-and-half is actually a combination of two very simple initialization methods called *grow* and *full*. Both techniques use a parameter *maximum depth* to control the size of the created programs. Program *size* is commonly defined as the number of nodes (primitives). Depth is calculated as the number of nodes between the root and a leaf, where the root node is considered to be at the depth of zero. *Tree depth* is the maximum depth between the root and any leaf, e.g., the tree depth of the tree in Figure 14 is four.

Grow Initialization

Grow is an uncomplicated initialization technique designed to create trees that vary in structure and size. Trees are created by simply randomly selecting a primitive from either the function set or the terminal set. If the selected primitive is a function, a new random primitive is selected for each argument of the function. The same procedure is repeated until there are no more functions that need argument, or until the maximum depth is reached. If the maximum depth is reached, only terminals which effectively stop the growth are selected. Figure 17 presents a pseudo code for recursively generating a random program using Grow. The maximum depth is given as an argument to the method, which then recursively adds primitives for each function argument until the maximum depth is reached, or no more function argument needs a terminal.

```
method: Grow(depth): primitive
if depth = 0
    p = random terminal
if depth > 0
    p = random function or terminal
    if p is a function
        for each argument a of p
        a = Grow(depth -1)
return p
```

Figure 17 – Pseudo code for Grow initialization

Since a node is selected randomly from both the functions' set and the terminals' set, the ratio between the number of functions and terminals will determine the average size of the created trees. If there are many more terminals than functions, terminals will be chosen more often, thereby terminating the growth of the tree. If, on the other hand, there are considerably more functions, the branches will tend to continue to grow until the maximum depth is reached.

Full Initialization

The name Full is due to the fact that it only creates full (balanced) trees where all the terminals are at the same depth. This is achieved by randomly selecting a function and continuing to select functions for all arguments until the maximum depth is reached. Thereafter, only terminals are selected. The pseudo code for recursively initializing a program using Full is given in Figure 18. In contrast to Grow, Full is not affected by different ratios of functions and terminals, due to its less random strategy. However, all trees will, of course, have the same maximum depth and, hence, are quite similar in size and structure.

```
method: Full(depth): primitive
if depth = 0
    p = random terminal
if depth > 0
p = random function
    for each argument a of f
        a = Full(depth -1)
return p
```

Figure 18 – Pseudo code for Full initialization

Ramped half-and-half Initialization

Even if both Full and Grow are simple to implement and, to some extent, create diverse populations, they are seldom used, since the size and shape of the solution are not known in advance. Full only creates trees of the maximum depth and Grow creates trees whose size is dependent on the ratio between the number of functions and terminals which do not need to have anything to do with the size of the solution. Instead, Koza suggested a more general initialization technique which combines Grow and Full.

In Ramped half-and-half, which Koza called the method, the maximum depth parameter is adjusted during creation to generate a more varied population. The population is created in n parts, where n is calculated as the maximum depth – 1. Each part contains *population size* / n programs created with an increasing value of depth parameter. Each part is created using i+1 as maximum depth, e.g., the first part will be created using 2 as maximum depth, the second using 3 and the last one with n+1 (which is the maximum depth decided by the user). Half of the trees in each part are created with Grow and half with Full. The pseudo code for initializing a population using ramped half-and-half is presented in Figure 19.

```
method: RampedHalf&Half(depth)
n = MAX_DEPTH -1, depth = 2;
DO n times
    DO POPULATION_SIZE / (n *2) times
        tree1 = Grow(depth)
        tree2 = Full(depth)
        add tree1 and tree2 to population
    depth  = depth +1
```

Figure 19 – Ramped half-and-half

Seeding

If there is prior knowledge of the type of structure of the solution, it is not necessary for the initial population to be created totally at random. A more informed initial population could be a better starting point for the evolution. However, Poli, Langdon, and McPhee (2008) point out two common problems when using a single seed. First, the seed may reduce the population diversity, since, in general, it is much fitter than randomly created individuals and thus tends to take over the population in a few generations. Secondly, a single seed may be lost, due to the probabilistic nature of some selection techniques. Both problems may be solved by simply introducing more copies or mutated versions of the seed. Langdon and Nordin (2004) did, for example, use copies of an unpruned C4.5 tree as the initial population and Hsu and Gustafson (2001) used several seeds, i.e., the final population of a previous GP-run, as the starting point of a GP process with a different fitness function.

2.5.3 Calculating fitness

In nature, fitness can be regarded as a measure of how well adapted an individual is to its environment. Fitter individuals tend to have a greater chance of survival and reproduction. In GP, fitness is calculated using a *fitness function* which describes how well a program performs on the training instances. For PM tasks, a fitness function is the same as the optimized score function and should hence be the same as the score function. However, for practical reasons, the fitness values are *standardized,* i.e., transformed so that lower fitness values are always better and the optimal solution will have a fitness value of zero. The original fitness values before standardization are called *raw* fitness and can have their optimum at a maximum or minimum value. If the fitness values are standardized, the GP task will always be to minimize the output of the fitness function and a general reference point for an optimal solution. Two fitness functions for PM tasks are presented in the equations

below, where p_i is the predicted value of program p for instance i, while a_i is the actual value of the dependent variable. Equation 29 presents a fitness function that can be used for classification tasks. No standardization of the fitness value is required, since it sums the misclassifications made by the program.

$$f_p = \sum_{i=1}^{n} (p_i! = a_i) \tag{29}$$

The second fitness function, (equation 30) below, is a defined regression task and sums the instances of error for all training instances. Both functions return standardized fitness values, where the best value is zero, which signifies a perfect prediction.

$$f_p = \sum_{i=1}^{n} |p_i - a_i| \tag{30}$$

However, normally, these simple types of fitness functions work poorly, since the size of the evolved programs tends to grow exponentially after a certain point in the evolution, a phenomena called *bloat* (Banzhaf, Nordin, Keller & Francone 1998).

Bloat and introns

Currently, there is no consensus regarding why bloat occurs, but the following section presents some of the more accepted theories. According to Angeline (1994), bloating helps the evolution by introducing redundant copies of strong subtrees and thus increasing the probability that the crossover operation will switch the complete subtree. Angeline calls these redundant parts and other unused parts of a program *intron,* i.e., an intron is an emergent part that does not directly affect the fitness of an individual. An *active* part of a program is a part that directly contributes to the fitness of a program.

Langdon and Poli (1997) argue that the correlation between complexity and fitness disappears after all programs have reached a high near-optimal fitness. Hence, since there are many more possible long solutions than short, the selection pressure will, over time, naturally select longer and longer programs.

Banzhaf et al. (1998) argue that the introns also provide strong global protection for a program. A strong program risks being disrupted by the genetic operation, if every part of the program is active for the predictions of the program. If, instead, the program contains large unnecessary parts, the risk that an active part will be

disrupted decreases. The exponential growth of introns typically occurs towards the end of the evolution when it becomes hard to find fitter programs. Since the evolution cannot find better individuals, it instead attaches introns to protect the offspring from genetic operators, because they would, very likely, be harmful at this point of the evolution.

Figure 20 shows a small program for the weather dataset with ten primitives. Every part of the three is active, if the whole dataset is considered as a training set. Any change made by a genetic operation could disrupt the program and result in an inferior tree.

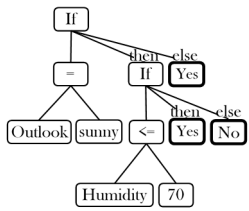

Figure 20 – Tree without introns

However, if the same program contained introns, the risk of disruption would decrease. Another version of the program above with introns can be seen in Figure 21. The program still makes exactly the same predictions but now has a total of 36 primitives, where the 21 marked in gray are unused introns. Since the first condition *(Outlook=sunny)* is repeated in the if-statement connected to the *else* part of the first if-statement, no instances with a sunny outlook will ever reach this part of the program. Hence, the primitives marked in gray will never be used and are thus insensitive to disruption. In fact, the risk that the program will be disrupted is now only 42% (*(36-21)/36*).

It should be noted that the if-statement (enclosed by a dotted line) connected to the first *else* and its conditions are also introns, since the same prediction can be made without them, i.e., with the original tree in Figure 20. However, these introns do not help protect the program, since the predictions may change if any of them were replaced.

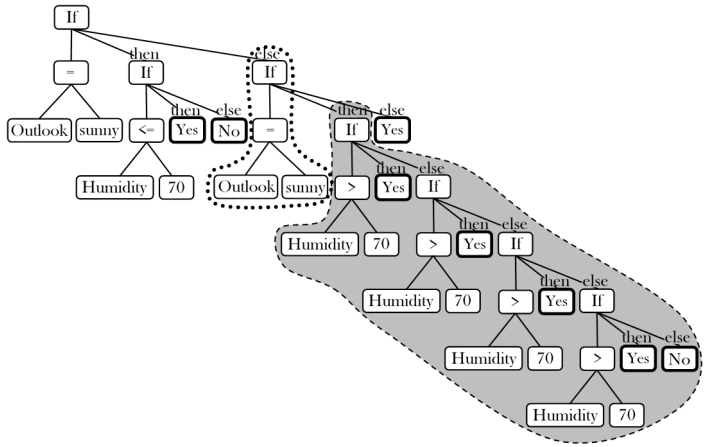

Figure 21 – Tree with introns

Banzhaf et al. (1998) stress the importance of distinguishing between the global protection, which gives rise to the exponential growth of introns, and the structural protection of strong subtrees noted by Angeline (1994). Global protection has a negative effect on GP runs, since the number of introns may lead to stagnation, (when only introns are swapped during crossover) as well as excessive use of CPU and memory. Structural introns, on the other hand, may help the evolution of fit programs by providing a local protection of strong subtrees, which thus may emerge despite the harmful effect of crossover and mutation.

Parsimony pressure

The most common and simplest way to counteract the exponential growth of introns is the parsimony pressure method (Koza 1992) which adds a small punishment to the fitness value related to the *complexity / size* of a program. Program complexity is normally defined as the number of primitives in the program. Complexity can, obviously, be calculated in many ways and it is, of course, also related to the type of operators used in a program. During the evolution, however, a program is always compared to another program with the same representation and, thus, the size of a program, calculated as the number of primitives, is normally used as an approximation of program complexity.

There is no consensus regarding how the parsimony pressure should be defined and combined with the performance measure, even if the approach described equation 31 is, according to Powidespreadee (2008), the most widely spread and accepted.

$$f_p = error_p + P * O_p \quad (31)$$

Here, $error_p$ is the error of program p according some measure, O_p is the complexity / size of the same program and P is the parsimony pressure, i.e., a weight used to scale the parsimony pressure. Any performance measure can be used in this approach, as long as it is designed to be minimized. The fitness function and performance measure could, of course, be designed for maximization, but as discussed in section 2.5.3, it is in general more practical to minimize a fitness function.

Since introns, by definition, do not contribute to the fitness of a program, the parsimony pressure will make them a burden and programs with many introns will hence eventually be removed during the selection process. In this way, the tradeoff between accuracy and complexity of the evolved programs is also somehow handled, since the parsimony pressure also affects the active parts of a program. The related fitness of each active part will be weighed against the extra complexity and thus more parsimony programs will be rewarded

It is important to select P with care, since a value that is too small will not stop the occurrence of bloating and a too large value makes the evolution favor small instead of accurate trees and, hence, will evolve very small useless programs. Normally, the parsimony coefficient is selected through initial experiments and remains constant during the whole evolution.

Recently, Poli and McPhee (2008) presented a theoretically sound method describing how to dynamically change P during evolution, in order to stop bloating and to gain better control of the mean complexity of the population. In the method proposed by Poli and McPhee, P is recalculated at each generation using equation 32, where $\mathrm{Cov}(O_g, f_g)$ is the covariance of the complexity and the fitness of the programs in the population at generation g and $\mathrm{Var}(O_g)$ is the variance of the complexity.

$$P(g) = \frac{Cov(O_g, f_g)}{Var(O_g)} \quad (32)$$

The advantage of this method is that P will be adapted in a way that keeps the average program size approximately constant, (a small drift may occur), during the evolution. Hence, a user can choose the average program size by controlling the creation of the initial population.

Finally, it should be noted that bloating can also be controlled by changes in the standard genetic operators. Langdon (2000), for example, suggests the use of a *size*

fair crossover and mutation, where the offspring are restricted to sizes close to those of their parents. According to Langdon, this effectively prevents bloating without any negative effect on the GP performance. However, most GP applications use parsimony pressure with a constant pressure.

Fitness functions for PM

A drawback with the fitness function presented in equation 31 is that *P,* the parsimony pressure, is dependent on the size of the training dataset. Hence, each experiment needs initial trial runs to determine a suitable setting for *P.* An obvious way to handle this problem is to use a percentage-based error metric like ACC, i.e., see equation 33.

$$f_{ACC_p} = 1 - ACC + P * O_p \tag{33}$$

Even if the approach described in the section above is widespread and can be used to optimize arbitrary performance metrics by replacing the accuracy calculation, there are also alternative implementations. Iba, De Garis, and Sato (1994) did, for example, use a fitness function based on the minimum description length, while Bojarczuk, Lopes and Freitas (2001), for example, used a combination of recall and specificity as performance measure and a parsimony pressure related to the maximum allowed complexity O_{max}, as presented in equation 34.

$$f_{SeSp_p} = Se_p * Sp_p * \frac{O_{max} - .5 * O_p - .5}{O_{max} - 1} \tag{34}$$

The parsimony pressure, used by Bojarczuk, Lopes and Freitas, gives no pressure for programs with a single element and a maximum pressure of .5 for trees that reach the maximum complexity. Note that this fitness function is designed for maximization.

2.5.4 Selection Schemes

When a fitness value has been calculated for each individual, a selection scheme is applied to decide which individuals will have the opportunity to reproduce through one of the genetic operations. A new generation is most often created in a *generational* or *steady state* manner. In generational creation, suggested by Koza (1992), a new generation of the population is created by applying a genetic operation to selected GPs and putting the offspring into the new generation. When the new generation has the same amount of programs as the previous, fitness values

are calculated for the new programs and the selection starts over to create yet another generation.

Reynolds (1993) introduced the steady state GP, (based on work conducted by Syswerda (1991)), which differs from generational GP by only working with a constant population, i.e., no new generations are created. Instead, a single offspring is created, evaluated, and used to replace a program with low fitness in the same population. Since a single program is created, evaluated, and inserted into the population, there are no generations in steady state GP. Another difference is that the population is often constrained to hold only unique programs which maximize diversity and reduce redundant calculations.

Except for the obvious advantage of just needing to keep a single generation in mind, there is no clear evidence that steady state GP is superior to generational GP. In fact, De Jong and Sarma (1993) argue that any difference in performance is more likely due to the selection schemes used rather than whether separate generations are used or not. Hence, since the generational method is more straightforward and it is the original method suggested by Koza, only selection schemes for generational GP are presented in this section.

In general, a selection scheme chooses programs based on a probability that is in some way related to the fitness of the programs. It is important that the selection scheme is not greedy but simply selects the best programs, since this will result in a simple hill climbing search. Since the selection schemes are always based on probabilities, a program may be chosen several times. In fact, *reselection,* which this property is called, is crucial, since it allows fitter programs to be selected several times and thus have a greater impact on the next generation. This tendency of selecting the same program several times is called *selection pressure* and it is what differs between selection schemes.

Fitness proportional selection

Fitness proportional selection, which is also known as *roulette wheel* selection in GP and based on Holland's (1992) work, simply gives each individual a possibility to reproduce proportionate to its relative fitness. This scheme can be pictured as a roulette wheel where each individual is given a slot the size of which corresponds to its relative fitness. The structure of the wheel stays the same for each selection, allowing an individual to be reselected several times. If the raw fitness r is larger for fitter individuals, equation 35 can be used to calculate the probability (P) that a program p is selected.

$$P_p = r_p / \sum_{i=1}^{n} r_i \tag{35}$$

However, if a lower fitness is a sign of a stronger individual, equation 36 must be used instead. Here r_{max} is the maximum raw fitness (error) that a program can achieve.

$$P_p = r_{max} - r_p / \sum_{i=1}^{n} r_{max} - r_i \tag{36}$$

A drawback of proportional selection is that programs which are very strong compared to the rest of the evolution tend to overcrowd the population and thus may result in a premature convergence. Furthermore, proportional selection also works poorly when all programs have similar fitness, e.g., when approaching convergence, since this will lead to all programs receiving approximately the same probability of being selected, resulting in a random search.

Rank selection

To hinder the premature convergence, Baker (1985) suggested *rank selection*, i.e., instead of fitness, the selection probabilities should be based on the programs' ranks within the population. In a population of n programs, the fittest program is assigned the rank of one, the second best two, and the worst program is given the rank n. Based on these ranks, the selection probabilities can be assigned in different ways. For linear ranking, the probability for a program with the rank of i can be calculated using equation 37, where s is a value between 1 and 2. A higher value of s results in a higher selection pressure.

$$P_i = s - \frac{2(i-1)(s-1)}{N-1} \tag{37}$$

Rank selection limits the selection pressure and effectively counteracts the domination of a few very strong programs. Furthermore, rank selection also exaggerates the difference between programs of very similar fitness in favor of the slightly better individuals and can thus drive the evolution forward, even when the differences are small.

Tournament selection

Another very popular selection scheme is *tournament selection* (Goldberg & Deb 1991) whose basic idea is to mimic the natural selection more closely, by basing the selection on small tournaments, i.e., competition, among a set of n randomly selected individuals. A new generation is simply created by conducting tournaments, until the new generation is the same size as the previous one. Tournament selection has become popular, since it can work directly on the raw fitness and is not dependent on a translation to ranks. The selection pressure can also be controlled by adjusting the tournament size. A low number of individuals increase the randomness of the selection and thus lower the selection pressure, while a higher value increases the selection pressure. The number of competitors in a tournament, normally between two to four, is constant during the whole evolution.

Goldberg and Deb (1991) also show that tournament selection using two individuals per tournament has the exact same properties as linear ranked selection. Hence, program domination can be counteracted and small differences can be exaggerated without calculating ranks.

2.5.5 GP Search and the Schemata Theory

Koza (1992) argued that GP is an implicit search among good building blocks, i.e., *schemata*. Good building blocks would grow exponentially, due to the selection pressure, and crossover would rearrange them into ever larger and stronger building blocks.

The schemata theory was developed by Holland (1992) and describes how selection affects the distribution of schemata in the next generation. However, Holland's schemata theory cannot be directly applied to GP, since it was developed for genetic algorithms (GA). GP is an extension of GA and the main difference between them is that GA uses a fixed-size binary strings representation, while GP's representation allows programs of varied size. However, Poli and McPhee (2003a; 2003b) have more recently presented an exact schemata theory for GP.

Koza also argued that crossover is a very creative force which makes the GP search more effective than random mutations. However, Nordin et al. (1996) showed that crossover is extremely destructive, especially at the beginning of the evolution. Based on Nordin et al. and other similar studies, Banzhaf et al. (1998) conclude that the standard crossover has a very negative effect on the offspring and actually primarily behaves as a macro mutation. During a normal GP evolution, approximately 75% of all crossovers have a damaging effect on the offspring.

In nature, crossover is much more successful, but it also differs from standard GP in three distinct ways (Banzhaf, Nordin, Keller & Francone 1998). First,

individuals only reproduce within their own species, which means they will be relatively similar. Secondly, biological crossover has a tendency to preserve semantics, i.e., the crossovers most often split genes with the same function. Finally, biological crossover is *homologous,* since the DNA strand is lined up before a crossover, resulting in offspring that are more similar to their parents in size and function. Furthermore, crossover is not restricted to a single point but can occur at several points where the genes match. Based on these facts, it is not strange that a random crossover on two randomly created programs is most often harmful.

Several researchers have tried to improve GP performance by introducing genetic operators which are more similar to biological crossover. Tackett (1994) suggested that crossover should over produce offspring, which is also common in nature, and only the best two should be selected. Poli and Langdon (1998) propose a uniform crossover which more closely resembles biological crossover. In uniform crossover, entire subtrees are not swapped, instead, only a proportion of the nodes that match up when two trees are aligned are swapped, if they are selected for a crossover. Two nodes are said to match if they have the same arity and if all the nodes of the branches leading to the node match.

Even if these and many other studies show promising results, there is still no consensus in favor of a certain crossover. Furthermore, even if standard crossover is only a macro mutation, it is still powerful and has been successfully applied on a wide variety of problems.

2.5.6 Parallel GP

GP has traditionally been considered slower than standard techniques, since GP is very computationally intensive. However, since GP is naturally parallelizable, the difference is rapidly decreasing, due to parallel GP implementation on many core CPUs and recently even more on massive core GPUs. The following section thus provides some examples of the current state of GPU parallel GP. For an extensive survey of the field see (Banzaf, Harding & Langdon 2009) or (Langdon 2011). All studies presented below used NVIDIAs CUDA for the GPU parallelization.

Lewis and Magoulas (2009) implemented a parallel GP for cyclic graphs and reached a speedup of 434 times over a single core CPU. However, Cyclic Graphs pose different evaluation problems than tree-based GP, which is the focus of this thesis. Classification using GP, which is a more typical predictive task, was evaluated by Cano, Zafra, and Ventura (2010). Three different types of GP rule classification schemes to evolve a rule for each class were parallelized. The GP GPU implementation was compared to a single thread GP on two UCI- datasets with speedups of 256 to 1000 as a result.

In more recent work, i.e. (Cano, Zafra & Ventura 2012) evaluated the GPU parallelization of decision tree-based GP. Here, the parallelization is focused on speeding up the fitness evaluations and a scalable CUDA GPU based programming model was presented. Experiments show that the proposed model significantly reduced the computational time up to 820 times, compared to a sequential approach. The approach reaches the highest speedups for high dimensional problems and problems with a large number of instances.

Even if the highest speedups for the studies presented above were achieved for high dimensional problems with a large number of instances, the results are still promising for GP, since these problems naturally take the longest time to run. The recorded speedups would reduce even a long GP run of ten minutes to a few seconds and more normal runs to fractions of a second. Hence, even if traditional decision tree techniques are still faster, the difference would not matter for all but real-time systems, which are rare for predictive models.

2.5.7 Extensions to standard GP

Several methods for improving the classification performance of GP systems have been suggested. Eggermont, Eiben, and van Hemert (1999) proposed a GP-system that used stepwise adaptation of weights (SAW). The basic idea is to dynamically assign higher weights for errors on "hard" instances, in an online fashion. However, the experiments show that the suggested SAWing approach fails to consistently reach a higher performance, compared to standard GP classification.

Eggermont and van Hemert (2001) later adapted SAW for regression problems. Here, the weights were again increased according to the errors made by the best individual in the population. Two variations of the SAW algorithm were evaluated:

- Classic SAW (CSAW) that increases the weight with 1, if the error is not zero, and the performance of the best individual in the population.
- Precision SAW (PSAW) which increases each weight with the absolute error made on its corresponding instance.

The techniques were evaluated against standard GP using different population sizes and number of generations on several simple polynomial expressions. Since the polynomials were relatively simple, the performance of the techniques were measured as both the mean error and the number of successful runs, i.e., runs where the best individual had an absolute error less than 10^{6}.

The experiments show that PSAW is the more promising of the two SAW variants. PSAW is slightly better than standard GP for small populations, i.e., 100 individuals, in terms of successful runs and MAE. For more normal populations, i.e., 500 individuals, the standard GP actually outperforms PSAW and CSAW.

In another study, Eggermont, Kok and Kosters (2004) present another approach where machine learning techniques are used to refine the GP search space. One method applies K-means clustering to reduce the huge number of possible primitives. Another approach uses C4.5's entropy gain criterion (calculated on the whole dataset) to create n new partitions that are used instead of the normal atomic representation. Both methods show promising results compared to normal GP and C4.5. A problem for both techniques is, however, that their performances are clearly dependent on certain non- trivial parameter values, i.e., the number of clusters used in K-means and the number of partitions.

A classifier system based on an alternative representation where several classification rules are evolved was used by Tan, Tay, Lee and Heng (2002). Each rule consists of enumerations of attribute value pairs, combined with only two Boolean functions, AND and NOT. If more than one rule covers an instance, the best rule will be used for classification. GPC, as the method is called, also uses a covering algorithm to calculate the overlap of different rules and penalize redundant rules. Experiments on nine datasets showed that the algorithm performs comparably to other classifiers, despite the simple representation.

Keijzer (2003) provides an example of the difficulties that can occur when using a straightforward GP approach for a symbolic regression problem. Keijzer evaluates the GP success rate for two simple target functions $t=x^2$ and $t=100+x^2$. The squared error was used as fitness function and 50 GP runs over 20 generations were performed for each function. The first function was often already found in the first generation and had a success rate of 98%; the second function, however, was seldom found and had a success rate of only 16%. Keijzer argues that the low success rate is the result of the selection pressure enforcing the GP process to spend most of its effort on getting the range of the constant right. Once found, the diversity is so low that the square function is never found. Hence, most of the runs for the second function converged on the average of the training data and only eight runs managed to find something better.

Keijzer shows how this problem can be eliminated for polynomial expressions, by scaling the output of each program with a simple linear regression. Given that p_i is the prediction of the program for the instance i and that a_i is the actual value for the same instance, the slope k and the intercept m for the linear regression can be

calculated using the least square, according to the equation below.

$$k = \frac{\sum_{i=1}^{n}(a_i - \bar{a})(p_i - \bar{p})}{\sum_{j=1}^{n}(p_i - \bar{p})^2}$$
$$m = \bar{a} - k\bar{p} \tag{38}$$

When k and m are known, all prediction made by the program is scaled using equation below.

$$p_{i_{scaled}} = k * p_i + m \tag{39}$$

For regression trees, the same problem of finding the correct range would appear in each leaf node. It would, however, not help to scale the final output of the model, since the range of the intercept would need to be in a scale appropriate for the associated variable.

In the light of the range problem, as demonstrated by Keijzer, the most straightforward approach for evolving model trees, even with simple regressions based on a single variable, will experience severe problems. Nevertheless, this approach has of course been applied by numerous researchers with varying amounts of success.

2.5.8 Application of GP to PM

Even if GP has been used successfully for over a decade, it is still not normally considered for predictive problems. Nonetheless, GP has been successfully applied to a wide variety of numerous real-world predictive problems. Here, a few problems to which GP has been applied and compared to other techniques are presented, to show just how flexible and powerful GP can be.

Archetti, Lanzeni and Messina (2007) evaluated the use of GP for predictive pharmacokinetics, with regard to the estimation of absorption, distribution, metabolism, excretion, and toxicity processes that a drug undergoes in the patient's organism. Their experiments on three pharmacokinetic datasets showed that GP outperforms linear regression, least square regression, ANN, and SVMs, both in terms of RMSE and correlation.

In another study, conducted by Bianco et al (2008), an evolutionary framework for the colorimetric characterization of scanners was evaluated. The problem concerned finding a polynomial which could perform the mapping between two color systems, i.e., RGB and CIELAB. Here, the GP approach is evaluated towards the state of the art polynomials which are optimized using least square, total square,

and two domain specific methods. GP is used to evolve a new polynomial which is then optimized using GA. The results show remarkable improvements for the evolved polynomials compared to the best known polynomials prior to the study.

Ocean component concentrations (phytoplankton, sediment, and yellow substance) were predicted in the basis of satellite images, by Fonlupt (2001). GP was compared to polynomial methods and ANN on two datasets, with promising results. The evolved models clearly outperform traditional polynomial techniques and rivaled the performance of ANN.

Lee (2006) used GP to successfully predict bankruptcy for Taiwan listed electronic companies. In the study, GP clearly outperforms CART, C5.0, ANNs and logit models on a dataset consisting of 55 companies in distress and 110 normal.

GP has also been applied to rhythmic stress detection in spoken NZ English, by Xie, Zhang and Andreae (2006). Several sets of features based on prosodic and vowel qualities were extracted from segments of speech. In the study, GP was compared to C4.5 and LIBSVM on 60 examples using three different feature sets. The experiments show that for this dataset GP was superior to both techniques in terms of accuracy and was also less sensitive to irrelevant features.

A totally different study, performed by Tsang, Li and Butler (1998), used GP to predict the winners of 180 handicapped horse races in the UK, which are notoriously hard to predict. Here, GP was compared to other common betting strategies and outperformed them by at least twice the return of investment.

Finally, Barriere, Lutton, and Baudrit (2008) used GP to model the ripening process of cheese, which is an extremely complex and not fully known process. Hence, human expert operators were studied to capture their knowledge and facilitate the creation of predictive models. Experiments showed that for this problem GP was about 10% better than multiple linear regressions and 5% better than Bayesian networks.

3 Criteria for GP frameworks for PM

The following section presents a set of general criteria for evaluating a GP framework for PM (GPPM). More specifically, the criteria have been selected to highlight the challenges and advantages that need to be considered when using GP for PM. Even if an extensive literature survey has been conducted as the basis for the thesis work, the survey is most probably not exhaustive and more criteria may certainly be added to the list. However, the ones presented here should be of value for any GPPM. The criteria are divided into five categories: *Generality, Accuracy, Comprehensibility, Interestingness,* and *Evaluability.* Reasonable execution time and accuracy are two vital criteria that, of course, must first be fulfilled for any of the criteria proposed below to be relevant. These criteria are, however, problem dependent and cannot be evaluated independently. Assuming that the criteria of reasonable execution time and accuracy are fulfilled, the following criteria GPPM can be identified from the key observations and the background presented in chapter 2.

1. **Generality**
   - 1.1. GPPMs should be general enough to handle n-class classification and regression.
   - 1.2. GPPMs should allow PM through a GUI, since it cannot be assumed that PM experts have programming skills.

2. **Accuracy**
   - 2.1. GPPMs should facilitate optimization of typical score functions, such as ACC, AUC, BRI for classification tasks and MAE, RMSE, MAPE, MUAPE, r for regression tasks, to ensure a better match of the actual score function.
   - 2.2. GPPMs should facilitate optimization of arbitrary score function, since all possible score functions cannot be predefined.
   - 2.3. GPPMs should be specially designed to handle large search spaces, since this is a challenge for GP-based techniques.

3. **Comprehensibility**
   3.1. To create comprehensible and accurate models, a GPPMs should allow a user to trade the importance of these criteria.
   3.2. Since comprehensibility is subjective, GPPMs should support tailoring of the model representation.
   3.3. GPPMs should facilitate the removal of introns, so that unnecessarily complex programs are not produced.
   3.4. GPPMs should be designed to exploit the inherent inconsistency, since it may be used to provide alternative programs and indicate informational instability in the data.

4. **Interestingness**
   4.1. GPPMs should, in some way, promote the discovery of interesting rules.

5. **Evaluability**
   5.1. Since there are no free lunches in PM, i.e., no technique can be best for all problems, GPPMs should facilitate easy benchmarking against other PM-techniques using n-fold cross validation and stratification.
   5.2. GPPMs should be open source to facilitate verification and evaluation of new techniques. This criterion is vital for the aim of this thesis, but not as important for most real-world applications.

3.1 Evaluation of GP frameworks for PM

There are numerous GP implementations, including commercial, freeware, and open source. This evaluation aims to describe the state of the art of GPPM software, i.e., applications or frameworks, and as a motivation of the framework used in this thesis. Applications are included in the evaluation of frameworks to explore the state of the art GP functionality for PM.

Thirteen of the most common and cited GP software for PM are evaluated. The selected software were recommended by Poli, Langdon and McPhee (2008), listed on http://www.kdnuggets.com/ or found by googling terms such as "genetic programming, predictive modeling, data mining, framework, application". Table 5 below summarizes the evaluation according to the suggested criteria. Criteria that are completely fulfilled are marked using "*", partially fulfilled using "/", while criteria that can easily be fulfilled by extending the code base are marked with "c". To complement the evaluation, each framework is described briefly after the table.

Criteria	Applications						Frameworks						
	Eureqa	GeneXproTools	Dicipulus	AI Solver Studio	GATree	DTREG	ECJ	GPC++	Open BEAGLE	Lil-gp	EpochX	WekaGP	G-REX
1.1 Regression & n-class	/	/	/	*	/	/	c	c	c	c	c	*	*
1.2 GUI	*	*	*	*	*	*						*	*
2.1 Predefined fitness function	/	*	/			/						/	*
2.2 Tailoring of fitness function		*	c				c	c	c	c	c	c	c
2.3 Large search spaces													
3.1 Trading Acc vs. Comp.		/	*	*	*	*	*	c	c	c	c		*
3.2 Tailoring of representation	*	*	/			*	c	c	c	c	c	*	*
3.3 Intron removal		*	*			*							
3.4 Handling of inconsistency	/	/	/										
4.1 Discovery of Interesting rules													
5.1 PM benchmarking						*						*	*
5.2 Open source							*	*	*	*	*	*	*

Table 5 – Evaluation of GP Frameworks

Eureqa

Cornel Creative Machine Lab develops the Eureqa software (Schmidt & Lipson 2009) for symbolic regression. Eureqa has a large set of predefined fitness functions for regression problems and facilitates tailoring of the representation. There is no way to create new fitness functions or control the parsimony pressure, but a list of alternative programs is presented to the user. Eureqa is free to use but is not open source, which of course limits the functionality to the predefined.

GeneXproTools

GeneXproTools 4.0 developed by GEPsoft Limited (2001) is another commercial tool based on gene expression programming (GEP), which is actually an approach for evolving decision trees using genetic algorithms (Ferreira 2001). However, even if GEP is fast and powerful, it restricts classification to binary tasks, since each gene represents an expression. Apart from this, GeneXproTools is a rather complete GP environment with many typical PM score functions built in, as well as

functionality which allows tailoring of the representation function and the fitness function, plus a kind of intron removal. A predefined parsimony pressure may also be applied, but it cannot be adjusted without implementing a new fitness function. Inconsistency is handled by presenting a set of alternative programs to the user. Even if the fitness functions are customizable, this is the only code that can be changed, since GeneXproTools is not open source.

Dicipulus

Discipulus v5.2 (RML Technologies, Inc. 2013) is a commercial GP system based on linear GP, where individuals are represented as a sequence of instructions. In Discipulus, the instructions are real machine code and the application is therefore much faster compared to typical single thread GP software. In addition to typical GP features, the system also supports optimization of AUC, MAE, RMSE, and externally defined fitness functions, as well as having functionality to remove introns. However, the evolved machine code is rather opaque, irrespective of whether the possibility to translate it into *C* or Java code is used. Furthermore, even if it is possible to select which terminal and functions should be used, the representation is restricted to machine code. Another drawback of this otherwise very powerful system is that only binary classification is supported and, hence, cannot be applied to many classification problems. Finally, it is interesting to note that the designers of Dicipulus do not try to reduce the inconsistency of GP but instead present a number of alternative solutions.

AI Solver Studio

AI Solver Studio is free but not open source software, developed by Perseptio Ehf. (2007), which supports both regression and n-class classification. The application has a minimalistic GUI where the only available settings are related to the complexity of the solutions and a weighting of classification errors. All other settings are hidden and use default or automatically set values. A big disadvantage of AI Solver Studio is that it does not present the evolved programs, which hence makes a comprehensible technique totally opaque.

GATree

Papagelis and Kalles (2002) have developed GATree, which is another commercial GPPM environment that uses genetic algorithms to evolve decision trees. Most typical GP settings including parsimony pressure can be modified, but regression is not supported and the fitness function and representation are fixed.

DTREG

DTREG is a commercial PM environment, developed by Sherrod (2003), which among most other common PM techniques also includes GEP for symbolic regression problems and binary classification. There are some fitness functions to choose from and the representation can be defined to fit a particular problem. Parsimony pressure is also implemented as well as an algebraic simplification of the evolved programs. Since DTREG is a full PM environment, benchmarking towards other techniques are easy to perform. However, the software is not open source and it is not possible to create new, or modify, existing fitness functions.

ECJ

ECJ 20 is developed by George Mason University's ECLab (2013), Evolutionary Computation Laboratory. It is an open source evolutionary computation programming framework written in Java which includes GP functionality. Since ECJ is a general framework, it does not have specific functionality for classification or regression. Instead, each problem must be defined in a Java file and GP settings are included through a property file. Of course, all required functionality could be implemented in Java code. With regard to GP functionality, ECJ is however a rather complete system.

Open BEAGLE

Open BEAGLE 4.0 is another open source programming framework for evolutionary computation, with support for GP written in C++, developed by Gagn´e and Parizeau (2007). Most standard GP functionality is implemented and parameters are set in code or through a configuration file. Except for example files regarding optimization of RMSE for symbolic regression problems, no specific PM functionality is available.

Lil-GP

Lil-GP is a general GP framework developed by Punch and Zongker (1998). Lil-GP was designed for speed, ease of use, and experiments with multiple populations, but has no specific functionality for PM. Since 1998 the framework is no longer under active development, but the source is available, so relevant PM functionality could be implemented.

EpochX

EpochX is another open source GP framework written in Java by Otero, Castle and Johnson (2012). Three types of representations are available, which can include any of a large set of predefined components and operators. EpochX is a

general GP framework and does not include any PM specific functionality, except a symbolic regression example problem.

GPC++

Fraser and Weinbrenner's (1997) GPC++ v0.5.2 is a minimalistic open source programming framework for GP. Only the most common GP functionality is available and no PM specific functionality is implemented. The current version was released 1997 and the framework is no longer under active development.

WEKA GP

WEKA GP is a GP module for WEKA, developed by Yan (2008), which supports regression and n-class classification. However, classification is done using multiple one against-all rules, which reduces the comprehensibility, since all rules must be evaluated to make a prediction. Classification tasks are optimized using ACC, while MAE and RMSE can be used for optimization of regression tasks.

G-REX

G-REX (König, Johansson & Niklasson 2008) is a GPPM and an open source programming framework. Both regression and n-class classification are supported using numerous predefined fitness functions for regression and classification. Parsimony pressure can be adjusted and the representation can be tailored both grammatically and by selection of the elements in the representation. G-REX can run from WEKA or stand alone with a GUI where all relevant settings are available, including benchmarking towards some of WEKA's more common techniques. New fitness functions and genes can be defined by extending the appropriate Java class. A unique feature of G-REX is that it supports pedagogical rule extraction, as described in section 2.2.6.

Other open source frameworks

The open source frameworks evaluated above are just a small sample of the ones available online, they are, however, deemed to be representative. Typical GP frameworks do not, in general, contain any specific functionality for PM, even if, of course, it is possible to implement. Other prominent typical frameworks not evaluated in this study are: Java Rapid GP (2009) and Groovy Java Genetic programming (2006).

3.2 Key observations regarding GP software for PM

The framework evaluation was not only done to motivate the use of G-REX, but also to show the state of the art regarding GPPMs. Table 5 clearly shows that many of the proposed criteria are fulfilled by most frameworks, but also that some are neglected by all. More specifically, the following key observations can be noted:

- Surprisingly few of the evaluated GP software, i.e., AI Solver, WekaGP and G-REX, completely fulfilled criterion 1.1 by allowing both regression and n-class classification.
- While all of the commercial applications have GUI, only G-REX and WekaGP of the open source frameworks have a GUI. Since a GUI has obviously been deemed important both in the literature, i.e., by O'Neill et al. (2010), and the commercial frameworks, they should also be given more attention by the open source GP frameworks. Finally, any "one button GP" must of course also have a GUI.
- Rather few frameworks had predefined fitness functions for optimization of typical PM score functions.
- The importance of criteria 2.1, 3.1, and 3.2 can be recognized by the fact that a clear majority of the evaluated frameworks facilitate tailoring of the fitness function, apply parsimony pressure to handle the accuracy vs. comprehensibility tradeoff, and all but two frameworks support tailoring of the representation.
- Criteria 2.3, which regard handling of large search spaces, are not met by any of the evaluated frameworks. Furthermore, there is no mention that large search spaces could be a problem and hence these criteria need empirical evaluation.
- Three of the frameworks handle inconsistency, i.e., criterion 3.4, by presenting a small set of alternative programs.
- Only three of the commercial frameworks fulfill criterion 3.3 by including functionality for the removal of introns. The other frameworks do not mention introns as a problem, even if it is a well-known fact in the literature. Hence, an empirical evaluation of how introns affect the comprehensibility is motivated.

- DTREG was the only commercial framework which together with G-REX and WekaGP fulfilled criterion 5.1 and facilitate benchmarking against other PM techniques. Not facilitating PM benchmarking must be regarded as a major deficiency, since it is well known that not any one technique can be best for all problems. Furthermore, without benchmarks against at least naïve techniques, it is impossible to know if the achieved accuracy is reasonable.

3.3 Motivation of the selected GP framework

One of the main reasons for using the G-REX GP framework as a base for the thesis work is that it has been developed by the author as a result of the licentiate thesis. Consequently, every intricate detail of the framework is well known and understood, which also Poli, Langdon and McPhee (2008) recognize as an important aspect when choosing a framework. However, as Table 5 indicates, G-REX is well suited for the task of PM and rivals the most competent commercial products, such as Dicipulus and DTREG, while still being open source. Since the thesis aims to develop and test novel techniques, the source code must of course be available. Except for G-REX, only WekaGP has any real PM functionality and WekaGP's PM functionality is only due to its integrations in WEKA. All other frameworks are rather general GP frameworks that would require the implementation and testing of a lot of extra functionality. WekaGP would be a reasonable choice if a more typical decision tree representation could be used. The current representation is unnecessarily complex, since a rule is evolved for each class, but all rules must still be evaluated to make a prediction. Furthermore, parsimony pressure is not implemented in WekaGP and there are only three predefined fitness functions. From the evaluation and discussion above, it is deemed that the use of G-REX is motivated, since it fulfills most of the proposed criteria and has been thoroughly tested both by the author and other researchers in numerous studies, e.g., see (Martens et al. 2007,2008),(Martens, Baesens,& Van Gestel 2009), (Johansson, Sönströd, & Löfström 2010) (Leichsenring, Hanheide & Hermann 2011) .

4 G-REX a GP Framework for PM

From the beginning, G-REX was created as a tool for performing pedagogical rule extraction using GP; i.e., see (Johansson, König & Niklasson 2003). During the following years, the functionality of G-REX was extended, by the author, as it was used in several successful studies, including the licentiate thesis work of the author (König 2009). Rule extraction is essentially a PM task, which naturally meant that most of the functionality implemented in G-REX could also be applied to PM. Hence, it eventually became natural to adapt G-REX to fully support PM and to launch it as an open source GP framework for PM, which was carried out by König, Johansson and Niklasson (2008). Since all techniques evaluated in this thesis have been implemented using G-REX, the following chapter presents the general GP design of G-REX's GP engine, based on the articles mentioned above. The framework itself, with all the practical details of how to use and extend it, is however presented in the appendix section 11.1.

4.1 Representation

G-REX can optimize arbitrary representations, since the GP optimization is inherently independent of the representation. A combination of constrained syntactic and strongly typed GP is employed to give the user full control of the structure of the programs without needing to bother about dataset specific details, such as defining how all independent variables should be handled. More specifically, functions are handled syntactically, while the input variables are strongly typed as categorical or numerical variables. For more details see section 11.1.8.

Scoring

A fundamental design choice in G-REX is to use *leaf scoring,* i.e., scoring instances when they reach a leaf node, rather than using *program scoring,* i.e., evaluating the program an instance at a time, by propagating the prediction made from a leaf back to the instance. More specifically, when using leaf scoring, the training set is presented to the root node of a program and then partitioned into smaller sets by the following child nodes. When a leaf is reached, all training instances that should be scored by that leaf are present and can be used to calculate a suitable prediction. Hence, it is not necessary for the predictions to be done using ERCs, instead, they can be calculated in different ways to simplify the search process, e.g., by predicting the majority class for classification tasks or the average value for regression tasks. Furthermore, additional metrics like probability estimates can also be calculated on the basis of the training instances in a leaf, thus giving more informed predictions.

Another advantage of the leaf scoring design is that it is well suited to handling large search spaces, since a local search may be carried out in each node. Consider, for example, the problem of optimizing AUC, which requires a probability estimate: *leaf scoring* would reduce the problem of finding the best partitioning of the dataset, since the probability estimates could be calculated on the basis of the distribution of the training instances in the leaf; *program scoring* using ERCs would however, in addition to the optimization of the partitioning of the dataset, require a search for the best probability estimate for each partition, which of course is a much harder problem.

A negative aspect of the node scoring design is that the whole training set must be held in memory during training, which can be a limitation for extremely large datasets. However, even if it is possible to save the training instances in each leaf node, this is normally not necessary, since the calculated prediction is all that is needed to score novel instances.

Finally, it should be noted that the *program scoring* design could also apply similar functionality in a second optimization phase, before the actual scoring, if not only the prediction but also the predicting leaf would be registered for each prediction. This approach would however require the local search functionality to be implemented separately from the leaf class, thus resulting in classes with lower cohesion and higher coupling, which is opposite to the object-oriented design principles.

Creation of initial population

G-REX supports *Grow*, *Full* and *Ramped half-and-half* initialization, with the addition that the chosen representation is enforced. The first function is selected randomly and each parameter of the function is then selected from a parameter list with a uniform probability. This means that functions and terminals, by default, will be selected with the same probability and not 80%/20% which Koza recommends. However, the exact selection probability can easily be changed by repeating the name of a function in the parameter list. This approach is more flexible and lets the user decide the probability for each function and terminal used in a representation.

4.2 Evolution

The evolution process implemented in G-REX closely follows Koza's (1992) original description, with a few exceptions. Each step of the evolutions is described in detail below.

Fitness functions in G-REX are minimized and have a built-in, typical parsimony pressure, according to equation 31. All predefined fitness functions in G-REX are

normalized to promote more similar effects of the same level of parsimony pressure. To speed up the fitness calculation, *n*-threaded concurrent evaluation is supported. Furthermore, only the predictions made by a part of a program affected by crossover or mutation are recalculated each generation.

A unique G-REX feature is that it supports rule extraction from other PM techniques (implemented in WEKA). When performing rule extraction, G-REX replaces the actual value of the target variable, of all training instances, with the prediction of the predictive model, i.e., the pedagogical approach described in section 2.2.6. Hence, all fitness functions can also be used for rule extraction.

G-REX supports both Roulette wheel- and Tournament selection and creates a mating pool of the same size as the original population. Crossover, mutation, and reproduction are then applied in user specified proportion. One difference from Koza's implementation of the genetic operators is the assurance that the programs produced by the crossover and mutation operators are legal according to the current representation. Another difference is that values of constants used in combination with a variable, e.g., x_i<*constant*, are represented by an internal value which is translated to a value of the corresponding variable, to ensure that the meaning of a constant is retained, even if (due to a genetic operation) it is associated with another variable. In other words, a low internal value will always correspond to a low value of the associated variable, regardless of the variable scale or interval; see section 11.1.8 for more details.

4.3 PM using G-REX

In the evaluation of G-REX against the identified GPPM criteria, G-REX fulfills a majority of the proposed criteria. G-REX can solve both regression and n-class classification problems, has predefined fitness functions for most typical score functions, handles the accuracy vs. comprehensibility tradeoff, allows tailoring of both the representation and the fitnessfuntion, and facilitates benchmarking against many other PM-techniques.

However, like all other evaluated frameworks, G-REX does not fulfill all the criteria. More specifically, G-REX, in the original version, lacks the ability to remove introns, handle large search spaces, handle the inherent inconsistency of GP, and it has no explicit functionality to promote the discovery of interesting rules.

5 Exploring the Advantages and Challenges of using GP for PM

Of the criteria for GPPM frameworks, proposed in chapter 3, criterion 2.2, tailoring the fitness function to the actual score function, and criterion 3.2, tailoring of representation, can be regarded as advantages of using GP for PM, since they are inherently fulfilled by the algorithm. The optimization is independent of the representation and the search performed by the evolutions is independent of the fitness function, as long as it somehow scores the strength of a program. The importance of these criteria can be motivated, both from the literature, see Espejo, Ventura and Herrera (2010), and by the fact that most GPPM frameworks implement this functionality, see Table 5. Nonetheless, even if the criteria are motivated, it is not entirely clear to what extent they are actually beneficial in practice. Hence, section 5.1 explores these criteria empirically, using G-REX which completely fulfills both criteria. Another advantage is GP global optimization which, in theory, should be superior to the greedy optimization performed by typical decision tree techniques.

According to the literature, e.g., (Jeroen Eggermont, Kok & Kosters 2004) and (Koza, 1999), GP also has some challenges when used for PM, e.g., problems with large search spaces, and an increased complexity due to introns and their inherent inconsistency. Of these challenges, which correspond to criteria 2.3, 3.3, and 3.4 respectively, only the removal of introns has been given any attention in practice, and then only in three of the thirteen evaluated frameworks, see Table 5. Obviously, there is a discrepancy in what theory and practitioners consider problematic. Hence, section 5.2 empirically explores how GP's global non-greedy search handles large search spaces and the extent of the inconsistency. Introns are discussed in section 7.1, together with a novel representation independent technique for intron removal.

5.1 Advantages of using GP for PM

The following sections explore the inherent advantages of using GP for PM. Section 5.1.1 first evaluates the advantage of the optimization of arbitrary score function in a straightforward way, by showing the effect of optimization and the possible use of different score functions. The following section, 5.1.2, evaluates a totally different use of GP's flexible optimization, i.e., G-REX's rather unique ability to perform rule extraction. Rule extraction can also be seen as a more advanced and powerful approach to the accuracy vs. comprehensibility tradeoff, i.e., criteria 3.1. Finally,

section 5.1.3 explores the possibilities and effects of the optimization of different representations.

5.1.1 Advantages of the Optimization of arbitrary score function

Most predictive techniques have a predefined score function that optimizes a single performance measure; e.g., decision trees maximize some measure of information gain and artificial neural networks, and most regression techniques minimize the sum of square error. Controversially, the optimized score function is rarely the same as the actual score function that defines the problem. Section 2.2 describes some of the more common performance metrics and it is clear that each metric has different properties that could be more or less favorable for a particular problem. Hence, it would of course be better to choose the score function that should be optimized, depending on the problem.

Since the only requirement of a fitness function is that it should be able to calculate the performance of a program, any performance measure can be used. Furthermore, a fitness function is not limited to the optimization of a single measure, but can be defined as a linear combination of measures or by using multi-objective GP. This is an important property, since Freitas (2002) argues that a predictive model should be accurate, comprehensible, and interesting. All three properties could be included in a fitness function and be optimized together as a whole. Decision trees also try to optimize accuracy and comprehensibility, but this is done greedily and often in two separate procedures, i.e., training and pruning. Espejo, Ventura & Herrera (2010) also note that GP has an advantage, since performance criteria, such as novelty, interestingness, utility, or interpretability, can be taken into consideration in the fitness function.

Empirical evaluation of the importance of the score function for classification problems

The following section presents an empirical evaluation of the importance of the score function, i.e. ACC, AUC, BRE, for classification problems, based on the article "*Using Genetic Programming to Increase Rule Quality*", by König, Johansson and Niklasson (2008b).

Method

The general method used in all experiments is to first optimize one model for each score function on the training data. Next, each model is evaluated on the test set against each of the performance metrics used in the evaluated score functions. G-REX is used to optimize the models in all experiments.

Missing values for continuous variables were replaced with the mean value of all non-missing values for the variable. Categorical missing values were, similarly, replaced with the mode value of the variable. When the missing values had been handled, each dataset was divided into ten folds stratified to ensure that each fold will have a representative class distribution.

In the first experiment, decision trees (according to the DT BNF defined in Table 66) are optimized for ACC, AUC, and BRE using f_{ACC}, f_{AUC} and $f_{BRE/ACC}$ according to equations 33, 40, and 41. Note that all fitness functions are minimized in G-REX and hence f_{AUC} is defined as *1-AUC*.

$$f_{AUC_p} = 1 - AUC_p + P * O_p \tag{40}$$

Both f_{ACC} and f_{AUC} use G-REX standard parsimony pressure, while $f_{BRE/ACC}$ uses the actual brevity (scaled with the parsimony pressure *P*) to control the size and structure of the evolved tree. Furthermore, since the definition of BRE in equation 15 (section 2.3.1) does not consider any predictive performance of the evaluated model, BRE cannot be the sole criterion for optimization. Hence, the fitness function is defined as a linear combination of ACC and BRE, *i.e.,* see equation 41 below. The first term of the fitness function is simply a calculation of the accuracy of the program, while the second term is the brevity scaled with *P,* the parsimony pressure.

$$f_{BRE/ACC_p} = \frac{1}{n}\sum_{i=1}^{n}(p_i! = a_i) + P * \frac{1}{n}\sum_{i=1}^{n}\#Conditions_{p_i} \tag{41}$$

Results - Optimization of decision trees

In the experiments optimizing decision trees, G-REX is executed in a batch of ten runs for each fold. The rule with the highest fitness is selected as the winner and is then evaluated on the test set. All fitness functions use *P=.05*, which should produce short and comprehensible rules.

All programs, evolved using G-REX, are compared to trees produced using CART. The representations do however differ slightly, since CART allows comparison of several categories in the same node (X = 1|2|...|n), while the DT BNF used by G-REX only allows a single category. Hence, the representation used by CART can be regarded as more powerful, since it can represent more complex models with fewer conditions.

Number of generations	100
Population size	1000
Crossover probability	.8
Mutation probability	.001
Creation type	Ramped half- and- half
Selection type	Roulette wheel
Maximum creation depth	6
Parsimony Pressure (P)	.05
Batch size	10

Table 6 – G-REX settings for optimization of ACC, AUC and BRE

Table 7 below shows the achieved accuracy for the different techniques on each dataset. All results are averaged values for ten stratified folds. The ranking of the evaluated techniques is presented next to the evaluated metric.

Dataset	ACC				Ranks			
	CART	f_{ACC}	f_{AUC}	$f_{BRE/ACC}$	CART	f_{ACC}	f_{AUC}	$f_{BRE/ACC}$
Breast-w	.943	.963	.941	.943	2.5	1	4	2.5
Colic	.853	.824	.848	.788	1	3	2	4
Credit-a	.855	.846	.852	.855	1.5	4	3	1.5
Credit-g	.740	.715	.707	.707	1	2	3.5	3.5
Diabetes	.756	.750	.737	.737	1	2	3.5	3.5
Heart-c	.758	.788	.749	.725	2	1	3	4
Heart-statlog	.774	.804	.726	.804	3	1.5	4	1.5
Hepatitis	.769	.819	.787	.814	4	1	3	2
Ionosphere	.897	.932	.877	.88	2	1	4	3
Labor	.747	.783	.817	.803	4	3	1	2
Liver-disorders	.661	.655	.626	.678	2	3	4	1
Sonar	.725	.711	.687	.721	1	3	4	2
Vote	.954	.949	.956	.956	3	4	1.5	1.5
Average	.802	.811	.793	.801	2.15	2.27	3.11	2.46

Table 7 – Average accuracy and Ranks (CD=1.21)

While f_{ACC} achieves the highest accuracy, CART actually has a slightly better average rank. The programs optimized for AUC have the lowest average ACC and the highest average rank. A Friedman test does however yield a p value of *.210* at a *.05* significance level, which illustrates that none of the techniques are significantly

better or worse than any other. This is also confirmed by a *Nemenyi* test, since there is no difference in average rank greater than *1.21*

Table 8 below shows the AUC values for the same programs previously evaluated on ACC. Here, f_{AUC} clearly outperforms the other techniques using this metric, achieving the highest AUC on 9 of 13 datasets. The second best technique is f_{ACC}, followed by CART and $f_{BRE/ACC}$. A *Friedman* test yields a *p-value* of $8.06*10^{-5}$, which indicates that there are significant differences among the techniques. A subsequent *Nemenyi* test, with a critical difference of *1.20,* indicates that the programs evolved with f_{AUC} and f_{ACC} have significantly higher AUC than $f_{BRE/ACC}$ and CART.

Dataset	AUC				Ranks			
	CART	f_{ACC}	f_{AUC}	$f_{BRE/ACC}$	CART	f_{ACC}	f_{AUC}	$f_{BRE/ACC}$
Breast-w	.933	.962	.972	.947	4	2	1	3
Colic	.853	.819	.883	.777	2	3	1	4
Credit-a	.862	.891	.922	.862	3.5	2	1	3.5
Credit-g	.719	.669	.705	.534	1	3	2	4
Diabetes	.720	.724	.798	.681	3	2	1	4
Heart-c	.780	.837	.838	.724	3	2	1	4
Heart-statlog	.771	.844	.834	.762	3	1	2	4
Hepatitis	.496	.755	.781	.567	4	2	1	3
Ionosphere	.902	.920	.918	.876	3	1	2	4
Labor	.740	.806	.833	.781	4	2	1	3
Liver-disorders	.642	.653	.640	.652	3	1	4	2
Sonar	.725	.745	.791	.737	4	2	1	3
Vote	.950	.968	.977	.959	4	2	1	3
Average	.776	.815	.838	.758	3.19	1.92	1.46	3.42

Table 8 - Average AUC and Ranks (CD=1.20)

Even if the *Nemenyi* test does not show a significant difference between f_{AUC} and f_{ACC} when all four techniques are compared together, f_{AUC} has a higher AUC than f_{ACC} for ten of thirteen datasets. A *Wilcoxon* test between f_{ACC} and f_{AUC} actually yields a *p*-value of .0178, which shows that f_{AUC} also achieves a significantly higher AUC than f_{ACC}.

The next table presents the results for all evolved models, in terms of brevity. Note that CART was not evaluated on BRE, since its representation allows category lists and thus does not facilitate a straightforward comparison.

Dataset	BRE			Ranks		
	f_{ACC}	f_{AUC}	$f_{BRE/ACC}$	f_{ACC}	f_{AUC}	$f_{BRE/ACC}$
Breast-w	3.04	2.83	1.54	3	2	1
Colic	7.51	9.61	3.09	2	3	1
Credit-a	6.56	7.03	3.00	2	3	1
Credit-g	6.60	12.55	1.06	2	3	1
Diabetes	2.04	3.95	1.11	2	3	1
Heart-c	7.65	1.73	1.18	2	3	1
Heart-statlog	2.91	3.69	1.89	2	3	1
Hepatitis	5.44	9.77	1.14	2	3	1
Ionosphere	4.37	4.82	1.77	2	3	1
Labor	7.67	6.54	2.23	3	2	1
Liver-disorders	2.48	4.00	1.58	2	3	1
Sonar	3.06	3.84	1.62	2	3	1
Vote	5.87	9.77	3.00	2	3	1
Average	5.02	6.86	1.86	2.15	2.85	1

Table 9 – Average Brevity and Ranks (CD=.879)

Obviously, $f_{BRE/ACC}$ produces rules with very low brevity compared to the other score functions. A Friedman test yields $p=1.228*10^{-5}$ which indicates that there are significant differences in BRE among the techniques. A subsequent *Nemenyi* test, which for thirteen datasets and three techniques requires a critical difference of *.879,* shows that $f_{BRE/ACC}$ produces rules with significantly lower brevity than both f_{ACC} and f_{AUC}. It is worth noting that the same programs were comparable to f_{ACC} and f_{AUC} when evaluated on ACC. Another interesting result is that f_{AUC} is significantly worse than all other techniques when compared for brevity.

Analysis

The main result of the empirical evaluation is that a program achieves the best performance for the score functions it is optimized for. Furthermore, the same accuracy could be achieved (for small comprehensible decision trees) while gaining an extra quality in regard to the optimized score function, e.g., AUC or BRE. One explanation could be that ACC is a less informed score function, i.e., $f_{BRE/ACC}$ is a linear combination of ACC and BRE, and AUC implicitly requires high ACC. Hence, all score functions either explicitly or implicitly optimize ACC. However, the complexity of the models must also be taken into account, since it affects both AUC and BRE.

Table 9 shows that programs evolved using f_{AUC} had a higher BRE than programs evolved using f_{ACC}, for eleven of thirteen datasets. Both f_{ACC} and f_{AUC} used the same complexity measure and parsimony pressure, which means that the difference in complexity can only be credited to the optimized score function, i.e., high AUC requires more complex programs than if only a high ACC is sought. A *Wilcoxon* test does, in fact, show that the trees evolved with f_{AUC} (see Table 9) have a significantly higher BRE than trees evolved with f_{ACC}, ($p=.0225$). Consequently, the results showing that programs evolved using f_{AUC} have a comparable accuracy and at the same time a better AUC can be explained by the arguably higher complexity of the f_{AUC} programs.

It should be noted that a comparison with $f_{BRE/ACC}$ cannot be done in a straightforward way, since the parsimony pressure is calculated using BRE instead of the size of the program. However, it is obvious and natural that $f_{BRE/ACC}$ does not perform as well when evaluated against AUC, since it produces very short programs leading to quite rough probability estimations. In the same way, f_{AUC} is clearly worse than all other techniques when compared for brevity. This could probably be explained using the same argument as above; i.e., to achieve high AUC, it is necessary to have good probability estimations, which can only be produced by more complex rule sets.

Finnaly, when compred to CART, the evolved programs had comparable ACC while having significantly higher AUC or BRE, when the same measure was used as score function and for evaluation.

Empirical evaluation of the importance of the score function for regression problems

In a latter study "*Optimization and Evaluation Criteria in GP Regression*" König, Johansson and Niklasson (2012), evaluated the importance of the optimized score functions importance for regression problems. Here, G-REX was used to evolve regression trees and are compared to regression trees created using M5P and REPtree. Five common score functions for regression tasks, i.e., MAE, RMSE, MAPE, MUAPE and r, are evaluated. The metrics have their own advantages and disadvantages and are presented in detail in section 2.3.2. The metrics were selected on the basis of their popularity and that they are all appropriate for different tasks.

In a classification tree, it is always best to select a leaf's majority class, as its prediction. Hence, the problem of creating a good decision tree lies in creating the optimal partitioning of the training set based on the independent variables. This is also true for regression trees, with the exception that there is no single optimal way of selecting the value to predict in each leaf. All error metrics have different

properties and are calculated in different ways; hence, there can be different optimal values, depending on which metrics are used. Consider for example a leaf node which observes three training instances with the target values 1, 2, and 4; the average value 2.33 would result in the minimum error according to RMSE. However, as can be seen in Table 10, it would, according to MAE and MUAPE, be better to predict the value 2, while a prediction of 1 would be optimal for MAPE. Correlation is, on the other hand, meaningless to calculate at a leaf level, since correlation is undefined for prediction of a single value.

Predicted Value	MAE	RMSE	MAPE	MUAPE	r
2.33	1.11	**1.25**	63.8%	49.3%	0
2	**1.00**	1.29	50.0%	**44.4%**	0
1	1.33	1.83	**41.76%**	62.2%	0

Table 10 - Prediction errors for values 1,2,4

As demonstrated by this simple example, it is crucial to consider the score function used for evaluation when creating decision trees. Surprisingly, decision tree techniques, like M5P, REPTree and CART, are designed to optimize a single measure.

Related work

A straightforward approach to evolving model trees, even with simple regressions based on a single variable, will, according Keijzer (2003), experience severe difficulties, due to the range problem, see section 2.5.7. Instead, Keijzer recommends scaling the output of the evolved programs, using least square and a simple linear regression. Scaling works well for polynomial expression, but is not applicable to regression trees, since the range problem appears in each leaf.

Another approach to handling the range problem was taken by Kretowski and Czajkowski (2010), where regression trees were evolved using a fitness function based on MAE. The initial population was created using an algorithm similar to CART, but to achieve diversity, each individual was created using randomly selected subsets of the training data. Since the ephemeral constants were created using the decision tree algorithm, they would always be in an appropriate range. In that study, mutation was also performed by calculating a new split with the same method. The proposed technique was evaluated against WEKA's REPTree on 14 UCI datasets. Surprisingly, the results were rather modest, with seven wins and seven losses when evaluated using RMSE. The evolved trees were, however, slightly smaller (i.e., easer to interpret) than the trees induced by REPTree.

Finally, it should be noted that this and similar methods automatically make the constants dependent on the criteria optimized by the decision tree algorithm, e.g., least square or least deviation. So, the constants were initialized based on RMSE, the tree was optimized on MAE, and the final model was evaluated on RMSE. Clearly, the implicit choice to mix these error metrics, especially evaluating on a metric not targeted during the model construction, may very well be the reason for the somewhat discouraging results.

Method

As demonstrated by the simple example in Table 10, using mean values as leaf constants is only optimal if using RMSE. Or, put the other way around, the method of finding good ephemeral constants is dependent on the optimization criterion, and, specifically, using mean values can actually be suboptimal for other metrics

It may be noted that it would, of course, be possible to calculate the optimal value for each measure, but this would restrict the generality of the GP process. If any changes were incorporated in the fitness function, like adding weights or introducing a new measure, the leaf class would have to be updated to produce the optimal value for the new fitness function.

In this study, a more flexible approach is taken instead, letting the evolution find the right constants, but then scaling the ephemeral constants to the range of the training instances reaching each node. Internally, all leaf constants C_i are initialized to a random value between zero and one. These internal constants are, however, scaled by the maximum and minimum value of the dependent variable *y* for the training instance reaching the leaf *l* according to equation 42.

$$C_{rnd} = min(y_l) * C_i * +(1 - C_i) * max(y_l) \tag{42}$$

This approach drastically reduces the range problem while, at the same time, allowing optimization of an arbitrary fitness function. Another advantage is that the general meaning of the internal constants is conserved through crossover, i.e., a high value will result in a prediction near $\max(y_l)$ in all leaves, even if $\max(y_l)$ is different in different leaves.

The error measures targeted in this study have been implemented into five different fitness functions, whose general form can be described by equation 43 .

$$f_{M_p} = M_p + P_1 * O_p + P_2 * max(O_p - O_{max}, 0) \tag{43}$$

M is the optimized measure, *p* is the program under evaluation, O_p is the complexity of *p* (nodes + leaves) and *P*i is the coefficient for the parsimony pressure.

P_2 is the coefficient for a second parsimony pressure aimed to facilitate the evolution of programs of a certain complexity. Since the complexity of a tree inherently decides how many different values it is able to predict, a higher complexity may lead to an unfair advantage. Hence, in this setting, it is desirable for all trees to have roughly the same complexity, since it is then actually the choice of ephemeral constants that has the greatest effect on the performance. To enforce this, the parsimony pressure P_1 is typically set to a small value and P_2 to a much larger value. Hence, a large punishment will be added for each extra element above a maximum complexity threshold O_{max}. Note that the maximum error of M_p, in equations 43, was trimmed to 1 for MAPE and MUAPE since both metrics have problems when the target values are close to zero.

MAPE, MUAPE and r are scale independent, but MAE and RMSE must be made scale independent by dividing with the mean of the actual value according to equation 44.

$$f_{M_p} = \frac{M_p}{\bar{a}} + P_1 * O_p + P_2 * max(O_p - O_{max}, 0) \quad (44)$$

Scale independence, of course, ensures that the same level of pressure will have a similar effect on the evolution, thus minimizing the amount of parameter tuning. Still, the effect may vary slightly, since each metric represents a different error function, i.e., the change in deviation seldom affects the error in the same way for all measures. Finally, r values were replaced with $1\text{-}r$ to facilitate minimization.

Results

To enable comparison with the evolved regression trees, two standard techniques also producing regression trees were included in the experimentation. For this benchmarking, REPTree and M5P, as implemented in the WEKA workbench, were chosen. These techniques are presented in detail in section 2.2.2. REPTree was used with the default setting, while M5P was set to produce regression trees without any smoothing, to create trees similar to the ones created by REPTree and G-REX.

G-REX used the same representation language as REPTree and M5P, i.e., the RDT representation presented in Table 71, with the exception that the ephemeral constants were initialized according to equation 42 in all experiments.

All the models, regardless of which technique was used to create them, were evaluated against all error metrics; i.e., MAE, RMSE, MAPE, MUAPE and r. For the actual evaluation, standard stratified 10-fold cross-validation was used; i.e., each technique was used to create one regression tree per fold. The result for a specific setup and dataset is the average over all folds.

To achieve a more robust performance of the GP optimization, a batch of three separate runs were performed for every fold in each dataset. The tree with the best fitness of the three batches was selected as the final tree, and used for evaluation. Since the optimization criteria, encoded in the fitness functions, were made scale independent and equal in range, the same set of parameters presented in Table 11 below, found by initial experimentation, could be used for all fitness functions.

GP Parameter	Value
Population Size	1000
Generations	100
Creation Type	Ramped Half- and -Half
Crossover Probability	.8
Mutation Probability	.01
Complexity Threshold (O_{max})	15
Parsimony pressure (P_1)	.01
Parsimony pressure (P_2)	.5
Batches	3

Table 11 – G-REX settings for optimization of score functions for regression

Sixteen regression datasets from the UCI-Repository were used in the experiments, i.e., *Auto-price, Basketball, Bolts, Cholesterol, Cloud, Gascons, Housing, Machine-cpu, Pharynx, Pollution, Quake, Sensory, Servo, Sleep, Veteran,* and *Wisconsin.* Since each model was evaluated on five metrics, no results from the individual datasets are reported. Instead, Table 12 shows the average ranks over all datasets for each technique and each metric. Results in bold signify the lowest rank obtained among all techniques, while the best result among the different GP fitness functions are underlined.

Technique	MAE		RMSE		MAPE		MUAPE		*r*	
	TRN	TST	TRN	TST	TRN	TST	TRN	TST	TRN	TST
M5P	4.44	3.44	4.09	3.44	4.63	4.03	4.13	3.28	4.94	4.44
REPTree	3.84	3.63	3.75	**3.31**	4.03	4.03	3.41	**3.25**	4.81	4.19
f_{MAE}	**2.13**	**3.16**	3.09	3.78	3.25	3.63	2.84	3.34	3.50	4.06
f_{RMSE}	3.34	3.91	**1.59**	3.34	5.09	4.47	4.66	3.78	2.19	3.75
f_{MAPE}	4.59	4.25	5.72	4.69	**1.69**	**2.44**	4.06	4.47	5.53	4.50
f_{MUAPE}	3.66	3.91	4.81	4.31	3.63	4.03	**2.53**	3.69	5.22	4.19
f_r	6.00	5.72	4.94	5.13	5.69	5.38	6.38	6.19	**1.81**	**2.88**

Table 12 - Comparison of error metrics: Mean ranks (CD =2.25)

One first observation is that the fitness functions work as intended, even if few results are significant due to the large number of compared techniques, i.e. a *Nemenyi* test yields a critical distance in ranks of 2.25. On the training set, the GP

optimizing that specific criterion is always the best setup overall. Another interesting pattern is that on the training data, GP using f_{MAE} actually outranks both regression tree techniques on all performance metrics. In addition, that setup is also nearly always better than all other GP setups not specifically targeting the criterion used for the evaluation. This, of course, indicates that using MAE in the fitness function will produce models that also have quite good results on the training data, according to the other metrics.

In general, the test results indicate that optimizing a specific criterion on the training data will produce models that rank well when that same criterion is used for the actual evaluation; f_{MAE}, f_{MAPE} and f_r all obtained the best rank overall on the test data, when the respective criterion was used for the evaluation. Regarding RMSE, f_{RMSE} was clearly the best among the GP setups, although slightly inferior to REPTree. On MUAPE, however, f_{MUAPE} was outperformed by both regression tree techniques and by f_{MAE}.

When GP is compared to the specialized techniques, GP appears to be the best option overall, but REPTree, as mentioned above, obtained slightly better RMSE and MUAPE. This is despite the fact that f_{RMSE} and f_{MUAPE} had a much lower training rank. It must, however, also be noted that the trees produced by the REPTree technique are almost twice as large.

Finally, it is very interesting to compare all the models, to see which approach gave the most robust model, i.e., had the best result considering all the evaluated metrics. To make this comparison as fair as possible, the mean ranks, presented in Table 12 above, were averaged over all evaluation metrics to produce an overall ranking for each technique on training and test data, see Table 13.

Technique	TRAIN	TEST
M5P	4.80	3.40
REPTree	3.60	2.70
f_{MAE}	2.00	2.60
f_{RMSE}	3.40	3.90
f_{MAPE}	5.00	5.20
f_{MUAPE}	3.60	4.40
f_r	5.60	5.80

Table 13 - Comparison of error metrics: Overall ranks

It is obvious that GP using f_{MAE} was the most robust technique, since it obtained the lowest overall rank on both training and test data. REPTree was the second best technique overall, but again it may be noted that the REPTrees were almost twice the size of the trees produce by all other techniques.

Technique	Rank	Size
M5P	2.81	11.50
REPTree	4.56	21.40
f_{MAE}	3.97	13.55
f_{RMSE}	4.56	13.76
f_{MAPE}	3.69	12.88
f_{MUAPE}	3.72	13.40
f_r	4.69	14.53

Table 14 - Comparison of error metrics: Size

Analysis

One unexpected result was that GP using f_{MUAPE} was outperformed by both regression tree techniques and by GP using f_{MAE} when evaluated on MUAPE. A possible explanation to this phenomenon is that MUAPE is measured in percent, which makes the measure less precise, especially since the percentage is affected by the levels of both the predicted and the actual value. Based on this, a model optimized on this criterion may appear to be overly specialized; i.e., will generalize poorly. This is also apparent in the results, where GP using f_{MUAPE} has the best MUAPE result on the training data, but among the worst on the test data.

MAPE, which also defines the error in percent, does not however suffer from this problem, which can be explained by the fact that MAPE is the only metric biased towards low estimations.

Another quite unexpected result is that REPTree received a slightly lower rank on RMSE, compared to GP optimizing the same measure. However, given the fact that REPTree produced trees almost twice as large as the GP trees, see Table 14, a decision maker would arguably choose a smaller tree, since the difference in predictive performance is negligible. Hence, the comparison to M5P is the more interesting, since M5P trees are of a more similar size. M5P's performance is however quite similar to REPTree, when compared to the GP setups. The only difference that appears in the ranks is that f_{RMSE} beats M5P but not REPTree. This result is probably explained by the fact that M5P trees are less complex, which translates to fewer separate prediction values. Hence, M5P models and the GP models (which are similar in size) cannot be fitted to as many extreme values as the REPTrees. Since RMSE punishes larger errors harder, this restriction makes the less complex models inferior. All other metrics give an error that is proportional to the size of the deviation and are thus less affected by the size of the models. This can also be seen, since REPTree and M5P perform very similarly on all other measures.

Summary

The main conclusion that can be drawn, from the two sections above, is that the optimized score function is very important for the performance and that a program achieves the best performance for the score functions it is optimized for. In addition, a more informed metric may add some extra quality to the evolved program.

The experiments did, for example, show that the same accuracy could be achieved by G-REX (for small comprehensible programs) while gaining an extra quality in regard to the optimized score function, e.g., AUC or BRE. However, the extra quality most often came from slightly more complex programs. This is in line with the idea of "one button GP", if a program with high AUC is sought, GP will automatically evolve larger trees than if ACC would be optimized without any need of further parameter tweaking.

When compared to CART, the programs evolved using GP had comparable ACC while having significantly higher AUC or BRE, when the same measure was used as score function and for evaluation. Of course, CART does not optimize BRE or AUC, but the point is that it is possible to choose the score function when using GP and not when using CART and other decision tree trechniques.

When evolving regression trees, the result show that when using the corresponding fitness function, G-REX outperformed regression trees produced by both M5P and REPTree on MAE, MAPE and r. On RMSE, REPTree obtained the best mean rank, but the programs optimized for RMSE had very similar results and were almost half as complex. Only when MUAPE was used for the evaluation, did the specialized techniques clearly outperform the GP approach. When evaluated using all error metrics, it was shown that optimizing MAE resulted in the most robust models.

It should also be noted that similar results, i.e. that it is favorable to optimize the actual score function, were presented for decision lists by Sönströd, Johansson, König, and Niklasson (2008) and for kNN hybrid ensembles by Johansson, König and Niklasson (2010). Consequently, the results is general and not dependent on the particular representation.

The results are clear, but since very few techniques can optimize an arbitrary score function, this advantage is rarely exploited in practice. The possibility of tailoring the optimized score function to the problem is, for example, not even mentioned in most data mining text books or, which is even more serious, in the two most widely used data mining methodologies, CRISP-DM and SAS instruments SEMMA. Of course, both methodologies stress the importance of choosing the best technique for the problem, but since they only consider standard

predictive techniques and not GP, they miss the importance of the optimized score function.

5.1.2 Rule Extraction using Genetic Programming

Pedagogical rule extraction (see Figure 7) is a totally different way of exploiting GP's ability to optimize the arbitrary fitness function. As previously mentioned, G-REX was originally developed for rule extraction based on this very idea. More specifically, G-REX uses a fitness function that linearly combines the fidelity towards the opaque model and the comprehensibility of the extracted rule. Fidelity is normally measured on the training data by calculating the ratio of predictions that are similar to the predictions of the opaque model. Equation 45 presents f_{XACC} which is the standard fitness function used when performing rule extraction with G-REX. The fitness function is very similar to the one presented in equation 33, with the exception that fidelity against the opaque model m is used instead of the accuracy on the training set.

$$f_{XACC_p} = \frac{1}{n}\sum_{i=1}^{n}(p_i! = m_i) + P * O_p \tag{45}$$

The following sections present an evaluation of G-REX against each of the six rule extraction criteria presented in section 2.2.6, i.e., *accuracy, comprehensibility, fidelity, consistency, scalability,* and *generality*. Empirical results are gathered from several studies, i.e., (Johansson, König & Niklasson 2003, 2005, 2007, 2008, 2010a), and summarized according to each criteria.

Accuracy

Accuracy measures the ability of extracted representations to make accurate predictions on previously unseen cases and is thus one of the more important criteria. G-REX has mostly been evaluated against different decision tree algorithms, since they are one of the most popular alternative producing comprehensible models. Most decision trees also have representations that are equal or very similar to G-REX's, thus minimizing any bias related to the chosen representation. Table 15 shows the result of a comparison to CART based on four studies that included both algorithms. Johansson, König, Niklasson (2005) and König, Johansson, Niklasson (2006) extracted rules from an ensemble consisting of five data mining techniques; CART, two different MLPs, a RBF neural network, and a PNN, and used four randomly created partitions consisting of 75% training- and 25% test instances. The other two studies, (Johansson et al. 2006a) and

(Johansson, König, Niklasson 2007), used tenfold cross-validation and ensembles consisting of only MLPs.

Even if the approach and evaluation differ slightly between the studies, they are deemed to be similar enough to be presented together. The results are combined by calculating the average accuracy for all the studies including a specific dataset. All but one dataset (Tic-Tac-Toe) are used in at least two studies. Combining the studies in this way facilitates a comparison on a wider range of problems, improves the robustness, and simplifies the analysis.

Table 15 shows that G-REX using f_{XACC} clearly outperforms CART with regard to accuracy, even if the difference is not significant, according to a Wilcoxon signed rank test, at a 95% confidence level, i.e., the critical distance is 73 and the minimum rank is 113. G-REX is however only inferior to CART on 8 of the 23 datasets, which is a strong result even if it is not significant.

G-REX has also been compared to C5.0 by Johansson et al. (2006a & 2006b) and J48 (König et al. 2010) with similar results, i.e., G-REX clearly outperformed J48 and C5.0, but the differences were not statistically significant.

Datasets	f_{XACC}	CART	Diff	R+	R-
Labor	**.900**	**.900**	.000	.5	.5
Vote	**.957**	.956	.001	2	
Wine	**.939**	.937	.002	3	
Iris	**.921**	.915	.005	4	
Credit-g	**.786**	.781	.006	5	
Tic-Tac-Toe	**.835**	.827	.008	6	
Ecoli	.778	**.787**	-.009		7
Zoo	.910	**.920**	-.010		8
Lymph	.786	**.797**	-.011		9
Waveform-5000	.680	**.692**	-.013		1.5
Ionosphere	**.899**	.886	.013	1.5	
Heart-statlog	**.758**	.740	.018	12	
Cmc	**.510**	.491	.019	13.5	
Liver-disorders	**.636**	.617	.019	13.5	
Breast-w	**.964**	.938	.026	15	
Heart-c	**.774**	.746	.028	16	
Hypothyroid	.962	**.995**	-.033		17
Tae	.547	**.581**	-.034		18
Diabetes	**.743**	.705	.038	19	
Credit-a	**.813**	.773	.040	20	
Vehicle	.630	**.671**	-.042		21
Glass	.617	**.667**	-.050		22
Sonar	**.770**	.695	.075	23	
Mean/Total	.787	.783	-	163	113

Table 15 – G-REX Accuracy (CD=73)

Comprehensibility

It should be noted that accuracy should not only be evaluated separately if comprehensibility is a relevant criteria. A larger tree naturally allows a more complex and possibly more accurate tree and hence gives an unfair advantage to techniques producing larger trees. It is however impossible to design a fair experiment, since algorithms handle comprehensibility in different ways. In the following experiments, presented by König, Johansson and Niklasson (2006) and Johansson et al. (2006b), accuracy and comprehensibility are evaluated together, using the default settings for all techniques, to make the comparison as realistic as possible.

Another problem related to evaluating comprehensibility is that the representation language may differ between different techniques, even if the techniques have the same characteristics. Quinlan's (1993 and 2005) C4.5 and C5 do for example allow multiple splits in the same node, while CART only allows binary splits. Furthermore, G-REX can evolve arbitrary representation including Boolean rules and splits with conjunction and disjunction. A simple straightforward way of counting the size of a tree is to calculate the number of primitives, since this is what a human needs to interpret when faced with a tree. It is however not a perfect metric, since the number of primitives required to form the same decision boundaries may vary for different representations. Figure 22 for example does show three programs, created for the diabetes dataset, that form the same decision boundaries with different numbers of primitives due to different representations. The programs are created using the DT, BDT and BR BNFs, i.e., see Table 66, Table 68, and Table 69. The DT program has eleven primitives, while the BTD program has ten and the BR program only seven.

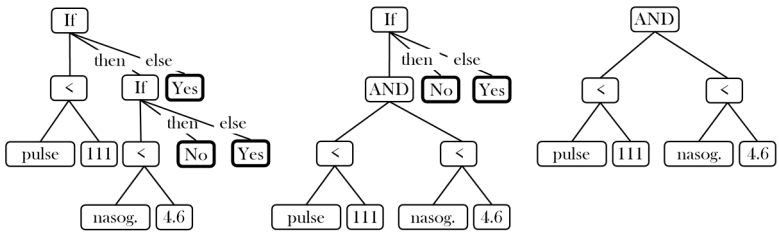

Figure 22 - A rule represented using the DT, BDT, and BR representations

In light of the discussion above, it may seem unfair to allow G-REX to use a Boolean representation for binary datasets, but it could however also be argued that it would be unfair towards G-REX not to use a more effective representation because another technique cannot. Hence, this approach was used in the experiments, performed in König, Johansson and Niklasson (2006), which is presented in Table 16. The last two columns in the table present the size of the G-REX programs and how big they were (in percent) compared to the CART trees, e.g., for the Ecoli dataset, the program extracted by G-REX had an average size of 17, which was 33% of the size of trees produced by CART, which on average had a size of *17/.333 = 51.1*. G-REX extracted rules from an ensemble consisting of five data mining techniques; CART, two different MLPs, a RBF neural network, and a PNN, and used four randomly created partitions consisting of 75% training- and 25% test instances.

Dataset	ACC		Diff	R+	R-	Size	
	f_{XACC}	CART				f_{XACC}	f_{XACC}/CART
Ecoli	.786	**.787**	-.001		1	17	.333
Heart-c	.761	**.763**	-.002		2.5	14.2	.582
Vote	**.958**	.956	.002	2.5		11.8	.123
Liver-disorders	**.625**	.621	.004	4.5		8.6	.818
Ionosphere	.895	**.899**	-.004		4.5	14.2	.374
Iris	**.921**	.915	.006	6		12	1.22
Waveform-5000	.683	**.692**	-.009		7	22	.318
Cmc	**.505**	.491	.014	8		23	.440
Heart-statlog	**.759**	.740	.019	9		11	.458
Breast-w	**.964**	.941	.023	10		15	.159
Wine	.933	**.965**	-.032		11	17	.176
Hypothyroid	.962	**.995**	-.033		12	7.3	.380
Glass	.611	**.651**	-.040		13.5	18	.305
Vehicle	.631	**.671**	-.040		13.5	32	.477
Credit-a	**.870**	.802	.068	15		14.5	.438
Diabetes	**.743**	.670	.073	16		15.8	.101
Mean	.788	.785	-	71.0	65.0	15.8	.316

Table 16 – G-REX Comprehensibility 1 (CD=30)

When evaluated on accuracy, the results for G-REX are quite modest, since it performs evenly, with eight wins for each technique. G-REX achieves a somewhat higher average accuracy, while CART receives a slightly better ranking in a Wilcoxon signed rank test. Obviously, there is no significant difference with regard to accuracy. However, with regard to comprehensibility, evaluated as the program size, G-REX produces trees that are significantly smaller than the CART trees. The programs produced by G-REX are smaller on all but the Iris dataset and are on average less than a third of the size of a CART tree. Significantly shorter rules explain why G-REX only performs evenly with CART and also demonstrates why accuracy and compressibility should be evaluated together. Of two equally accurate trees, most decision makers would naturally prefer a significantly smaller tree.

Of course, some of the difference in size can be contributed to G-REX's Boolean representation for binary problems, but another study conducted by Johansson et al. (2006b) confirms the result without the advantage of Boolean representation. Johansson et al. used the BTD representation, e.g., the middle program in Figure 22, and calculated the size as the number of splits in a program. The BTD representation differs from the DT representation used by CART, e.g., the left program in the same figure, in that conjunctions are allowed in the BTD

representation. However, since only the splits using the < > = operators are calculated, both trees will receive a size of two, which is fair, since they form the same decision boundaries.

Table 17 below shows the accuracy and size (calculated as the number of splits) for G-REX and CART for experiments performed by Johansson et al. (2006b). The G-REX trees were extracted from an ensemble consisting of 20 MLPs trained on randomly selected subsets covering 70% of the training data, using a randomly selected number of hidden units. Here, the CART trees were produced with the Clementine data mining workbench (2006 version) using *simple* mode, which sets all parameters automatically, based on dataset properties.

Dataset	Acc		Diff	R+	R-	Size	
	f_{XACC}	CART				f_{XACC}	f_{XACC} /CART
Iris	**.967**	**.967**	.000	.5	.5	3.2	.653
Credit-a	.849	**.852**	-.003		2	6.6	.246
Credit-g	**.723**	.719	.004	3		7.1	.246
Diabetes	**.746**	.739	.007	4		5	.269
Glass	.643	**.652**	-.009		5	5.8	.276
Tic-tac-toe	**.835**	.822	.013	6		13.5	.412
Lymph	.786	**.800**	-.014		7	6.3	.453
Labor	**.900**	.883	.017	8		5.4	1.00
Cmc	**.565**	.547	.018	9		11.1	.529
Liver-disorders	**.650**	.629	.021	10		7.8	.363
Breast-w	**.952**	.929	.023	11		6.4	.889
Tae	.500	**.527**	-.027		12	12.4	.216
Zoo	.900	**.930**	-.030		13	7.5	.926
Ionosphere	**.903**	.857	.046	14		5.2	.650
Wine	**.947**	.900	.047	15		5.3	.898
Heart-c	**.793**	.737	.056	16		7.3	.397
Sonar	**.770**	.700	.070	17		7.2	.600
Mean	.790	.776	.014	113.5	39.5	7.2	.531

Table 17 – G-REX Comprehensibility 2 (CD=35)

Here, G-REX performs much better in terms of accuracy, with twelve wins against five for CART. Another Wilcoxon test shows that G-REX is very close to being significantly more accurate than CART, i.e., the minimum rank difference is 39.5 and 35 is a significant difference. This is of course a much better result for G-REX, especially since the G-REX programs are still significantly smaller. The fact that G-REX clearly outperforms CART with regard to accuracy in this study can possibly

be explained by the difference in size being smaller. Programs extracted by G-REX are however still about half the size of the CART trees.

G-REX was also compared to C5.0 with similar results in the same study, i.e., G-REX clearly outperformed C5.0, with regard to accuracy, while producing significantly smaller programs. The difference in size was however smaller, with a 15% advantage for G-GREX.

Finally, it should be noted that G-REX uses a novel simplification technique to remove introns in both studies presented above. Hence, the difference for a standard GP implementation may vary, since the size of the extracted programs will in general be larger. The benefit of this novel simplification technique used in G-REX is evaluated in detail in section 7.1.

Fidelity

Fidelity is the extent to which extracted representations accurately model the opaque model from which they were extracted. Fidelity is a crucial criterion for rule extraction, since the idea is that the extracted rule should be a comprehensible version of the opaque model. Even if fidelity is crucial, it should however be evaluated together with accuracy, since the main goal for all PMs is models that are accurate on novel data. Table 18 presents the results of the ensemble accuracy, the train and test fidelity for G-REX, as well as the accuracy for G-REX and CART. The results are taken from (Johansson et. al. 2006a) where rules were extracted from an ensemble consisting of twenty randomly initiated MLPs. All the results are the average accuracy over ten folds.

Overall, G-REX achieves a fidelity of 86.6%, which is a high result, even if there is no specific predefined level of a sufficient fidelity. In practice, a decision maker would set a minimum fidelity level based on his own subjective opinion and the problem at hand. G-REX also performs well with regard to accuracy, when compared to CART and the ensemble from which the programs are extracted. G-REX clearly outperforms CART and has on average only a 3.8% lower accuracy than the opaque ensemble.

An interesting result is that even if the predictions of the extracted programs differ from the predictions of the opaque model on 15.7% of the test instances, the difference in accuracy is only 3.8%. If the fiddle predictions were evenly distributed over the correct and incorrect predictions made by the opaque model, the accuracy of the extracted programs would be *.843*.831+(1-.843)*(1-.831) = .727*. The average accuracy of the extracted programs is, however, .793, (6.6% higher), which means that the non-fiddle predictions are done more often when the opaque model makes incorrect predictions.

Dataset	Ensemble	*f*xacc			CART	*f*xacc vs CART		
	ACC	Train Fid	Test Fid	ACC	ACC	Diff	R+	R-
Labor	.917	.998	.850	**.900**	**.900**	0	.5	.5
Diabetes	.766	.840	.836	**.746**	.741	.005	2	
Credit-g	.848	.895	.923	**.849**	.842	.007	3.5	
Credit-a	.742	.767	.815	**.723**	.716	.007	3.5	
Lymph	.850	.855	.793	.786	**.793**	-.007		5
Tic-Tac-Toe	.911	.847	.874	**.835**	.827	.008	6	
Zoo	.970	.923	.920	.900	**.910**	-.01		7
Ionosphere	.934	.944	.906	**.903**	.889	.014	8	
Breast-w	.967	.977	.977	**.952**	.933	.019	9	
Liver-disorders	.685	.791	.753	**.650**	.597	.053	10	
Glass	.690	.701	.729	.643	**.700**	-.057		11
Wine	.988	.968	.947	**.947**	.888	.059	12	
Tae	.573	.782	.727	.500	**.560**	-.06		13
Heart-c	.800	.869	.820	**.793**	.720	.073	14	
Sonar	.825	.836	.775	**.770**	.690	.08	15	
Mean	.831	.866	.843	.793	.780	-	83.5	36.5

Table 18 – G-REX fidelity (CD=25)

A closer examination shows that the expected accuracy (for evenly distributed fidelity) is significantly lower than the actual accuracy, since the expected accuracy is lower for all but the *Tae* dataset, see Table 19.

A possible explanation for this result is that since the extracted model is comprehensible, it is simpler and cannot represent as complex decision boundaries as the opaque model. Hence, it will not be able to separate atypical instances. Since atypical instances per definition have fewer training instances, they are more likely not to be representative for the dataset and lead to misclassification of new instances. It is therefore not certain that a small increase in fidelity would lead to a higher accuracy for the extracted model and, consequently, it could be argued that G-REX achieves a sufficiently high fidelity. It should however be noted that if the aim of the rule extraction is to explain the predictions of the opaque model, it is of course always better to be more fiddle.

Dataset	e-Acc	Acc	Diff	R+	R-
Zoo	.895	.900	.005	1	
Breast-w	.946	.952	.006	2	
Wine	.936	.947	.011	3	
Tic-Tac-Toe	.807	.835	.028	4	
Tae	.533	.500	-.033		5
Ionosphere	.852	.903	.051	6	
Credit-g	.794	.849	.055	7	
Glass	.587	.643	.056	8	
Liver-disorders	.594	.650	.056	9	
Diabetes	.679	.746	.067	10	
Credit-a	.652	.723	.071	11	
Lymph	.705	.786	.081	12	
Sonar	.679	.770	.091	13	
Heart-c	.692	.793	.101	14	
Labor	.792	.900	.108	15	
Mean/Total	.727	.793	.050	115	5

Table 19 – Expected vs. real accuracy of extracted model (CD=25)

Consistency

If users are to accept and rely on any technique that automatically produces predictive models, it must be consistent. There is of course more than one way to measure consistency of the produced models, where the two most straightforward are the similarity of splits and the similarity of predictions. Johansson, König and Niklasson (2007) used the latter approach to evaluate the consistency of rules extracted by G-REX during different runs. Table 20 shows the consistency on both the training and the test sets for the dataset and the corresponding accuracy for G-REX and CART.

The consistency varies slightly between datasets, but overall G-REX is quite consistent on about 90% of both the training and the test instances. As usual, G-REX performs better than CART and there is no obvious correlation between the consistency and the accuracy.

Dataset	Consistency		ACC	
	TRAIN	TEST	f_{XACC}	CART
Breast-w	.986	.971	**.977**	.937
Diabets	.908	.893	**.746**	.739
Heart-c	.893	.845	**.797**	.737
Liver-disorders	.805	.751	**.650**	.629
Sonar	.910	.898	**.770**	.700
Zoo	.897	.903	.920	**.930**
Mean	.900	.877	**.810**	.779

Table 20 – G-REX Consistency

Generality and Scalability

Due to the pedagogical approach taken by G-REX, the generality criterion is clearly fulfilled, since the GP optimization is independent of the opaque model. This is also demonstrated in practice in (Johansson, König, Niklasson 2005) and (König, Johansson, Niklasson 2006), where rules were successfully extracted from a heterogeneous ensemble consisting of five different techniques.

The scalability of G-REX has also been demonstrated in several studies where G-REX has been applied to a wide range of datasets. G-REX has for example been successfully applied to the *Waveform-5000* dataset with 5000 instances, 21 attributes, and 3 classes; the *Zoo* dataset with 101 instances, 17 attributes, and 7 classes; the *Sonar* dataset with 208 instances, 60 attributes, and 2 classes, as well as several regression datasets. All GP implementations are of course computationally expensive, compared to most traditional data mining techniques, but it was possible to handle all the tested datasets within reasonable execution times.

Evaluation against other rule extraction techniques

A final criterion for a new rule extraction technique is of course that it should perform comparably or better than existing techniques. In (Johansson, König and Niklasson 2003), G-REX was evaluated against Craven and Shavliks' well-known rule extraction technique, TREPAN. Both G-REX and TREPAN extracted rules from a MLP, since TREPAN can only extract rules from a single network. The datasets used in the experiments concerned predicting the advertising impact for seven different car brands. The aim was to predict whether the advertisements carried out a certain week resulted in a high or low impact. Table 21 above shows that even if the difference in average accuracy was only 2%, G-REX outperformed TREPAN on five of seven datasets. It should be noted that this is a rather strong

result for G-REX, since TREPAN was especially designed to extract rules from MLPs.

Due to the small number of datasets, there was of course no significant difference in accuracy between G-REX and TREPAN. In a latter study, Johansson (2007) did, however, show that G-REX was in fact significantly better than both TREPAN and RX, another well-known rule extraction algorithm.

Dataset	MLP	f_{ACC}	TREPAN
BMW	.86	.82	**.88**
Ford	.92	.84	**.88**
Opel	.84	**.90**	.84
Saab	.96	**.96**	.86
Toyota	.92	**.86**	**.86**
Volvo	.94	**.96**	.92
VW	.94	**.88**	.82
Mean	.91	.89	.87

Table 21 – Comparing Rule Extraction Techniques

Rule Extraction Optimizing brier score

Another advantage that stems from GP is that it is possible to choose an alternative optimization criterion. Normally, G-REX optimizes the fidelity as a 0/1 metric, i.e., either the prediction is fiddle or not. However, most predictive techniques can produce probability estimates for their prediction and, since the certainty of prediction varies, probability estimates could be a more informed rule extraction target. The idea is that it should not be as bad for the extracted model to make a different prediction when the opaque model is uncertain.

Johansson, König and Niklasson (2010) evaluated the approach of using probability estimates by optimizing the generalized brier score by incorporating it in the fitness function of G-REX, according to equation 46. BRI is the generalized brier score (see equation 14) calculated towards the probability estimates of the opaque model.

$$f_{XBRI_p} = 1 - BRI_p + P * O_p \tag{46}$$

However, to optimize the brier score, G-REX had to produce probabilities itself. Two strategies were evaluated: the first strategy ($f_{XBRI\text{-}i}$) is a simple method based on the relative frequencies of the classes of the training instances reaching each leaf. Consequently, this strategy tries to minimize the difference in probability estimates between the extracted and the opaque model, at an instance-for-instance level. The

second strategy ($f_{XBRI\text{-}l}$), on the other hand, tries to optimize the probability estimate for each instance falling in a certain leaf, by calculating the average probability estimate from the opaque model of these instances. Johansson, König and Niklasson (2010) extracted rules from both a random forest and a MLP ensemble using the G-REX setting presented in Table 22. Since the results were very similar for the rules extracted from the random forest and the MLP ensemble, Table 23 - Table 25 only present the results related to the random forest.

Number of generations	100
Population size	1000
Crossover probability	.8
Mutation probability	.001
Creation type	Ramped half -and -half
Selection type	Roulette wheel
Maximum creation depth	10
Parsimony Pressure (P)	.01
Batch size	10

Table 22 – : G-REX settings for rule extraction using brier score

Table 23 shows the accuracy for the random forest (RF),i.e., the opaque model, J48 as implemented in WEKA, the original G-REX using 0/1 fidelity, i.e., f_{XACC} and the two novel G-REX strategies ($f_{XBRI\text{-}i}$ and $f_{XBRI\text{-}l}$) optimizing the brier score. All three G-REX versions extracted rules from the same random forest. The results are the averages over ten folds.

As in the previous evaluation against CART, f_{XACC}, the original G-REX, clearly achieves a higher accuracy than J48, with only 10 losses on the 26 datasets used in the experiment. A Friedman test followed by a Nemenyi post-hoc test does however again confirm that the difference is not significant, i.e., the critical distance in rank when comparing four classifiers (for α=.05) is .92.

A difference that is statistically certain is that both $f_{XBRI\text{-}i}$ and $f_{XBRI\text{-}l}$ outperform J48 significantly. Both strategies only have five losses against J48. An interesting observation is that four of these five losses occur on the same datasets, i.e., *Tae, Credit-G, Colic,* and the *Ionosphere* datasets.

Datasets	RF	J48	f_{XACC}	$f_{XBRI\text{-}i}$	$f_{XBRI\text{-}l}$
Balance-scale	.813	.787	.790	.790	**.798**
Breast-cancer	.678	.699	.727	**.731**	.717
Breast-w	.966	.956	**.961**	.957	.956
Cmc	.512	.506	**.547**	.532	.534
Colic	.840	**.856**	.845	.840	.848
Credit-a	.864	.855	.841	.855	**.859**
Credit-g	.769	**.730**	.724	.718	.727
Diabetes	.760	.733	.734	**.738**	.736
Ecoli	.816	.786	.807	**.813**	.789
Glass	.785	.668	.659	**.673**	**.673**
Haberman	.647	.739	**.761**	.745	.739
Heart-h	.813	.779	**.810**	.789	.793
Heart-l	.828	.759	.776	.779	**.782**
Heart-statlog	.829	.770	.759	**.774**	**.774**
Hepatitis	.839	.807	.813	.819	**.826**
Ionosphere	.926	**.900**	.846	.855	.866
Iris	.947	.953	.960	**.967**	**.967**
Labor	.930	.737	.825	**.877**	.842
Lcancer	.531	.406	**.438**	**.438**	**.438**
Liver-disorders	.739	.620	.620	.632	**.652**
Lymph	.845	.730	**.824**	.777	.791
Sonar	.846	.755	.736	**.760**	.726
Tae	.629	.550	.550	.523	.523
Vote	.956	.956	.949	**.963**	**.963**
Wine	.972	.916	.910	**.933**	.916
Zoo	.951	.931	.911	.911	**.941**
Mean	.809	.765	.774	**.776**	**.776**
Mean rank		3.10	2.71	2.12	2.08

Table 23 – Rule extraction using brier score: Accuracy

When comparing the techniques on AUC, see Table 24, the results are even stronger. f_{XACC} only loses against J48 on nine datasets, $f_{XBRI\text{-}i}$ on three and $f_{XBRI\text{-}l}$ on two. Here, $f_{XBRI\text{-}i}$ and $f_{XBRI\text{-}l}$ achieve an average rank that is significantly lower than both J48 and f_{XACC} using 0/1 fidelity.

It is also interesting to note that both f_{XACC} and $f_{XBRI\text{-}i}$ calculate their probability estimates in the same way, i.e., as the relative class frequency in a leaf, which means that the difference in AUC can only be contributed to the choice of optimization criterion.

Datasets	RF	J48	f_{XACC}	$f_{XBRI\text{-}i}$	$f_{XBRI\text{-}l}$
Balance-scale	.947	.834	.853	.880	**.890**
Breast-cancer	.621	.596	.557	**.694**	.612
Breast-w	.99	.956	.967	.969	**.970**
Cmc	.674	.654	.679	.696	**.702**
Colic	.886	.829	.813	.829	**.840**
Credit-a	.932	.871	.888	**.904**	.903
Credit-g	.799	.667	.627	**.728**	.718
Diabetes	.82	.726	.713	**.779**	.766
Ecoli	.963	.892	.912	**.930**	**.930**
Glass	.929	.775	.814	**.832**	**.823**
Haberman	.655	.538	.654	.673	**.680**
Heart-c	.894	.763	.803	.795	**.829**
Heart-h	.876	.740	.779	.834	**.844**
Heart-statlog	.888	.751	.755	.826	**.838**
Hepatitis	.858	.609	.767	.721	**.820**
Ionosphere	.981	**.871**	.822	.831	**.871**
Iris	.995	.968	.968	.972	**.982**
Labor	.974	.720	.819	.859	**.882**
Lcancer	.648	.489	.565	.566	**.649**
Liver-disorders	.767	.631	.565	.612	**.661**
Lymph	.925	.792	.831	.836	**.883**
Sonar	.944	.757	.757	.792	**.829**
Tae	.765	.698	**.708**	.707	.688
Vote	.99	**.977**	.921	.951	.967
Wine	.999	.937	.922	.951	**.968**
Zoo	.998	.976	.949	**.979**	.996
Mean	.874	.770	.785	.813	**.829**
Mean rank		3.36	3.24	2.00	1.40

Table 24 – Rule extraction using brier score: AUC

Regarding the size of the extracted programs and J48, presented in Table 25, it should be noted that multi splits were used by J48 and that these were given the score of one, the same as a single split. Hence, J48 was given a small advantage in terms of the size of the calculation, compared to the different G-REX strategies that did not use multiple splits.

Datasets	J48	f_{XACC}	$f_{XBRI\text{-}i}$	$f_{XBRI\text{-}l}$
Balance-scale	76	51	**45**	46
Breast-cancer	**16**	36	21	32
Breast-w	20	**13**	16	15
Cmc	245	29	**17**	22
Colic	19	25	18	**15**
Credit-a	24	15	**11**	**11**
Credit-g	98	21	**11**	13
Diabetes	38	17	**16**	21
Ecoli	30	33	24	**14**
Glass	42	31	**19**	27
Haberman	**2**	23	21	17
Heart-c	35	42	**23**	32
Heart-h	14	20	**11**	16
Heart-statlog	34	32	32	**31**
Hepatitis	**12**	22	23	25
Ionosphere	22	21	**20**	29
Iris	**8**	11	13	13
Labor	**7**	14	11	11
Lcancer	33	24	**17**	18
Liver-disorders	**14**	23	26	33
Lymph	25	27	**24**	**24**
Sonar	**23**	31	28	32
Tae	40	35	**27**	45
Vote	**9**	15	14	15
Wine	**10**	22	18	27
Zoo	**14**	23	16	18
Mean	35	25	**20**	23

Table 25 – Rule extraction using brier score: Size

An important result of the comparison with regard to size is that both $f_{XBRI\text{-}i}$ and $f_{XBRI\text{-}l}$ that optimized the brier score actually produce slightly smaller trees than both J48 and f_{XACC}. Hence, the significant increase in accuracy and AUC was achieved without sacrificing any comprehensibility.

Otherwise, the main result with regard to comprehensibility is that in general all the techniques produce rather comprehensible models. J48 does however create very large trees, which would probably be considered incomprehensible by most decision makers, for the *Cmc, Credit-G,* and the *Balance-scale* datasets.

Summary

It is clear that G-REX is well suited for rule extraction. G-REX extracts programs with similar or higher accuracy, while still being considerably shorter than typical decision tree techniques. The key features facilitating this result are the ability to weigh accuracy against comprehensibility in the fitness function and a global non-greedy optimization; both features are only possible due to the use of GP. Furthermore, the pedagogical rule extraction approach used makes G-REX very general, since it is totally independent of the opaque model. G-REX is also quite consistent, fiddle and scales reasonably well. In addition, G-REX's GP based approach also facilitates optimization of probability estimates, thus further extending its rule extraction capabilities. When optimizing the brier score, for example, G-REX extracts programs that have a significantly higher ACC and AUC than both the original G-REX optimizing 0/1 fidelity and the decision tree technique J48.

All in all, G-REX must be considered an excellent rule extraction technique, since it fulfills all the criteria for rule extraction techniques while outperforming existing rule extraction techniques, i.e., TREPAN and RX.

5.1.3 Advantages of representation free optimization

Normally, overfitting is considered to be caused by an unnecessarily complex model. Domingos (1999), however, claims that a model does not have to be complex to suffer from overfitting. Instead, Domingos argues that overfitting occurs when a model happens to fit the data due to chance rather than because it accurately models the underlying relationships. Hence, the search of the best model should be restricted as much as possible. A simple and direct way of restringing the search space is incorporating as much domain knowledge as possible in the search, e.g., by defining illegal combination of attributes. As described in section 2.5.1, this can easily be done in GP, if constrained syntactic structures are used to define the representation. Another way to include domain knowledge is of course to ensure that only functions that are appropriate to the problem and comprehensible to the decision maker are used. Finally, there are also numerous different types of representations that could be used: Boolean rules, If-else rules, discriminant functions, decision lists, fuzzy rules, etc., see section 2.5 for more details.

GP's flexibility in representation is not only important in order to restrict the search space, but also to make the models more comprehensible to the decision maker. Since comprehensibility is a subjective property, it is not possible to define an optimal representation. A representation must be tailored to each person, if maximum comprehensibility is to be achieved. Naturally, this is only possible if a technique, like GP, is independent of the representation.

The most common representation language for decision tree techniques, such as CART and C4.5, is if-else rules. The if-statements can be seen as a series of tests, between an independent variable and a constant, dividing the instances of a dataset into subgroups. Each subgroup is associated with the value of a certain terminal. Neither C4.5 nor CART allows representation free optimization. If the representation were to change even slightly, like allowing comparisons between independent variables, the algorithm would have to be modified. Hence, the choice of a certain decision tree technique is also implicitly a choice of representation. This is contradictory to the previous discussion which argues that the representation should be tailored to the problem at hand and the decision maker's preference. It would therefore make more sense to use a technique like GP, which optimizes a candidate solution independent of its representation. The following sections demonstrate various representations and how they may benefit the accuracy or comprehensibility of the solutions.

Normally, a GP representation is given by its function set and terminal set. However, if constrained syntactic structures or strongly typed GP are considered, a BNF is also required to define the representation, since not all combinations of the terminals and functions may be allowed. Hence, each representation is presented with an example tree. The function set, terminal set and BNF is given in the appendix. G-REX uses a combination of constrained syntactic and strongly typed GP. The advantage is that the user has full control of the structure of the programs but does not have to bother about dataset specific details, such as defining how all independent variables should be handled. Only example programs are given in the following section, but the BNFs can be found in the appendix section 11.2.

Representations for regression

In the article "*Genetic Programming – A Tool for Flexible Rule Extraction*", by König, Johansson and Niklasson (2007) presented in detail in section 6.1, three different regression representations were evaluated on eight regression datasets. The first representation, *extended decision tree* (EDT), see Table 67, is a slight extension of the typical simple *decision tree* representation (DT), see Table 66, used by CART. The only difference is that CART's syntax only allows comparisons between a variable and a constant value in a condition, while EDT also allows comparisons between two variables. Note that doing this would be impossible using CART, without changes to the original algorithm. In both DT and EDT a terminal is always defined as a constant, which means that all instances (known and novel) belonging to a certain terminal will be predicted to have the same value.

An example program from the study is presented in Figure 23. The program was created for *Auto-price* dataset, another UCI-dataset, and concerns the prediction of the price of 156 different cars, based on 16 attributes.

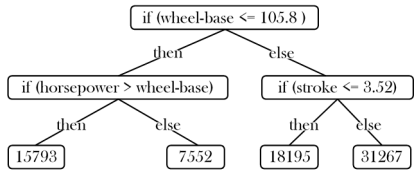

Figure 23 - EDT program for the Auto-price dataset

The lower left split is typical for the EDT representation, since it is based on a comparison between two attributes, *horsepower* and *wheel-base*. Cars which have greater horsepower in comparison to the *wheel-base,* i.e. the distance between the two wheel axes, are sold at a higher price. Put in another way, cars with an engine that is powerful relative to its size are more expensive. Naturally, this type of relationship would be much harder to find without comparison between attributes.

EDT is one of the simplest representations that can be used for regression. Another popular but slightly more complex representation is to use multiple linear regressions (MREG), which consists of a set of linear regressions (see section 2.2.1). Figure 24 shows a model created using MREG for the same dataset. As is typical for most MREG models, all variables are used and contribute to the final prediction.

```
-57183   +
-131.56  *  symboling +
10.889   *  losses +
131.4 *  wheelBase +
-95.736  *  length +
828.3 *  width +
16.846   *  height +
6.1278   *  weight +
47.855   *  engineSize +
-1687.5  *  bore +
-1815.7  *  stroke +
103.57   *  compressionRatio +
15.65 *  horsepower +
0.82669  *  peakRpm +
-47.336  *  cityMpg +
28.109   *  highwayMpg
```

Figure 24 – MREG model for the Auto-price dataset

A well-known downside of MREG is that it handles categorical variables poorly and that it cannot handle more complex nonlinear problems (Dunham 2003). For

example, all variables do not necessarily contribute as much to the prediction for all instances. One solution to this problem could be to first divide the dataset into smaller subsets and then create a linear regression for each subset. Figure 25 shows an example program of GP-LM BNF (see Table 75), another representation used in the same study. Here, linear regressions, (in the form of $y=wx+m$, where w and m are constants calculated using the least square method), are used as terminals in a typical DT representation. This means that instances reaching the same leaf may be predicted with different values, in contrast to the DT representation that predicts a single value in each leaf.

An example program created using G-REX for the *Auto-price* dataset can be seen below. It is apparent that the car curb *weight* has different importance for instances reaching different terminals, since each linear regression has different constants for the importance of *weight.*

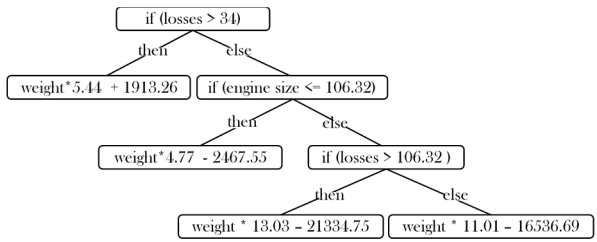

Figure 25 – GP-LM program for the Auto-price dataset

Of course, the representation would be even more powerful with a MREG in each terminal, but this would also make the program less comprehensible.

Representation for classification

Figure 26 shows a DT tree for the Toyota IM dataset which contains Toyota commercials' data and whether they had a high or low effect. TV0 is the amount of money spent on commercials the week of the measurement, TV1 is the investment for the previous week. The program was created using G-REX, but could as well have been produced by CART or C4.5.

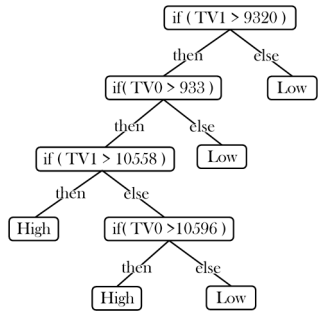

Figure 26 – DT program for the Toyota IM dataset

The program is identical to the one in Figure 27 below, which was also evolved using G-REX, but with a *Boolean* representation, in a study conducted by Johansson, König, and Niklasson (2004). The *Boolean* representation is less common than the if-else representation, since it only works for problems with two classes. However, the program actually has fewer primitives than the equivalent *if-else* and neither CART nor C4.5 can produce trees with this representation. Of course, comprehensibility is subjective to the individual; therefore, the choice of representation should be up to the decision maker.

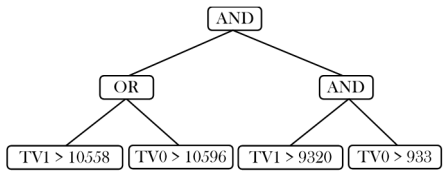

Figure 27 – Boolean program for the Toyota IM dataset

Another program evolved with EDT is presented in Figure 28. Again, the only difference is that comparison among variables is allowed, which is not the case for most decision tree techniques.

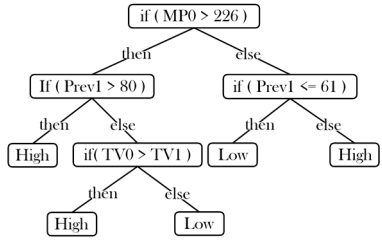

Figure 28 – EDT program for the Toyota IM dataset

The program is fairly easy to comprehend and has an interesting interpretation, especially the left part of the tree. If the first two conditions to the left are true, the interpretation can be that a small investment in *MP0* (morning press) is enough to achieve a high impact, if the commercials of the previous week had a high impact (*Prev1 > 80* signifies a high impact the previous week). If, on the other hand, the impact of the previous week's commercials was low, the investments in TV must increase, compared to the previous week, to achieve a high impact. This rule makes sense, since commercials tend to linger in our memory for a time and because TV-commercials are generally very powerful.

Regardless of how the program is interpreted, it is important to note that since normal decision tree techniques do not allow comparisons among variables, they would not be able to find this type of relationship. Naturally, this type of relationship does not occur in all problems, but a decision maker should have the possibility to include them when appropriate.

Of course, it is also possible to include complex conditions using Boolean as is in the DTB tree in Figure 29. This tree uses *And* to combine conditions in the root node, indicating that the impact will be *High* if both the investment in TV and morning press is high.

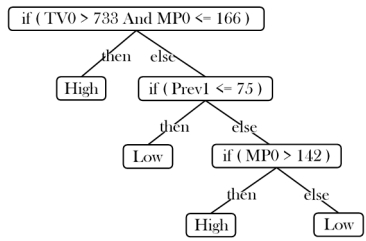

Figure 29 – DTB program for the Toyota IM dataset

Another slight variation of the standard DT representation is *decision lists* (DL), which are normal DT programs with the exception that each condition (except the last) must have at least one terminal. Hence, the program can be seen as an ordered rule set which some practitioners consider to be more comprehensible than decision trees. The program presented in Figure 29 was evolved using the DTB representation, however, by chance it also classifies as a decision list since the first two splits have one terminal each. There are several decision list techniques, such as IREP (Fürnkranz & Widmer 1994) and RIPPER (Cohen 1995), but they are only defined for a certain representation and cannot change representation without changes to the algorithm. However, decision lists can easily be evolved using GP, see for example (Sönströd, Johansson, König & Niklasson 2008), with a small change to the DT BNF, i.e., see Table 70.

Several slightly different representations based on fuzzy logic (Zadeh 1965) have been proposed and optimized using GP, for example, see (Edmonds, Burkhardt & Adjei. 1995) or (J Eggermont 2002). Fuzzy logic has become popular since it can express the vagueness of the real world and because fuzzy rules are close to how people tend to express rules in natural language. Yet another representation was used in (Johansson, König, & Niklasson 2004); first two membership functions, *high* and *low,* were created for each variable, according to the diagram in Figure 30. The data was then fuzzified using the 20-percentile and the 80-percentile for each variable, as values for constants *a* and *b.*

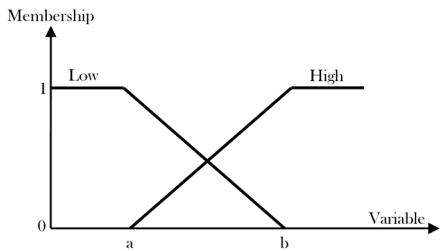

Figure 30 – Fuzzy membership function for the Toyota IM dataset

Finally, the operators *AND*, *OR*, *VERY*, and *RATHER* were defined according to the definitions below:

$$\begin{aligned} \mu_A(x)\ AND\ \mu_B(x) &= min\{\mu_A(x), \mu_B(x)\} \\ \mu_A(x)\ OR\ \mu_B(x) &= max\{\mu_A(x), \mu_B(x)\} \\ VERY(\mu_A(x)) &= \mu_A(x)^2 \\ RATHER(\mu_A(x)) &= \sqrt{\mu_A(x)} \end{aligned} \tag{47}$$

G-REX was again used to evolve programs which turned out to be as accurate, or even more accurate, than the other representation, while being surprisingly short. Figure 31 below shows one of the short fuzzy programs evolved for the Toyota IM dataset using the FUZ BNF presented in Table 72.

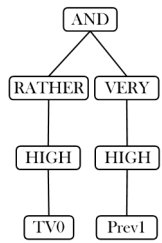

Figure 31 – Fuzzy program for the Toyota IM dataset

The program's easy interpretation indicates that a high impact is obtained by a rather considerable investment in TV, if the commercials of the previous week had a very high impact. At an elevated level, interpreting the program is trivial and gives a general idea of the underlying relationships, i.e., TV commercials are important. However, if actual resources were invested on the basis of the program, the fuzzy values would have to be transformed into crisp values by reversing the fuzzification process. Again it is up to the decision maker to judge which representation suits him and the problem best.

Empirical evaluation: Using a flexible representation to increase k-NN performance

The section above has provided several examples of representations that are more or less suitable in different situations. A certain representation may be favorable either because a certain type of condition may affect the predictive performance or

because it may simply make more sense to a decision maker. However, the examples have mainly been discussed on the basis of common sense, hence, the following section provides an empirical evaluation of whether a specialized representation optimized using GP could increase the predictive performance compared to a standard representation.

More precisely, a novel instance-based learner proposed by *Johansson, König, and Niklasson (2008), based on GP and a three-tailored kNN representation, is evaluated using G-REX. The aim is to counter the inherent drawbacks of standard kNN (described in section 2.2.4), i.e., that it is extremely sensitive to the *k*-value and restricted to a single *k*-value for the whole dataset. Note that it would be hard to optimize these kNN representations in a traditional mathematical way, but with GP it is just a matter of defining the representations.

Method

The overall idea of the proposed technique called G-kNN, is to counter the drawbacks of standard kNN, by evolving decision trees, where the leaf nodes use some version of kNN. Three different versions of kNN leaves (*global*, *local* and *mixed*) are evaluated in the experiments.

In *global* G-kNN, a leaf only specifies the *k* that should be used to classify instances reaching the leaf. It does not considered to which leaf those *k* neighbors belong. Figure 32, show an example where some G-kNN tree has divided a set of instances into for leaves, which each has its own *k* value. A test instance (black) reach a leaf with, *k*=5, and is classified using the five closest instances (gray). Consequently, *Global* G-kNN optimizes the separation in regions, and the *k*-values for each region.

Figure 33 depicts a short *global* G-kNN program, evolved by G-REX (using the KN presented in Table 73) for the Toyota IM dataset, that clearly demonstrates another benefit of this representation. First, the program provides some insight into the underlying relationships of the dataset. Morning press (MP1) is clearly an important attribute, since it is used in two of three splits. In cases with a very high MP1, i.e., the lower left leaf, only the three nearest neighbors are used, while more neighbors are used for weeks with lower investments in MP. Looking at the data this makes sense, since only 14 of 150 weeks have an investment in MP1 higher than 1068.8 and it is therefore more appropriate to use a low *k*-value. A traditional kNN approach would enforce a single *k* value for all instances and would thus probably reduce the accuracy for weeks with a high MP1.

---

*König and Johansson are equal contributors

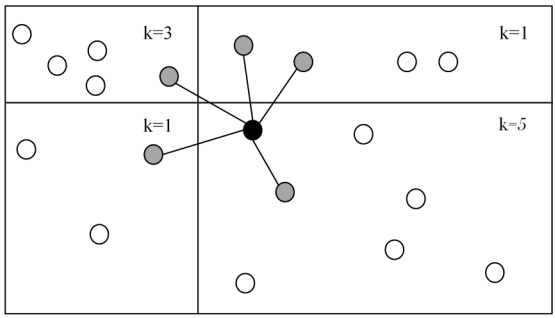

Figure 32 - Selection of five neighbors using global G-kNN

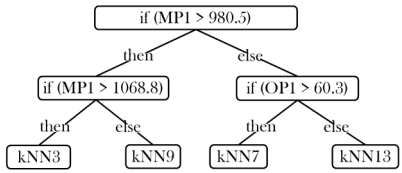

Figure 33 – Sample global G-kNN program for the Toyota IM dataset

In *local* G-kNN leaves, the *k* value is specified exactly like in *global* G-kNN nodes, but here only instances in the same leaf are considered when selecting the nearest neighbors. Figure 34 below show the same example as above, but with a local G-kNN node. Here, the test instance is classified using the five closest instances in the same leaf.

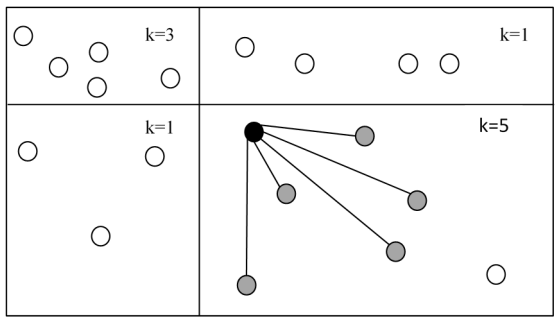

Figure 34 - Selection of five neighbors using local G-kNN

The third variant, called *mixed*, allows both *local* and *global* G-kNN leaf nodes in the same tree. Figure 35 below show a sample mixed G-kNN tree from the

Cancer-w dataset evolved using KN bnf presented in Table 73. It should be noted that the strategy (i.e., *global*, *local* or *mixed*) is simply determined by including different kNN node functions in the grammar. As seen in the figure, G-kNN here uses both local and global kNN, depending on the different regions.

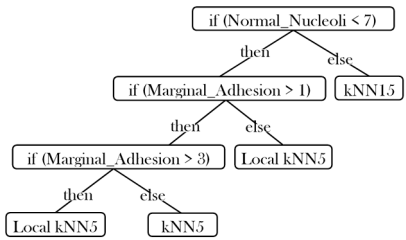

Figure 35 - Sample mixed G-kNN program for the Cancer-w dataset

To summarize, the global technique mostly resembles standard kNN, but locally optimized k-values are used instead of just one global k. Local G-kNN is more similar to decision trees, but the classification of a test instance is based on neighboring instances in the specific leaf. The mixed strategy, finally, permits a locally optimized combination of both the local and global strategy.

A standard accuracy-based fitness function, i.e., f_{ACC} as defined in equation 33, was used to evolve the programs and since kNN in itself is quite powerful, but also rather computationally intense, small populations and few generations were deemed to be sufficient. The GP settings used by G-REX are found in Table 26.

Number of generations	50
Population size	100
Crossover probability	.8
Mutation probability	.001
Creation type	Ramped half –and- half
Selection type	Roulette wheel
Maximum creation depth	6
Parsimony Pressure (P)	.001
Persistence	20
Batch size	10

Table 26 – G-REX settings for optimization of all G-kNN strategies

In the experiments, performed on 27 datasets from the UCI repository, a normal kNN technique was used as a performance benchmark. Since many of problems used in the study are binary, only odd *k*-values were used. More specifically, the three *k*-values evaluated for normal kNN are 5, 11 and 17, while

possible k-values for G-kNN are all odd numbers between 1 and 25. 10-fold stratified cross-validation is used for evaluating both the standard kNNs and G-kNN. However, due the inconsistency of GP, G-kNN is run 10 times on each fold. Hence, the reported results for G-kNN are the average test set accuracies over the ten folds, where the result of each fold is the average from 10 runs.

When preprocessing, all attributes were linearly normalized to the interval [0, 1]. For numerical attributes, missing values were handled by replacing the missing value with the mean value of that attribute. For nominal attributes, missing values were replaced with the mode; i.e., the most common value.

Of course, how distance is measured when using instance-based learners is very important. In this study, standard Euclidean distance between instance vectors is used. For nominal attributes, the distance is either 0 or 1; i.e., the distance is 0 if the values are identical, otherwise 1.

Nominal and ordered categorical attributes could potentially be handled differently; e.g., ordered attributes could use the same distance function as continuous attributes. On many UCI datasets, it is quite obvious from the problem descriptions that several categorical attributes are actually ordered. In a previous study by Johansson, Boström and König (2008), where another kNN algorithm was suggested, an effort was made to identify the character of each attribute in the datasets used. Although this was not explicitly targeted in the evaluation, it was obvious that this identification of attribute characteristics generally increased kNN accuracy. Here, however, all attributes marked as categorical were, for simplicity, treated as nominal.

Results

First of all, it is interesting to note that all three standard versions of kNN obtained quite similar overall results; i.e., see Table 27. Even if the overall performance was comparable, it varied on different datasets. For some datasets it was clearly better to use k=5 and for others k=11 or k=17, gave the best results. Clearly, the k-value is very important and no single value is considerably better overall.

Table 27 show that all three G-kNN versions obtained higher average accuracies than the three standard kNNs. Especially the global G-kNN performed remarkably well, winning 7 datasets and almost always (on 23 of 27 datasets) finishing among the top three techniques.

Dataset	kNN			G-kNN		
	k=5	k=11	k=17	Global	Local	Mix
Breast-cancer	.693	.749	**.752**	.733	.719	.725
Breast-w	.964	**.966**	**.966**	.957	.956	.959
Cmc	.465	.476	.473	.467	**.518**	.517
Colic	.810	.818	.826	**.831**	.826	.829
Credit-a	**.871**	.867	.864	.865	.864	.861
Credit-g	.732	**.739**	.733	.733	.732	.730
Cylinder-bands	.732	.713	.698	**.801**	.794	.753
Diabetes	.729	.733	**.738**	.734	.732	.732
Ecoli	.869	.866	.834	**.871**	.864	**.871**
Glass	.678	.636	.618	.694	.687	**.702**
Haberman	.693	**.745**	.722	.725	.713	.719
Heart-c	**.828**	.825	**.828**	.825	.825	.821
Heart-statlog	.804	.815	**.822**	.816	.818	.810
Hepatitis	**.846**	**.846**	.820	.839	.835	.834
Inonosphere	.855	.846	.843	.870	.873	**.879**
Iris	.953	.947	**.967**	.955	.954	.957
Labor	**.887**	.837	.800	.872	.857	.853
Liver-disorders	.612	.623	.638	.651	.640	**.643**
Lymph	.839	.826	.840	**.850**	.842	.844
Sick	.963	.960	.956	.964	.964	**.966**
Sonar	.836	.726	.706	**.853**	.811	.824
Tae	.551	.510	.498	**.580**	.530	.546
Tic-tac-toe	.884	.953	.981	.990	.978	**.992**
Vehicle	**.698**	.679	.685	.689	.695	.691
Votes	.934	.929	.927	.934	**.958**	.954
Wine	.966	**.978**	**.978**	.974	.967	.969
Zoo	.950	.882	.883	**.965**	.936	.952
Mean	.802	.796	.792	.816	.811	.812

Table 27 - Accuracy for all kNN models

To assess statistical differences, a Friedman test, followed by a Nemenyi post-hoc test, was performed. Table 28 shows the ranks for each technique on every dataset. For 27 datasets and six techniques, the critical distance (for α=0.05) of average rank is *1.45* for the Nemenyi post hoc test.

Dataset	kNN			G-kNN		
	k=5	k=11	k=17	Global	Local	Mix
Breast-cancer	6	2	1	3	*5*	4
Breast-w	3	1	1	*5*	6	4
Cmc	6	3	4	*5*	1	2
Colic	6	*5*	3	1	4	2
Credit-a	1	2	*5*	3	4	6
Credit-g	*5*	1	2	3	4	6
Cylinder-bands	4	*5*	6	1	2	3
Diabetes	6	3	1	2	4	*5*
Ecoli	3	4	6	2	*5*	1
Glass	4	*5*	6	2	3	1
Haberman	6	1	3	2	*5*	4
Heart-c	1	*5*	2	4	3	6
Heart-statlog	6	4	1	3	2	*5*
Hepatitis	1	1	6	3	4	*5*
Inonosphere	4	*5*	6	3	2	1
Iris	*5*	6	1	3	4	2
Labor	1	*5*	6	2	3	4
Liver-disorders	6	*5*	4	1	3	2
Lymph	*5*	6	4	1	3	2
Sick	4	*5*	6	2	3	1
Sonar	2	*5*	6	1	4	3
Tae	2	*5*	6	1	4	3
Tic-tac-toe	6	*5*	3	2	4	1
Vehicle	1	6	*5*	4	2	3
Votes	4	*5*	6	3	1	2
Wine	6	1	1	3	*5*	4
Zoo	3	6	*5*	1	4	2
Mean	3.96	3.96	3.93	2.44	3.48	3.11

Table 28 - Average rank for all kNN models (CD=1.45)

According to the Nemenyi post hoc test, *global* G-kNN obtained significantly higher accuracy than all three kNN versions. Furthermore, the ranks also show that both *mixed* and (to a lesser degree) *local* G-kNN clearly outperformed standard kNN, even if the differences are not statistically significant.

Analysis

The importance of the *k*-value for the standard kNN techniques can clearly be seen in Table 28. In general, the standard techniques are either very good (ranks 1-2) or poor (ranks 5-6). G-kNN performance, is however, much more consistent and better overall. All three G-kNN versions perform at least fairly well (ranks 1-4) on a large majority of the datasets. Note that the reported results are the averages of 10 G-kNN runs. Hence, the performance could probably be improved considerably by a reasonable heuristic that selects one of the ten programs.

A drawback of G-kNN is that it is much more computationally expensive than standard kNN. Hence, even if it could be argued that G-kNN should be compared to more powerful techniques like ANN or SVM. On the other hand, the decision tree part of a G-kNN program adds some comprehensibility to the prediction and can lead to new insights. Consequently a comparison to opaque techniques will only lead to another case of the accuracy versus comprehensibility tradeoff.

Summary

From the discussion and examples above, it should be clear that the fact that GP can optimize arbitrary representation is a great strength. Of course, there are other specialized techniques that can optimize one or two of the presented representations, but like decision trees or decision lists, they cannot optimize all. Hence, a decision maker cannot specialize in a single technique, since this would tie him/her to a certain representation.

In a concrete example a G-kNN trees, evolved using the *global* strategy, were significantly more accurate than three standard kNN versions. Furthermore, the decision tree part of the evolved program adds some comprehensibility to the model, while standard kNN is totally opaque. There are three additional advantages of G-kNN, compared to standard kNN; G-kNN does not require a specific *k*-value to be set, it can exploit local information to select a different *k* for different regions of the input space, and it is optimized globally using GP, which may avoid local optima.

Finally, neither the CRISP-DM nor SAS Instruments' SEMMA give the representation individual attention. They do, however, stress the importance of choosing the best predictive technique for the problem, but this means choosing a predefined package of a representation, an optimized score function, and a training algorithm. Hence, the representation can seldom be chosen as freely as would be desirable, if not using GP.

5.2 Evaluation of the Challenges of using GP for PM

As mentioned in the introduction to this chapter, the following sections are motivated by a discrepancy between theory and practice; i.e., the challenges identified in the literature are not recognized by typical GPPM frameworks. Hence, section 5.2.1 explores criterion 2.3, i.e., how large search spaces may affect GP performance when used for PM. Criterion 3.4, inconsistency, is also normally viewed as a disadvantage of evolutionary techniques, but only three of the thirteen evaluated frameworks have any functionality related to inconsistency and they all simply presented a small set of alternative programs. To facilitate a more informed evaluation of strategies for handling inconsistency, section 5.2.2 explores the true extent of GP's inconsistency when used for PM. Introns are discussed later in section 7.1, together with a novel, representation-independent technique for intron removal.

5.2.1 GP and large search spaces

Since finding the smallest optimal decision tree for a specific training set is a NP-complete problem (Hyafil & Rivest 1976), machine learning algorithms for constructing decision trees tend to be non-backtracking and greedy in nature (Jeroen Eggermont, Kok & Kosters 2004). Even if greedy splitting heuristics are efficient and adequate for most applications, Murthy (1998) argues that they are essentially suboptimal, since they optimize each split locally without considering the model as a whole. Consequently, due to the local non-backtracking search, decision trees may fasten in local minima. Therefore, Eggermont, Kok and Kosters (2004) argue in favor of GP, since it performs a more global search that reaches a substantially larger part of the space of possible trees. However, the authors also point out that the search space for evolutionary approaches tends to be very large and sometimes even infinite. The maximum tree size grow exponentially with the allowed depth, and there by the search space, since the attributes and possible splits can be combined in more ways. Since a program can, in theory, have infinite size, the search space naturally also becomes infinite. De Jong (1988) makes the same observation and notes that even if, theoretically, every point in the search space has a nonzero probability of being sampled, the search space of most problems of interest is so large that waiting long enough for guaranteed global optimums is impractical.

The results presented earlier in Table 7 section 5.1.1 shows no significant difference between GP and CART. Even if f_{ACC} has a slightly better average ACC, CART actually wins on 7 of 13 datasets. Hence, in this case, the greedy search is not outperformed by GP global optimization. All the experiments of this thesis have been conducted with a constant, rather high parsimony pressure to favor small

comprehensible programs. Since smaller rules are more comprehensible, the choice of parsimony pressure becomes a crucial design choice. On one hand, a pressure favoring smaller trees may result in sub-optimal accuracy, since larger complex rules may not be evolved. On the other hand, if larger trees are favored, it may result in an unnecessarily complex or sub-optimal solution, due to the very large search space. Table 55 in section 7.1 does for example show that a lower parsimony pressure can actually lead to clearly inferior programs.

The following section evaluates whether the results presented in Table 7 can be explained by the size of the produced models.

Method

The results presented below are taken from a study conducted by König, Johansson, Löfström and Niklasson (2010). Here, f_{ACC} is compared to *J48* (as implemented in WEKA) using default settings over eighteen datasets employing 4-fold cross-validation. The G-REX settings applied when evolving the program using f_{ACC} are presented in Table 29.

Number of generations	100
Population size	200
Crossover probability	.8
Mutation probability	.001
Creation type	Ramped half -and -half
Selection type	Roulette wheel
Maximum creation depth	6
Parsimony pressure (P)	0.001
Batch size	5

Table 29 – G-REX settings for ACC comparison

Results

The results reported in Table 30 are the average ACC of 4-fold cross-validation with stratification. Clearly, there is no significant difference in ACC between J48 and f_{ACC}, even if the average ACC is again slightly higher for f_{ACC}. It is also apparent that the trees produced by *J48* are larger than the ones produced using f_{ACC}.

In general, both techniques perform very comparably, but there are some datasets where the difference is bigger. For four of the datasets, *Colic, Tic-Tac-Toe, Hepatitis* and *Glass, J48* achieves an ACC that is approximately *5%-12%* higher than the programs evolved using f_{ACC}. The opposite is true for *Cylinder-bands, Labor,* and *Lymph,* where f_{ACC} outperforms *J48* by between 8%-15%.

Datasets	ACC		Size	
	J48	f_{ACC}	J48	f_{ACC}
Breast-cancer	**.699**	.696	**6**	45
Breast-w	**.956**	.946	**25**	29
Colic	**.860**	.739	***5***	25
Credit-a	.855	**.867**	42	**9**
Credit-g	**.730**	.723	140	**29**
Cylinder-bands	.578	**.665**	**1**	17
Diabetes	.733	**.737**	**39**	45
Ecoli	.786	**.815**	43	**21**
Glass	**.668**	.612	59	**33**
Haberman	.739	**.745**	25	**21**
Heart-c	.759	**.782**	51	**49**
Heart-statlog	.770	**.800**	35	**25**
Hepatitis	**.806**	.735	21	**17**
Iris	**.953**	.947	**9**	13
Labor	.737	**.895**	***5***	21
Liver-disorders	**.620**	**.620**	51	**49**
Lymph	.730	**.797**	34	**17**
Tic-tac-toe	**.849**	.788	141	**13**
Average	.768	**.773**	40.7	26.6

Table 30 - Average ACC and Size

Analysis

For five of the seven datasets, where there were big differences between the techniques, the winning model had higher complexity. In the cases where J48 was superior and larger, the lower accuracy for f_{ACC} could be the result of a too high parsimony. While a too hard pruning could explain the cases where J48 was the inferior model. The two remaining datasets where the winning model actually had a lower complexity could be the result of overfitting. Nonetheless, the results show one of PM's hardest problems, i.e., how to create comprehensible models with high predictive performance from novel data. The point here is that GP's global search does not, in this case, result in a superior model, due to a too hard parsimony pressure or because the evolution did not have enough time to explore the whole search space, thus resulting in a suboptimal solution.

Some datasets obviously require a large model, since the underlying relationships are complex. A typical example is the UCI dataset Tic-Tac-Toe, which encodes the complete set of possible board configurations at the end of tic-tac-toe games. Since there is no noise, and there is a true underlying concept, a

larger solution will most often perform better. The poor performance of f_{ACC} for *Tic-Tac-Toe (.788* for f_{ACC} vs. *.849* for J48*)* can therefore be explained by a model that is too simple, due to a parsimony pressure that is too high. The same holds for the *Glass* dataset, where *J48* is 5.6% better than f_{ACC} but almost twice as large. *Hepatitis* is yet another dataset where a larger *J48* model clearly outperforms a small f_{ACC} model. A solution to the problem would of course be to perform several GP runs, with varied parsimony pressure to find the best setting, but since GP is computational expensive, this is normally not practical.

Summary

Even if global optimization of the GP search should, in theory, be superior to the non-backtracking search of decision tree techniques, it seems not to be the case in practice. All studies discussed above showed that GP did not outperform CART or *J48* when compared with regard to ACC. One explanation could be that the programs evolved using f_{ACC} were too simple and another is that the GP process was not run long enough. A further explanation is that some programs were overfitted; all three reasons are probably true, depending on the dataset.

5.2.2 Empirical evaluation of instability

A well-known deficiency present in most decision tree algorithms is that they are unstable; i.e., slight variations in the training data can result in quite different attribute selections in the splits, see, e.g., (Roiger & Geatz 2003). The problem then, of course, becomes which decision tree should be trusted if small variations in the data will produce very different trees. Turney (1995) notes that engineers are disturbed and lose confidence in the decision trees when different batches of data from the same process result in radically different trees.

According to Dietterich (1996), the most fundamental source of instability is that the hypothesis space is too large. If an algorithm searches a very large hypothesis space and outputs a single hypothesis, then in the absence of huge amounts of training data, the algorithm will need to make many more or less arbitrary decisions, decisions which might be different if the training set were only slightly modified. This is called informational instability, instability caused by the lack of information. Informational instability is for example, often experienced in the medical domain, where datasets often contain a small number of instances (sometimes 100 or less) but a still relatively large number of features. In such cases (large spaces with sparse samples), it is quite likely that different logical expressions may accidently classify some data well, thus many data mining systems may find solutions which are precise but not meaningful according to experts; see, e.g., (Grąbczewski & Duch 2002).

Informational instability is obviously a problem for all machine learning techniques. Even if deterministic techniques, such as decision trees, always produce the same tree for the same dataset, the tree produced is just one arbitrary tree. There is no evidence that a tree created using a specific measure for information gain should be better.

GP, on the other hand, is inherently nondeterministic, due to the probabilistic nature of the evolution process. Consecutive runs made on the same data may result in several different solutions. Hence, GP's inconsistency is in fact nothing else than information instability, which of course differs for each dataset, depending on the data.

Instead of evaluating the instability of GP, as the heading suggests, this section actually evaluates the informational instability of some well-used UCI datasets, in order to demonstrate how unstable they are. Naturally, deterministic techniques cannot be used for this purpose, since the idea is to generate as many alternative solutions as possible.

There are two main approaches to evaluating the instability of a technique, i.e., prediction instability and structural instability. Prediction instability disregards the program and only concerns how similar the predictions of two programs created from the same data are. However, structural instability concerns how similar the actual programs are as rule sets.

Related work

A simple and common way of generating different solutions for a certain dataset is to train different models on different parts of the training data. Since most machine learning techniques are unstable, this will result in different models. Bagging is a well-known example of this technique which creates a new bootstrap training set for each model, by randomly selecting instances (with replacement) from the original training set. This technique is, however, not suitable for evaluating informational instability, since each bootstrap creates a new dataset with a new, unknown level of informational instability.

Plish (1998) suggests an algorithm for generating alternative solutions for multi-criterion linear programming models. Alternative solutions are generated by slightly altering the objective or the constraints and then solving the resulting problem with the help of supplementary constraints. Structural changes are also considered, but the algorithm is constrained to multi-criterion linear problems and cannot be applied to decision trees.

An algorithm for generating alternative decision trees is presented by Grabczewski and Duch (2002) who use a variant of the beam search to create a heterogeneous forest of decision trees. The number of possible alternative

solutions is restricted to the beam size. To find a forest, all trees found during the search are ordered according to their accuracy estimated on a validation set. An infinitum beam size corresponds to a breadth first search, which is impractical for most problems. Grąbczewski & Duch report that their algorithm finds several good alternative solutions for three UCI datasets. However, it is unclear exactly how the solutions were evaluated and there is no account of how many alternative solutions were found.

Li and Liu (2003) present a simple method for creating ensembles called cascading trees. First, all features are ranked according to their gain ratio. Next, trees are created in a cascading manner where the root node of the *ith* tree corresponds to the *ith* ranked features; the rest of the tree is created as normal. The authors stress the fact that cascading trees, unlike bagging or boosting, do not in any way modify the original data. This is of critical concern in bio-medical applications, such as the understanding and diagnosis of a disease where it is important that all training instances are classified correctly. The method was evaluated on two medically-related datasets using tenfold cross-validation. Several trees that could classify the training data without error were found for all folds. An interesting observation made by the authors is that the tree with the best test accuracy often did not have the feature with the highest gain ratio in its root. The number of alternative trees is however naturally bounded by the number of features in the dataset.

Finally, it should be noted that ensemble creation techniques often create base classifiers by sacrificing accuracy for diversity. Ensemble members are usually less accurate than single models, while the ensemble is more accurate. Hence, most ensemble methods are not suitable for creating several alternative standalone solutions.

Method

Predictive instability

Johansson, König, and Niklasson, (2007) investigated how GP's probabilistic nature affected the predictions of the evolved programs. The idea was to quantify how much the predictions of two programs evolved for the same data differed. It should be noted that this was not the main purpose of the study and hence the dataset was produced for rule extraction, i.e., the target variable was replaced by the prediction of an opaque model consisting of an ANN ensemble. The dataset was however

identical for each GP run, so the result in terms of instability is still sound. Tenfold cross-validation was used and five programs were evolved and compared for each fold.

Structural instability

Structural instability was evaluated in a latter study by König, Johansson, and Niklasson, (2010). Here, a single decision tree was first created using the *J48* algorithm in the WEKA (Witten & Frank 2005) workbench. An alternative solution was defined as being a decision tree that had equal or less complexity, or less than the original *J48* tree, while having equal or better training accuracy. Furthermore, an alternative decision tree should classify the data with a unique partitioning of the training instances, i.e., to be an alternative, the solution needs to base its decision on different facts. Two decision trees can classify the data in exactly the same way but still partition the instances differently.

Naturally, the numbers of alternative solutions are dependent on the size of the tree, i.e, the number of conditions in the original tree and the number of possible attribute values in the dataset. In the experiments (and in most decision tree algorithms), only relevant conditions are considered, since the aim is to present truly alternative solutions. Relevant conditions for an attribute are the conditions needed to divide the dataset into pure and impure partitions. A pure partition is a set of instances of the same target class.

For an original tree with n conditions and a dataset with r relevant conditions, there are n^r possible trees. Of these solutions, only the ones with equal or higher accuracy were considered to be alternative solutions. Furthermore, if several alternative trees partitioned the dataset in the same way, only the smallest tree was considered to be an alternative solution. Since the number of possible solutions is related to the size of the original trees, trees of the same size must be evaluated for all datasets. Equal tree size will facilitate an evaluation of the relation between size, accuracy, and the lack of information in each dataset.

Creation of original trees

Since the pruning in J48 does not support the creation of trees of a certain size, the algorithm cannot be used in its original form. Instead, a very large J48 tree is first created by setting the confidence factor to .5 (higher values yield warnings in WEKA). Next, the tree is pruned to a certain size in the following manner:

1. CUT_DEPTH is set to MAX_SIZE
2. All branches are cut at CUT_DEPTH and replaced with leaf nodes predicting the majority class of the training instance reaching the new leaf.
3. Redundant leaves are removed, i.e., if both leaves of any root split are predicting the same class, the split is replaced by one of the leaves. This is done recursively, since replacing a split can result in new redundancy higher up in the tree.
4. If the tree SIZE > MAX_SIZE then CUT_DEPTH is set to = CUT_DEPTH-1. Return to 2.

Even if this pruning algorithm does not guarantee an exact size of the final tree either, it is much more consistent than the original pruning algorithm of J48.

Creation of Alternative Trees

To be able to generate alternative trees, the method needs to guarantee that the solutions have a certain training accuracy, size, and partition the dataset in a unique way. None of the techniques discussed in the related work fulfill all three requirements.

The approach taken in this study, which was based on GP, continuously evolves a population of decision trees. If any of the trees in the population meets the requirement of an alternative tree (accuracy, size and uniqueness), it is put in a growing list of alternative solutions. If two trees partition the dataset in the same way, only the smallest tree is kept in the list. In order to always drive the evolution towards new alternative solutions, a fitness function based on three metrics is used:

- A reward based on accuracy.
- A parsimony pressure in relation to the tree size, which increases if a tree is bigger than the original tree.
- A punishment in relation to how similar the tree is to other alternative solutions.

Similarity is calculated by counting the number of identical splits that occur in the same position in both trees. To ensure that each tree makes a unique partition of the dataset, the GP is only allowed to search among relevant splits. If all splits are relevant and the tree is not a copy of another tree, it will partition the data in a unique way.

The GP used in the experiments is more or less standard, employing tournament selection. A difference is that only two trees are selected for each

tournament, to slow down the convergence of the population. The idea is to look for more alternative trees around each discovered solution. Another important difference from standard GP is that five batches are used in the experiments. A batch always starts from a new, randomly generated population, but the list of alternative trees is kept during all batches. In this way, each population can start to look for solutions in new directions, even if a previous batch has converged on a certain solution.

Number of generations	100
Population size	300
Crossover probability	.8
Mutation probability	.001
Creation type	Ramped half-and -half
Selection type	Tournament Selection
Maximum creation depth	6
Parsimony pressure (P)	-
Batch size	*5*

Table 31 – G-REX settings for creation of alternative trees

Results

Predictive Instability

Table 32 shows the average predictive instability, i.e., the disagreement in the prediction of programs evolved for six UCI datasets. The results are average over ten folds, where the instability for each fold is calculated by comparing the prediction of five programs pairwise. On average, *10%* of the training set predictions differ between two randomly chosen trees evolved for the same dataset. Surprisingly, the result is almost the same for the test predictions which have an predictive instability of *12%*.

Structural Instability

The results of structural instability are presented for the three different program sizes, i.e., for programs with *3, 5* and *7* conditions (*S3, S5,* and *S7*), in Table 33. *Actuacl Size* is the actual size of the J48 tree after the second phase of pruning has been performed. The average number of alternative trees and groups (for ten folds) is presented for each of the evaluated program complexities.

A large number of alternative solutions could be found for all of the evaluated complexities. As could be expected, a higher size resulted in more alternative solutions. Ten or less alternative solutions could only be found for some datasets.

Datasets	Train	Test
Breast-w	.045	.029
Diabetes	.092	.107
Heart-c	.107	.155
Liver-disorders	.195	.249
Sonar	.090	.102
Zoo	.103	.097
Mean	.100	.123

Table 32 – Average predictive instability

Dataset	Actual Size			#Programs		
	S3	S5	S7	S3	S5	S7
Breast-cancer	2.6	4.2	6.1	64.3	9.3	159.9
Breast-w	2.0	5	5.3	113	325.4	302.4
Colic	2.0	3.5	5.7	4.6	24.7	95.7
Contact-lenses	2.9	2.9	2.9	8.2	7.4	7.4
Credit-a	2.5	4	6	26	106.8	237.2
Cylinder-bands	2.8	4.4	6.9	218.4	249.2	42.5
Diabetes	2.5	3	4.4	5.2	29	13.4
Glass	2.8	4.8	5.6	427.4	15.5	22.9
Haberman	2.4	3.9	6.4	142.5	324	551.9
Heart-c	2.6	4.5	5.5	76.9	17.3	34.1
Heart-statlog	1.8	4.5	5.4	9.7	43.1	3.5
Hepatitis	1.9	3.6	5.3	19	145.9	28.2
Iris	2.9	3.8	3.8	28.5	21.8	23.8
Liver-disorders	2.1	4.6	5.6	66.7	106.2	88.1
Lymph	2.2	4.5	5.4	155.8	292.2	284.9
Tae	2.7	2.9	6.3	176.4	164.1	648.9
Tic-tac-toe	1.2	1.2	6	1.1	1.1	72.7
Wine	2.9	4.2	4.5	4.2	1.4	12.7
Zoo	3.0	5	6.8	2.7	1.9	16.4
Mean	2.4	3.9	5.5	81.6	103.5	173.9

Table 33 – Average size and number of alternative solutions

Another interesting result of the experiment is how the *J48* trees compared, in terms of test ACC, to the evolved alternative programs. The test accuracies are presented in Table 34, where **RND** is the average accuracy of all alternative

solutions, which represents selecting a tree at random. For each dataset and target complexity, the best result is marked with bold numbers.

Dataset	S3		S5		S7	
	J48	RND	J48	RND	J48	RND
Breast-cancer	.707	**.724**	.710	**.726**	.707	**.721**
Breast-w	.929	**.937**	.944	**.950**	.946	**.951**
Colic	.837	**.852**	.845	**.851**	**.851**	.850
Contact-lenses	.750	**.776**	.750	**.780**	.750	**.779**
Credit-a	**.857**	.852	.841	**.848**	.843	**.848**
Cylinder-bands	.663	**.674**	.680	**.692**	**.709**	.689
Diabetes	.743	**.744**	.745	**.749**	**.746**	.744
Glass	.472	**.534**	.640	**.650**	**.645**	.641
Haberman	.683	**.724**	.690	**.721**	.677	**.725**
Heart-c	**.752**	.744	.782	**.799**	.772	**.808**
Heart-statlog	**.722**	.705	.763	**.785**	.774	**.804**
Hepatitis	**.801**	.788	.788	**.813**	.762	**.779**
Iris	.947	**.950**	.940	**.952**	.940	**.954**
Liver-disorders	.644	**.665**	**.650**	.644	**.655**	.645
Lymph	.595	**.690**	**.783**	.756	.750	**.765**
Tae	.458	**.491**	.465	**.496**	.485	**.547**
Tic-tac-toe	**.688**	**.688**	**.688**	**.688**	.716	**.753**
Wine	.880	**.908**	.916	**.929**	.920	**.933**
Zoo	**.833**	.815	**.912**	.902	.911	**.931**
Mean	.735	**.751**	.765	.775	.766	**.782**

Table 34 – Average ACC per target size

As could be expected, a more complex *J48* tree is in general more accurate. This result also holds for the evolved alternative programs. In terms of test accuracy, the average evolved alternative solutions clearly outperform the *J48* trees, with thirteen wins and one draw for S3, fifteen wins and one draw for S5, and fourteen wins for S7.

Analysis

Predictive Instability

The experiment regarding predictive instability shows that there is, on average, a difference of *10%* on training data and *12%* on test data. The difference for instability between the training- and test data was quite small for all datasets. No direct evaluation was done, in terms of which of the predictions were different.

Hence, it is hard to say if the instability relates to a few atypical instances or it is spread more uniformly over all instances.

Structural instability

On average, there are a large number of alternative solutions for all target sizes, even if the number varies greatly for different datasets. More specifically, the ten most unstable datasets constitute 90% of the total instability, which clearly shows that it is the instability of the datasets which is the main reason for GP's inconsistency.

As could be expected, there are many alternative solutions and higher complexity results in even more alternative solutions. It is, however, more surprising that even if the accuracy increases for more complex programs, the possible number of solutions also increases. Of course, larger programs facilitate more combinations of the splits, but the search for an accurate program automatically becomes harder, since there are more programs to search among and fewer trees that have the required accuracy. Furthermore, the average alternative tree clearly outperformed the original *J48* tree, in terms of test accuracy. *J48* only outperformed the average alternative solution on, at most, five of nineteen datasets. Finally, it should be noted that the alternative pruning mechanism that was used may have lowered the generalization capabilities of the J48 tree, compared to the standard pruning mechanism. However, it is clear that there are many competitive solutions to the ones found by a J48 tree.

Summary

If numerous competitive alternative solutions exist, as the experiments demonstrated, the obvious question then becomes why the best solutions should be the ones that happened to be found by a decision tree algorithm. Why would the decision tree represent the true underlying relationship, when there are so many others? Furthermore, even if the choice of program does not affect the predictive performance, it will have a noticeable effect on the interpretation of the underlying relationships, since, on average, two evolved programs differ on 10% of the predictions.

Since informational instability is a consequence of a dataset that is not fully representative for the problem domain, more data needs to be gathered, or a decision maker needs to use domain knowledge to select the most probable solutions. Presenting all possible solutions is however not a good alternative either, since the average number of solutions (*81.5* for *S3*, *103.5* for *S5*, and *173.9* for *S7*) is too large for a decision maker to handle manually.

Hence, there is need of a decision support technique that helps a decision maker select solutions, based on domain knowledge not present in the dataset, in a rational and effective way.

5.3 Implications for a GP framework

The sections above demonstrate the importance of an optimization independent of the representation and validated the importance of optimizing the score function actually required by the problem at hand, i.e., criteria 2.2 and 3.2. The results regarding optimization of alternative score functions are significant for all score functions except ACC. On behalf of the score functions that optimized AUC and BRE, this may be explained by the fact that both metrics optimized ACC implicitly. The draw, in terms of ACC compared to decision tree and decision list techniques, is however more surprising, even if they also implicitly optimize ACC. GP's global search should in theory outperform the greedy non-backtracking search strategy used by decision tree algorithms. Two possible explanations are both related to large search spaces, i.e., criteria 2.3:

- An ill-chosen Parsimony pressure may constrain the GP search in a search space that is too small or unnecessarily large.
- When the problem requires large solutions, it may be impractical to run the evolution long enough.

Both explanations are well-known weak spots of GP and need further attention. Experiments also showed that criterion 3.1., the accuracy vs. comprehensibility tradeoff, could be somewhat handled by applying parsimony pressure and even more by extracting rules from an opaque model. Actually, when optimizing BRI the extracted rules achieved significantly higher ACC and AUC compared to J48 while being considerably smaller in size.

Informational instability is another ever present challenge which needs to be handled. As demonstrated, GP may produce numerous alternative solutions which can confuse and overwhelm a decision maker. Decision trees are, on the other hand, deterministic and only produce a single tree of numerous possible trees with comparable predictive performance. Hence, decision tree algorithms may lure a decision maker into a false security. Three of the evaluated frameworks present a small set of alternative programs. However, considering the large number of alternative programs that, in general, exists, presenting five or ten programs instead of one is not much better.

Obviously, none of the examples are satisfactory. Therefore, critieria 3.4 is well motivated and GPPM frameworks would benefit from:

- Techniques explicitly designed to handle instability in a structured and rational way.

Considering the advantages that GP's representation-free optimization provides, techniques for intron removal should also be independent of the underlying presentation. In practice, however, intron removal techniques rely on algebraic simplification and are thus inherently dependent on the representation. Hence, criteria 3.3 could be better fulfilled by:

- A representation-independent technique for the removal of introns.

Summarizing the results for criteria 2.2 – 4.1, which regard the more general aspects of GP, it is clear that criteria 2.2, Tailoring of the fitness function; 3.1, Trading Acc vs. Comp; and 3.2, Tailoring of the representation; are all well motivated and have reasonable solutions, both in theory an in practice. The remaining criteria, i.e., 2.3, Handling of large search spaces; 3.3, Intron removal; 3.4, Handling of inconsistency; and 4.1, Discovery of interesting rules; are not satisfactorily handled in theory or practice and require more attention. Of these criteria, 2.3 relates to the predictive performance and 3.3, 3.4, and 4.1 to the comprehensibility of the evolved models.

6 Enhancing Accuracy

The previous chapter shows that GP outperforms traditional decision tree techniques when GP optimizes the evaluated score function, while the decision trees optimize another, i.e., some kind of purity or information gain metric. It did however also show that standard GP was not significantly better when compared on ACC, since ACC was implicitly optimized by the decision tree techniques. One possible explanation could be that the GP process had not been run long enough and/or that a too high parsimony pressure had obstructed the search among large programs. For other datasets, the opposite can have led to overfitting. Jeroen Eggermont et al. (2004) argue that the standard GP algorithm fails to achieve a decent performance when the search space becomes very large. Hyafil and Rivest (1976) point out that the search space for classification tasks tends to become extremely large. Hence, several methods for improving the performance of GP systems have been suggested; see section 2.5.7. Nevertheless, the challenge of how to best choose the parsimony pressure and how to find good programs when the search space grows very large still remains. The following section purposes two novel techniques related to large search spaces, i.e., criteria 2.3, in sections 6.1 and 6.2. Section 6.3 approaches criteria 3.4, by presenting three novel techniques that utilize GP's inherent inconsistency to improve predictive performance.

6.1 Using local search to increase the accuracy of regression models

Dobra and Gerke (2002) note that there has been an increased interest in accurate and comprehensible regression models for large datasets and that model trees, with linear regression in the leaves, fulfill both properties. Of course, GP could also be used to evolve model trees. However, even if GP is very suitable for evolving models trees since the structure and parameters are evolved simultaneously, the search space grows very large because the degrees of freedom increase drastically compared to a regression tree. Hence, care needs to be taken when choosing the type of model and optimization approach. The following sections are based on the article "*Genetic Programming – A Tool for Flexible Rule Extraction*", by König, Johansson and Niklasson (2007), and aims to propose a technique for the creation of regression model trees using GP. In general, regression tasks are harder for GP compared to classification tasks, due to a numerical target and hence a larger search space. Here, model trees with linear regressions as leaves are evolved, which further increases the search space, due to a complex representation. The main idea explored in this section is to simplify the search for a good model tree using a local search, i.e., by calculating the linear regressions using least square instead of evolving both the tree and the optimal regression for each leaf.

Related Work

Potgieter and Engelbrecht (2008) evolved model trees with multivariate nonlinear models at the leaves. The main idea was to let the GP process select terminals from a pool of good, multivariate nonlinear models which were continuously evolved using GA. In the GA process, the fitness of a terminal was based on how frequently it was used by the programs in the GP process. Experiments show that GPMCC produces programs that are less complex, but still have a MAE comparable to two commercial regression techniques, NeuroLinear and Cubist.

The down side of nonlinear models is of course that they are very hard to interpret. Even model trees with linear regression based on several variables, e.g., created using M5, are naturally often much harder to fully comprehend than regression trees. A middle way, taken by Alexander and Grimshaw (1996) in their decision tree technique, *Treed*, is only to allow a single attribute in the regressions. With a single attribute in the regressions, even a large tree should be possible to interpret.

In light of the range problem demonstrated by Keijzer (2003), see section 5.2.1, a straightforward approach to evolving model trees with even simple regressions based on a single variable seems unlikely to succeed. The same problem of finding the correct range would appear in each leaf node. In this case, Keijzer's approach of scaling the final output of the model would not work either, since the range of the intercept would need to be in a scale appropriate for the associated variable. At the same time, the approach taken by Potgieter and Engelbrecht (2008) appears unnecessarily complex, if only simple regressions are sought.

Method

The following section extends Keizer's idea by calculating the optimal slope and intercept using least square in each leaf, instead of once for the whole program. The main idea is to guide the GP so that it only searches among reasonable building blocks, i.e., linear regressions optimized using least square. In this way, the problem is divided into a global GP search for the best partitioning of the dataset and uses a local greedy search for the best linear regressions.

In an attempt to mimic Koza's ERCs, all linear regression is calculated at the creation of a program and then remains constant, even if swapped to another leaf by a crossover operation. However, to ensure that the evolution is not constricted to the regressions of the initial population, mutation can introduce new linear regression in the same way that Koza introduced new ERCs. Each linear regression is created for a randomly chosen variable and optimized for the training instances reaching the leaf. The BNF for the GP optimized linear model tree, henceforth named *fomt*, can be found in Table 75 in the appendix. An important motivation

for this design was the need of reducing the computational costs. Calculating new linear regression for each leaf of 500 programs during 100 generations could be a rather expensive task computationally, especially for larger datasets. Another motivation was that the leaf regressions should be rather general, to reduce the risk of over fitting. By not recalculating the regressions every time they are swapped to a new branch, the selection pressure should favor regressions that are general enough to work on several subsets of the training set and hence, hopefully, also would work well on new unseen data.

The fitness function used in the study is based on the absolute error (AE) and uses a standard parsimony pressure based on the complexity of the programs. However, since the scale and the difficulty of a dataset greatly affect the magnitude of the AE, it is standard practice to use initial experiments to find a factor that can scale the pressure to an appropriate level. The scales of different datasets can easily be handled by normalizing the AE with the total value of the dependent variable, but this does not account for the difficulty of the datasets. In an attempt to remove this step, which can be tedious and time consuming, especially when using several datasets and 10-fold cross-validation, the AE is automatically scaled using a scaling factor *s* initially set to the total value of the dependent variable. The scaling factor is adjusted on the basis of the fitness of the best individual in the population (f_{best}).

$$f_p = \frac{\sum_{i=1}^{n} |p_i - a_i|}{s} + P * O_p$$

$$s = \begin{cases} \sum a & if(generation\ 0) \\ s * 1.5 & if(f_{best} > .9) \\ s & if(.1 \geq f_{best} \geq .9) \\ s/1.5 & if(f_{best} < .1) \end{cases} \quad (48)$$

To evaluate if the scaling of AE will make the parsimony pressure more independent of the dataset, the parsimony coefficient P is set to .01 for all datasets. This value was of course found using some initial experiments, but the point is that this only had to be done once and not ten times for each dataset (10-fold cross-validation was used in the experiments). A summary of the used values of the standard GP can be seen in Table 35.

A secondary objective of this study was to demonstrate the use of GP to extract rules from an opaque, highly accurate ensemble. However, rule extraction using GP is evaluated separately in section 5.1.2 and is therefore not analyzed in detail in this section. Nevertheless, the rule extraction procedure may of course influence the results.

Number of generations	100
Population size	500
Crossover probability	.8
Mutation probability	.001
Creation type	Ramped half and half
Selection type	Roulette wheel
Maximum creation depth	6
Parsimony Pressure (P)	.01
Batch size	1

Table 35 – G-REX settings for optimization of ACC

The ensembles used in the experiments consist of 20 fully-connected ANNs, each having one hidden layer. Each ANN is trained on an individual subset of the training set. The subset is created by randomly picking 80% of the training instances without replacement. The number of input variables is also reduced by randomly selecting only 80% of the features for each ANN. The purpose of this is, of course, to increase the ensemble diversity. Each ANN is initiated with random weights and a random number of hidden units in the range of 1 to 20 units. When the ensemble is trained, a new training set is created, which is identical to the original training set, except that the values of the dependent variable are replaced with the average value of the predictions made by the ensemble members.

Results

To facilitate a benchmark of f_{OMT}, comparisons are made to GP not using the local optimization (f_{MT}), multiple linear regressions (MREG), as implemented in MatLab, and to a regression tree created using CART.

GMRAE, presented in section 2.3.2, is used as a measure of predictive performance, according to the recommendation of Armstrong and Collopy (1992). GMRAE is recommended after a comparison to other measures based on reliability, construct validity, sensitivity to small changes, protection against outliers, and their relationship to decision making. GMRAE is designed for times series prediction, but can also be used for estimation tasks, if the reference to random walk ($p_{s,rw}$) in equation 21 is replaced with the mean value of the dependent variable.

Size is calculated as the total number of functions and terminals used in a model to facilitate some kind of quantitative comparison, in spite of the evaluated techniques' difference in representation. This measure only gives a rough idea of the complexity, since the representations are quite different; CART predicts using constant values in the leaves, f_{MT} and f_{OMT} with simple linear regressions, while MREG produces a single expression. The results presented below are the average

results over ten folds which were stratified to ensure that each fold was representative for the dataset.

Dataset	GMRAE				Size			
	f_{MT}	f_{OMT}	CART	MREG	f_{MT}	f_{OMT}	CART	MREG
Auto-price	.341	**.262**	.303	.317	11.8	26.5	3.5	61.0
Diabetes-numeric	3.19	**.818**	.971	.942	15.8	35.1	11.0	9.0
Housing	.734	**.488**	.501	.506	12.2	24.5	32.0	57.0
Machine-cpu	.403	**.289**	.320	.383	11.1	38.0	21.5	25.0
Pharynx	.946	.725	1.049	**.672**	10.6	24.5	1.0	49.0
Sleep	.777	.665	**.238**	1.104	10.1	44.5	12.5	29.0
Veteran	.992	**.772**	1.034	.947	11.1	32.9	1.0	29.0
Wisconsin	.886	**.902**	1.008	.903	18.5	38.0	1.5	129.0
Mean	1.03	**.615**	.678	.722	12.7	33.0	11.2	48.5

Table 36 – Average GMRAE and Size

f_{OMT} is clearly the best technique with seven wins over each of the other techniques and a clearly lower GMRAE. Table 37 below, which presents the *p*-values of the Wilcoxon signed rank test between f_{OMT} and the other techniques, shows that the difference is significant for all comparisons except against CART. With regard to this result, it is interesting to note that the f_{MT}, whose only difference is that it does not use the local optimization, is clearly the worst of the evaluated techniques. The programs produced by f_{OMT} are in general more complex than the ones produced by f_{MT} and CART, but less complex than the MREG models.

	f_{MT}	CART	MREG
f_{OMT}	.0156	.1953	.0391

Table 37 – Pairwise comparison of local optimization

Analysis

The main result of the experiments is of course that the local search performed by f_{OMT} significantly improves the predictive performance, compared to a straightforward GP approach f_{MT}. It is also interesting to note that the programs produced by f_{MT} are less complex than the ones produced using f_{OMT}, even if both representations were optimized with the same parsimony pressure. A possible explanation is that the parsimony pressure makes it hard to find better programs, since each new linear regression, which consists of 5 primitives, results in an increase in predictive performance of $0.01 * 5 = .05$ to overcome the parsimony pressure. In the initial population, the effect of the parsimony pressure is not so

strong, since all programs have a rather similar size and hence receive the same pressure. However, when the initial programs evolve and become fitter, it also becomes increasingly more unlikely that a new random linear regression will add the necessary fitness to overcome the parsimony pressure.

The same reasoning applies to the f_{OMT} which uses the exact same representation, but f_{OMT} only searches among reasonable building blocks, i.e., pre-optimized linear regression, which increases the possibility of finding a new good part to add to the program. The parsimony pressure is the same for both methods and the f_{MT} representation produces less complex programs, simply because it fails to find more complex programs with a higher performance.

The fact that f_{OMT} outperforms MREG could of course be credited to the generalization capabilities of the ensemble. However, since f_{MT} has worse predictive performance than MREG, this cannot be the only reason why f_{OMT} outperforms MREG. It is more likely that increased performance is due to the local search of the linear regressions. Finally, it should be noted that even if the result compared to CART is not significant, f_{OMT} is still clearly superior and wins over CART on seven of eight datasets.

Summary

Even if GP should, in theory, be able to evolve more powerful programs using a more complex representation, this is always not true in practice. A more complex unrestricted representation creates a larger search space which requires a larger population evolved during more generations. Extending the population and the number of generations naturally leads to longer running times, the performance of which in a reasonable time is not always practical. An alternative is of course to use a greedy non-backtracking algorithm, but this will more often lead to a suboptimal model, since the search is not global.

This result of this study suggests a middle way, to use simple greedy models (linear regression) as building blocks for the GP. The experiments show that the suggested technique clearly outperforms traditional GP and multiple linear regressions, by winning on seven of eight datasets against each technique.

6.2 Enhancing accuracy by injection of decision trees.

In "*Improving GP performance by injection of decision trees*", König, Johansson, Löfström and Niklasson (2010) suggest a hybrid technique especially designed to handle the challenges demonstrated in section 5.2.1:

- The GP search is very powerful and may produce overfitted programs.
- A too high parsimony pressure may have prevented a search among sufficiently complex programs.
- The search space was too large to be fully explored by the GP search in a reasonable time.

Overfitting is a well-known problem of all machine learning algorithms and has been extensively studied, without reaching a consensus regarding the best general approach, see section 2.4.1. The most common way to handle overfitting is, however, to use a separate validation dataset to judge when the training should be stopped or to choose a model among several alternative models. This approach is, however, most suitable for problems with large amounts of data, since a portion of the training instances must be set aside for the validation dataset.

Choosing parsimony pressure is another well-known challenge normally handled by performing a set of initial experiments. Typically, several programs are evolved, using different parsimony pressures, and evaluated using a holdout sample. The procedure is often rather time consuming, especially if all reasonable parsimony pressures should be evaluated. When a simple program is sufficient, this procedure may perform many unnecessary evolutions evaluating parsimony pressures, resulting in excessively complex programs. Furthermore, when more complex programs are required, the procedure may be error prone, since a larger search space requires a larger population evaluated over more generations than are practical to use in initial experiments.

Large search spaces are the last of the listed challenges and maybe the one that is furthest from a good solution. Since the flexibility of GP makes it easy to combine with other techniques, most methods suggested for improving the GP search are some kind of hybrid system. Numerous variations of GP classification systems exist today, including decision trees, artificial neural networks, fuzzy logic, and support vector machines. However, the related work only presents some typical hybrid methods for evolving decision trees, since decision trees are the main focus of this thesis.

The technique proposed by König, Johansson, Löfström and Niklasson (2010) is based on the main idea of using decision tree performance as a benchmark to facilitate a dynamic adjustment of the parsimony pressure. Furthermore, if the GP

search do not reach at least a comparable performance, decision trees are injected into the population to guide the search into more fertile regions of the search space. The technique is described in detail after the related work has been discussed.

Related work

Langdon and Nordin, (2004), used copies of an unpruned C4.5 tree as the initial population. The aim was to evolve a smaller, more general version of the decision tree. Their experiments showed that the GP succeeded in finding smaller, more general trees, but it is unclear if the evolved trees were better than standard pruned C4.5 trees.

In another similar study, Lee (2006) used decision trees to create the initial GP population. More specifically, each individual in the population was created using C4.5. To achieve some diversity, each C4.5 tree was created from a random subset of the original training set. In the reported experimentation, the suggested method performs remarkably well, compared to the other techniques. This is very surprising, since CART reaches the same training accuracy but has almost 20% lower test accuracy. Since both techniques use almost identical representations, and the results are averaged over five folds, more similar accuracies should be expected. On one fold, both CART and the hybrid method achieve 100% accuracy on the training data, but CART only reaches 69% on the test data compared to 90% for the hybrid method. Furthermore, the benefit of using C4.5 as creation method is hard to evaluate based on this study alone, simply because the results for comparable standalone runs of normal GP and C4.5 are lacking, i.e., there is no way of knowing if GP increased the accuracy of the original rules.

Yet another method utilizing C4.5 to create the initial population was presented by Oka and Zhao (2007). The aim of this study was to produce short and comprehensible rules. This was motivated by the statement that C4.5 produced much shorter and comprehensible rules, but could not be trained incrementally. The suggested method created the initial population in the same way as the previous study, but used C4.5 settings that created smaller trees. Experiments were performed on a digit recognition dataset, where the proposed method produced smaller and slightly more accurate trees, compared to a normal GP implementation. However, when the number of generations increased, the GP trees achieved the same accuracy as the hybrid method. Finally, it can be noted that GP in general produce comprehensible rules. A survey of thirteen related articles conducted by Espejo et al. (2010) showed that GP produced more comprehensible rules, than the competing technique, for 66% of 60 analyzed occasions.

Method

The proposed method is a novel hybrid technique called DT Injection GP ($_{DTI}GP_{ACC}$). The method starts by estimating a benchmark level for reasonable accuracy, based on decision tree performance on bootstrap samples of the training set. Next, a normal GP evolution is started with the aim of producing an accurate GP. At even intervals, the best GP in the population is evaluated against the accuracy benchmark. If the GP has higher accuracy than the benchmark, the evolution continues as normal until the maximum number of generations is reached. If the accuracy falls below the benchmark, two things happen. First, the fitness function is modified to allow larger GPs to be able to handle more relationships. Secondly, a decision tree with increasing size (trained on a bootstrap of the training data) is injected into the population.

The overall purpose of the method is to automatically force the GP to search for models of suitable size; i.e., as small as possible but not overly simplified. Due to this, after the threshold for reasonable accuracy is established, the GP starts out with small trees and a high length penalty, but is allowed to search for increasingly larger trees until the accuracy is higher than the threshold. In more detail, this is accomplished by decreasing the length penalty and injecting accurate models into the population. Once the level of acceptable accuracy is met by the GP, normal GP optimization is used for the rest of the search.

As described above, the method consists of three parts: estimating a reasonable level of accuracy, evolving GP trees, and conditionally injecting decision trees.

Each part is described in detail and motivated in the sections below. Evaluation of the method is done against normal GP, the decision tree algorithm used for the injected trees, and *J48*.

Estimating Reasonable Accuracy

To obtain trees that are as comprehensible as possible, a high parsimony pressure will initially be used to favor small solutions, regardless of the dataset. A smaller pressure could lead to unnecessarily complex and specialized trees that may generalize poorly, when the data contains noise. However, some datasets require a large complex solution and do not contain noise. In these cases, a more complex solution is actually required. A typical example is the UCI dataset Tic-Tac-Toe, which encodes the complete set of possible board configurations at the end of tic-tac-toe games. Since no noise exists, and there is a true underlying concept, a larger solution will most often perform better.

However, since the true complexity of the solution is not known, it is impossible to directly set the parsimony pressure in an optimal way. Instead, the proposed method will dynamically lower the pressure, until a reasonable accuracy is achieved.

Several GP runs, with varied parsimony pressure, could of course be used to find the best setting, but since GP is computationally expensive, this is normally not practical. Decision trees are, however, fast to train and usually perform quite well.

To obtain a good approximation of the performance, 10 bootstrap samples are first drawn from the training set. Training instances not selected for a specific bootstrap are used instead as an out-of-bag test set for that bootstrap. Next, decision trees are created using jaDTi (Francois 2004), an open source implementation based on the C4.5 algorithm. The main difference to the real C4.5 algorithm is that jaDTi uses pre-pruning based on an entropy threshold, and a score threshold instead of the usual subtree based post pruning of C4.5. The entropy threshold is set to .5 and the score threshold is set to .01* the size of the dataset, i.e., the number of instances in the dataset. When all trees have been created, both average training and test accuracies are calculated. However, since reasonable training accuracy is sought, the mean test accuracy would be a value that is too pessimistic, and the mean training accuracy would be too optimistic. Instead, the average of these values is used as the benchmark of reasonable training accuracy for the GP system.

Results

Table 38 shows the average test accuracies and ranks over 4 folds for each of the tested techniques. The best result for each dataset is marked with bold letters. $_{DTF}ACC$ performs best and has the highest score (with three split victories) on 10 of the 18 datasets. Normal GP is the second best technique, with 7 overall wins, including two split victories. $_{DTF}ACC$ also has the highest mean accuracy, followed by normal GP, J48, and jaDTi.

Dataset	ACC				Ranks			
	jaDTi	J48	f_{ACC}	$_{DN}f_{ACC}$	jaDTi	J48	f_{ACC}	$_{DN}f_{ACC}$
Breast-cancer	.688	.699	.696	**.717**	4	2	3	1
Breast-w	.936	**.956**	.946	.947	4	1	3	2
Colic	.754	**.860**	.739	.826	3	1	4	2
Credit-a	.822	.855	**.867**	.864	4	3	1	2
Credit-g	.677	**.730**	.723	.720	4	1	2	3
Cylinder-bands	**.715**	.578	.665	**.715**	1	4	3	1
Diabetes	.680	.733	**.737**	**.737**	4	3	1	1
Ecoli	.753	.786	**.815**	.795	4	3	1	2
Glass	**.673**	.668	.612	.673	1	3	4	1
Haberman	.676	.739	**.745**	.735	4	2	1	3
Heart-c	.726	.759	.782	**.789**	4	3	2	1
Heart-statlog	.726	.770	**.800**	.778	4	3	1	2
Hepatitis	.735	.806	.735	**.832**	3	2	3	1
Iris	.940	.953	.947	**.960**	4	2	3	1
Labor	.790	.737	.895	**.895**	3	4	1	1
Liver-disorders	.638	.620	.620	**.641**	2	3	3	1
Lymph	.784	.730	**.797**	.784	2	4	1	2
Tic-tac-toe	**.936**	.849	.788	.899	1	3	4	2
Average	.758	.768	.773	.795	3.11	2.61	2.28	1.61

Table 38 – Average Accuracy (CD=1.03)

A *Bonferroni-Dunn* test yields a critical distance of 1.03 at a 95% confidence level, which shows that $_{DN}f_{ACC}$ is significantly more accurate than *jaDTi* and extremely close to being significantly better than *J48*. In fact, a pairwise *Wilcoxon* signed rank test shows that the difference between $_{DN}f_{ACC}$ and *J48* is significant, see Table 39.

Dataset	ACC			
	jaDTi	J48	f_{ACC}	$_{DN}f_{ACC}$
jaDTi	-	.360	0.202	0.002
J48		-	0.723	0.020
f_{ACC}			-	0.098

Table 39 – p-values for a wilcoxon sign test

In terms of tree size, calculated as number of nodes, there are no differences that are even close to being significant. However, f_{ACC} produces trees that on average almost have the size compared to the other techniques.

Dataset	Size				Ranks			
	jaDTi	J48	f_{ACC}	$_{DT}f_{ACC}$	jaDTi	J48	f_{ACC}	$_{DT}f_{ACC}$
Breast-cancer	49	6	45	17	4	1	3	2
Breast-w	12	25	29	9	2	3	4	1
Colic	61	5	25	5	4	1	3	1
Credit-a	32	42	9	61	2	3	1	4
Credit-g	28	140	29	57	1	4	2	3
Cylinder-bands	2	1	17	121	2	1	3	4
Diabetes	42	39	45	36	3	2	4	1
Ecoli	57	43	21	72	3	2	1	4
Glass	53	59	33	73	2	3	1	4
Haberman	74	25	21	25	4	2	1	2
Heart-c	31	51	49	109	1	3	2	4
Heart-statlog	36	35	25	37	3	2	1	4
Hepatitis	29	21	17	17	4	3	1	1
Iris	24	9	13	21	4	1	2	3
Labor	18	5	21	9	3	1	4	2
Liver-disorders	144	51	49	173	3	2	1	4
Lymph	93	34	17	45	4	2	1	3
Tic-tac-toe	89	141	13	41	3	4	1	2
Average	48.6	40.7	26.6	51.6	2.9	2.2	2.0	2.7

Table 40 – Average Complexity (CD=1.03)

Analysis

It is interesting to note that $_{DT}f_{ACC}$ is 11% better than normal GP on the Tic-Tac-Toe dataset. Obviously, this is due to the high performance of jaDTi, which reaches an even higher accuracy by building a very large tree. Since this dataset does not contain any noise, a bigger tree will almost always be better. However, this is exactly what $_{DT}f_{ACC}$ was designed to achieve. Since the benchmark for reasonable performance is also calculated using jaDTi, it becomes rather high and forces $_{DT}f_{ACC}$ to increase its training accuracy by lowering the parsimony pressure and injecting larger jaDTi trees. The exact same scenario also happens on *cylinder-bands* and glass, where jaDTi clearly outperforms normal GP, thus forcing the GP to aim for higher training accuracy by using larger programs.

It is not surprising that $_{DTI}f_{ACC}$ accuracy increases when jaDTi outperforms normal GP, since similar trees will be injected into the population. However, $_{DTI}f_{ACC}$ is not bounded by the accuracy of jaDTi. As seen in Table 38, jaDTi only outperforms normal GP on five datasets, but $_{DTI}f_{ACC}$ has 10 wins over normal GP. In fact, $_{DTI}f_{ACC}$ is not bounded by the performance of either technique, since it has seven outright wins, when compared to jaDTi and normal GP. Colic and Hepatitis are two good examples where $_{DTI}f_{ACC}$ wins over both techniques with large margins (7% and 10%, respectively).

J48 is significantly worse than $_{DTI}f_{ACC}$, but still better than jaDTi on 12 datasets. J48 wins over jaDTi, probably due to the more advanced pruning algorithm, since both use a creation method based on C4.5. This result is interesting, since it shows a possible improvement by using J48 in $_{DTI}f_{ACC}$ instead of jaDTi.

Finally, it is interesting to note that normal GP and J48 perform comparably with nine and eight wins respectively (with one draw). Again, the settings used for normal GP are not sufficient to constantly outperform decision tree algorithms when evaluated on ACC. In that light, the fact that $_{DTI}f_{ACC}$ significantly outperforms J48 must be considered a strong result. Furthermore, $_{DTI}f_{ACC}$ also clearly performs better than normal GP, losing on only 6 datasets of 18 against normal GP. In addition, in all six losses, the differences in accuracy obtained by the two techniques are all less than 2%. On the other hand, $_{DTI}f_{ACC}$ has 10 wins against normal GP, where five of these wins are between 5% and 10% better than normal GP.

Even if the aim of this study was not to evaluate the size of the produced trees, initial results presented in Table 40 show that $_{DTI}f_{ACC}$ generally evolved trees that were shorter than trees produced by jaDTi and J48, but slightly larger than the original GP. The average complexity was f_{ACC}=*24.2*, $_{DTI}f_{ACC}$=*35*, *J48=41.9* and *jaDTi=46.5.* However, these results disregard *Liver-disorders, Heart-c,* and *Cylinder-bands* where $_{DTI}f_{ACC}$ creates quite large programs. For *Liver-disorders,* this can be explained by the fact that the injection algorithm created very large trees and for *Heart-c* and *Cylinder-bands* it is probably due to bloating caused by a parsimony pressure that is too low. Of course, the increased ACC could be contributed to the higher complexity, but for these three datasets it is only *Cylinder-bands* that show a big difference in ACC compared to the result of the original GP.

Even if the technique is not perfect with regard to the complexity of the produced trees, they are still very promising results, since $_{DTI}f_{ACC}$ clearly outperforms the other techniques with regards to accuracy.

Summary

The main strength of *DTfACC* is that it increases the robustness of normal GP, thus performing well over a large range of problems. In the experimentation, *DTfACC* was significantly better than J48, and clearly outperformed normal GP.

The robustness of *DTfACC* comes from the use of a benchmark level for reasonable training accuracy, but also from the injection of decision trees and an adjustable parsimony pressure. It is clear that the use of the benchmark greatly improves *DTfACC*'s result in at least three of the datasets where normal GP performs poorly. Nevertheless, the benchmark is not enough in itself when the search space is too big. Hence, a very important feature of *DTfACC* is the injection of accurate trees to help the evolution, when the benchmark training accuracy cannot be reached.

Lowering the parsimony pressure and injecting jaDTi trees greatly improve the robustness and accuracy of *DTfACC*. Furthermore, the method not only improves the results on hard datasets, where standard GP performs poorly, but also shows to increase the accuracy, even when GP outperformed jaDTi. All in all, the experiments clearly show that the hybrid solution of injecting decision trees into a GP population gives synergetic effects, resulting in higher accuracies than using the basic techniques separately.

6.3 Turning Challenges into Advantages by Exploiting Instability

As described in section 5.2.2, even deterministic techniques are unstable in practice, due to informational instability, which will almost always be present in real-world applications. Hence, there is no idea in trying to enforce a more deterministic behavior of GP to only produce a single program. Instead, new techniques that make use of the instability are the only reasonable solution. The following section proposes three novel techniques that do exactly this, i.e., a method that identifies the best of a set of good solutions, a technique that improves probability estimates by considering alternative solutions, and an approach that evolves accurate *k-NN* ensembles.

6.3.1 Enhancing accuracy by selecting the best of the best

Section 5.2.2 shows that there are normally numerous alternative models with the same accuracy for a certain dataset. Nevertheless, there are situations where a decision maker wants to use only one comprehensible model. Naturally, this model should then be "the best possible". Exactly what criterion this translates into is not obvious, but, as always, test accuracy, i.e., the ability to generalize to novel data, must be considered very important. So, the overall question addressed here (based on the article "*Using imaginary ensembles to select GP classifier*", by Johansson, König, Löfström & Niklasson(2010)) is whether access to a number of evolved decision tree models will make it easier to select one, single model, likely to have good test accuracy.

Given a pool of independently trained (or evolved) models, there are two basic strategies for producing one model; either you somehow use the trained models to create a brand new model or you choose one of the available models. The first strategy could, for example, be approached with rule extraction, by considering the available models as an ensemble, e.g., as described in section 5.1.2.

Here, the second strategy is investigated, i.e., how do we choose a specific model from the pool. The most straightforward alternative is, of course, to compare all models and select the one with the highest accuracy on either training data or on an additional (validation) dataset. However, the hypothesis tested here is that we can do better by somehow using the fact that we have a number of models available.

Related Work

Section 2.2.5 describes ensembles and how uncorrelated errors made by the ensemble members lead to a superior accuracy of the ensemble, compared to the individual models used in the ensemble. In addition, an ensemble could also be used to generate predictions for novel instances with unknown target values, as they become available. Since the ensembles, in general, are very accurate, these "extra"

predictions can then be used as extra training instances when training a single model. In the field of semi-supervised learning, this is referred to as *coaching*. Specifically, ensemble predictions could be produced even for the test instances, as long as the problem is one where predictions are made for sets of instances, rather than one instance at a time. Fortunately, in most real-world data mining projects, bulk predictions are made, and there is no shortage of unlabeled instances. As an example, when a predictive model is used to determine the recipients of a marketing campaign, the test set, i.e., the dataset actually used for the predictions, could easily contain thousands of instances.

It must be noted that to use the test instances in this way is not "cheating". Specifically, it is not assumed that the actual values of the target variable are available for the test instances. Instead, new semi-artificial training instances are created with the prediction of the ensemble as target instead of the missing actual value. In this way, new instances will cover a larger part of the instance space and will thus hopefully facilitate the creation of more accurate models.

Method

In this specific setting, the ultimate goal is to choose one specific model (base classifier) from a set of models. The main idea is to investigate whether it is better to pick a model that agrees as much as possible with the (imaginary) ensemble consisting of all available models, or use the standard approach of selecting the one with the highest possible individual training- or validation accuracy. Furthermore, the coaching approach of creating semi-artificial instances using the ensemble predictions for test instances is also evaluated.

Four different selection strategies are evaluated in the experiments. Three of these use imaginary ensembles and the concept ensemble fidelity, which is defined as the number of instances classified identically to the ensemble.

- TrnAcc: The tree with the highest training accuracy is selected.
- TrnFid: The tree with the highest ensemble fidelity on training data is selected.
- TstFid: The tree with the highest ensemble fidelity on test data is selected.
- AllFid: The tree with the highest ensemble fidelity on both training data and test data is selected.

In addition, the average accuracies of the base classifiers (which correspond to evolving only one tree or picking one of the trees at random) and results for the J48 algorithm as implemented in WEKA are also presented.

Some may argue that most data miners would not select a model based on high training accuracy, but would instead use a separate validation set; i.e., a dataset not used for training the models. However, for small datasets like the UCI datasets used in the experiments, this approach may have negative consequences, since fewer training instances may result in less accurate models.

Nevertheless, since the separate validation approach is very common, a preliminary experiment evaluating the selection of models based on validation accuracy (ValAcc) against TrnAcc is performed first.

Results

For the experimentation, 25 datasets and 4-fold cross-validation was used. The reported test set accuracies are therefore averaged over the four folds. For each fold, 15 decision trees were evolved. If several trees had the best score (according to the selection strategy), the result for that strategy was the average test set accuracy of these trees. All selection strategies except ValAcc used the same pool of trees, where each tree was evolved using all available training instances. When using ValAcc, 75% of the available training instances were used for the actual training and the remaining 25% for validation. The 25 datasets used are all publicly available from the UCI Repository. Each tree was evolved using the typical G-REX setting according to Table 41. J48, which is an implementation of the C4.5 algorithm, used default settings in WEKA.

Number of generations	100
Population size	1000
Crossover probability	.8
Mutation probability	.001
Creation type	Ramped half -and -half
Selection type	Roulette wheel
Maximum creation depth	6
Parsimony pressure(P)	.001
Batch size	1

Table 41 – G-REX settings for creation of imaginary ensembles

Table 42 below shows the results from the preliminary experiment. The main result is that it is clearly better to use all data for actual training, rather than reserving some instances to be used as a validation set.

Datasets	TrnAcc	ValAcc	Rnd	Rnd75
Breast-cancer	**.727**	.706	**.742**	.703
Breast-w	**.962**	.958	**.957**	.953
Cmc	**.550**	.537	**.519**	.516
Colic	.836	**.849**	**.845**	.834
Credit-a	.845	**.855**	.847	**.851**
Credit-g	**.726**	.693	**.710**	.694
Cylinder-bands	**.694**	.676	**.673**	.655
Diabetes	**.734**	**.743**	**.738**	**.733**
Ecoli	**.813**	.802	**.790**	.780
Glass	.654	**.666**	**.649**	.636
Haberman	.733	**.736**	.734	**.749**
Heart-c	.767	**.774**	**.763**	.757
Heart-statlog	.773	**.777**	**.770**	.752
Hepatitis	**.825**	.811	**.815**	.812
Iono	**.890**	.867	**.873**	.871
Iris	**.960**	.942	.943	**.944**
Labor	**.850**	.848	**.863**	.838
Liver-disorders	.609	**.636**	**.629**	.607
Lung-cancer	**.670**	.573	**.677**	.579
Lymph	**.757**	.735	**.760**	**.760**
Sonar	**.760**	.721	**.732**	.717
Tae	**.543**	.525	**.554**	.542
Vote	.932	**.940**	.947	**.948**
Wine	.884	**.909**	**.904**	.900
Zoo	**.934**	.909	**.919**	.884
Mean	**.777**	.768	**.774**	.761
p-value	.074		3.94e-4	

Table 42 – Comparing selection based on train- and validation accuracy

Comparing TrnAcc to ValAcc, TrnAcc wins 15 of 25 datasets, and also obtains a higher mean accuracy over all datasets. A Wilcoxon test yields a *p-value* of *.0735,* which is very close to being a significant result. Not surprisingly, it is significantly better, i.e. *p-value =3.94e-4*, to use all available instances (Rnd) than to use 75% of the instances for the training (Rnd75) and the rest for validation, i.e., Rnd only loses against Rnd75 on four of 25 datasets.

Table 43 below shows the results from the main experiment. The overall picture is that all selection strategies outperform both J48 and *Rnd*; i.e., picking a random tree.

Dataset	J48	TrnAcc	TrnFid	TstFid	AllFid	Rnd
Breast-cancer	.699	.727	**.748**	**.748**	**.748**	.742
Breast-w	.956	.962	**.966**	.964	.965	.957
Cmc	.506	**.550**	.517	.517	.517	.519
Colic	**.856**	.836	.843	.842	.842	.845
Credit-a	.855	.845	**.860**	.851	**.860**	.847
Credit-g	**.730**	.726	.715	.724	.719	.710
Cylinder-bands	.578	.694	.674	**.700**	.676	.673
Diabetes	.733	.734	.734	.734	.734	**.738**
Ecoli	.786	.813	.816	**.819**	.813	.790
Glass	.668	.654	**.687**	.682	.678	.649
Haberman	.739	.733	.740	**.745**	.742	.734
Heart-c	.759	.767	.768	**.776**	.767	.763
Heart-statlog	.770	.773	.767	**.781**	.778	.770
Hepatitis	.806	.825	**.830**	.817	.816	.815
Iono	.900	.890	**.907**	.902	.904	.873
Iris	.953	**.960**	.948	.947	.947	.943
Labor	.737	.850	.850	**.876**	**.876**	.863
Liver-disorders	.620	.609	**.681**	**.681**	**.681**	.629
Lung-cancer	.656	.670	.670	**.781**	**.781**	.677
Lymph	.730	.757	**.777**	.764	**.777**	.760
Sonar	.755	**.760**	.745	.728	.748	.732
Tae	.550	.543	.550	.550	.550	**.554**
Vote	.956	.932	.951	**.957**	.955	.947
Wine	.916	.884	.899	**.938**	.930	.904
Zoo	.931	.934	.914	**.970**	.965	.919
Mean	.766	.777	.782	.792	.791	.774
#Wins	2	3	8	12	6	2
Mean rank	4.34	3.98	3.12	2.48	2.76	4.28

Table 43 – Evaluation of selection based on ensemble fidelity

The best results were obtained by the selection strategy utilizing the imaginary ensemble on only the test instances.To determine if there are any statistically significant differences, a Friedman test, followed by a Nemenyi post-hoc test was performed. With six classifiers and 25 datasets, the critical distance (for α=0.05) is 1.51, therefore, based on these tests, TstFid and AllFid obtained significantly higher accuracies than J48 and *Rnd*. Furthermore, the difference between TstFid and TrnAcc is very close to being significant at α=0.05. Finally, TrnAcc is slightly better than choosing a tree at random, but the difference is not significant.

Since the most common and straightforward alternative is to simply use a decision tree, it is particularly interesting to compare the suggested approach to J48. As seen in Table 44, all selection strategies clearly outperform J48. A Wilcoxon sign rank test yields that the selection strategies TrnFid, TeFid, and AllFid are significantly better than J48 at *a=0.05*, also when using pair-wise comparisons.

	TrnAcc	TrnFid	TstFid	AllFid	Rnd
J48	.3394	.0454	.0027	.0026	0.8751

Table 44 – Win draws and losses against J48

Another interesting comparison is to look specifically at selection strategies not considering test instances. After all, not all predictions are performed in bulk. Table 45 below therefore compares J48 to TrnAcc, TrnFid, and Rnd. Even now the picture is clear; it is significantly better to select a tree based on ensemble fidelity.

Strategy	J48	Rnd	TrnAcc
TrnFid	.0454	.0001	.0145
TrnAcc	.3394	.3393	
Rnd	.8751		

Table 45 – Comparing non bulk selection strategies

Analysis

The results of the first experiments clearly show that it is better to use all data for training than setting aside data for a validation set. Models produced using all training data were only worse than *rnd75* on four of the 25 datasets. Obviously, it is the considerably higher average accuracy of the models that makes TrnACC superior to ValAcc. It should be noted, however, that the UCI datasets are all quite small, which is favorable for TrnACC, but for larger datasets the result would probably be different.

All in all, it is obvious that basing the selection on fidelity to the imaginary ensemble is beneficial, especially when also considering the ensemble predictions on test instances. Interestingly enough, this is despite the fact that J48 wins almost half of the datasets when compared to a random tree. So, the success of the suggested approach must be credited to the selection strategies, rather than having only very accurate base classifiers.

A potential objection to the suggested approach is that using evolutionary algorithms to produce decision trees is much more computationally intense and, therefore, slower than using standard techniques. This is certainly true, but the

novel part, i.e., the evolution of several trees just to pick one, could easily be run in parallel, making the entire process no more complex and time consuming than evolving just one tree.

In this study, GP was used to evolve all decision trees. However, the suggested approach is also potentially applicable to standard algorithms like C4.5 and CART. The most natural setting would probably be quite large datasets, where each tree would have to be induced using a sample of all available instances, thereby introducing some implicit diversity.

Summary

The performed experiments clearly show that the predictive performance of genetically evolved decision trees can compete successfully with trees induced by more specialized machine learning techniques, here J48. The main advantage for an evolutionary approach is the inherent instability that facilitates the creation of a number of decision trees without having to sacrifice individual accuracy to obtain diversity. Each tree was evolved on the basis of all available training data, which is in contrast to standard techniques that normally have to rely on some resampling technique to produce diversity. Consequently, the proposed method evolves a collection of accurate yet diverse decision trees, and then uses some selection strategy to choose one specific tree from that pool. The key idea, suggested here, is to form an imaginary ensemble of all trees in the pool, and then base the selection strategy on that ensemble. Naturally, the assumption is that individual trees which, to a large extent, agree with the ensemble are more likely to generalize well to novel data. In the experimentation, the use of several selection strategies produced trees significantly more accurate than the standard rule inducer J48. The best performance was achieved by selection strategies utilizing the imaginary ensemble on actual predictions, thus limiting the applicability to problems where predictions are made for sets of instances. Nevertheless, the results also show that even when bulk predictions are not possible, the suggested approach still outperformed J48.

6.3.2 Improving probability estimates

Although the normal operation of a decision tree is to predict a class label based on an input vector, decision trees can also be used to produce class membership probabilities; in which case they are referred to as probability estimation trees (PETs). Probability estimates are important, since uncertain predictions may be analyzed and adjusted by a domain expert, especially if they are done by a comprehensible model. In this way, some prediction errors may be avoided and a user's confidence in the model may even increase.

The easiest way to obtain a class probability is to use the proportion of training instances corresponding to a specific class in each leaf. If 50 training instances reach a specific leaf, and 45 of those belong to class A, the assigned class probability for class A, in that leaf, would become 0.9. Consequently, a future test instance classified by that leaf would be classified as class A, with the probability estimator 0.9.

Normally, however, a more refined technique, called Laplace estimate or Laplace correction is applied; see section 2.2.2. The main reason is that the basic, maximum-likelihood estimate, described above, does not consider the number of training instances reaching a specific leaf. Intuitively, a leaf containing many training instances is a better estimator of class membership probabilities.

The purpose of this section, which is based on work by Johansson, König and Niklasson (2007), is to show that the inherent inconsistency of GP may actually be an advantage, when calculating probability estimates.

Related work

Provost and Domingos (2003) created what they called "well behaved probability estimate trees" (PETs) by combining the Laplace estimator with the standard ensemble creation technique bagging. More specifically, an ensemble consisting of several, independently trained PETs was created and final probability estimates were computed by averaging the individual probability estimates from all ensemble members. To obtain some diversity, each PET was trained using only part of the training data, here a bootstrap sample. So, if the Laplace estimator for a certain leaf p_l was:

$$p_l(k) = \frac{N_k + \lambda_k}{N + \sum_{k=1}^{K} \lambda_k} \tag{49}$$

where N_k is the number of instances belonging to class k, N is the total number of instances reaching the leaf and λ_k is the prior probability for class k, the ensemble probability estimates p_e for instance i and class k were calculated using (50) below, where L signifies the leaf in each base model that classifies the instance i.

$$p_e(i|k) = \frac{1}{L} \sum_{l=1}^{L} p_l(k) \tag{50}$$

In order to obtain what they call well-behaved PETs, Provost and Domingos changed the C4.5 algorithm by turning off both pruning and the collapsing

mechanism. This obviously led to much larger trees, which, in fact, turned out to be much better PETs.

Method

The general technique investigated in this section is similar to the one used by Provost and Domingos (2003) in that it uses an ensemble to compute the probability estimates. However, here the ensemble is only used to calculate the probability estimates and a single model is selected on the basis of training accuracy, for the actual classification. In this way, the classification model will remain comprehensible, since a single comprehensible model is used. The reasons for the probability estimates will, however, be opaque, since they are calculated using the ensemble. Furthermore, GPs inherent inconsistency is exploited to create diverse programs using the entire training set. Deterministic techniques, such as C4.5 and CART which are used as a benchmark, must use some kind of subset of the training set to create diverse classifiers and will thus, in general, be less accurate for all but huge datasets. Here, a subset of only 70% randomly selected instances is used to create each CART tree in the ensemble.

Both the programs evolved using GP and the trees created using CART calculate their uncertainty estimates using the Laplace estimate, based on the distribution of the training instances in the leaves.

The experimentation is performed in three stages to compare the normal performance of CART and G-REX, the uncertainty estimates using a single model and uncertainty estimates calculated using an ensemble. Standard performance is calculated as the accuracy of a single CART tree and a GP program trained on the whole training set. The same trees are then evaluated on the basis of their uncertainty estimates, using the accuracy of the 80% of the instances that each technique ranked as least uncertain, here called Top80. Of course, any percentage could have been used for the evaluation of the uncertainty estimates; here, 80% was chosen to allow the techniques to discard a sufficient number of instances to facilitate clear differences in the evaluation.

Finally, an ensemble is created for each technique as described above. A single model is selected for the classification of the Top80, based on the respective uncertainty estimates of each technique.

All GP programs were extracted from an ensemble of 20 ANNs. All ANNs are fully connected feed-forward networks; 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 based on the number of features and classes in the current dataset. For an ANN with one hidden layer, the number of hidden units is determined from equation 51 below.

$$h = \left\lfloor 2 * rnd * \sqrt{v * c} \right\rfloor \quad (51)$$

v is the number of input features and c is the number of classes. rnd 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 layer is determined using equations 52 and 53, respectively.

$$h_1 = \left\lfloor \frac{\sqrt{v * c}}{2} + 4 * rnd * \frac{\sqrt{v * c}}{c} \right\rfloor \quad (52)$$

$$h_2 = \left\lfloor rnd * \frac{\sqrt{v * c}}{c} + c \right\rfloor \quad (53)$$

To further increase the diversity, each ANN is trained using only 80% (randomized without replacement) of the available training instances.

The GP programs are extracted from the ensembles using f_{XACC}, see equation 45, with the settings presented in Table 46.

Number of generations	1000
Population size	1000
Crossover probability	.8
Mutation probability	.001
Creation type	Ramped half -and -half
Selection type	Roulette wheel
Maximum creation depth	8
Parsimony pressure (P)	0.01
Batch size	1

Table 46 – G-REX settings for improved uncertainty estimates

Results

Ten UCI-datasets were used to compare the different techniques and the results presented in Table 47 were produced using ten-fold cross-validation. Bold numbers are used to highlight the best result in each double column

Dataset	ACC		ACC Top80		Ensemble ACC Top80	
	CART	*fxACC*	CART	*fxACC*	CART	*fxACC*
Breast-w	.937	**.977**	.971	**.986**	.995	**.996**
Credit-g	**.745**	**.745**	.753	**.767**	**.791**	.786
Diabetes	.728	**.762**	.744	**.749**	.780	**.815**
Heart-c	.800	**.847**	.812	**.868**	.848	**.904**
Ionosphere	**.897**	**.960**	.928	**.952**	.948	**.983**
Liver-disorders	**.668**	**.668**	**.707**	**.707**	**.721**	.704
Lymph	.771	**.857**	.783	**.875**	.833	**.917**
Sonar	.660	**.895**	.618	**.894**	.741	**.929**
Wine	.912	**.982**	.893	**.979**	.979	**1.00**
Zoo	.880	**.950**	.933	**.967**	.944	**.978**
Mean:	.800	**.864**	.814	**.874**	.858	**.901**

Table 47 – Top80 accuracy for probability estimates

There are several interesting results in the table above. First, it is apparent that the programs extracted using *fxACC* are more accurate than its CART counterpart. Single *fxACC* programs outperform single CART trees with eight wins and two draws, in terms of accuracy, and nine wins and one draw, when compared using Top80. Of course, these results are significant which can be seen in Table 48 below, which presents the *p*-value of a Wilcoxon test when single *fxACC* and CART models are compared to the other techniques.

Single model All	Single model Top80		Ensemble Top80	
	CART	*fxACC*	Ensemble CART	Ensemble *fxACC*
fxACC	0.064	0.106	0.919	0.002
CART	0.185	0.002	0.002	0.002

Table 48 – Wilcoxon p-values for All vs. Top80

The uncertainty estimates for single models are quite accurate for both techniques, since the Top80 accuracy is significantly higher than the normal accuracy for CART and clearly higher for *fxACC* (*p=.106*). However, the uncertainty estimates produced by the CART and *fxACC* ensembles are even better and both techniques are significantly better than respective single model. It is also clear that the ensemble method works, since both the G-REX and CART ensembles are significantly better than a single model of respective type

Table 49 finally shows that the G-REX ensemble is in turn significantly better than all other techniques including a single G-REX tree which is the second best technique.

	CART	f_{XACC}	Ensemble CART
Ensemble f_{XACC}	0.004	0.004	0.020
Ensemble CART	0.002	0.636	
f_{XACC}	0.009		

Table 49 – Wilcoxon p-values for comparisons of Top80 results

Analysis

There are two main reasons that may have contributed to the superior performance of programs evolved using f_{XACC}, i.e., the global non-greedy search of GP and the fact that the programs were extracted from an ensemble. The inferior result of the CART ensemble with regard to Top80 may be further explained by the fact that the individual CART tree of the ensemble could only be trained on 70% of the instance. On the other hand, GP could use all training data for each program, since it is inherently inconsistent. Not surprisingly, experiments in section 6.3.1 also showed that it is in fact the accuracy that is significantly higher for models created using all training data rather than a subset.

Another important result is that the GP ensemble significantly outperforms the individual GP programs. Obviously, the inherent inconsistency is an advantage here, since the produced programs are not only accurate but apparently also diverse. A later study conducted by Johansson et al. (2009) did, in fact, confirm these results related to ensembles created using bagging with J48 and RepTree as base models. The result showed that the GP base classifiers were not only more accurate, but also more diverse with regard to the diversity measures *double fault* and *disagreement*.

Summary

There are two main conclusions that can be drawn from the experiment; when creating base classifiers for an ensemble, inconsistency is an advantage rather than a disadvantage and probability estimates produced using an ensemble are superior to those estimated using a single model. Hence, the proposed technique of creating a single model to make classifications and using an ensemble to estimate the uncertainty is very feasible. The proposed technique is especially well suited for GP, since ensemble base classifiers can be created using training data, thanks to the inherent inconsistency of GP. Finally, it should again be noted that the ensemble is

only used for the uncertainty estimates and hence does not affect the comprehensibility of the model used for classification.

6.3.3 Evolving accurate kNN Ensembles

Historically, kNN models have only rarely been combined into ensembles. The main reason for this is that kNN turns out to be very robust with respect to dataset variations, i.e., resampling methods like bagging and boosting, will normally only produce limited diversity. The following section instead examines a totally different approach where the inherent inconsistency of GP is used to create a kNN-ensemble with the necessary diversity. The work is based on the article "*Genetically Evolved kNN Ensembles*",* Johansson, König and Niklasson (2010b).

Related work

Based on section 2.2.5, the overall goal when designing ensembles seems to be fairly simple, i.e., to somehow combine models that are highly accurate but diverse. Base classifier accuracy and diversity are, however, highly correlated, so maximizing diversity will most likely reduce the average base classifier accuracy. According to Domeniconi and Yan (2004), kNN is very robust with respect to the type of dataset variations used in bagging, boosting, but is, at the same time, sensitive to the features and the distance function used. Hence, feature weighting or feature sampling seem to be more reasonable approaches for creating diversity. However, most often, feature reduction leads to lower base classifier accuracy.

As described in section 2.2.4, standard kNN is popular technique since it is both simple and in general performs well. However, it also suffer from some inherent drawbacks, i.e., that it is extremely sensitive to the *k*-value and restricted to a single *k*-value for the whole dataset. Wettschereck and Dietterich (1994), tried to overcome this problem by using locally optimized kNN, but experiments showed that local kNN methods in general, on real-world datasets, performed no better than standard kNN. G-kNN (presented in section 5.1.3) is, however, a local optimized kNN technique that succeeds in significantly outperforming standard kNN using a predefined *k*. More specifically, G-kNN uses GP to build decision trees, partitioning the input space into regions, where each leaf node (region) contains a kNN classifier with a locally optimized *k*.

*König and Johansson are equal contributors

Method

The aim of this study is to create accurate kNN ensembles. The basic idea is to exploit GP's inherent inconsistency, in order to create the necessary diversity of the ensemble. Naturally, the representation is also important for the amount of diversity that can be introduced in an ensemble. Hence, two types of base models are evaluated in the experiments; the first is a standard kNN with feature weights, while the second is an extended *global* G-kNN model with feature weights, see the BNF in Table 74 and Figure 36.

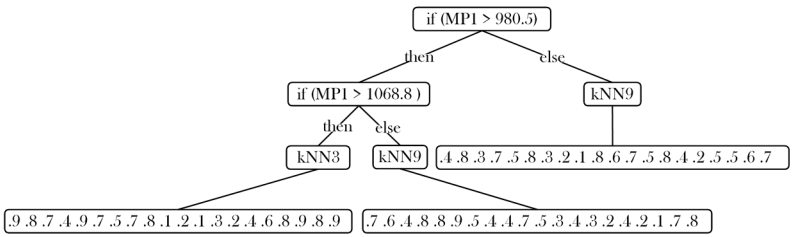

Figure 36 - KNW program for the Toyota IM dataset

Naturally, neither of these models require the number of neighbors to be considered as a parameter, but instead evolution is used to find optimal *k*-values and feature weights. However, only odd values in the range 1-21 are considered, which in practice means that there are 10 possible *k*-values and hence ten possible different base models for a kNN-ensemble.

Feature weighting will "stretch" the significant input axes, making them more important in the neighbor selection process. Here, all feature weights are between 0 and 1. Setting a weight to 0 will eliminate that dimension altogether. Weight vectors are initialized to random values and treated as a single primitive during optimization. Hence, weight vectors themselves are unchanged and the optimization only rearranges the position and frequency of the initial vectors. New weight vectors are, however, created continuously during the evolution, due to the GP mutation operation. In practice, this crude optimization of the weight vectors will introduce more diversity to the ensemble, since the vectors will most often differ slightly, even for kNN models optimized for the same set of training instances.

All attributes were preprocessed by linearly normalization to the interval [0, 1]. For numerical attributes, missing values were handled by replacing the missing value with the mean value of that attribute. For nominal attributes, missing values were replaced with the mode, i.e., the most common value. Standard Euclidean distance between feature vectors was used as distance measure. For nominal

attributes, the distance is 0 for identical values and 1 otherwise. Nominal and ordered categorical attributes could potentially be handled differently, since ordered attributes often use the same distance function as continuous attributes. Here, for simplicity, all attributes are handled as nominal.

Obviously, the key concept in this study is the use of GP, which not only facilitates the straightforward evaluation of different representation languages, but also makes it easy to introduce diversity among the base classifiers. As a matter of fact, in this specific context, the inherent inconsistency of GP (meaning that different runs on identical data can produce quite different models) must be regarded as an asset, since it makes obtaining a number of accurate, but still diverse, base classifiers uncomplicated.

Results

In the experimentation, all ensembles consisted of ten base classifiers, where each base classifier was optimized using all available training data. It must be noted that each base classifier was individually evolved, and that all parameters, including fitness function, were identical between the runs. The ensembles are compared to standard kNN, using majority and weighted voting, where k is found by cross-validation. For the actual evaluation, 4-fold cross-validation, measuring accuracy, was used.

Parameter	kNN base classifier	G-kNN base classifier
Number of generations	10	30
Population size	500	300
Crossover probability	.800	.800
Mutation probability	.001	.001
Creation type	N/A	Ramped half –and- half
Selection type	Roulette wheel	Roulette wheel
Maximum creation depth	N/A	5
Parsimony Pressure (P)	N/A	0.01
Batch size	1	1

Table 50 – G-REX settings for optimization of kNN- and GkNN ensembles

Since all evolved kNN models in this study were to be used in ensembles, relatively weak and small models were evolved. More specifically, the number of generations and the number of individuals in the GP population were kept quite small. Programs were optimized using f_{ACC} employing small length penalty, in order to encourage smaller and potentially more general models. For the exact GP parameters used in the experimentation see Table 50 above. Table 51 shows the accuracy and mean rank for all techniques, with the best marked in bold.

Dataset	Singel		Ensemble	
	kNN-ma	kNN-wv	kNN-wv	G-kNN
Breast-cancer	.717	**.745**	.741	.731
Breast-w	.966	.966	**.969**	**.969**
Cmc	.466	.475	.500	**.502**
Colic	.834	.832	.829	**.837**
Credit-a	.860	**.861**	.851	.855
Credit-g	**.742**	**.742**	.733	.735
Cylinder-bands	**.774**	.706	.754	.748
Dermatology	.954	.956	.956	**.962**
Diabetes	**.746**	.741	.736	.742
Ecoli	**.872**	.860	.860	.851
Glass	.673	.696	.687	**.697**
Haberman	.739	.719	.732	**.745**
Heart-c	.825	.825	.828	**.835**
Heart-statlog	.826	**.830**	.826	.822
Hepatitis	.845	.845	**.858**	**.858**
Ionosphere	.843	.846	**.877**	**.877**
Iris	.947	.940	.960	**.967**
Labor	.877	.877	.930	**.964**
Liver-disorders	.617	.606	**.673**	**.673**
Lymph	.804	.791	.824	**.838**
Post-operative	**.712**	**.712**	.644	.678
Primary-tumor	.422	.422	.425	**.431**
Sick	**.964**	**.964**	.957	.957
Sonar	**.851**	.803	.822	.822
Tae	.622	.556	.649	**.662**
Tic-tac-toe	.985	.985	**.992**	**.992**
Vehicle	.675	.677	.690	**.693**
Votes	.927	.924	.943	**.947**
Wine	**.972**	.972	.961	.966
Zoo	**.951**	.931	.930	.930
Mean	.800	.794	.805	**.810**
Mean Rank	2.68	2.82	2.52	**1.92**

Table 51 – Accuracy for kNN-models and ensembles (CD =.856)

Regarding the standard kNN techniques, the results show that it is, in fact, better to use straightforward majority voting. The most important result of Table 51 above is, of course, that the G-kNN ensemble clearly outperforms

the other techniques. The G-kNN ensemble is the best ranked method of all four on 18 of 30 datasets, which must be considered a strong result, and a Nemenyi test shows that the result is close to being significant at a 5% significance level (*CD= .856*).

A pair-wise Wilcoxon test does, however, show significant differences between several techniques, as can be seen in Table 52 below, which presents the *p*-value for each test. Consequently, the tests show that G-kNN ensembles were significantly more accurate than kNN-wv and kNN ensembles at a 5% significance level and against kNN-ma at a 7.5% significance level. In addition, the kNN-wv ensemble was significantly more accurate than kNN-wv, but far from significant when compared to kNN-ma.

	kNN-ma	kNN-wv	kNN ensemble
G-kNN ensemble	.075	.008	.010
kNN-wv ensemble	.289	.039	
kNN-wv	.117		

Table 52 – Wilcoxon p-values for comparisons of kNN- models and ensembles

In the second experiment, presented in Table 53, the accuracy of the base classifiers in each ensemble is evaluated against the ensemble performance.

Dataset	kNN-wv		G-kNN	
	Base	Ensemble	Base	Ensemble
Breast-cancer	.736	**.741**	.717	**.731**
Breast-w	.964	**.969**	.965	**.969**
Cmc	.491	**.500**	.490	**.502**
Colic	**.831**	.829	.827	**.837**
Credit-a	**.851**	**.851**	.850	**.855**
Credit-g	**.733**	**.733**	.733	**.735**
Cylinder-bands	.751	**.754**	**.746**	.748
Dermatology	.949	**.956**	.957	**.962**
Diabetes	.732	**.736**	.737	**.742**
Ecoli	.845	**.860**	.846	**.851**
Glass	.677	**.687**	.683	**.697**
Haberman	**.737**	.732	.736	**.745**
Heart-c	.823	**.828**	.829	**.835**
Heart-statlog	.823	**.826**	.815	**.822**
Hepatitis	.848	**.858**	.845	**.858**
Ionosphere	.870	**.877**	.866	**.877**
Iris	**.961**	.960	.957	**.967**
Labor	.928	**.930**	.935	**.964**
Liver-disorders	.672	**.673**	.668	**.673**
Lymph	.807	**.824**	.808	**.838**
Post-operative	**.649**	.644	.648	**.678**
Primary-tumor	.420	.425	.417	**.431**
Sick	.956	**.957**	.956	**.957**
Sonar	.819	**.822**	.805	**.822**
Tae	.633	**.649**	.618	**.662**
Tic-tac-toe	.987	**.992**	.987	**.992**
Vehicle	.671	**.690**	.676	**.693**
Votes	.942	**.943**	.941	**.947**
Wine	**.962**	.961	.965	**.966**
Zoo	**.937**	.930	**.935**	.930
Mean	.800	**.805**	.799	**.810**

Table 53 – Comparison of kNN-base models and ensembles

It is obvious that the base models are weaker than the ensembles. Or, put in another way, the ensemble approach really works. Wilcoxon tests also show that the ensembles are significantly more accurate than their base classifiers.

	G-kNN-base	kNN-vw ensemble
kNN-vw-base	.317	.001
G-kNN ensemble	.000	.010

Table 54 – Wilcoxon p-values for comparisons of kNN- models and ensembles

Analysis

The main result is, of course, that G-kNN ensembles clearly outperform both standard kNN using majority- and weighted voting and the kNN-ensemble technique. Since the accuracy of the G-kNN base models is lower than the ensemble accuracy and standard kNN using majority voting, the superior performance must be credited to the approach of using GP to optimize the individual ensemble members.

However, even if both ensembles are better than both individual standard kNN techniques, the representation of the base models is evidently important. The G-kNN representation clearly facilitates more diverse base models compared to the single global kNN model, since the G-kNN significantly outperforms the kNN ensemble, in spite of lower base classifier accuracy. This is a quite natural result, since a kNN-ensemble can only contain ten different models due to the restrictions on the possible *k*-values.

Another interesting observation is that genetically evolved base classifiers, on several datasets, perform worse than the best kNN-cv from the first experiment. However, this is not strange, since optimization of the weight vector was very crude and aimed to create more diversity rather than higher accuracy. If feature weights were used for a single G-kNN model, a different GP representation facilitating application of crossover and mutation during the evolution should be used.

Summary

Most importantly, GP has an inherent ability to produce several, quite different models, all having similar individual performance. Naturally, this is exactly what is sought when building ensembles, i.e., accurate, yet diverse base classifiers. One specific advantage of the GP ensemble approach is, of course, the possibility to optimize the arbitrary score function; i.e., see section 5.1.1. Another advantage is the fact that each model is still built using all features and all instances. This is in contrast to resampling methods, where implicit diversity is often introduced mainly by making all base classifiers less accurate, i.e., they do not have access to all relevant data. On the other hand, the different solutions produced by the GP are all models of the original problem.

In this study, two different kinds of genetically evolved base classifiers were evaluated. The first was a global kNN model, similar to standard kNN, but with optimized feature weights and an optimized k-value. The second was the G-kNN representation using feature weights, i.e., a decision tree where each leaf node contains optimized feature weights and an optimized k-value. In the experimentation, the G-kNN ensemble clearly outperformed both the corresponding base classifiers and standard kNN with optimized k-values. Comparing the two different ensemble versions, the G-kNN ensembles which used more complex decision tree models, potentially utilizing several locally optimized k-values, obtained significantly higher accuracy. Interestingly, this superior performance was achieved despite the fact that base classifiers had similar performance, indicating that the ensembles built from the more complex models had greater diversity.

7 Enhancing comprehensibility

Comprehensibility is an important property of predictive models, since opaque models are often met with skepticism by the users. Furthermore, comprehensibility facilitates verification of the model, which can be important for safety- or legal reasons, and rational adjustments of predictions in case there is a drastic change in the environment. GP is well suited for evolving comprehensible models, since it allows a decision maker to tailor the representation to his/her own preference, which is crucial as comprehensibility is naturally a subjective quality, i.e., see section 5.1.3. Additionally, parsimony pressure or rule extraction; i.e., see section 5.1.2, can also be applied to handle the accuracy vs. comprehensibility tradeoff and thus increase the comprehensibility of the evolved models.

However, GP most often produces models that contain introns, which add unnecessary complexity and thus lower the comprehensibility. Even if there are techniques for removing introns, they are based on algebraic simplification and are thus dependent on the representation of the evolved models. To better fulfill criterion 3.3 concerning removal of introns, section 7.1 presents a novel technique to remove introns independent of the underlying representation.

Another challenge for PM techniques is informational instability, i.e., the data is ambiguous in that several models with the same accuracy can be created. Most PM techniques are deterministic and will only produce a single model, in spite of informational instability, while GP, which is probabilistic in nature, may produce different models every run. Normally, this is considered a disadvantage for GP, but as seen in section 6.3, it can be utilized to increase the predictive performance. It can also be seen as an approach to criterion 4.1, i.e., the discovery of interesting rules by presenting several alternative solutions to the decision maker. However, since there are usually more alternative solutions than a decision maker could analyze manually, i.e., see section 5.2.2, section 7.2 presents a novel technique that guides the decision maker among the different solutions. This increases the comprehensibility by finding a better match between the decision maker's domain knowledge and the model, while at the same time promoting the discovery of truly interesting rules.

7.1 A simplification technique for enhanced comprehensibility

As previously explained in section 2.5.3, introns are a crucial part of the evolution. However, for the final program, introns only add unnecessary complexity and should therefore be removed. Several approaches based on algebra, the numerical approximation and their combination, have been suggested, but none is general enough to handle arbitrary representations. Consequently, they limit the GP's advantage of an arbitrary representation. Hence, the following section describes and evaluates a novel simplification technique, suggested by König, Johansson and Niklasson (2006), which is independent of the representation language.

Related work

When considering the simplification of GP programs, it is natural to first turn to the extensive research related to the pruning of decision trees. Bohanec and Bratko (1994) do, however, make a clear distinction between *pruning* and *simplification*; *pruning* considers trees as imperfect, due to noisy data, and prunes them so they are more accurate on future unseen instances, while *simplification* instead considers the trees as a perfect representation of the data and tries to make them more comprehensible. In other words, pruning creates smaller trees at the cost of lower training accuracy, while simplification tries to create smaller trees without loss of training accuracy.

Since decision tree algorithms are deterministic, standard pruning techniques like *Cost-Complexity Pruning* (Breiman, et al., 1982) or *Reduced Error Pruning* (Quinlan 1987) only consider pruning trees that are a subset of the original tree. GP, on the other hand, is non-deterministic and may produce different programs for each run. There is no reasonable argument that can motivate why a tree created using a decision tree algorithm should be the one that is most true to the underlying concept. It could hence be argued that limiting pruning and simplification of decision trees to subsets of the original tree is a suboptimal solution.

The effect of introns and how to best remove them have been studied from the beginning of GP. Koza (1992) suggested an editing operation that would continuously simplify expression during the evolution, according to predefined domain dependent rules. There are two main simplification approaches, algebraic and numerical. In algebraic approaches, for example, see (M. Zhang, Wong & Qian 2006), the program is seen as a mathematical expression and algebraic rules are applied step wise to simplify the expression. Numerical approaches, for example see (Kinzett, M. Zhang & Johnston 2008), evaluate a subtree on its training instances to see if it can be approximated by a smaller tree. Finally, Johnston, Liddle & M. Zhang (2010) proposed a technique that uses both approaches. However, even if these approaches have been successful, none is general enough to deal with

an arbitrary representation, which is one of GP's main strengths. Hence, they would constrict the choice of representation or demand that new rules be developed for each new function added to a representation. Eggermont (2005) suggests another simple method that removes unused nodes, i.e., nodes that do not process any of the training instances, and propagates the used node upwards in the tree recursively.

Method

Following the reasoning of Bohanec and Bratko, a reasonable requirement of a simplified program is that it is 100% fiddle to the original program. However, fidelity can be defined in many ways. Here it is argued that fidelity should be measured towards the predictions made on the training data by the original program and that it should be a 100% match.

Due to the fact that even stable deterministic decision tree techniques become unstable in the case of informational instability, as discussed in section 5.2.2, neither the structure, order, constants or the used attributes of the program itself are deemed to be important. Without domain knowledge related to the structure and form of the sought model, there is simply no way of pointing out the correct tree. All trees making the same predictions must therefore be considered equally likely to be the best fit of the underlying concept. Following this line of reasoning, a shorter program that is 100% fiddle to the original rule will always be preferable when performing simplification.

Two approaches of intron elimination are evaluated in this study. The first approach, described in the next section, aims to prevent the creation of introns by applying a hard parsimony pressure. The second approach instead tries to eliminate introns from the final rules by applying a simplification technique described below.

Comparing parsimony pressures

A parsimony pressure can be implemented in many ways. Here, two types of parsimony pressure are implemented in two different fitness functions and evaluated in this study; f_{PK} see equation 55 and f_{XC} i.e. equation 54.

f_{XC} is similar to f_{XACC} but uses the number of fiddle predictions instead of the ratio to somewhat reduce the complexity of analyzing the effect of different parsimony pressures and the effect of the simplification technique.

$$f_{XC} = \sum_{i=1}^{n} (p_i! = m_i) - P * O_p \qquad (54)$$

Two versions of f_{XC} with different pressure were used in the experiments. The first $f_{XC\text{-}l}$ used a low pressure of .0125 to produce rather large trees, while the other fitness function $f_{XC\text{-}s}$ used a pressure of .33 to evolve small programs.

A parsimony pressure of .0125 in practice means that an increased size of up to 80 *(1/.0125=80)* is compensated with a single extra fiddle prediction, since this will increase the fitness with the same amount. A $f_{XC\text{-}s}$ pressure of .33 in the same way means that an extra fiddle prediction can compensate for an increase of up to 3 primitives. Hence, $f_{XC\text{-}l}$ implicitly allows more introns, since a program may make an extra fiddle prediction with less than 80 primitives, use the rest to form introns, and still be better than a tree with just one less fiddle prediction.

f_{PK}, the other fitness function used in this study, was suggested by Papagelis and Kalles (2001) as a way of handling the accuracy versus comprehensibility tradeoff. Here, the parsimony pressure is more direct and forceful; the pressure is more direct, since the product of the predictive performance and the size is used instead of the sum, and it is more forceful because the difference in predictive performance and size is exaggerated by the use of their square. The accuracy versus comprehensibility tradeoff is handled by adjusting the pressure using a constant x typically in the range of 1000-10000. A smaller value of x means a bias towards smaller trees, while a larger x prioritizes accuracy.

$$f_{PK_p} = \left(\sum_{i=1}^{n} (p_i = a_i) \right)^2 * \frac{x}{{O_p}^2 + x} \qquad (55)$$

In an attempt to hamper the creation of introns by applying a hard parsimony pressure, an x value of *1000* was used in the experiments.

An example of the effect of the two different types of parsimony pressure is presented in Figure 37. The x-axis denotes the size of the program, while the y-axis denotes the relative fitness compared to the maximum fitness for each of the two fitness functions $f_{XC\text{-}s}$ and f_{PK} $(x = 1000)$. Two plots are made for each fitness function; one for programs that make the maximum of 150 correct predictions and one for programs that make 140 correct predictions.

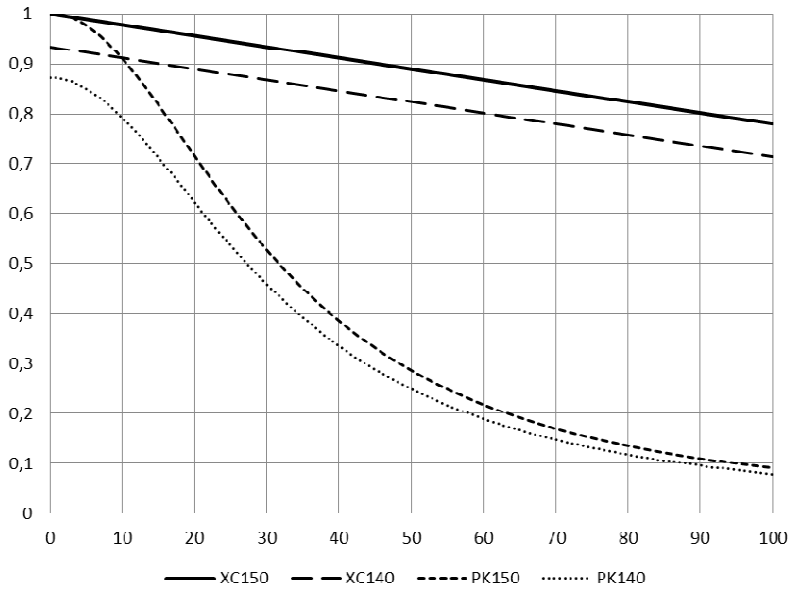

Figure 37 - Comparing parsimony pressure

The biggest difference between the fitness functions can be seen when comparing the maximum and minimum values of programs with the same accuracy in the diagram, i.e., the maximum and minimum for a single plot. For $f_{XC\text{-}s}$ the biggest difference in fitness of two programs of the same accuracy is 22%, while the biggest difference for f_{PK} is 91%. A bigger difference in the fitness of different individuals will naturally drive the evolution to converge faster around small programs with very few or no introns.

Another interesting but probably less important difference is while the fitness that relates to the difference in accuracy is constant for $f_{XC\text{-}s}$ it diminishes as the programs get larger for f_{PK}. This can be seen in the diagram as the two f_{PK} plots converge. For two programs of size one with 150 respectively 140 correct predictions, the difference in fitness is about 13%, while the difference for programs of size 100, with the same accuracy, is only around 1%.

Simplification

When trying to simplify an extracted rule, G-REX starts by replacing the original target variable with the corresponding prediction from the program that should be simplified. This new dataset is then used by G-REX to evolve a new population of rules. The fitness function used during simplification is very similar to f_{XACC} the standard fitness function for rule extraction using G-REX. A difference between

the two fitness functions is that f_{Simp} is based on fidelity towards the program p that should be simplified, instead of the fidelity towards an opaque model. Another important difference is that f_{Simp} is based on the number of fiddle predictions used instead of the ratio. In combination with a parsimony pressure of *.01,* this ensures that the evolution will search for programs that are very faithful to the original program, i.e, a fiddle prediction is 100 times more important than one that is less primitive.

$$f_{Simp_p} = \sum_{i=1}^{n} (p_{s_i}! = p_i) - .01 * O_p \quad (56)$$

A final very important part of the simplification is that the initial population is created using the program that should be simplified as a seed. More specifically, the population is divided into five equally sized parts, each subjected to different levels of mutation. One part contains exact copies of the original rule, while the remaining parts consist of copies of what has been mutated one to four times. Since there will be numerous copies of the original rules and fiddle prediction is much more important than the removal of a single primitive, it is all but guaranteed that the extracted program will be 100% faithful to the original program.

Results

To facilitate a comparison of the two approaches taken to eliminate introns, two experiments are performed. The first experiment aims to evaluate the accuracy and size of programs extracted using fitness f_{XC} and f_{PK}. In the second experiment, all programs extracted by the fitness functions used in experiment one are simplified using the proposed simplification technique. Here the aim is to evaluate whether the simplification technique succeeds in eliminating potential introns.

The opaque model from which the programs are extracted is an ensemble consisting of five data mining techniques; CART, two MLPs, a RBF, and a PNN. The result of CART as an individual technique is also used as a benchmark for accuracy and size.

The table below shows the accuracy and relative rank for respective fitness function on each dataset, the last row gives the mean result over all datasets.

Dataset	Accuracy				Ranks			
	CART	$f_{XC\text{-}l}$	$f_{XC\text{-}s}$	f_{PK}	CART	$f_{XC\text{-}l}$	$f_{XC\text{-}s}$	f_{PK}
Breast-w	.941	**.964**	**.964**	**.964**	4	2	2	2
Cmc	.491	.497	**.505**	.497	4	2.5	1	2.5
Credit-a	.802	.857	**.870**	.850	4	2	1	3
Diabetes	.670	.725	**.743**	.728	4	3	1	2
Ecoli	**.787**	.769	.786	.76	1	3	2	4
Glass	**.651**	.596	.611	.596	1	3.5	2	3.5
Heart-c	**.763**	.744	.761	.728	1	3	2	4
Heart-statlog	.740	.730	**.759**	.731	2	4	1	3
Hypothyroid	**.995**	.960	.962	.950	1	3	2	4
Ionosphere	.899	**.905**	.895	.889	2	1	3	4
Iris	.915	**.947**	.921	.926	4	1	3	2
Labor	.956	**.961**	.958	.954	3	1	2	4
Liver-disorders	.621	.616	**.625**	.614	2	3	1	4
Vehicle	**.671**	.624	.631	.635	1	4	3	2
Waveform-5000	**.692**	.657	.683	.683	1	4	2.5	2.5
Wine	**.965**	.933	.933	.951	1	3.5	3.5	2
Mean	.785	.780	.788	.778	2.25	2.72	2.00	3.03

Table 55 – Accuracy of CART and original programs (CD=1.17)

All techniques perform quite comparably and a *Nemenyi* test does not show any significant differences in accuracy between the different techniques. However, there is a large difference in the average rank between $f_{XC\text{-}s}$ and f_{PK} and a *Wilcoxon* test shows that there is actually a significant difference, i.e., the lowest rank's sum is 22.5 and the critical distance is 30.

Table 56 presents the average size of the extracted programs in relation to the size of the trees produced by CART. A score below one signifies that the extracted program was smaller than the cart tree. The smallest models are marked with bold letters.

Dataset	G-REX/CART				Ranks			
	CART	$f_{XC\text{-}l}$	$f_{XC\text{-}s}$	f_{PK}	CART	$f_{XC\text{-}l}$	$f_{XC\text{-}s}$	f_{PK}
Breast-w	94.1	1.105	**.159**	.234	3	4	1	2
Cmc	522.2	.337	.630	**.420**	4	3	2	1
Credit-a	33.1	1.360	**.462**	.492	3	4	1	2
Diabetes	156.6	1.226	.142	**.126**	3	4	2	1
Ecoli	51.0	3.588	**.490**	.824	3	4	1	2
Glass	59.0	2.441	**.339**	.746	3	4	1	2
Heart-c	58.2	3.488	**.313**	.478	3	4	1	2
Heart-statlog	45.8	2.096	**.310**	.415	3	4	1	2
Hypothyroid	19.2	2.396	**.380**	.677	3	4	1	2
Ionosphere	37.4	1.952	.401	**.380**	3	4	2	1
Iris	**9.80**	4.592	1.224	1.837	1	4	2	3
Labor	95.6	.554	**.123**	**.123**	4	3	1.5	1.5
Liver-disorders	81.8	1.577	**.125**	**.125**	3	4	1.5	1.5
Vehicle	67.1	3.010	.790	**.775**	3	4	2	1
Waveform-5000	69.2	1.199	.578	**.390**	3	4	2	1
Wine	96.5	1.275	**.207**	.332	3	4	1	2
Mean	93.5	2.012	.382	.500	3	3.88	1.38	1.63

Table 56 – Size of original programs in relation to CART (CD=1.17)

In Table 57 the size of the simplified programs are presented, again in relation to CART. The size of the simplified models are divided with the size of the CART tree, which means that a value below one signifies that a program is smaller than the CART tree. The smallest model of the four techniques is marked with bold letters.

f_{Simp} clearly worked well when applied to the programs extracted using $f_{XC\text{-}l}$, for which all programs could be simplified, and for $f_{XC\text{-}s}$, for which all but programs from four datasets could be simplified. The result for simplification of $f_{XC\text{-}s}$ and $f_{XC\text{-}l}$ programs does, however, stand in stark contrast to the result of f_{PK}, for which none of the programs could be simplified. A *Nemenyi* test shows that both $f_{XC\text{-}s}$ and f_{PK} are significantly smaller than CART and programs extracted using $f_{XC\text{-}l}$. Furthermore, a *Wilcoxon* test between $f_{XC\text{-}s}$ and f_{PK} shows that $f_{XC\text{-}s}$ programs are significantly smaller, i.e., the minimum sum of ranks is 4.5 in favor of $f_{XC\text{-}s}$ and the critical difference is 30.

Dataset	Size				Ranks			
	CART	$f_{XC\text{-}l}$	$f_{XC\text{-}s}$	f_{PK}	CART	$f_{XC\text{-}l}$	$f_{XC\text{-}s}$	f_{PK}
Breast-w	94.1	1.103	**.159**	.234	4	3	1	2
Cmc	522.2	.128	.044	**.042**	4	3	2	1
Credit-a	33.1	.915	**.438**	.492	4	3	1	2
Diabetes	156.6	.418	**.101**	.126	4	3	1	2
Ecoli	51.0	1.235	**.333**	.824	4	3	1	2
Glass	59.0	1.288	**.305**	.746	4	3	1	2
Heart-c	58.2	.450	**.244**	.478	4	2	1	3
Heart-statlog	45.8	.834	**.240**	.415	4	3	1	2
Hypothyroid	19.2	.641	**.380**	.677	4	2	1	3
Ionosphere	37.4	1.193	**.380**	**.380**	4	3	1.5	1.5
Iris	9.80	2.245	**1.224**	1.837	4	3	1	2
Labor	95.6	.550	**.123**	.123	4	3	1.5	1.5
Liver-disorders	81.8	.291	**.105**	.125	4	3	1	2
Vehicle	67.1	1.341	**.477**	.775	4	3	1	2
Waveform-5000	69.2	.977	**.318**	.390	4	3	1	2
Wine	96.5	.528	**.176**	.332	4	3	1	2
Mean	93.5	.884	**.316**	.500	4	2.875	1.125	2

Table 57 – Size in relation to CART after simplification (CD=1.17)

Analysis

The proposed simplification process works very well for programs extracted with $f_{XC\text{-}s}$ and $f_{XC\text{-}l}$ for which the programs could be made significantly smaller. With regard to accuracy, there were no significant differences between $f_{XC\text{-}s}$ and $f_{XC\text{-}l}$ but the latter clearly performed worse with a *.72* higher average rank. Theoretically, a larger program should be able to express more complex relationships and hence be more accurate, but here the results are the opposite. A larger program inevitably leads to a larger search space and, in this case, the population size and the number of generations that were the same for both $f_{XC\text{-}s}$ and $f_{XC\text{-}l}$ were insufficient for $f_{XC\text{-}l}$.

Programs extracted with f_{PK} could, however, not be simplified at all and had significantly lower accuracy compared to $f_{XC\text{-}s}$. In this case, both fitness functions evolved programs of similar size, i.e., the average size was *20.8* for $f_{XC\text{-}s}$ and *22.4* for f_{PK}. Hence, the difference in accuracy cannot be explained by the size of the search spaces. Instead, it is likely that f_{PK}'s inferior accuracy is due to a smaller number of introns resulting from the much harder parsimony pressure. Nordin, Francone and Banzhaf (1996) give support to this theory, since they show that introns have a protective role against destructive crossovers and can hence improve the effective fitness of an individual. The difference in the number of introns can be seen in the

result of the simplification, where the $f_{XC\text{-}s}$ programs' size could, on average, be reduced by 24% with the removal of introns; f_{PK} on the other hand, could not be simplified at all.

Summary

The most important result of this study is that the novel simplification feature of G-REX succeeded in simplifying programs extracted using f_{XC} by removing unnecessary parts. As a matter of fact, even rather small programs could often be further simplified. All simplified programs were 100% fiddle towards the unsimplified programs on the training data. In general, the extracted programs were slightly more accurate than the CART and after simplification they were significantly shorter than both CART and the original unsimplified program.

Programs extracted using f_{PK} did, however, seem not to contain any introns and could hence not be simplified, probably due to the relatively high parsimony pressure. However, the reduction of introns came at the cost of significantly larger and less accurate programs, compared to the simplified programs evolved using f_{XC}.

It is important to point out that all simplified programs were 100% fiddle towards the unsimplified programs on the training data.

It should be noted that even if this study concerned programs extracted from an opaque model, there is no difference to applying to programs evolved from the original dataset, since the technique is based on a pedagogical approach. Furthermore, since the GP optimization is independent of the evolved representation, the simplification process can naturally be applied to programs with arbitrary syntax.

7.2 Guiding among Alternative Models

Chapter 5.2.2 clearly demonstrates that there are often numerous alternative models with comparable accuracy and complexity.

Due to the ever present challenge of informational instability, there is no hope of a future stable GP technique. In the absence of huge amounts of data covering all possible variations of a concept, there will always be numerous alternative models with comparable predictive performance. Obviously, when alternative models appear, there will be no experts with sufficient domain knowledge to select the models that best capture the modeled concept, since that knowledge would have eliminated the need of PM in the first place. It may, however, be possible that domain experts have some knowledge of the nature of the modeled concept or that they seek a particular type of solution. In these cases, a system that guides the decision maker among the numerous alternative models would be very valuable. The sections below evaluate a technique, presented by König, Johansson and Niklasson (2010), that does exactly this, i.e., a technique that guides a decision maker among numerous solutions. The method used to create alternative solutions is described in detail in section 5.2.2.

Related work

Grąbczewski and Duch (2002) argue that many sets of rules with similar complexity and accuracy may exist, using for example different feature subsets, and that they bring more information of interest to a domain expert. Providing experts with several alternative descriptions makes it easier for them to find interesting explanations that are compatible with their own experience and may lead to a better understanding of the problem. Consequently, data mining methods aimed at finding several different descriptions of the same relationship are potentially valuable and deserve investigation. The approach of providing experts with alternative solutions is also supported by, for instance, Plish (1998), who notes that modern support systems for group decisions in situational centers largely depend on the availability of a procedure that generates "reasonable" (nearly optimal) alternative decisions in real time. If more than one solution exists, it would actually be misleading to present a single solution to a decision maker.

Selecting solutions

The most straightforward approach to selecting a single model from several alternative models is, of course, to compare all models and pick the one with the highest accuracy on either training data or on an additional (validation) dataset. In this study, however, the starting point is models, all with comparable training accuracy which could complicate selection based on training accuracy. Holding

back a validation set is still applicable, but previous work, such as presented in section 6.3.1, has shown that even if a validation set is useful for selecting models, it also lowers the accuracy for the generated model, since all data is not available for training. Again, the use of all available data for the actual modeling is especially important for datasets with relatively few instances to start with. Hence, selection based on validation accuracy is not considered in this study. A different approach is to select the trees with the highest gain ratio, but as seen in the work of Li & Liu (2003), this does not yield very promising results.

A more promising approach, suggested by Johansson et al. (2010) and presented in detail in section 6.3.1, is to select the model whose predictions are most faithful to an ensemble consisting of all alternative solutions. An ensemble of models is usually better than its individual members and, hence, the individual that is faithful to the predictions of the ensemble should also be more accurate. Experiments in section 6.3.1 show that ensemble fidelity was clearly a better indicator of an accurate model than the results of a validation set.

Method

From the results in section 5.2.2, it is clear that there are usually more alternative models than a decision maker can analyze manually. Here, it is argued that the work of a decision maker could be reduced by only presenting a few, better than average models that represent different types of models. The decision maker would select the model that best fits his/her domain knowledge and only if not fully satisfied could he/she be shown another small subset of models similar to the selected model. In this way, the number of alternative models that a decision maker needs to analyze would be drastically reduced and the possibility that the selected model would have a higher than average accuracy would increase. Selecting models with higher than average accuracy from a set of models with equal training accuracy may seem like a paradox, but it is, of course, the accuracy on novel data that should be maximized.

The suggested method is based on two assumption; that the alternative solutions can be categorized into groups of similar models and that models with better than average performance can be identified for each group.

An alternative decision tree is defined as a tree that is equal or smaller and at least as accurate as an original tree created using J48. Furthermore, if several trees partition the training data in the same way, only one of them is considered to be an alternative solution. The method of generating alternative decision trees is described in detail in section 5.2.2.

Grouping of similar models

Alternative models can of course be grouped in different ways, depending on their representation. One representation independent approach would be to group models that make similar predictions on the training set. However, a serious drawback with this approach would be that models which predict similarly do not necessarily need to have a similar appearance, i.e., they can use different attributes, structure, or order and still make similar predictions. A decision maker could select a promising model according to his/her domain knowledge, but if he/she wanted to investigate similar solutions, their appearance could be totally different if they were chosen on the basis of prediction similarity.

In this study, the selected model type is decision trees, which makes it natural to instead group solutions on the splits used in the trees. One or more consecutive splits could be used to group the trees, and more splits would naturally lead to more groups with fewer members in each group. To minimize the number of trees that would need to be analyzed and to ensure that each group has a reasonable number of trees, the grouping here is based only on the root split, i.e., trees with the same split in the root node will be placed in the same group. Furthermore, since the root split normally has a strong influence on a decision tree, trees with the same root split may have more splits in common than trees that only share a non-root split. Finally, only groups of three or more programs are counted as groups, since this is the minimum number of programs that would make sense to the selection techniques described below.

Selecting good models

As described in the related work, there are several ways of selecting a single tree from a group of trees. This study evaluates four different techniques: *random* (RND), *training accuracy* (TRN), *ensemble fidelity* (ENS), and *similarity* (SIM). *Random* selects a tree randomly and will be estimated by calculating the average accuracy of all trees. The other three techniques select one tree from each group of solutions. TRN the tree with the highest training accuracy from each group. Note that even if all trees have similar accuracy, it may differ slightly, since the definition of an alternative tree is that it should have at least the same training accuracy as the original J48 tree and an equal or less number of splits. Validation accuracy was not considered in this study, since previous work presented in section 6.3.1 has shown that training accuracy is often a better predictor of test accuracy.

Ensemble fidelity is based on the coaching technique of Johansson et al. (2010), but here an ensemble is created for each group and the tree that is most fiddle to its ensemble is selected from each group. Finally, *similarity* selects the trees that have most in common with the other trees in the same group. Similarity is

calculated as the number of identical splits that appear in the same place in both trees. The idea is that important splits will be used more often and hence the tree that has most in common with the other group members should contain more important splits.

Results

In the following section, the results based on tenfold cross-validation for 19 UCI datasets are presented. Since larger trees can naturally express more complex, and hence possibly more accurate trees, and since the accuracy and size will, of course, affect the number of possible solutions, alternative trees with the target size three, five and seven are analyzed in the experiments. The G-REX settings used for evolving the alternative trees can be found in Table 31 in section 5.2.2.

Grouping of similar models

Table 58 below shows the average number of trees generated per target size for each dataset, how many groups they consisted of and how similar the trees in each group were compared to the average similarity of trees created for the same dataset. The left column also shows the target size that was used when pruning the J48 trees and the resulting actual size that was used as an upper limit for the evolved programs.

Size		Number of			Similarity	
Target	Actual	Trees	Groups	Trees/Group	RND	ENS
3	2.4	81.6	6.9	11.8	0.27 / 11%	1.04 / 43%
5	3.9	103.5	5.2	19.9	0.75 / 19%	1.94 / 48%
7	5.5	173.9	4.8	36.2	1.02 / 19%	2.56 / 46%

Table 58 – Group similarity

The most important result of this experiment is that grouping trees based on their root split works reasonably well. Trees in the same groups have more in common than two randomly selected trees and they have more split than the root split in common. All target sizes produced groups where the trees in the same group were more similar than the average similarity of all trees. While trees of target size three only had 4% more splits in common than the average similarity, the larger trees had considerably higher similarity. If the number of similar splits is related to the target size of the trees, the result is however more equal, i.e., 46% for trees in the same group compared to only 16% for all trees. Finally, it is interesting to note that even if the number of alternative trees grows with the target size, which could be expected, the number of groups decreases.

Selecting good models

The second part of the purposed methods aims to select models that are more accurate than the other models in the same group. Table 59 shows the average accuracy for J48 and the selection techniques over all datasets for each target size.

Size		ACC				
Target	Actual	J48	RND	TRN	ENS	SIM
3	2.4	.735	.751	.755	.761	.749
5	3.9	.765	.775	.777	.782	.765
7	5.5	.766	.782	.779	.789	.766

Table 59 – Summary of ACC

An obvious result of the accuracy comparison, which could be expected, is that larger trees have a higher accuracy. The comparison among the selection techniques is, however, more interesting. First it should be noted that the average evolved program, i.e., RND, is better than trees created using J48 with the adapted pruning technique. Hence, it is no surprise that all selection techniques are better than J48. Selections based on training accuracy are only slightly better than the average tree and selection based on the similarity of splits is actually marginally worse. All in all the performance of RND, TRN, and SIM is very similar. Selections based on group ensemble fidelity is, however, clearly the best selection method, as a matter of fact a Wilcoxon signed rank test at a 0.05 significance level shows that ensemble fidelity is, in general, significantly better than J48 and all other selection techniques. Table 60 presents the result of the tests between ENS and the other selection techniques. The tests show significance, if the result of the test is less than the critical distance, i.e.CD = 46 for 19 datasets. As can be seen in the table, all results are significant except between ENS and TRN for target size five.

α=0.05	J48	RND	TRN	SIM
3 Ens	22	12	22	2.5
5 Ens	21.5	35	75	21
7 Ens	18	7.5	6.5	18

Table 60 – Wilcoxon test for ENS (CD=46)

Table 59 shows the average accuracy over all datasets. To show a more detailed picture of the experiments, the result of each target size and dataset is presented in Table 61, Table 62, and Table 63.

Dataset	J48	RND	TRN	ENS	SIM	Groups	Trees/Group
Breast-cancer	.707	.724	**.730**	**.730**	.727	6.7	9.60
Breast-w	.929	.937	.946	**.950**	.943	12.2	9.26
Colic	.837	.852	**.859**	**.859**	.837	0	0
Contact-lenses	.750	.776	.750	**.783**	.750	1.6	5.13
Credit-a	**.857**	.852	.844	.850	.850	1.9	13.68
Cylinder-bands	.663	.674	.680	**.685**	.681	13.7	15.94
Diabetes	.743	**.744**	.740	**.744**	.743	.5	1.40
Glass	.472	.534	**.565**	.559	.533	28.9	14.79
Haberman	.683	.724	.722	**.725**	.721	12.1	11.78
Heart-c	.752	.744	.740	**.753**	.749	3.5	21.97
Heart-statlog	**.722**	.705	.708	.714	.713	1.0	9.70
Hepatitis	**.801**	.788	.783	.782	.777	1.6	11.88
Iris	.947	.950	.949	**.952**	.948	4.5	6.33
Liver-disorders	.644	.665	.665	**.672**	.662	5.8	11.50
Lymph	.595	.690	.723	**.724**	.675	19.3	8.07
Tae	.458	.491	.516	**.540**	.488	17.1	1.32
Tic-tac-toe	**.688**	**.688**	**.688**	**.688**	**.688**	0	0
Wine	.880	.908	.907	**.914**	.905	.4	1.50
Zoo	**.833**	.815	**.833**	**.833**	**.833**	.1	27.00
Mean	.735	.751	.755	**.761**	.749	6.9	11.83

Table 61 – Tree size 3: Accuracy of J48 and selection methods

The results for target size three are clear; ENS only loses three times against J48, two times against RND and TRN and never against SIM. It is also interesting to note that TRN and RND perform very similarly, with nine wins for TRN and eight for RND. For *Colic* and *Tic-Tac-Toe,* no groups of at least three trees with the same root split could be found. In these and similar cases, ENS and SIM consider all trees as a single group.

Dataset	J48	RND	TRN	ENS	SIM	Groups	Trees/Group
Breast-cancer	.710	.726	.730	**.733**	.710	5.4	16.72
Breast-w	.944	.950	.953	**.954**	.944	13.8	23.58
Colic	.845	.851	.849	**.853**	.845	.8	3.88
Contact-lenses	.750	.780	.750	**.817**	.750	1.4	5.29
Credit-a	.841	**.848**	.844	.847	.841	3.4	31.41
Cylinder-bands	.680	.692	.695	**.698**	.680	9.3	26.80
Diabetes	.745	.749	**.751**	.749	.745	2.1	13.81
Glass	.640	.650	**.657**	**.657**	.640	1.7	9.12
Haberman	.690	.721	.729	**.731**	.690	17.2	18.84
Heart-c	.782	**.799**	**.799**	.797	.782	2	8.65
Heart-statlog	.763	.785	.796	**.788**	.763	1.2	35.92
Hepatitis	.788	.813	**.822**	.819	.788	6.1	23.92
Iris	.940	.952	**.953**	.950	.940	2.7	8.07
Liver-disorders	.650	.644	**.652**	.647	.650	7.6	13.97
Lymph	**.783**	.756	.755	.773	**.783**	8.4	34.79
Tae	.465	.496	.511	**.530**	.465	15.7	1.45
Tic-tac-toe	**.688**	**.688**	**.688**	**.688**	**.688**	0	0
Wine	.916	**.929**	.922	.922	.916	.1	14.00
Zoo	**.912**	.902	.902	.902	**.912**	.2	9.50
Mean	.765	.775	.777	**.782**	.765	5.2	19.90

Table 62 – Tree size 5: Accuracy of J48 and selection methods

As mentioned above, ENS is not significantly better than TRN for target size five. Nevertheless, Table 62 clearly show that ENS is a better selection method, since for nineteen datasets it only loses four times against TRN. Here, TRN has eleven wins, three draws, and six losses, compared to a program chosen at random, i.e., RND.

For the largest trees with target size seven, see Table 63, ENS is again significantly better than all other techniques. ENS only loses twice against J48, once against RND and TRN, and four times against SIM. There is no significant difference between RND and TRN, and here RND even achieves fourteen wins over TRN.

Dataset	J48	RND	TRN	ENS	SIM	Groups	Trees/Group
Breast-cancer	.707	.721	.717	**.735**	.707	3.6	44.42
Breast-w	.946	.951	**.954**	**.954**	.946	12.9	23.44
Colic	**.851**	.850	.844	.850	**.851**	2.5	38.28
Contact-lenses	.750	.779	.750	**.783**	.750	1.3	5.69
Credit-a	.843	**.848**	**.848**	.846	.843	2.7	87.85
Cylinder-bands	**.709**	.689	.684	.700	**.709**	6.3	66.75
Diabetes	**.746**	.744	.743	.744	**.746**	0.6	22.33
Glass	.645	.641	.639	**.648**	.645	1.2	19.08
Haberman	.677	.725	.722	**.727**	.677	20.1	27.46
Heart-c	.772	.808	.810	**.813**	.772	2.1	16.24
Heart-statlog	.774	.804	.797	**.819**	.774	1.4	21.79
Hepatitis	.762	.779	.782	**.785**	.762	6.5	43.11
Iris	.940	.954	.947	**.948**	.940	3.2	7.44
Liver-disorders	**.655**	.645	.637	.645	**.655**	3.8	23.18
Lymph	.750	.765	.749	**.768**	.750	6.4	44.52
Tae	.485	.547	.545	**.557**	.485	12.1	53.63
Tic-tac-toe	.716	.753	.774	**.777**	.716	2.5	29.08
Wine	.920	.933	.922	**.946**	.920	0.7	18.14
Zoo	.911	.931	**.941**	**.941**	.911	0.7	23.43
Mean	.766	.782	.779	**.789**	.766	4.8	36.23

Table 63 – Tree size 7: Accuracy of J48 and selection methods

Analysis

From the results presented in the previous chapter, it is clear that the grouping of programs based on the root split works reasonably well. On average, the programs of a group had 29% more splits in common compared to the average tree.

Another interesting result is that even if larger target sizes naturally result in more alternative programs, the number of groups is, at the same time, decreasing. An explanation could be that since the larger trees are more accurate, see Table 59, the root split is more important, because it is the first split and hence has a big effect on the predictions. At the same time, larger trees mean more leaves and fewer training instances in each leaf, which increases the chance that there will be more with similar performance. As a result, there will be less possible root splits resulting in fewer groups, but more small variations of each tree, resulting in more variations of each tree.

J48 achieves significantly lower accuracy than the average program, which is a worse result for J48 compared to other studies, such as (König et al. 2010) and

(Johansson, König and Niklasson 2010) where J48 was clearly inferior, but not significantly so. Here, the lower accuracy of J48 can probably be explained by the specific pruning algorithm used to achieve trees of a certain size. With this said, it should be noted that the comparison with J48 is not the main point, but only a way of getting an estimate of a reasonable accuracy for trees of a certain size.

Of the tree selection techniques, it was only ENS that was significantly better than choosing a tree at random. Since training accuracy and validation accuracy are rather common criteria for model selection, it is also interesting to note that here it was in no way better to select a tree based on training accuracy than selecting a tree at random.

Summary

The purposed method of guiding a decision maker among numerous alternative models is dependent on two assumptions, which are both confirmed by the experiments. First, it is clear that grouping alternative trees based on the root split works well, since the trees in each group are relatively similar and because it results in a practical number of groups. Secondly, fidelity against an ensemble consisting of all members in a group can be used to select trees that are more accurate than the average tree. Training accuracy and split similarity were, however, not better than selecting a tree at random.

Hence, it is obvious that the purposed technique is feasible; it is possible to create and group alternative models and then use ensemble group fidelity to select a better than average model from each group. In this way, a decision maker could be given a few accurate alternative models, instead of analyzing all possible alternative models. Since each model is selected from a group of similar trees, it would also be possible to look at other versions of a certain promising model.

8 Conclusions

The research question of this thesis is "Can the deficiencies and strengths of Genetic Programming, when used for Predictive Modeling, be reduced and exploited by novel techniques to enhance the accuracy and comprehensibility of the generated models?" In order to answer this question, five research objectives were identified. These are discussed below to facilitate a well-founded answer to the overall question.

8.1 Criteria for a GP framework for PM

A relative small, but general set of criteria for GP frameworks for PM, that in addition to the basic requirements highlights the challenges and advantages of using GP for PM, has been identified in Chapter 3. The criteria have been based on an extensive but in no way exhaustive literature survey and more criteria may certainly be added to the list. However, the suggested criteria should be a good base for the evaluation of GP frameworks for PM, since they were selected to highlight the challenges and advantages of using GP for PM. Finally, it should be noted that accuracy and execution time are two vital problem-dependent criteria that are assumed to be fulfilled for the proposed criteria to be relevant. In short, to exploit and handle the inherent advantages and challenges, a GPPMs should:

- support regression and n-class classification
- support PM through a GUI
- support optimization of common score functions
- support tailoring of the fitness function
- be able to handle large search spaces
- handle the accuracy versus comprehensibility tradeoff
- support tailoring of the representation
- support intron removal
- exploit GP's inconsistency to present alternative programs, indicate informational instability and increase predictive performance
- support discovery of interesting rules
- support benchmarking against common PM techniques
- be open source, if used for research purposes.

8.2 A GP framework for PM

Numerous GP applications and frameworks exist, but only a few are designed with PM in mind. Section 3.1 evaluates the suitability of six commercial applications and six frameworks for PM.

First, it is clear that the advantages of using GP for PM, i.e., tailoring of the fitness function and the representation, plus the possibility to handle the accuracy vs. comprehensibility tradeoff using parsimony pressure, are recognized in practice and are implemented in a majority of the evaluated software. The challenges of using GP for PM have, on the other hand, not received as much attention; i.e., no software has any special functionality for handling large search spaces or for finding interesting rules, only three have some kind of intron removal functionality, and inconsistency is only handled by three applications that present a small subset of alternative rules.

Regarding the more general PM criteria, surprisingly few, i.e., only three, fulfill the basic requirement of supporting both regression and n-class classification, and the same number support benchmarking against other PM-techniques.

Of the evaluated applications, DTREG is arguably most suited for PM. DTREG is essentially a PM application with support for genetic programming using a set of predefined fitness functions, adaptable representation, parsimony pressure, algebraic-based removal of introns, and functionality for benchmarking against other techniques. It is, however, not possible to create new fitness functions and only binary classification tasks are supported. Furthermore, since DTREG is a commercial application, it is not open source and does not support extensions or verification of the code base, which is naturally vital if new techniques are supposed to be designed and tested.

Naturally, only open source GP frameworks are of interest for research purposes. Of these, only the WekaGP and G-REX have any explicit functionality related to PM. All others are general GP frameworks that would require substantial extensions to be used for PM. G-REX and WekaGP are quite similar in functionality, with the exception that WekaGP lacks the ability to control the parsimony pressure and has an arguably less comprehensible representation where a program is evolved for each class. Due to these reasons, G-REX was selected and further developed during the thesis work.

More specifically, the current version of G-REX v9.0 Unleashed has been extended with functionality for handling large search spaces, as described in sections 6.1 and 6.2, and intron removal, as described in section 7.1. With this addition, G-REX is a rather complete framework for PM and clearly superior, according to the proposed criteria for GPPM, to both the commercial applications and the other open source frameworks evaluated in this thesis. Furthermore, since

G-REX was used to implement and test all techniques proposed in this study, it can be extended with the techniques for exploiting inconsistency and discovery of interesting rules, proposed in sections 6.3 and 7.2, and thus fulfill all of the suggested criteria.

8.3 Advantages and challenges of using GP for PM.

According to the literature, GP has both advantages and challenges when used for PM. The advantages are, in general, GP's flexibility, which allows optimization of the arbitrary fitness function and representations, plus a global optimization which should be superior to the typical greedy searches used by decision tree algorithms. The challenges are the presence of introns in the evolved programs, difficulties with large search spaces, and inconsistency. Speed, which is another challenge, is not considered in this thesis, since it is problem dependent. Furthermore, since GP is naturally parallelizable, most improvements in speed can be referred to parallelization of the basic GP algorithm.

The conclusions that can be drawn, based on Chapter 5 which explores these advantages and challenges, are presented in the sections below.

8.3.1 Optimization of arbitrary score function

The general results of section 5.1.1 is that it is clearly favorable to optimize the actual score function. Regarding classification, decision trees optimized using f_{AUC} were significantly better in terms of AUC compared to trees optimized using f_{ACC}, f_{BRE} and trees created using CART, and the same holds for f_{BRE} when evaluated on BRE. The result for optimization of ACC is not as clear, even if f_{ACC} achieves the highest average ACC of the evaluated techniques. A possible explanation is that the trees optimized using f_{ACC} were less complex than trees optimized using the other techniques. On a lesser note, another contributing reason for this result may be that f_{BRE} and f_{AUC} explicitly and implicitly also optimize ACC.

It should also be noted that these results hold, regardless whether it is decision trees, decision lists, or the G-kNN representation that are evolved.

For regression problems, the results were similar, even if not as strong. G-REX succeeded in optimizing MAE, RMSE, MAPE, MUAPE and r, which resulted in superior models, in terms of respective metrics, except for f_{MUAPE} which was slightly outperformed by f_{MAE} on the holdout sets. Compared to the decision tree techniques M5P and REPTree, the trees evolved using f_{MAE}, f_{MAPE} and f_r were clearly superior when evaluated on respective metric. In terms of RMSE, f_{RMSE} was outperformed by REPTree and f_{MUAPE} was inferior to both M5P and REPTree in terms of MUAPE. While M5P had a similar complexity to the evolved trees,

REPTrees superiority, in terms of MUAPE and RMSE, can be explained by trees that were almost twice as complex.

A more or less standard way of utilizing the optimization of arbitrary score functions is to apply a parsimony pressure as a way to handle the accuracy versus comprehensibility tradeoff. Section 5.2.1 shows that parsimony pressure can be applied to evolve programs that are, on average, almost half the size of trees produced using J48. However, the result was mainly due to two datasets where J48 produced very large programs and, although they were large, the difference in size was not significant.

Sections 5.1.2 and 7.1 show that rule extraction from an ensemble, which is another way to exploit GP's flexible optimization, produces programs that are significantly smaller compared to CART, while having higher accuracy. In one study, where the programs produced by G-REX were, on average, only 32% of the CART trees' size, the G-REX programs were also still slightly more accurate. In the other case, the difference in size was smaller, but still large, i.e., 53.1%, while having an almost significantly higher accuracy, i.e., the *p*-value of a Wilcoxon signed rank test was *0.083.* Obviously, parsimony pressure in combination with rule extraction is a good approach to the accuracy versus comprehensibility tradeoff. In fact, when optimizing BRI, the extracted rules where significantly better in regard of ACC and AUC compared to J48, while being smaller in size.

8.3.2 Optimization of arbitrary representation

GP's flexibility in representation is especially important when comprehensible models are sought, since comprehensibility is naturally a subjective property. Hence, a decision maker should be able to tailor the representation to his own preference. Furthermore, tailoring the representation is a way of incorporating domain knowledge into the modeling process and may lead to more accurate models, since irrational operators and functions can be avoided, thus reducing the search space and the risk of overfitting. Section 5.1.3 presents several studies where G-REX was been used to evolve very competitive programs using different types of representations, i.e., Boolean- and if-else decision trees, regression trees, decision lists, fuzzy rules, and a novel hybrid decision tree representation with kNN classifiers in the leaves. Obviously, this wide range of representations will make it easier to create a better fit for the decision maker's preference. The importance of the ability to tailor the representation is also recognized in practice, which can be seen by that fact that ten of the twelve GP software that were evaluated in section 3.1 had some kind of related functionality.

Section 5.1.3 essentially shows that different representations can be optimized using GP and only, in general, argues why this should be considered an advantage.

However, G-kNN, a novel hybrid kNN- decision tree technique, which is only possible due to GP's flexibility in representation and fitness function, is also presented. G-kNN (with global optimization) significantly outperformed all traditional variants of kNN. Furthermore, the hybrid technique had three other advantages compared to standard kNN techniques; i.e., it does not require a specific *k*-value to be set, it can exploit local information to select different *k* for different regions of the input space, and it is optimized globally using GP which may avoid local optima.

8.3.3 Global optimization vs. large search spaces

GP's global search should, in theory, outperform the greedy non-backtracking search strategy used by decision tree algorithms. Nonetheless, experiments from two studies discussed in section 5.2.1 show that G-REX did not outperform CART or J48 when compared with regard to ACC. Specifically datasets that required a very large program were problematic for G-REX. Two possible explanations which are well-known weak points of GP and relate to large search spaces are:

- An ill-chosen Parsimony pressure may constrain the GP search in a too small or unnecessarily large search space.
- When the solution required is large, it may be impractical to run the evolution long enough.

For regression problems which naturally have a larger search space due to a numeric target, the problem is even more obvious. Section 5.1.1 shows that G-REX can evolve regression trees that are superior, according to several common score functions, to the counterpart created using decision tree algorithms. However, when a more complex representation is evolved, like the model trees presented in section 6.1, the optimization fails and creates programs that are significantly inferior to the more simple regression trees. Here, it is obvious that the more complex representation creates a search space that is too large for the experimental setup. Furthermore, section 7.1 shows that a too low parsimony pressure may lead to inferior programs, since it allows larger programs and thereby may results in a too large search space.

Obviously, the global optimization does not outweigh the challenge of large search spaces, which hence must be handled in another way, e.g., with a local search, as presented in section 8.4.1.

8.3.4 Inconsistency

Inconsistency is mentioned as a disadvantage for evolutionary techniques in the literature and, as section 5.2.2 shows, GP is quite inconsistent in that it may produce numerous programs that differ in both their predictions and their structure, in spite of similar predictive performance.

In regards to predictive instability, i.e., if only the predictions are considered, GP is relatively consistent. When evaluated on six UCI datasets, the predictions of five alternative programs differed on *10%* of the training data and *12%* of the test data.

In terms of structural inconsistency, i.e., the used functions, terminals and their structure, GP is, however, much more inconsistent and may produce numerous programs with similar predictive performance. Naturally, the size of the evolved programs affects the number of alternative programs where the production of larger trees is more inconsistent. It should be noted that the inconsistency is higher for larger trees, in spite of the fact that they are more accurate. Experiments on 19 datasets show that there are, on average, 80 alternative programs for trees of size 3, 104 for trees of size 5, and 174 for trees of size 7. However, the amount of instability varies greatly among datasets and 10 of the datasets stand for 90% of the instability. Hence, it is clear that it is the informational instability of the datasets which is the main reason for GP's inconsistency.

Clearly, informational instability is a problem for all predictive techniques and there is no reason why a deterministic decision tree technique would find the true underlying relationships. Hence, GP's inconsistency should be seen as an advantage, since it makes detecting the informational instability of the data possible. Since informational instability is a consequence of a dataset that is not fully representative of the problem domain, more data needs to be gathered or a decision maker needs to use domain knowledge to select the most probable solutions. However, if the informational instability is high, there may be so many alternative programs that a decision maker would need help choosing between them, to avoid becoming overwhelmed, e.g., as suggested in section 8.5.1.

8.4 Techniques for enhancing accuracy of predictive models

Two main perspectives were explored in relation to how they may enhance accuracy, namely, handling large search spaces and informational inconsistency.

8.4.1 Handling large search spaces

As concluded in section 8.3.3, it can sometimes be impractical to use GP, if a large or complex program is needed to solve a problem. The underlying reason is simply that the number of possible programs grows exponentially with the number of

possible functions and terminals combined with the allowed program size. Given sufficient time, this should not be a problem, but in practice, it sometimes is. Furthermore, in search of a comprehensible model, a parsimony pressure that is too hard may also limit the search to a suboptimal space.

Section 6.1 shows how a local search may increase the search for model trees with linear regression as leaves. With a straightforward GP approach, the evolved programs are inferior to the less complex regression trees created using CART and to models created using multiple linear regressions. If, instead, the simple regressions of the programs are initiated using least square, the search becomes the task of finding the best tree for the available regression, which is a much simpler task. Consequently, the suggested technique significantly outperforms both straightforward GP and multiple linear regressions, and outperforms CART on all but a single dataset.

For classification tasks, a slightly more complex technique that both handles the parsimony pressure and speeds up the search, is proposed in section 6.2. Here, decision trees are first used to calculate a reasonable accuracy using bootstraps from the training set. A check against the reasonable accuracy is done each *nth* generation and, if the evolution has not found a program surpassing this accuracy, two things happen: first, the parsimony pressure is decreased with 10% to allow a search among larger programs; secondly, increasingly larger decision trees, also created using training set bootstraps, are mutated and injected into the population. Consequently, in the cases where the evolution is struggling to find a reasonable solution, it is guided to a region known to hold at least reasonable programs. Experiments showed that the proposed technique was significantly more accurate than the decision tree technique used to create the injected tree, i.e., jaDTi, and J48. Furthermore, standard GP was clearly outperformed and a Wilcoxon test gave a value of 0.09 in favor of the proposed technique. Injecting trees was especially successful for datasets that required a large program, which is exactly what the technique was designed for. This can also be seen in that the trees produced using tree injection were almost twice as large as the ones produced with standard GP and were, instead, more comparable to the J48 trees.

Concluding, large search spaces are problematic for GP, but they can be somewhat handled by performing some kind of local search.

8.4.2 Exploiting inconsistency to improve predictive performance

GP is inherently inconsistent in that different runs may produce different solutions. However, as concluded in section 8.3.4, it is the informational instability of a dataset that is normally the main reason for the inconsistency. One of the points made in this thesis is that inconsistency should be considered an advantage rather than a

disadvantage for GP. As a basis for this claim, three novel techniques that exploit the inconsistency of GP in order to increase the predictive performance are proposed.

First, section 6.3.1 proposes a technique for selecting a single program among a set of programs with comparable training accuracy and size. Decision tree techniques are deterministic and, if the very common case of informational instability is in the dataset, the final tree is decided by the order in which different splits are evaluated. More precisely, if there are several splits resulting in the same purity gain, the selection is normally based on their order, which in essence is random, and has nothing to do with the underlying relationships.

The proposed techniques do, however, manage to select trees that are better than the average tree, even if they all have the same training accuracy and comparable size. The main idea exploited in the technique is that ensembles are more robust than single models and, hence, the model whose predictions most resemble the predictions of the ensemble should also be more robust. Experiments show that the program which predicts most similarly to the ensemble on the training set is significantly better than a random tree selected from the same set of programs, or a tree selected on the basis of training accuracy. Furthermore, the ensemble selected program is also significantly better than a J48 tree, which is not the case for a tree selected at random or based on training accuracy. In those cases where the test predictions can be made in a batch, the ensemble predictions for the test instance can also be used as a benchmark. Furthermore, the experiments also show that selecting programs based on the ensemble test predictions is even better than using the training predictions.

Secondly, section 6.3.2 presents a technique that uses alternative GP programs as an ensemble, to enhance the probability estimates for a single comprehensible model. More specifically, experiments show that the GP ensemble produces probability estimates that are significantly better compared to the probability estimate done by a single GP or a CART tree and, more important, significantly better than an ensemble of CART trees. An explanation for that superior GP ensemble is that since GP is inconsistent, it can produce different programs using all training data, while CART, which is deterministic, uses bootstraps to obtain diverse trees. Since bootstraps are subsets of the training data, each CART tree in the ensemble naturally misses potentially important instances, resulting in fewer accurate trees. An important point in the suggested techniques is that the predictions are still done using a single comprehensible model and only the probability estimates are calculated using the ensemble and are thus opaque.

Finally, section 6.3.3 exploits the inconsistency by creating a diverse ensemble of hybrid G-kNN trees. The main result is, of course, that G-kNN ensembles

clearly outperform both standard kNN using majority- and weighted voting and the kNN-ensemble technique. Since the accuracy of the G-kNN base models is lower than the ensemble accuracy and standard kNN using majority voting, the superior performance must be credited to the approach of using GP to optimize the individual ensemble members.

However, even if both ensembles are better than both individual standard kNN techniques, the representation of the base models is evidently important. The G-kNN representation clearly facilitates more diverse base models compared to the single global kNN model, since the G-kNN significantly outperforms the kNN ensemble, in spite of slightly lower base classifier accuracy. This is a quite natural result, since a kNN-ensemble can only contain ten different models, due to the restrictions on the possible *k*-values. Finally, it should be noted that the lower base accuracy was a result of the conscious choice of using a GP setting that favored diversity over accuracy, by using a less extensive search, i.e., a smaller population and fewer generations.

8.5 Techniques for enhancing comprehensibility of predictive models

The comprehensibility of predictive models is enhanced in two main ways, namely, by removing introns and by exploiting inconsistency to discover interesting rules.

8.5.1 Enhancing comprehensibility by removing introns

It is well known that introns are necessary during evolution but add unnecessary complexity to the final programs. Normally, intron removal techniques are algebraic and hence dependent on the underlying representation, consequently hampering the advantage of being able to use arbitrary representation. To allow GP to fully support arbitrary representation, section 7.1 presents a novel representation independent technique for the removal of introns. To remove introns, a secondary evolution is performed where the target variable is replaced by the predictions of the program to be simplified. This approach is similar to pedagogical rule extraction, but differs in that the simplified rule must be 100% fiddle to the original rule and that the population is initiated with mutated copies of the original rule.

Experiments show that when a typical fitness function, like f_{XC}, is used, the final program can be made significantly shorter, compared to the original program and CART, and still be 100% fiddle to the original program. Another interesting result is that the harder parsimony pressure used in f_{PK} resulted in programs without introns. However, these programs were both significantly larger and less accurate than the simplified program created using f_{XC}, which confirms the beneficial role of introns during evolution.

8.5.2 Exploiting inconsistency to discover interesting rules

Due to GP's inconsistency, it has a potential for discovering truly interesting rules, which is a criterion for PM techniques sometimes mentioned in the literature. At the same time, section 5.2.2 shows that when the informational instability is high, there may be more alternative programs than what are practical to analyze manually. A solution to this problem and a step towards a strategy for the discovery of interesting rules are presented in section 7.2. The main idea of the proposed techniques is to generate as many alternative solutions as possible and then group them according to some criteria. A single representative tree is then selected from each group and presented to the decision maker. If one of the solutions is more interesting to the decision maker, he/she can explore additional similar solutions from the same group. In this way, the decision maker can be guided among all possible programs without having to analyze each program separately.

Experiments show that a simple grouping based on the root split is sufficient to create a practical number of groups. Furthermore, even if only grouped on the basis of the root split, the programs in each group have more splits in common, which is vital for the proposed technique. Once again, the imaginary ensemble technique, proposed in section 6.3.1, is used to select a program from each group and the results are, once again, significantly more accurate programs, compared to programs selected on the basis of training accuracy.

Obviously, the purposed technique is feasible; it is possible to create and group alternative models and then use ensemble group fidelity to select a better than average model from each group. In this way, a decision maker could be presented with a few accurate alternative models instead of having to analyze all possible alternative models. Since each model is selected from a group of similar trees, it is also possible to look at other similar versions.

8.6 Final conclusion

This thesis aims to enhance GP for predictive modeling, in terms of accuracy and comprehensibility, by reducing its deficiencies and exploiting its strengths. Based on the conclusions related to each objective above, some final thoughts are presented below:

The advantages of being able to optimize arbitrary representation and fitness functions are recognized both in the literature and in practice and could also be demonstrated empirically in this thesis. Section 5.1.2 does, for example, show how GP's flexible optimization facilitates rule extraction from opaque models. The extracted rules had a high fidelity, were highly accurate and significantly more comprehensible, i.e., smaller, compared to trees produced with typical decision tree techniques.

The global optimization performed by GP, which is another often mentioned advantage, did not, however, outperform the greedy non-backtracking optimization

used by typical decision tree techniques when evaluated empirically. This result is most likely due to a search space that is too large or an ill-chosen parsimony pressure, two well-known weak spots of GP. However, these deficiencies can be handled by incorporating an amount of local search into the evolution, as done in the novel techniques presented in sections 6.1 and 6.2. Programs evolved using the suggested techniques significantly outperformed typical decision tree methods and clearly outperformed standard GP. Based solely on predictive performance, GP enhanced using the suggested techniques is hence a better method for creating decision trees.

Introns are another well-known disadvantage of GP, since they add unnecessary complexity to a program and hence lower the comprehensibility. To handle this deficiency, section 7.1 introduces a novel technique for the automatic removal of introns. After simplification, the programs were significantly smaller, compared to standard GP and decision tree techniques, while retaining the same predictive performance. An advantage of the suggested technique, compared to typical intron removal methods, is that it is independent of the underlying representation and does not need to be adapted to each new representation.

Finally, the inherent inconsistency of GP is often mentioned as a disadvantage. In this thesis, the contrary has instead been argued. Inconsistency is, in essence, a sign of informational instability, which a decision maker should be made aware of. Furthermore, section 6.3 presents two novel techniques which enhance probability estimates of the produced programs and create accurate kNN-ensembles by exploiting inconsistency. If a comprehensible model is required, a decision maker must, however, in the end choose a single program. In these situations, the novel technique suggested in section 7.2 can be used to guide the decision maker towards an interesting program that fits the available domain knowledge. Clearly, this must be considered a better approach than only presenting a single model selected on the basis of the arbitrary decision inherent in a specific algorithm. Consequently, inconsistency should be considered yet another advantage of GP, if exploited using the suggested enhancements.

Based on the conclusions above, it is quite clear that GP is well suited, in terms of the predictive accuracy and the comprehensibility of the evolved programs, for predictive modeling. Depending on which suggested enhancements were used, the evolved programs were either significantly more accurate or comprehensible than typical decision tree techniques. An important final note is that most of the suggest enhancements are implemented in the open source GPPM framework G-REX (www.grex.se), thus making them available to both practitioner and researchers.

9 Discussion and Future work

- Section 5.1.1 clearly shows the benefit of being able to optimize arbitrary score function for a single model. However, in general, ensembles are more accurate than single models, which can also be seen in section 6.3.1, where GP's inherent inconsistency was exploited to create accurate and diverse ensembles. Hence, it would be interesting to investigate whether GP can be used to evolve ensembles specialized for a certain score function. Today, all ensemble creation techniques are aimed at producing ensembles of traditional data mining techniques that are typically predefined to optimize a single score function. GP is exceptionally well suited for this task, since it may produce diverse classifiers using all training data while optimizing arbitrary score function.
- The work on guiding the decision maker to the best solution for him or her should be continued. As seen in the above examples, several representations can be used and numerous programs can be found for each representation. Hence, there is a need of better support for choosing representation and selecting the best solution of the chosen representation. Restricting the representation may also lead to fewer overfitted trees.
- The technique of using an imaginary ensemble, proposed in section 6.3.1, was very successful and its basic idea is, in essence, not constricted to GP. Therefore, it would be interesting to evaluate whether it could be adapted for decision trees and other techniques that aim to produce comprehensible models. A set of alternative decision trees could, for example, be created using different settings or random subsets of instances and attributes. Naturally, an imaginary ensemble could be created in the same way and used to select the tree most similar to the ensemble. Alternatively, a real ensemble technique, like random forest, could be used to create the benchmark predictions. Since random forests are both very powerful and robust, a decision tree that produces similar predictions should also generalize better.
- Tree injection seems to be a very successful approach for handling cases where GP struggles due to large search spaces. However, the proposed technique was not evaluated on regression problems, which naturally often have even larger search spaces due to a numerical target. Future

work regarding tree injection should hence use another decision tree technique which can produce both classification and regression trees, e.g., J48, to facilitate the injection of regression trees. With regard to classification, some work remains, to ensure that the injected trees are not unnecessarily large, which was the case for a few of the datasets in section 6.2.

10 References

Alexander, W.P. & Grimshaw, S.D. 1996. Treed regression. *Journal of Computational and Graphical Statistics*, pp.156–175.

Andrews, R. & Geva, S. 1995. RULEX \& CEBP networks as the basis for a rule refinement system. *Hybrid problems, hybrid solutions*, pp.1–12.

Andrews, R., Diederich, J. & Tickle, A. 1995. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6), pp.373–389.

Angeline, P.J. 1994. Genetic programming and emergent intelligence. *Advances in genetic programming*, 1, pp.75–98.

Archetti, F. et al., 2007. Genetic programming for computational pharmacokinetics in drug discovery and development. *Genetic Programming and Evolvable Machines*, 8(4), pp.413–432.

Armstrong, J.S. 2001. *Principles of forecasting: a handbook for researchers and practitioners*. Boston, Mass. London: Kluwer Academic Publishers.

Armstrong, J.S. & Collopy, F. 1992. Error measures for generalizing about forecasting methods: Empirical comparisons. *International Journal of Forecasting*, 8(1), pp.69–8.

Baker, J.E. 1985. Adaptive selection methods for genetic algorithms in *Proceedings of the 1st International Conference on Genetic Algorithms*. L. Erlbaum Associates Inc., pp. 101–111.

Banzhaf, W., Nordin, P., Keller, R.E. & Francone, F. 1998. *Genetic Programming an Introduction, On the Automatic Evolution of Computer Programs and Its Applications.* San Francisco, California: Marogan Kaufmann Publishers, Inc.

Banzhaf, W., Harding, S., Langdon, W. B. & Wilson, G., 2009. Accelerating Genetic Programming through Graphics Processing Units. *Genetic Programming Theory and Practice VI*, pp. 229-247.

Berry, M.J.A. & Linoff, G. 2004. Data mining techniques: for marketing, sales, and customer relationship management. New York, Chichester: Wiley.

Bianco, S. et al., 2008. An Evolutionary Framework for Colorimetric Characterization of Scanners. , pp.245–254.

Bianco, S. & Gasparini, F., 2008. An evolutionary framework for colorimetric characterization of scanners. Applications of Evolutionary Computing, *Lecture Notes in Computer Science*, 4974, pp.245–254.

Bishop, C., 1995. *Neural networks for pattern recognition*, Oxford University Press.

Blanco, V.R., Hernandez, O.J. & Ramirez, Q.M., 2004. Analysing the trade-off between comprehensibility and accuracy in mimetic models. *Lecture Notes in Computer Science*, 3245, pp.1–8.

Blake, C. & Merz, C. 1998. UCI repository of machine learning databases.

Bohanec, M. & Bratko, I., 1994. Trading accuracy for simplicity in decision trees. *Machine Learning*, 15(3), pp.223–25.

Bojarczuk, C.C., Lopes, H.S. & Freitas, A.A. 1999. Discovering comprehensible classification rules using genetic programming: a case study in a medical domain in *Proceedings of the Genetic and Evolutionary Computation Conference*. Citeseer, pp. 953–958.

Bojarczuk, C.C., Lopes, H.S. & Freitas, A.A. 2001. Data Mining with Constrained-Syntax Genetic Programming: Applications in Medical Data Set. *Intelligent Data Analysis in Medicine and Pharmacology - a workshop at MedInfo-2001*, 6, p.7.

Bot M. and Langdon, W. 2000, "Application of genetic programming to induction of linear classification trees," in *European Conference on Genetic Programming*, pp. 247–258.

Breiman, L. 1996. Bagging predictors. *Machine Learning*, 24(2), pp.123-14.

Breiman, L. 1984. *Classification and regression trees*. Chapman & Hall/CRC.

Brier, G.W. 1950. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1), pp.1–3.

Brown, G., Wyatt, J., Harris, R., Yao, X., 2005. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1), pp.5–20.

Cano, A., Zafra, A. & Ventura, S., 2010. Solving Classification Problems Using Genetic Programming Algorithms on GPUs. *Proceedings of the 5th international conference on Hybrid Artificial Intelligence Systems*, pp 17-26, Springer-Verlag.

Cano, A., Zafra, A. & Ventura, S., 2012. Speeding up the evaluation phase of GP classification algorithms on GPUs. *Soft Computing*, Volume 16, pp. 187–202.

Chapman, P. et al. 1999. *CRISP_DM 1.0 Step-by-step data mining guide*. CRISP DM Consortium. Available at: http://www.crisp-dm.org.

Cohen, W.W. 1995. Fast effective rule induction in *International Conference on Machine learning*. Tahoe City, CA: Morgan Kaufman, pp. 115–123.

Craven, M.W. & Shavlik, J.W. 1996. Extracting tree-structured representations of trained networks. *Advances in neural information processing systems*, pp.24–3.

Craven, M.W. & Shavlik, J.W. 1999. Rule extraction: Where do we go from here. University of Wisconsin Machine Learning Research Group Working Paper, pp.99–100.

Darwin, C. 1859. *The origin of species* 1985th ed. (J. Murray, ed.). Penguin Classics.

Demšar, J. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7, p.3.

Dietterich, T.G. 1996. Editorial. *Machine Learning*, 2(24), pp.1-3.

Dietterich, T.G. 1997. Machine learning research: four directions, *The AI Magazine*, 18: 97-136.

Dobra, A. & Gehrke, J. 2002. SECRET: a scalable linear regression tree algorithm, *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 481–487.

Domeniconi, C., Yan, B. 2004: Nearest neighbor ensemble. *Proceedings of the 17th International Conference on Pattern Recognition*, vol. 1, pp. 228–231. IEEE Computer Society, Los Alamitos, CA, USA.

Domingos, P., 1998. Knowledge discovery via multiple models. *Intelligent Data Analysis*, pp.1–18.

Domingos, P. 1999. The role of Occam's razor in knowledge discovery. *Data Mining and Knowledge Discove*ry, 3(4), pp.409–425.

Dunham, M.H. 2003. *Data mining: Introductory and advanced topics*. Pearson Education India.

Dunn, O.J. 1961. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293), pp.52–64.

Edmonds, A.N., Burkhardt, D. & Adjei., O. 1995. Genetic programming of fuzzy logic production rules in *Evolutionary Computation*. IEEE, pp. 765-77.

Eggermont, J. & Hemert, J.I. van 2001. Adaptive Genetic Programming Applied to New and Existing Simple Regression Problems in *Genetic Programming - Lecture Notes in Computer Science.*, pp. 23-35.

Eggermont, J. 2002. Evolving fuzzy decision trees with genetic programming and clustering. *Genetic Programming*.

Eggermont, J. 2005. Data Mining using Genetic Programming*: Classification and Symbolic Regression*. , 27(2), p.179.

Eggermont, J., Eiben, A. & Hemert, J.I. van 1999. Adapting the fitness function in GP for data mining. *Lecture Notes in Computer Science*, (2), pp.193–202.

Eggermont, J., Kok, J.N. & Kosters, W.A. 2004. Genetic programming for data classification: Partitioning the search space in Proceedings of the 2004 *ACM symposium on Applied computing*. ACM, pp. 1001–1005.

Espejo, P.G., Ventura, S. & Herrera, F. 201. A Survey on the Application of Genetic Programming to Classification. *IEEE Transactions on Systems, Man and Cybernetics*, 40(2), pp.121-144.

Falco, I. De, Della Cioppa, A. & Tarantino, E. 2002. Discovering interesting classification rules with genetic programming. *Applied Soft Computing*, 1(4), pp.257–269.

Fawcett, T. 2006. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8), pp.861-874.

Fix, E. & Hodges, J.L., 1952. Discriminatory analysis. Nonparametric discrimination: Consistency properties, Randolpf Field, Texas.

Fonlupt, C., 2001. Solving the ocean color problem using a genetic programming approach. *Applied Soft Computing*, 1, pp.63–72.

Fonseca, C. & Fleming, P., 1993. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. Conference on genetic algorithe on genetic algorithms, (July).

Francois, J.M. 2004. jaDTi - Decision Trees: a Java implementation.

Fraser, A., Weinbrenner, T. 1997. GPC++ v0.5.2. http://www0.cs.ucl.ac.uk/staff/ucacbbl/ftp/weinbenner/gp.html

Ferreira, C., 2006. Gene expression programming: mathematical modeling by an artificial intelligence, Springer.

Freitas, A.A. 2007. A review of evolutionary algorithms for data mining. *Soft Computing for Knowledge Discovery and Data Mining*. Springer: Berlin, pp.61–93.

Freitas, A.A. 2002. A survey of evolutionary algorithms for data mining and knowledge discovery. *Advances in Evolutionary Computation*, pp.819–845.

Friedman, M. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200), pp.675–701.

Fürnkranz, J. & Widmer, G. 1994. Incremental reduced error pruning in *International Conference on Machine Learning*. San Mateo, CA: Citeseer, pp. 70-77.

Gagn´e, C. Gardner, M. & Parizeau, M. 2007. Open BEAGLE v4.0. https://code.google.com/p/beagle/.

George Mason University's ECLab, 2013. ECJ 20. http://cs.gmu.edu/~eclab/projects/ecj/.

Goldberg, D.E. & Deb, K. 1991. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*, 1, pp.69–93.

Goodwin, P. 2002. Integrating management judgment and statistical methods to improve short-term forecasts. *Omega*, 30(2), pp.127–135.

Grąbczewski, K. & Duch, W. 2002. Heterogeneous Forests of Decision Trees. *Artificial Neural Networks* (ICANN).

Groovy Jaca Genetic Prgoramming 2006, jgprog, http://jgprog.sourceforge.net/

Hand, D. 2009. Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine learning*, 77(1), pp.103-123.

Hassan, G., 2010. Multiobjective genetic programming for financial portfolio management in dynamic environments. University College London.

Holland, J.H. 1992. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. The MIT Press.

Hornik, K. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2), pp.251–257.

Hsu, W.H. & Gustafson, S.M. 2001. Wrappers for automatic parameter tuning in multiagent optimization by genetic programming in *Workshop on Wrappers for Performance Enhancement in Knowledge Discovery in Databases* (KDD). Seattle, Washington, USA.

Hyafil, L. & Rivest, R.L. 1976. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1), pp.15–17.

Iba, H., De Garis, H. & Sato T., 1994, "Genetic programming using a minimum description length principle," *Advances in genetic programming*, pp. 265–284.

Iman, R.L. & Davenport, J.M. 1979. Approximations of the critical region of the Friedman statistic.

Jabeen, H. & Baig, A.R. 201. Review of Classification Using Genetic Programming. *International Journal of Engineering Science and Technology*, 2(2), pp.94–103.

Java Rapid GP, 2009. rgp v1.01, http://jrgp.sourceforge.net/

Johansson, U., Boström, H. & König, R., 2008. Extending Nearest Neighbor Classification with Spheres of Confidence. In International Florida AI Research Symposium Conference (FLAIRS). AAAI Press, pp. 282–287.

Johansson, U., König, R. & Niklasson, L. 2003. Rule extraction from trained neural networks using genetic programming in *International Conference on Artificial Neural Networks and International Conference on Neural Information Processing*. Istanbul Turkey, pp. 13-16.

Johansson, U., König, R. & Niklasson, L. 2004. The truth is in there-rule extraction from opaque models using genetic programming in *International Florida AI Research Symposium Conference (FLAIRS)*. Miami Beach, USA, pp. 658-663.

Johansson, U., König, R. & Niklasson, L. 2005. Automatically balancing accuracy and comprehensibility in PM in *International Conference on Information Fusion*. Philadelphia, U.S.A, pp. 1554-156.

Johansson, U. Löfström, T. König, R. Niklasson, L., 2006a. Why Not Use an Oracle When You Got One. *Neural Information Processing-Letters and Reviews*, 10(8-9), pp.227–236.

Johansson, U., Löfström, T., König, R., Sönströd, S, Niklasson, N., 2006b. Rule Extraction from Opaque Models–A Slightly Different Perspective. In *International Conference on Machine Learning and Applications, (ICMLA)*. pp. 22–27

Johansson, U., König, R. & Niklasson, L. 2007. Inconsistency - Friend or Foe in *International Joint Conference on Neural Networks*. IEEE, pp. 1383-1388.

Johansson, U., König, R. & Niklasson, L. 2008. Evolving a Locally Optimized Instance Based Learner in *International Conference on Data Mining (DMIN)*. Las Vegas, Nevada, pp. 124-129.

Johansson, U., König, R. & Niklasson, L. 2010a. Genetic rule extraction optimizing brier score in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, pp. 1007–1014.

Johansson, U., König, R. & Niklasson, L. 2010b. Genetically evolved kNN ensembles. Data Mining, *Annals of Information Systems*, 8, pp.299–313.

Johansson, U., König, R., Löfström, T., & Niklasson, L. 2008. Increasing rule extraction accuracy by post-processing GP trees. IEEE Congress on Evolutionary Computation (pp. 3005-3010). IEEE.

Johansson, U., König, R., Löfström, T. & Niklasson, L. 201. Using Imaginary Ensembles to Select GP Classifiers in *European Conference on Genetic Programming.* Springer, pp. 278–288.

Johansson, U., König, R., Löfström, T., Sönströd, C., & Niklasson, L. 2009. Post-processing Evolved Decision Trees. *Foundations of Computational Intelligence*, vol 4: Bio-Inspired Data Mining, 149-164. Springer.

Johansson, U., Sönströd, C., Löfström. T., König, R. 2009. Using Genetic Programming to Obtain Implicit Diversity. *In IEEE Congress on Computational Intelligence*. Trondheim, Norway, pp. 2454 – 2459.

Johansson, U. Sönströd, C. & Löfström,T. 2010. Oracle Cohached Decision Trees and Lists*. Advances in Intelligent Data Analysis IX, Lecture Notes in Computer Science,* v.6065, pp. 67-78.

Johnston, M., Liddle, T. & Zhang, M. 2010. A Relaxed Approach to Simplification in Genetic Programming. *Genetic Programming*, pp.110–121.

Jong, K.A. De 1988. Learning with genetic algorithms: An overview. *Machine Learning*, 3(2), pp.121–138.

Jong, K.A. De & Sarma, J. 1993. Generation gaps revisited. *Foundations of genetic algorithms*-2, pp.19–28.

Kass, G.V. 198. An exploratory technique for investigating large quantities of categorical data. Journal of the Royal Statistical Society. Series C (*Applied Statistics*), 29(2), pp.119–127.

Kecman, V. 2001. Learning and soft computing: support vector machines, neural networks, and fuzzy logic models. Cambridge, Mass.: The MIT press.

Keijzer, M. 2003. Improving symbolic regression with interval arithmetic and linear scaling. *Genetic Programming*, pp.275–299.

Kinzett, D., Zhang, M. & Johnston, M. 2008. Using numerical simplification to control bloat in genetic programming. *Simulated Evolution and Learning*, pp.493–502.

Koza, J.R. 1992. Genetic programming: on the programming of computers by means of natural selection. Cambridge, Massachusetts: MIT Press.

Koza, J.R. 1989. Hierarchical genetic algorithms operating on populations of computer programs in *International Conference on Artificial* Intelligence (IJCAI). Detroit, MI, USA: Morgan Kaufman, pp. 768-774.

Kretowski, M. and Czajkowski, M. 2010. An evolutionary algorithm for global induction of regression trees. *Artifical Intelligence and Soft Computing: Part II* (2010), 157–164.

Krogh, A. & Vedelsby, J. 1995. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, 7, pp.231–238.

König, R., Johansson, U. & Niklasson, L. 2010. Finding the Tree in the Forest in *IADIS International Conference Applied Computing. Timisoara*: IADIS Press, pp. 135-142.

König, R., Johansson, U. & Niklasson, L. 2008a. G-REX: A Versatile Framework for Evolutionary Data Mining in *IEEE International Conference on Data Mining Workshops*, 2008. ICDMW'08., pp. 971–974.

König, R., Johansson, U. & Niklasson, L. 2007. Genetic programming - a tool for flexible rule extraction in I*EEE Congress on Computational Intelligence*. IEEE, pp. 1304-131.

König, R., Johansson, U. & Niklasson, L. 2006. Increasing rule extraction comprehensibility. *International Journal of Information Technology and Intelligent Computing*, 1(2), pp.303–314.

König, R., Johansson, U. & Niklasson, L. 2008b. Using Genetic Programming to Increase Rule Quality in *International Florida AI Research Symposium Conference FLAIRS Proceedings*. AAAI Press, pp. 288–293.

König, R., Johansson, U., Löfström, T. & Niklasson, L. 2010. Improving GP classification performance by injection of decision trees in *IEEE Congress on Computational Intelligence*. IEEE, pp. 1–8.

Langdon, W.B. 2000. Size fair and homologous tree crossovers for tree genetic programming. *Genetic Programming and Evolvable Machines*, 1(1), pp.95–119.

Langdon, W. B., 2011. Graphics processing units and genetic programming: an overview. Soft Computing, 15, pp. 1657-1669.

Langdon, W.B. & Nordin, P. 2004. Seeding genetic programming populations. *Genetic Programming*, pp.304–315.

Langdon, W.B. & Poli, R. 1997. Fitness causes bloat. *Soft Computing in Engineeringn Design and Manufacturing*, (June), pp.23-27.

Laplace, P.S. 1814. *Théorie analytique des probabilité*. Paris, France: Paris, Ve. Courcier.

Lee, W.-C. 2006. Genetic Programming Decision Tree for Bankruptcy Prediction. *Proceedings of the 9th Joint Conference on Information Sciences (JCIS)*, pp.4-7.

Leichsenring, C., Hanheide, M., & Hermann, T. 2011. A Hybrid Symbolic-Subsymbolic Transformation Architecture as a Whiteboxing Tool for Machine Learning Systems Targeted at Non-Expert Users. *Technical report*. Bielefeld, Germany

Lewis, T. E. & Magoulas, G. D., 2009. Strategies to minimise the total run time of cyclic graph based genetic programming with GPUs. *Genetic and Evolutionary Computation Conference Proceedings,* ACM, pp. 1379-1386, Montréal Québec.

Li, X.R. & Zhao, Z. 2001. Measures of performance for evaluation of estimators and filters in *Proc. 2001 SPIE Conf. on Signal and Data Processing of Small Targets*. Citeseer, pp. 530–541.

Li, X.R. & Zhao, Z. 2005. Relative error measures for evaluation of estimation algorithms in *Proceedings 8th International Conference on Information Fusion*, 25-28 July 2005, Philadelphia, PA. Citeseer, pp. 211-218.

Yan, L. 2008, WEKA GP v3.4.12. http://www.leyan.org/tiki-index.php?page=Genetic%20Programming.

Makhoul, J., Kubala, F., Schwartz, R. & Weischedel, R. 1999. Performance measures for information extraction in *Broadcast News Workshop'99 Proceedings*, p. 249.

Makridakis, S. 1993. Accuracy measures: theoretical and practical concerns. *International Journal of Forecasting*. 9, 4 (1993), 527–529.

Margineantu, D.D. 2003. Improved class probability estimates from decision tree models. *Lecture notes in statistics*, pp.173–188.

Marmelstein, R.E. & Lamont G., 1998, "Pattern Classification using a Hybrid Genetic Program Decision Tree Approach," in *Genetic Programming*, pp. 223–231.

Martens, D., Baesens, B., Van Gestel, T., & Vanthienen, J. 2007. Comprehensible credit scoring models using rule extraction from support vector machines. *European journal of operational research*, 183(3), 1466-1476.

Martens, D., Huysmans, J. Vanthienen, J., Baesens, B. 2008. *Rule Extraction from Support Vector Machines: An Overview of Issues and Application in Credit Scoring*, Studies in Computational Intelligence (SCI) 80, 33–63, Springer verlag.

Martens, D., Baesens, B., & Van Gestel, T. 2009. Decompositional rule extraction from support vector machines by active learning. *Knowledge and Data Engineering*, IEEE Transactions on, 21(2), 178-191.

Mitchell, T.M. 1997. *Machine Learning*. New York: McGraw-Hill.

Montana, D.J. 1995. Strongly Typed Genetic Programming. *Evolutionary Computation*, 3(2), pp.199-23.

Mugambi, E.M. & Hunter, A., 2003. Multi-objective Genetic Programming Optimization of Decision Trees for Classifying Medical Data. *Lecture Notes in Computer Science*, 2773, pp.293–299.

Murthy, S.K. 1998. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4), pp.345–389.

Nemenyi, P. 1963. *Distribution-free multiple comparisons*, Princeton University.

Nordin, P., Francone, F. & Banzhaf, W. 1996. Explicitly defined introns and destructive crossover in genetic programming. *Advances in Genetic Programming*, pp.111-134.

Oka, S. & Zhao, Q. 2007. Design of Decision Trees through Integration of C4. 5 and GP in *Japan-Australia Joint Workshop on Intelligent and Evolutionary Systems.*, pp. 128-135.

O'Neill, M., Vanneschi, L., Gustafson, S.M. & Banzhaf, W. 201. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4), pp.339-363.

Otero, E.B., Castle, T., Johnson, G. 2012. EpochX: Genetic Programming in Java with Statistics and event Monitoring. In proceeding of Genetic and Evolutionary Computation Conference. , http://www.epochx.org.

Papagelis, A. & Kalles, D. 2001. Breeding decision trees using evolutionary techniques in *International Conference on Machine Learning*. Citeseer, pp. 393–400.

Papagelis, A. & Kalles, D. 2002. GATree v2. http://www.gatree.com/-

Perceiptio Ehf. 2007. AI Solver Studio v1.0, http://www.aisolver.com/.

Piatetsky-Shapiro, G. 2007. KDnuggets Methodology Poll. , pp.[2011-07-19]. Available at: http://www.kdnuggets.com/polls/2007/data_mining_methodology.htm.

Plish, V.E. 1998. Algorithms generating alternative solutions for a multicriterion linear programming model. *Cybernetics and Systems Analysis*, 34(2), pp.301–304.

Poli, R. & Langdon, W.B. 1998. On the search properties of different crossover operators in genetic programming. *Genetic Programming*, pp.293–301.

Poli, R. & McPhee, N.F. 2008. Covariant parsimony pressure in genetic programming. *Technical Report*, Department of Computing and Electronic Systems University of Essex, UK.

Poli, R. & McPhee, N.F. 2003a. General schema theory for genetic programming with subtree-swapping crossover: Part I. *Evolutionary Computation*, 11(1), pp.53–66.

Poli, R. & McPhee, N.F. 2003b. General schema theory for genetic programming with subtree-swapping crossover: Part II. *Evolutionary Computation*, 11(2), pp.169–206.

Poli, R., Langdon, W.B. & McPhee, N.F. 2008. *A field guide to genetic programming*.

Potgieter, G. & Engelbrecht, A.P. 2008. Evolving model trees for mining data sets with continuous-valued classes. *Expert Systems with Applications*, **35**(4), pp.1513-1532.

Provost, F. & Domingos, P., 2003. Tree induction for probability-based ranking. Machine Learning, 52(3), pp.199–215.

Provost, F., Fawcett, T. & Kohavi, R. 1998. The case against accuracy estimation for comparing induction algorithms in *Proceedings of the Fifteenth International Conference on Machine Learning*. San Franciscio, CA: Morgan Kaufmann, pp. 445-453.

Punch, B. & Zongker, D. 1998. Lil-gp v1.1. http://garage.cse.msu.edu/software/lil-gp/

Quinlan, J. R. 1992. Learning with continuous classes, *5th Australian joint conference on artificial intelligence.*, pp. 343–348.

Quinlan, J. R. 1993. C4.5: *programs for machine learning*. Morgan Kaufmann.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning*, 1(1), pp.81-106.

Quinlan, J. R. 1987. Simplifying decision trees. *International Journal of man-machine studies*, 93.

Quinlan, J. R. 2005. See5 Version 2.02, http://www.rulequest.com

Reynolds, C.W. 1993. An evolved, vision-based behavioral model of coordinated group motion in From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior., pp. 0–262.

Rissanen, J. 1978. Modeling by shortest data description. *Automatica*, 14(5), pp.465–471.

RML Technologies, Inc. 2013. Dicipulus 5.2, www.rmltech.com.

Roiger, R. & Geatz, M. 2003. *Data Mining: A tutorial-based primer*. Addison Wesley.

SAS Instruments 2011. SEMMA.

Schaffer, C. 1994. A conservation law for generalization performance in *International Conference on Machine Learning.*, pp. 259-265.

Schapire, R.E. 1990. The strength of weak learnability. *Machine Learning*, 5(2), pp.197-227.

Scherrod, H.P. 2003. DTREG v.10.7.3. http://www.dtreg.com

Sheskin, D. 2004. Handbook of parametric and nonparametric statistical procedures. CRC Pr I Llc.

Schmidt M., Lipson H. (2009) "Distilling Free-Form Natural Laws from Experimental Data," *Science*, Vol. 324, no. 5923, pp. 81 - 85.

Stefanova, L. & Krishnamurti, T.N. 201. Interpretation of seasonal climate forecast using Brier skill score. *Journal of Climate*, 15(5), pp.537-544.

Syswerda, G. 1991. A study of reproduction in generational and steady-state genetic algorithms in *Proceedings of the First Workshop on Foundations of Genetic Algorithms*. Indiana, USA: Morgan Kaufman, pp. 94–101.

Sönströd, C., Johansson, U. & König, R. 2007. Towards a Unified View on Concept Description in *International Conference on Data Mining (ICDM)*. Las Vegas, Nevada, pp. 59-65.

Sönströd, C., Johansson, U., König, R. & Niklasson, L. 2008. Genetic Decision Lists for Concept Description in *International Conference on Data Mining (ICDM)*. Las Vegas, Nevada, pp. 450-457.

Tackett, W.A. 1994. Recombination, selection, and the genetic construction of computer programs.

Tan, K.C., Tay, A., Lee, T.H. & Heng, C.M. 2002. Mining multiple comprehensible classification rules using genetic programming. *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02* (Cat. No.02TH8600), pp.1302-1307.

Thrun, S.B. 1993. Extracting provably correct rules from artificial neural networks. University of Bonn.

Towell, G. & Shavlik, J.W. 1993. The extraction of Refined Rules from knowledge-basedneural networks. *Machine Learning*, 13(1), pp.71-101.

Turney, P. 1995. Technical note: Bias and the quantification of stability. *Machine Learning*, 20(1-2), pp.23-33.

Wettschereck, D. & Dietterich, T., 1994. Locally adaptive nearest neighbor algorithms. *Advances in Neural Information Processing Systems.*

Wilcoxon, F. 1945. Individual comparisons by ranking methods. Biometrics Bulletin, 1(6), pp.80–83.

Wilt, C.M., Thayer, J.T. & Ruml, W. 2001. A Comparison of Greedy Search Algorithms in *Third Annual Symposium on Combinatorial Search*.

Winkler, S.M., Affenzeller, M. & Wagner, S. 2008. Using enhanced genetic programming techniques for evolving classifiers in the context of medical diagnosis. *Genetic Programming and Evolvable Machines*, 10(2), pp.111-14.

Witten, I.H. & Frank, E. 2005. Data mining: practical machine learning tools and techniques 2nd ed. Amsterdam: Morgan Kaufmann.

Wolpert, D.H., 1992. Stacked generalization. *Neural networks,*5 (505).

Wolpert, D.H., 1995. 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.

Xie, H., Zhang, M. & Andreae, P., 2006. Genetic programming for automatic stress detection in spoken english. *Applications of Evolutionary Computing*, pp.460–471.

Zadeh, I.A. 1965. Fuzzy Sets. *Information and control*, 8, pp.338-353.

Zavrel, J., 1997. An empirical re-examination of weighted voting for k-nn. Proceedings of the 7th Belgian-Dutch Conference on Machine Learning, pp.139–148.

Zhang, G., Patuwo, E.B. & Hu, M.Y. 1998. Forecasting with artificial neural networks::: The state of the art. *International journal of forecasting*, 14(1), pp.35–62.

Zhang, M., Wong, P. & Qian, D. 2006. Online program simplification in genetic programming. *Simulated Evolution and Learning*, pp.592–600.

11 Appendix

11.1 The G-REX framework

G-REX is an open source programming framework for PM using genetic programming, but it also has a GUI that facilitates access to predefined functionality. The GUI comprises two main parts; a left pane containing all relevant settings and a right desktop pane where windows monitoring the evolution and the results are shown during execution. The settings pane is itself divided into two parts; i.e., the upper pane consists of setting concerning the dataset and the representation, while the tabbed pane below contains settings regarding the evolution process, benchmarking, how the results should be reported and how larger experiments should be performed. Each of these parts is described in detail below.

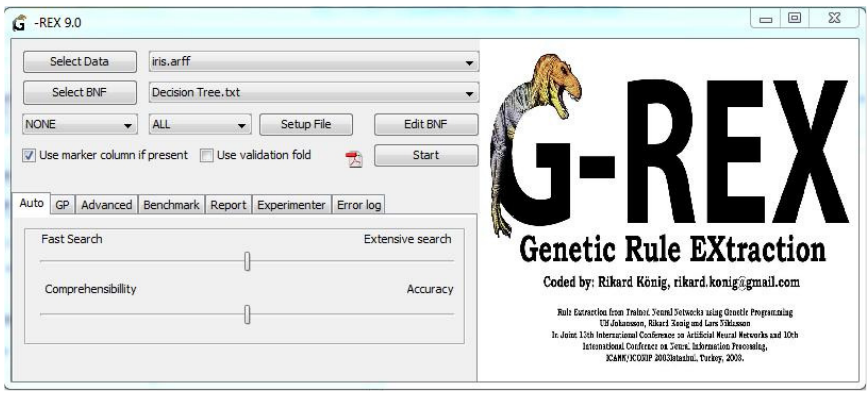

Figure 38 – Overview of the G-REX GUI

11.1.1 Settings for dataset and representation

The basic settings of G-REX are always available in the top part of the settings pane of the G-REX GUI, see Figure 38 above. Here, a dataset can be selected using the topmost combo box which contains all datasets present in the *Dataset* folder in G-REX's root folder. Files that reside in other locations may be selected using the *Select Data* button. A BNF for a specific representation can be chosen in the same way using the button and combo box in the following row. In the standard G-REX package, the following representations are present in the G-REX's *BNF* folder and will hence be available in the combo box:

- Boolean Tree - e.g., see Figure 27
- Decision Tree – e.g., see Figure 26
- Decision Tree with Boolean conditions – e.g., see Figure 29
- Decision List – e.g., see Figure 29
- Decision List with Boolean Conditions
- Fuzzy rules - e.g., see (Eggermont 2002).
- Hybrid k-NN – e.g., see Figure 33

The third row contains two more combo boxes which are used to select the part of the dataset that should be used as validation set and test set. *NONE* specifies that no validation set should be used and is default, since G-REX's standard fitness functions do not utilize validation sets. If *n*-fold cross-validation is used, *ALL* signifies that G-REX should evolve one program for each fold. Finally, there are also two buttons which open internal windows for manipulation of the dataset, i.e., Figure 39, and the chosen representions, Figure 40.

The last row contains two checkboxes which decide whether a predefined partitioning of the dataset should be used if availible, or if G-REX should be responsible for creating the partitions. To use a predefined partitioning, the last column of the dataset specifies the role of each instance, using *TRAIN, VAL,* and *TEST.*

Finally, the *Start* button starts the evolutions using the selected dataset, BNF, and the setting defined in the tabbed setting pane. The *pdf* icon presented to the left of the *Start* button is used throughout the GUI to link to related research papers. Clicking on a *pdf* icon will open one of the articles residing in the *Publications* folder in G-REX's root folder in an external browser.

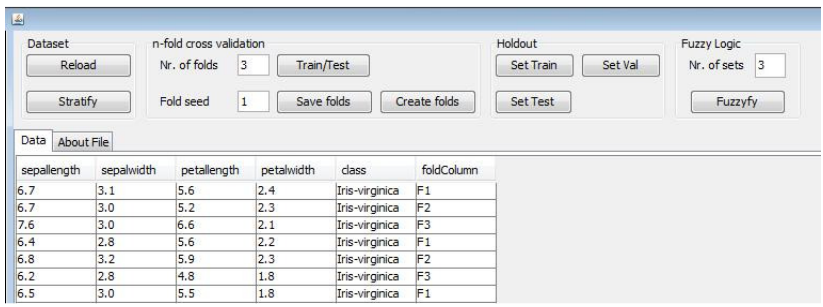

Figure 39 - G-REX's data manipulation options

The lower part of G-REX's data manipulation window consists of a tabbed pane with one tab, *Data,* that presents the data and another, *About File,* that shows any

comments included in the file. The upper pane divides the functionality into four distinct groups, where the first contains a button for reloading the dataset and another for stratifying the data. Stratifcation first sorts the instances that are based on the target variable and then randomizes, based on the *fold seed*, the order of instances with the same target value. Stratification is mainly used in combination with *n*-fold cross-validation which is specified by setting the number of folds and clicking the *Create folds* button. Folds are then created by simply assigning each *n*-instance to a specific fold. A simple holdout sample can also be created by pressing *Train/Test,* which assigns each *n*th instance to the test set and the remaining to the training set. Finally, it is possible to save the folds in *n* separate files.

A holdout set can also be created by marking instances using the mouse and then pressing *Set Train, Set Val,* or *Set Test* to assign the corresponding role to a certain part of the dataset.

Finally, the dataset can be automatically fuzzified, as described by Eggermont 2002, by finding *k* mediods using the *k-means* clustering algorithm and then creating *k* triangular membership functions with the mediods as centers. When programs are evolved using the *fuzzy rules* BNF, the dataset is automatically fuzzified using this functionality.

The *Edit BNF* button in the basic settings opens a simple text editor which can be used to modify the currently selected BNF, see section 11.1.8 for more details about how to design a BNF.

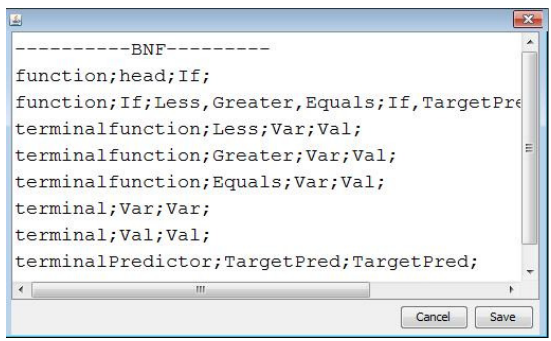

Figure 40 - G-REX's BNF text editor

11.1.2 Settings for Evolution

Settings related to the evolution and the reporting of the results are availible in the tabbed pane below the basic setting. The first tab, *Auto*, selected in Figure 38, contains a simple way of setting reasonable settings for users who are not familiar

with GP. Two sliders are used to control how thorough the search should be, i.e., fast to extensive, and whether accuracy or comprehensibility should be favored during evolution. The sliders affect the settings in the *GP* tab to make the effect of the sliders transparent.

11.1.3 Basic GP Settings

The *GP* tab, shown in Figure 41, contains all settings related to the evolution of programs for the BNF, dataset, and evaluation scheme selected in the basic settings.

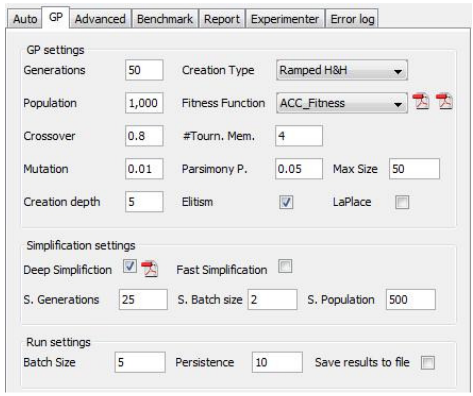

Figure 41 – G-REX's GP settings

The settings are divided into three groups, where *GP settings* contains typical GP settings, *Simplification settings* regards the removal of introns, and *Run settings* concerns general settings for the execution.

GP settings

Besides the typical GP settings that could be expected of a GP framework, several predefined fitness functions are available for both classification and regression tasks:

Fitness functions for classification:

- *ACC_Fitness* – optimizing accuracy, as defined in equation 33.
- *AUC_Fitness* – optimizing the area under the ROC curve, as defined in equation 40.
- *BRI_Fitness* – optimizing the brier score, as defined in equation 46.
- *BRE_Fitness* – optimizing brevity, as defined in equation 41.

Fitness functions for regression:

- *MAE_Fitness* – optimizing the mean absolute error, according to equation 44.
- *RMSE_Fitness* – optimizing the root mean square error, according to equation 44.
- *MAPE_Fitness* – optimizing the mean absolute percentage error, according to equation 43.
- *MUAPE_Fitness* – optimizing the mean unbiased absolute percentage error, according to equation 43.
- *CORR_Fitness* – Optimizing the Pearson correlation, i.e., (1-*r*).

Note that all fitness functions are minimized, which means that the fitness relates to the error of the optimized performance metric. Even if these fitness functions were sufficient for most applications, it is simple to create new ones by extending the *FitnessFunction* class and simply overriding the *calcPredictionError* method.

It may also be noted that G-REX supports both *tournament* and *roulette wheel* selection and that *#Tourn. Mem.* defines the number of programs that should be selected in each tournament, when using *tournament selection*. *Roulette wheel* selection is used, if the number of tournament members is set to zero.

Parsimony P. sets the parsimony pressure, as defined in equation 33, with the addition that a maximum size can be set, i.e., *Max Size*. Since all G-REX fitness functions should be normalized in the range of zero to one, a reasonable parsimony pressure should be between *0.1* and *0.001,* with a typical pressure of *0.01*. Programs which become larger than the *Max Size* will receive an additional large parsimony pressure that all but guarantees they are not selected for the next generation.

Finally, the *LaPlace* check box decides whether probabilities should be calculated as the relative frequency of the class values of the training instances in a leaf or whether this estimate should be corrected using *laplace*.

Simplification settings

The next group of settings regards the removal of introns, i.e., simplification of the final program of an evolution. There are two types of simplification that can be used together or individually. *Deep Simplification* is an implementation of the technique suggested in section 7.1 which performs a second evolution with the predictions of the final program as target. *Deep Simplification* is applicable to arbitrary representation and can remove both unused and unnecessary nodes.

Fast Simplification is a fast way to remove unused nodes, i.e., nodes not reached by any training instances, of a program, by recursively replacing all nodes that are parents to an unused leaf node, with its other child node. Even if this technique is

fast and effective, it is only applicable to BNFs based on the *If* node and cannot remove nodes that are used but unnecessary for the final prediction. It can be noted that the results reported in section 7.1 only use *Deep Simplification,* even if the techniques can be used together.

Run settings

The final group relates to settings regarding how a G-REX run should be performed. *Batch Size* decides how many evolutions should be performed for each fold. Each batch creates a new randomly initialized population and, if more than one batch is used, the program with the highest training fitness is reported as the final program. More than one batch is normally used to reduce the risk that the evolution could fasten in a local minimum.

Persistence is a stopping criterion that stops the evolution, if no improvement in fitness has been done in the specified number of generations. If set to zero, all generations specified in the *GP-settings* will be performed. Finally, *Save results to file* decides whether the results of a run should be saved to files or just displayed in the GUI.

11.1.4 Advanced GP settings

The advanced settings of G-REX relate to three extensions of standard GP, i.e., rule extraction, Tree Injection, and hybrid kNN models.

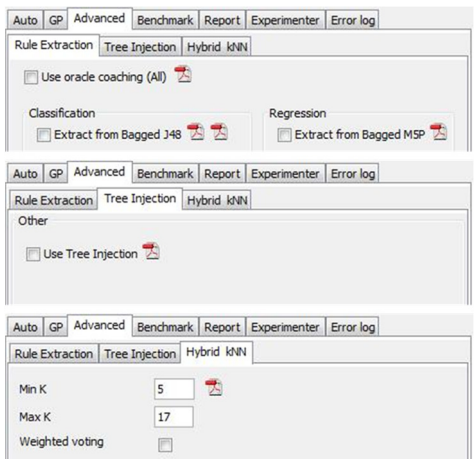

Figure 42 – G-REX's advanced setting

The benefit of rule extraction is demonstrated in section 5.1.2. It is one of the most powerful features of G-REX and yet very simple to use. For classification problems, rules are extracted from an ensemble of bagged J48 trees, while an ensemble of bagged M5P trees is used for regression problems. Rules can be extracted from any WEKA model, but at the moment this requires a few simple changes in the code base, for details see www.grex.se. Finally, oracle coaching, which is even more powerful than traditional rule extraction, see (Johansson et al. 2006a), can be applied in the cases where a batch of test instances is available.

Tree injection, i.e., the techniques suggested in 0, is controlled in the *Tree Injection* tab by a single checkbox which decides whether tree injection should be used or not. Here, however, the injected trees are created using J48 instead of jaDTi. J48 is used, since it clearly outperformed jaDTI in the original study, it is more well-known, and it is integrated in the WEKA framework which eliminates another external dependency.

Finally, settings regarding the hybrid kNN trees, suggested in section 5.1.3, are available under the *Hybrid kNN* tab. Note that these settings are only relevant if the *Hybrid k-NN* BNF is selected. Three different versions of the hybrid representation were evaluated in the study, but here only the *global* variant, i.e., different *k*s are used for different parts of the dataset but all instances are considered when finding the nearest neighbors, is available, since it was the most successful technique. The available settings are the lowest and highest *k* that should be considered and whether to use weighted voting or not.

11.1.5 Result settings

There are two ways to control which results G-REX reports. First, it is possible to select the techniques that G-REX should be benchmarked against, under the *Benchmark tab.* The techniques are implemented in WEKA and use the exact same data partitions as G-REX, to facilitate a fair comparison. The only difference to how they are implemented in WEKA is that G-REX's own functionality for handling missing values is used, even if a technique has an alternative way of handling missing values. G-REX handles missing values by replacing them with the mean values for numeric variables and the mode, i.e., the majority class, for categorical variables.

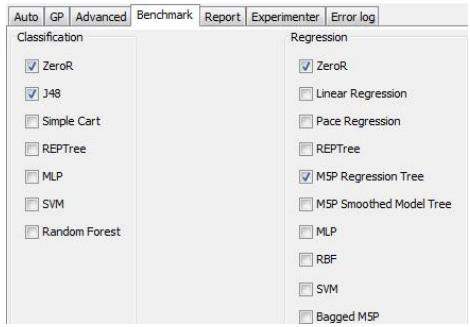

Figure 43 - G_REX's benchmark settings

Secondly, the *Report* tab contains settings for the performance metrics that should be used and whether the results of the training set should also be reported. It is also possible to use other metrics, by specifying the class name and package in the *Custom Metrics* text fields.

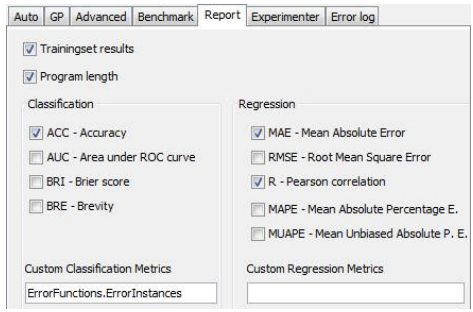

Figure 44 - G-REX's report settings

11.1.6 Settings for large experiments

The *experimenter* facilitates easy definition of larger experiments over several datasets and settings. The *Add* button simply adds an experiment using the specified dataset and settings. If the same settings are to be tested over several datasets, the *Add folder* can be used to add experiments with the same settings for all datasets in a folder. *Add TSFolder* does the same thing, but sets up a time series experiment by specifying that *1*%, i.e. 1 / number of folds of the last instances in the dataset that should be used as test set. Finally, *Del* or *Dell All* can be used to delete the selected experiment or all experiments.

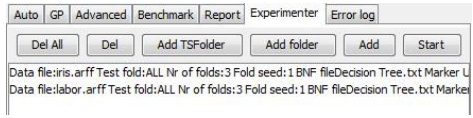

Figure 45 - G-REX's experimenter tab

When the tab's own *Start* button is clicked, G-REX will run each experiment in order and report the result, after each completed experiment, to the GUI and a file named using settings of the experiment. If some settings differ between experiments, these will be replaced with an X in the file name e.g.:

```
Exp-DecisionTree-X-gen50-popX-crs0.8-mut0.01-bat3-
per10-max50-crd5-crt0-pap0.05-lpcFALSE-extFALSE-ID-
1369338211710.grex
```

Here, the first X shows that different BNFs have been used in the experiment and the second X shows that the population size also differed. In the file, the results of a certain experiment are identified using the dataset file name and the used fitness function or the name of WEKA techniques. An experiment result file is most easily analyzed using a pivot-table in Excel.

11.1.7 Monitoring Evolution

When evolving programs, G-REX displays a window where a GP run can be monitored in the application desktop pane. Since a GP run may consist of several evolutions, the fold and batch will be presented as a tab attached to the left side of the window.

Each tab is divided into three parts, where the first presents the current fitness and the selected performance metrics on the left side, together with the currently best-known program on the right. The middle part of the window presents the predictive performance graphically. For classification, each class is represented on a separate row, with the correct class marked using a black border. Correct predictions are marked in green and incorrect classifications with red. The probability estimate for incorrect classes is also displayed using red, but with opacity related to the magnitude of the estimate. Training instances are depicted on the left side of the black vertical line and test instances on the right. Since G-REX uses stratification by default, the true classes are sorted during the partitioning of the dataset, which explains the stair-like pattern.

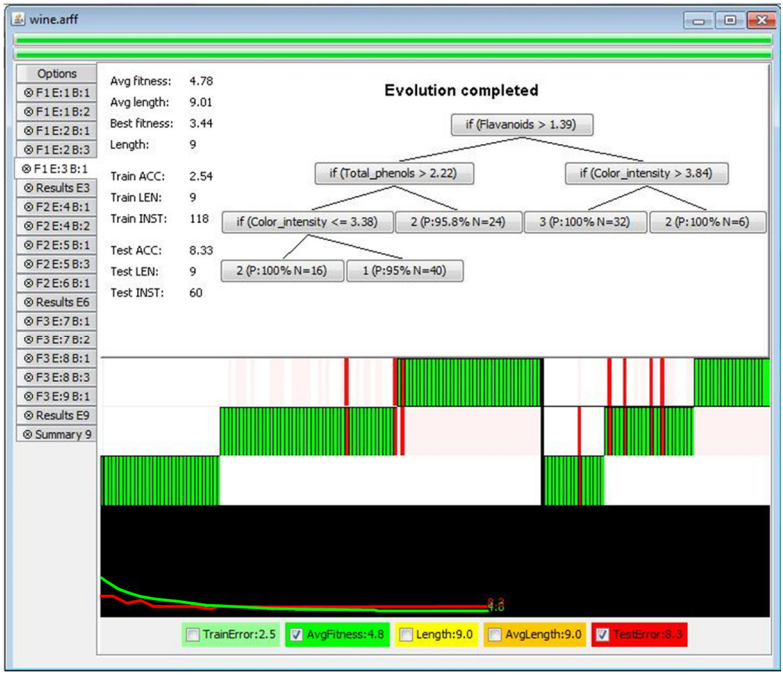

Figure 46 – Monitoring Evolution for classification tasks

Figure 46 shows an example evolution of a G-REX run for the wine datasets. The datasets have three different classes and, hence, the graph window has three rows. Red predictions are incorrect, but there are also light pink markings which signify that the program has made the correct classification but not with 100% confidence.

The lower part of an evolution tab displays statistics related to the evolution. By checking the corresponding checkboxes, it is possible to display the fitness, train error, length, and test error of the best individual and the average fitness and length of the entire population for each generation. Train and test error is calculated using the fitness function with the parsimony pressure set to zero.

For regression problems, the actual values are plotted using red and the predicted using blue. Figure 47 below shows an example evolution tab for the *fishcatch* dataset. Again, the dataset is stratified and hence sorted in relation to the target variable, which explains why this regression dataset looks like a typical time series problem when displayed. Here, the evolved program uses the *Model tree* BNF, i.e., the *fomt* technique suggested in section 6.1. For normal regression trees, clicking on a program leaf will highlight the corresponding prediction in the graph.

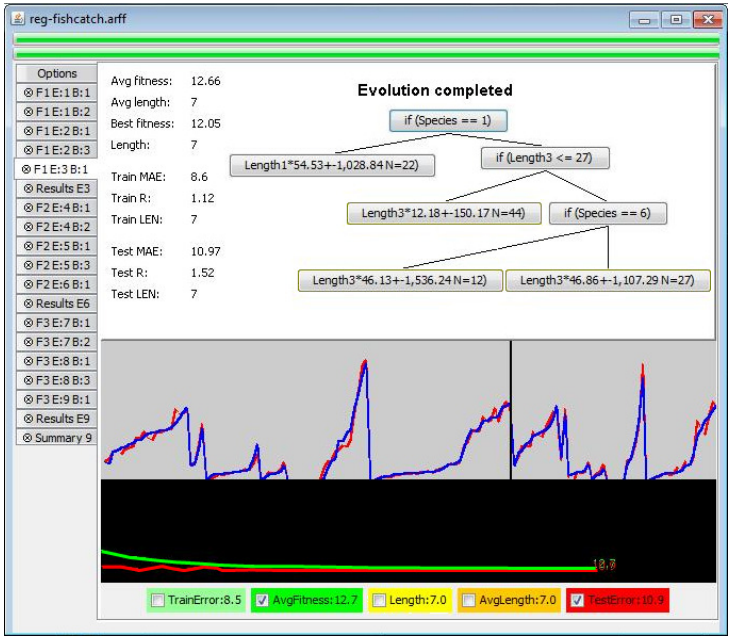

Figure 47 – Monitoring Evolution for regression tasks

11.1.8 Customizing representations

Several representations are predefined in G-REX, but a user can also create a new representation in external text files. The syntax used to define a representation is similar to the Backus-Naur form and the text files are hence called BNFs. Rules for how to define a G-REX representation are presented in Table 64.

Reserved words: **function, terminal, terminalfunction, terminalPredictor, head**	
program	::= **function**;**head**;parameter;\nline...line
line	::= functionDef|terminalDef\n
functionDef	::= function;class;parameter;...;parameter;
parameter	::= arg,...,arg
arg	::= class that is an allowed argument for the associated parameter
terminalDef	::= terminalType;class;compatibalClass,...,compatibalClass;
terminalType	::= **terminal**|**terminalfunction**|**terminalpredictor**
class	::= G-REX java class extending Gene. Must be defined in the program.
compatibleClass	::= class which can take the place of associated class in a program

Table 64 - BNF for the G-REX's representation specification language

Table 65 shows the actual BNF for the DT representation definied in Table 66. Bold letters are used to mark reserved words ";" to divide arguments "," and to separate items in a list; all other words are names of G-REX classes.

```
function;head;If;
function;If;Less,Greater,Equals;If,TargetPred;If,TargetPred;
terminalfunction;Less;Var;Val;
terminalfunction;Greater;Var;Val;
terminalfunction;Equals;Var;Val;
terminal;Var;Var;
terminal;Val;Val;
terminalPredictor;TargetPred;TargetPred;
```

Table 65 - BNF for the Decision Tree representation

The first line defines that the head of the program should be an *If* function, but it is also possible to allow more than one starting function or even to start with a terminal. The actual definition of the representation starts at line two. Since an *If* function has three arguments, i.e., its condition and what to do if it is *true* or *false*, it also has three parameter lists that are separated using ";" after the name specification. As mentioned in Chapter 4 , the elements in a list can be repeated to adjust its selection probability during the creation of the initial population.

The next three lines define the three conditions that are allowed as *terminalfunction,* which are functions that only have terminals as parameters. It is necessary to specify terminal functions, to allow the *Grow* creation scheme which creates trees with a specific depth. Note that this representation does not allow conditions based on two variables, but that this can be achieved by simply extending the last argument list of the conditions with *var:*

```
terminalfunction;Less;Var;Val;
```

becomes:

```
terminalfunction;Less;Var;Val,Var;
```

Next, the two terminals *var* and *val* are defined. The difference between terminals and functions is that terminals do not take any parameters. The terminals' second parameter list instead defines which function and terminals they are compatible with; i.e., can be switched with in a crossover. Here, the terminals are only allowed to be switched with the same type of variable.

Finally, the last row specifies that the class *TargetPred* is a *terminalPredictor* and should be used to make the actual predictions.

As mentioned above, there are several predefined representations in the BNF folder of the G-REX root folder. However, they can all be modified and a new one can be created using any of G-REX predefined nodes.

Basic predefined nodes

G-REX has a multitude of predefined nodes that can be used in any representation. The following section describes the most important and frequently used nodes. Note that a node must be defined in a BNF with the exact name and letter case given below, since G-REX uses Java's reflection functionality to create objects.

- *TargetPred* - Makes a prediction based on the training instances reaching the node. The majority class is predicted for classification problems, while a randomly initialized value between the highest and lowest value of the target variable, of the instances reaching the leaf, is the prediction for regression problems.
- *If* - A typical if-else function that evaluates its first argument as true, if it returns a value greater than zero. All other values are evaluated as false. If the first argument is true, the second argument is evaluated, if not, the third.
- *Less, Greater, Equals* - Conditional nodes, representing <=, >,=, typically used as the conditional argument in an *If* statement. Returns 1 (true) or 0 (false) according to its condition. Note that *Less* actually stands for less or equal, while *Greater* is strictly greater. *Greater* and *Less* only take numerical variables as input, while *Equals* takes categorical variables.
- *Var* – Node representing an input variable. Automatically types a variable as categorical or numerical based on the specifications in data files which use WEKA's .arff format.
- *Val* – Represents a value of a corresponding variable and must hence always be argument to a function with a *Val* and a *Var* parameter, e.g., *Less*, *Greater* or *Equals*.
- *Val* – Represents a value of a corresponding variable and must hence always be argument to a function with a *Val* and a *Var* parameter, e.g., *Less*, *Greater* or *Equals*. *Val* nodes have internal values between 0 and 1 which are assigned randomly and then remain constant during the evolution. When evaluated in a function, the internal value is translated to a value of the corresponding *Var* by first calculating a value index by multiplying it with the number of unique values of the variable and then

selecting the value corresponding to that index from a sorted list of the unique variable values. In this way, a low internal value will always correspond to a low real value, regardless which variable the node is associated with. Hence, the meaning of the variable is retained, even if a genetic operator like crossover were to move the variable to another program.

- *And, Or* – Boolean nodes typically used in the conditional part of an *If* statement or to create Boolean decision trees, returns 1 if true and 0 if false;
- Plus, Times – Mathematical node operators which return the sum respectively the product of the node's two children nodes.

11.2 G-REX BNFs

Functions	{ *if*, <=, >, = }
Terminals	{ *conV*, *catV*, C_{depV}, C_{conV} , C_{catV} }
program	::= *expression*
expression	::= *if-statement* | C_{depV}
if-statement	::= if *condition* then *expression* else *expression*
condition	::= *conV* <= C_{conV} | *conV* > C_{conV} | *catV* = C_{catV}
C_{depV}	::= Constant initialized to a random value of the dependent variable
C_{conV}	::= Constant initialized to a random value of *conV*
C_{catV}	::= Constant initialized to a random value of *catV*
conV	::= Independent continuous variable
catV	::= Independent categorical variable

Table 66 - BNF for Decision trees (DT)

Functions	{ *if*, <=, >, = }
Terminals	{ *conV*, *catV*, C_{depV}, C_{conV} , C_{catV} }
program	::= *expression*
expression	::= *if-statement* | C_{depV}
if-statement	::= if *condition* then *expression* else *expression*
condition	::= *conV* <= C_{conV} | *conV* > C_{conV} | ***conV* > *conV*** | *catV* = C_{catV} / ***catV* = *catV***
C_{depV}	::= Constant initialized to a random value of the dependent variable
C_{conV}	::= Constant initialized to a random value of *conV*
C_{catV}	::= Constant initialized to a random value of *catV*
conV	::= Independent continuous variable
catV	::= Independent categorical variable

Table 67 – BNF for extended decision trees (EDT)

Functions	{ *if*, <=, >, =, ***AND***, ***OR*** }
Terminals	{ *conV*, *catV*, C_{depV}, C_{conV} , C_{catV} }
program	::= *expression*
expression	::= *if-statement* | C_{depV}
if-statement	::= if *condition* then *expression* else *expression*
condition	::= *conV* <= C_{conV} | *conV* > C_{conV} | *catV* = C_{catV} | ***boolOp***
boolOp	::= *condition* ***AND*** *condition* | *condition* ***OR*** *condition*
C_{depV}	::= Constant initialized to a random value of the dependent variable
C_{conV}	::= Constant initialized to a random value of *conV*
C_{catV}	::= Constant initialized to a random value of *catV*
conV	::= Independent continuous variable
catV	::= Independent categorical variable

Table 68 – BNF for Boolean decision trees (BDT)

Functions	{ *if*, <=, >, =, *AND*, *OR*}
Terminals	{ *catV*, *conV*, C_{conV} , C_{catV}}
program	::= *boolOp*
boolOp	::= *condition AND condition* | *condition OR condition*
condition	::= *conV* <= C_{conV} | *conV* > C_{conV} | *catV* = C_{catV} | *boolOp*
C_{conV}	::= Constant initialized to a random value of *conV*
C_{catV}	::= Constant initialized to a random value of *catV*
conV	::= Independent continuous variable
catV	::= Independent categorical variable

Table 69 – BNF for Boolean rules (BR)

Functions	{ *if*, <=, >, =}
Terminals	{ *catV*, *conV*, C_{depV}, C_{conV} , C_{catV}}
program	::= *expression*
expression	::= *if-statement* | C_{depV}
if-statement	::= if *condition* then C_{depV} else *expression*
condition	::= *conV* <= C_{conV} | *conV* > C_{conV} | *catV* = C_{catV}
C_{depV}	::= Constant initialized to a random value of the dependent variable
C_{conV}	::= Constant initialized to a random value of *conV*
C_{catV}	::= Constant initialized to a random value of *catV*
conV	::= Independent continuous variable
catV	::= Independent categorical variable
catV	::= Independent categorical variable

Table 70 – BNF for Decision Lists (DL)

Functions	{ *if*, <=, >, =}
Terminals	{ *catV*, *conV*, *C*, C_{conV} , C_{catV}, ***Prediction***}
program	::= *expression*
expression	::= *if-statement* | ***Prediction***
if-statement	::= if *condition* then *expression* else *expression*
condition	::= *conV* <= C_{conV} | *conV* > C_{conV} | *catV* = C_{catV}
C_{conV}	::= Constant initialized to a random value of *conV*
C_{catV}	::= Constant initialized to a random value of *catV*
conV	::= Independent continuous variable
catV	::= Independent categorical variable
Prediction	::= ***leafMin*** * ***C*** +***leafMax**** (1-***C***)
C	::= Constant initialized to a random value of 0-1.
leafMin	::= The minimum value of dependent variable for the instances reaching the leaf.
leafMax	::= The maximum value of dependent variable for the instances reaching the leaf.

Table 71 – BNF for regression trees (RDT)

Functions	{ *high, low, rather, very, AND, OR*}
Terminals	{ *catV, conV,* C_{conV} , C_{catV}}
program	::= *expression*
expression	::= *fuzzyFunction* | *hedge* | *fuzzyOp*
fuzzyOp	::= *expression* AND *expression* | *expression* OR *expression*
hedge	::= *rather(fuzzyFunction)* | *very(fuzzyFunction)*
fuzzyFunction	::= *high*(*conV*) | *low*(conV)
conV	::= Independent continuous variable

Table 72 – BNF for fuzzy trees (FUZ)

Functions	*{ if, <=, >, = }*
Terminals	{ *1,3,...25, catV, conV,* C_{conV} , C_{catV}}
program	::= *expression*
expression	::= *if-statement* / ***kNN***
if-statement	::= if *condition* then *expression* else *expression*
condition	::= *conV* <= C_{conV} | *conV* > C_{conV} | *catV* = C_{catV}
C_{conV}	::= Constant initialized to a random value of *conV*
C_{catV}	::= Constant initialized to a random value of *catV*
conV	::= Independent continuous variable
catV	::= Independent categorical variable
kNN	::= ***global-kNN k | local-kNN k***
k	::= 1,3,..,25

Table 73 – BNF for mixed G-kNN programs (KN)

Functions	*{ if, <=, >, = }*
Terminals	{ *1,3,...25, catV, conV,* C_{conV} , C_{catV}, **[*Aw*$_1$,*Aw*$_2$,..., *Aw*$_i$]**}
program	::= *expression*
expression	::= *if-statement* | ***kNN***
if-statement	::= if *condition* then *expression* else *expression*
condition	::= *conV* <= C_{conV} | *conV* > C_{conV} | *catV* = C_{catV}
C_{conV}	::= Constant initialized to a random value of *conV*
C_{catV}	::= Constant initialized to a random value of *catV*
conV	::= Independent continuous variable
catV	::= Independent categorical variable
kNN	::= ***attribute-weights global-kNN k***
attribute-weights	::= **[*Aw*$_1$,*Aw*$_2$,..., *Aw*$_i$]**
Aw$_i$	::= Constant initialized to a random value of 0-1.
k	::= 1,3,..,25

Table 74 – BNF for G- kNN programs with attribute weights (KNW)

Functions	*{ if, <=, >, = }*
Terminals	*{ catV, conV, C_{dep}, C_{conV} , C_{catV}, **linear-regression** }*
program	::= *expression*
expression	::= *if-statement* | ***linear-regression***
if-statement	::= if *condition* then *expression* else *expression*
condition	::= *conV* **<=** C_{conV} | *conV* > C_{conV} | *catV* = C_{catV}
C_{conV}	::= Constant initialized to a random value of *conV*
C_{catV}	::= Constant initialized to a random value of *catV*
conV	::= Independent continuous variable
catV	::= Independent categorical variable
linear-regression	::= ***k * conV + m*** where *k* and *m* are calculated using the least square method for the instances reaching the node.

Table 75 – BNF for GP-LM

11.3 Datasets Used for Evaluation

The experiments in this study are performed on datasets from a wide range of domains. All datasets are available to the public from the UCI Repository and are frequently used to evaluate machine learning algorithms. The actual datasets were downloaded from the dataset collection packages at the WEKA homepage. The actual file name is used as the name for each data set. The motivation for using these benchmark datasets is that they are all gathered from real domains and therefore will contain problems that a decision support system may face in the real world. Each dataset used in the experiments is described briefly below.

11.3.1 Classification Datasets

- **Balance-scale.** A dataset concerning the prediction of the balance of a scale, depending on different weights.
- **Breast-cancer.** A breast cancer domain obtained from the University Medical Centre, Institute of Oncology, in Ljubljana, Slovenia (formerly Yugoslavia).
- **Breast-w.** A dataset collected by University of Wisconsin Hospitals, Madison, Wisconsin, USA, concerning the prediction of whether a detected case of breast cancer is malignant or benign.
- **Cmc.** This dataset is a subset of the 1987 National Indonesia Contraceptive Prevalence Survey. The samples are married women who were either not pregnant or did not know if they were at the time of interview. The problem is to predict the current contraceptive method choice (no use, long-term methods, or short-term methods) of a woman based on her demographic and socio-economic characteristics.
- **Colic.** A retrospective study concerning whether the lesion in cases of horse colic was surgical or not. Attributes contain the age of the horses, data from different medical diagnoses, the clinic at which they were treated, and whether the horse survived.
- **Contact-lenses.** Prediction of the proper contact-lenses for 24 patients.
- **Credit-a.** 690 credit card applications where all attribute names and values have been changed to meaningless symbols to protect confidentiality of the data.
- **Credit-g.** Data related to 1000 German credit card applications which among other things contain information about credit history, purpose of credit, amount, employment, age, etc.

- **Cylinder-bands.** A dataset concerning the presence of cylinder-bands in rotogravure printing.
- **Dermatology.** Classification of six different skin diseases within the erythemato-squamous family.
- **Diabetes.** Prediction of whether or not a patient shows signs of diabetes according to the World Health Organization criteria. All patients are females at least 21 years old of Pima Indian heritage.
- **Ecoli.** A biological dataset concerning classification of the localization site of a protein. Attributes consist of measurements of cytoplasm, persiplasm, and six membrane properties.
- **Glass.** This study was motivated by criminological investigation and concerns classification of glass using the proportions of the ingredients.
- **Haberman.** The data set concerns "five years survival" of patients who undergone surgery for breast cancer at the University of Chicago's Billings Hospital.
- **Heart-c.** Prediction of whether a patient at the Cleveland Clinic Foundation, has a heart disease or not. The attributes include age, gender, and the measurements of eight relevant characteristics of the patient's heart.
- **Heart-h.** Another dataset related to the prediction of possible heart disease for 273 patients at the Hungarian Institute of Cardiology.
- **Heart-statlog.** Similar to *heart-c* and *heart-s*, but in a slightly different form.
- **Hepatitis.** Prediction of the survival of patients with hepatitis, depending on treatment and disease-related measurements.
- **Hypothyroid.** A slightly different version of the Sick dataset where a few instances have been removed, making it even more unbalanced, and where the positive class has been changed to compensated hypothyroid.
- **Ionosphere.** This radar data was collected by a system in Goose Bay, Labrador. The system consists of a phased array of 16 high-frequency antennas with a total transmitted power in the order of 6.4 kilowatts. The idea is to predict good radar returns; i.e., those showing evidence of some type of structure in the ionosphere.
- **Iris.** This famous dataset contains 3 classes of 50 instances where each class refers to a type of iris plant. One class is linearly separable from the other two, but not linearly separable from each other.

- **Labor.** Final labor settlements in collective agreements reached in the business and personal services sector for organizations with at least 500 members (teachers, nurses, university staff, police, etc.) in Canada in 1987 and the first quarter of 1988.
- **Liver-disorders.** The problem is to predict whether or not a male patient has a liver disorder, based on blood tests and alcohol consumption. The first five attributes are all the blood tests which are thought to be sensitive to liver disorders that might arise from excessive alcohol consumption and the sixth is the number of alcoholic beverages drunk per day. Each instance constitutes the record of a single male individual.
- **Lung-cancer.** Identification of three different types of lung cancer based on 56 nominal attributes.
- **Lymph.** Detection of cancer using x-ray of lymph nodes and lymphatic vessels made visible by the injection of a special dye. This lymphography domain was obtained from the University Medical Centre, Institute of Oncology, in Ljubljana, Slovenia (formerly Yugoslavia).
- **Post-operative.** Here, the classification task is to decide if a patient should be sent to the next recovery area after a surgery. The attributes are mainly related to body temperature measurements.
- **Primary-tumor.** Identification of whether a primary tumor is present in the lungs or stomach based on seventeen nominal attributes.
- **Sick.** The task for this dataset is to diagnose whether a patient is sick with hyperthyroid or not.
- **Sonar.** This dataset contains instances obtained by bouncing sonar signals off a metal cylinder at various angles and under various conditions. The transmitted sonar signal is a frequency-modulated chirp, rising in frequency. Each attribute represents the energy within a particular frequency band, integrated over a certain period of time.
- **Tic-Tac-Toe.** This dataset contains all possible endgame board configurations for the tic-tac-toe game. The goal is to decide if player X won the game.
- **Vehicle.** 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 vehicle may be viewed from one of many different angles.

- **Vote.** This dataset includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the CQA. The task is to classify whether a congressman is a democrat or a republican, depending on these votes.
- **Waveform-5000.** This is an artificial three-class problem based on three waveforms.
- **Wine.** Recognition of wine based on the chemical analysis of wines grown in the same region of Italy, but derived from three different cultivars. The analysis determines the quantities of 13 constituents found in each of the three types of wines.
- **Zoo.** A database containing 17 Boolean-valued animal attributes. The "type" to be predicted is: mammal, bird, reptile, fish, amphibian, insect, or invertebrate.

For a summary of the characteristics of the datasets see Table 76. *Size* is the number of instances in the dataset. *Classes* are the number of output classes in the dataset. *Num.* is the number of numeric attributes, *Nom.* is the number of nominal attributes and *Naïve* is the proportion of the majority class, i.e., the accuracy achieved by only predicting the majority class. *Naïve* is presented as reference point for each dataset and shows the worst possible performance that a reasonable classifier should achieve.

Dataset	Size	Classes	Num.	Nom.	Naïve
Balance-scale	576	2	4	0	50%
Breast-cancer	286	2	0	9	78%
Breast-w	699	2	9	0	66%
Cmc	1473	3	2	7	43%
Colic	368	2	7	15	63%
Contact-lenses	24	3	0	4	63%
Credit-a	690	2	6	9	56%
Credit-g	1000	2	7	13	70%
Cylinder-bands	540	2	18	20	58%
Dermatology	366	6	1	33	31%
Diabetes	768	2	8	0	65%
Ecoli	336	8	7	0	43%
Glass	214	7	9	0	36%
Haberman	306	2	2	1	74%
Heart-c	303	2	6	7	54%
Heart-h	294	2	6	7	64%
Heart-statlog	270	2	5	8	56%
Hepatitis	155	2	6	13	79%
Hypothyroid	3675	2	7	22	98%
Ionosphere	351	2	34	0	64%
Iris	150	3	4	0	33%
Labor	57	2	8	8	65%
Liver-disorders	345	2	6	0	58%
Lung-cancer	32	3	0	56	41%
Lymph	148	4	3	15	55%
Post-operative	90	3	1	7	71%
Primary-tumor	123	2	0	17	68%
Sick	3772	2	7	22	94%
Sonar	208	2	60	0	53%
Tic-tac-toe	958	2	0	9	65%
Vehicle	846	4	18	0	26%
Vote	435	2	0	16	61%
Waveform-5000	5000	3	21	0	34%
Wine	178	3	13	0	40%
Zoo	101	7	0	17	41%

Table 76 - Characteristics of classification datasets

11.3.2 Estimation Datasets

The following section presents the different estimation datsets used in the various experiments. The actual datasets were downloaded from the dataset collection at the WEKA hompage, i.e. *datasets-numerics.jar*. The actual file name is used as the name for each data set.

- **Auto-price.** The dataset consists of car specifications from which the price of the cars should be predicted. There are three types of attributes that describe an instance: car specifications, assigned insurance risk rating and its value loss compared to other cars.
- **Basketball.** Prediction of the number of points scored per minute for 96 basketball players.
- **Bolts.** A data set concerning the optimization of a machine packaging bolts.
- **Cholesterol.** Prediction of the cholesterol level of 303 heart disease patients. The patients are the same as in the classification dataset *heart-c*.
- **Cloud.** Data from a cloud-seeding experiment in Tasmania. The target is the amount of rain in the eastern target region.
- **Diabetes-numeric.** The objective is to investigate the dependence of the level of serum C-peptide on the various other factors, in order to understand the patterns of residual insulin secretion. The response measurement is the logarithm of C-peptide concentration (pmol/ml) at the diagnosis, and the predictor measurements of age and base deficit, a measure of acidity.
- **Gascons.** A data set concerning the gasoline consumption between 1960 and 1986.
- **Housing.** The input variables are statistics related to the living standard in the different suburbs of Boston. The target variable is the median value of owner-occupied homes in a certain suburb.
- **Machine-cpu.** The problem concerns predicting the relative CPU performance based on six CPU properties, such as cycle time and memory size, etc.
- **Pharynx.** A dataset consisting of patients with squamous carcinoma in 3 sites of the mouth and throat (in the oropharynx). The objective of the study was to compare the two treatment policies with respect to patient survival time.

- **Pollution.** Prediction of the mortality rate per 100.000 habitants based on the different air pollution- and demographic related attributes.
- **Quake.** Estimation of the magnitude of earthquakes, on the richter scale, based on longitude, latitude and the focal depth.
- **Sensory.** Data from a sensory evaluation experiment. The target is the score of different wines, given by the judges participating in the experiment.
- **Servo.** A data set concerning a very non-linear phenomenon, i.e., the prediction of the rise time of a servomechanism.
- **Sleep.** Includes brain and body weight, life span, gestation time, predation and danger indices for 62 mammals. The target is the total time the mammal spends sleeping.
- **Veteran.** The dataset contains data about veterans with lung cancer. The input variables consist of information about the patients, the type of cancer and the treatment. The target variable is the patient's survival time.
- **Wisconsin.** Each record in the dataset represents follow-up data for a breast cancer study. The target is the recurrence time of the cancer and the input variables are measures for the cell nucleus.

Table 77 shows the general properties of the datasets with the same abbreviations that are used for the classification datasets. In addition the mean value (*Mean*), the standard deviation (*Std*), the maximum value (*Max*) and the minimum value (*Min*) are also presented.

Dataset	Size	Num.	Nom.	Mean	Std	Max	Min
Auto-price	159	14	1	11446	5878	35056	5118
Basketball	96	4	0	0.42	0.11	0.83	0.16
Bolts	40	7	0	33.9	27.4	89.3	7.32
Cholesterol	303	6	7	246.7	51.8	564	126
Cloud	108	4	2	1.23	1.08	6	0
Diabetes-numeric	43	2	0	4.8	.72	6.6	3
Gascons	27	3	0	207.0	43.8	269.4	129.7
Housing	506	13	1	22.5	9.2	50	5
Machine-cpu	209	6	0	105.6	161	1150	6
Pharynx	195	2	10	555.5	422	1823	0
Pollution	60	15	0	940.4	62.2	1113.2	790.7
Quake	2178	3	0	5.98	0.19	6.9	5.8
Sensory	576	0	11	15.1	0.82	17.5	12.5
Servo	167	0	4	1.39	1.56	7.1	0.131
Sleep	62	7	0	8.5	5.8	20	0
Veteran	137	3	4	121.6	158	999	1
Wisconsin	194	32	0	46.9	34.5	125	1

Table 77 - Characteristics of estimation datasets

Publications *in the series*
Örebro Studies in Technology

1. Bergsten, Pontus (2001) *Observers and Controllers for Takagi – Sugeno Fuzzy Systems*. Doctoral Dissertation.
2. Iliev, Boyko (2002) *Minimum-time Sliding Mode Control of Robot Manipulators*. Licentiate Thesis.
3. Spännar, Jan (2002) *Grey box modelling for temperature estimation*. Licentiate Thesis.
4. Persson, Martin (2002) *A simulation environment for visual servoing*. Licentiate Thesis.
5. Boustedt, Katarina (2002) *Flip Chip for High Volume and Low Cost – Materials and Production Technology*. Licentiate Thesis.
6. Biel, Lena (2002) *Modeling of Perceptual Systems – A Sensor Fusion Model with Active Perception*. Licentiate Thesis.
7. Otterskog, Magnus (2002) *Produktionstest av mobiltelefonantenner i mod-växlande kammare*. Licentiate Thesis.
8. Tolt, Gustav (2003) *Fuzzy-Similarity-Based Low-level Image Processing*. Licentiate Thesis.
9. Loutfi, Amy (2003) *Communicating Perceptions: Grounding Symbols to Artificial Olfactory Signals*. Licentiate Thesis.
10. Iliev, Boyko (2004) *Minimum-time Sliding Mode Control of Robot Manipulators*. Doctoral Dissertation.
11. Pettersson, Ola (2004) *Model-Free Execution Monitoring in Behavior-Based Mobile Robotics*. Doctoral Dissertation.
12. Överstam, Henrik (2004) *The Interdependence of Plastic Behaviour and Final Properties of Steel Wire, Analysed by the Finite Element Metod*. Doctoral Dissertation.
13. Jennergren, Lars (2004) *Flexible Assembly of Ready-to-eat Meals*. Licentiate Thesis.
14. Jun, Li (2004) *Towards Online Learning of Reactive Behaviors in Mobile Robotics*. Licentiate Thesis.
15. Lindquist, Malin (2004) *Electronic Tongue for Water Quality Assessment*. Licentiate Thesis.
16. Wasik, Zbigniew (2005) *A Behavior-Based Control System for Mobile Manipulation*. Doctoral Dissertation.

17. Berntsson, Tomas (2005) *Replacement of Lead Baths with Environment Friendly Alternative Heat Treatment Processes in Steel Wire Production*. Licentiate Thesis.

18. Tolt, Gustav (2005) *Fuzzy Similarity-based Image Processing*. Doctoral Dissertation.

19. Munkevik, Per (2005) *"Artificial sensory evaluation – appearance-based analysis of ready meals"*. Licentiate Thesis.

20. Buschka, Pär (2005) *An Investigation of Hybrid Maps for Mobile Robots*. Doctoral Dissertation.

21. Loutfi, Amy (2006) *Odour Recognition using Electronic Noses in Robotic and Intelligent Systems*. Doctoral Dissertation.

22. Gillström, Peter (2006) *Alternatives to Pickling; Preparation of Carbon and Low Alloyed Steel Wire Rod*. Doctoral Dissertation.

23. Li, Jun (2006) *Learning Reactive Behaviors with Constructive Neural Networks in Mobile Robotics*. Doctoral Dissertation.

24. Otterskog, Magnus (2006) *Propagation Environment Modeling Using Scattered Field Chamber.* Doctoral Dissertation.

25. Lindquist, Malin (2007) *Electronic Tongue for Water Quality Assessment*. Doctoral Dissertation.

26. Cielniak, Grzegorz (2007) *People Tracking by Mobile Robots using Thermal and Colour Vision*. Doctoral Dissertation.

27. Boustedt, Katarina (2007) *Flip Chip for High Frequency Applications – Materials Aspects*. Doctoral Dissertation.

28. Soron, Mikael (2007) *Robot System for Flexible 3D Friction Stir Welding*. Doctoral Dissertation.

29. Larsson, Sören (2008) *An industrial robot as carrier of a laser profile scanner. – Motion control, data capturing and path planning*. Doctoral Dissertation.

30. Persson, Martin (2008) *Semantic Mapping Using Virtual Sensors and Fusion of Aerial Images with Sensor Data from a Ground Vehicle*. Doctoral Dissertation.

31. Andreasson, Henrik (2008) *Local Visual Feature based Localisation and Mapping by Mobile Robots*. Doctoral Dissertation.

32. Bouguerra, Abdelbaki (2008) *Robust Execution of Robot Task-Plans: A Knowledge-based Approach*. Doctoral Dissertation.

33. Lundh, Robert (2009) *Robots that Help Each Other: Self-Configuration of Distributed Robot Systems.* Doctoral Dissertation.

34. Skoglund, Alexander (2009) *Programming by Demonstration of Robot Manipulators*. Doctoral Dissertation.

35. Ranjbar, Parivash (2009) *Sensing the Environment: Development of Monitoring Aids for Persons with Profound Deafness or Deafblindness*. Doctoral Dissertation.

36. Magnusson, Martin (2009) *The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loop Detection.* Doctoral Dissertation.

37. Rahayem, Mohamed (2010) *Segmentation and fitting for Geometric Reverse Engineering. Processing data captured by a laser profile scanner mounted on an industrial robot.* Doctoral Dissertation.

38. Karlsson, Alexander (2010) *Evaluating Credal Set Theory as a Belief Framework in High-Level Information Fusion for Automated Decision-Making.* Doctoral Dissertation.

39. LeBlanc, Kevin (2010) *Cooperative Anchoring – Sharing Information About Objects in Multi-Robot Systems*. Doctoral Dissertation.

40. Johansson, Fredrik (2010) *Evaluating the Performance of TEWA Systems*. Doctoral Dissertation.

41. Trincavelli, Marco (2010) *Gas Discrimination for Mobile Robots.* Doctoral Dissertation.

42. Cirillo, Marcello (2010) *Planning in Inhabited Environments: Human-Aware Task Planning and Activity Recognition.* Doctoral Dissertation.

43. Nilsson, Maria (2010) *Capturing Semi-Automated Decision Making: The Methodology of CASADEMA*. Doctoral Dissertation.

44. Dahlbom, Anders (2011) *Petri nets for Situation Recognition.* Doctoral Dissertation.

45. Ahmed, Muhammad Rehan (2011) *Compliance Control of Robot Manipulator for Safe Physical Human Robot Interaction.* Doctoral Dissertation.

46. Riveiro, Maria (2011) *Visual Analytics for Maritime Anomaly Detection*. Doctoral Dissertation.

47. Rashid, Md. Jayedur (2011) *Extending a Networked Robot System to Include Humans, Tiny Devices, and Everyday Objects*. Doctoral Dissertation.

48. Zain-ul-Abdin (2011) *Programming of Coarse-Grained Reconfigurable Architectures*. Doctoral Dissertation.

49. Wang, Yan (2011) *A Domain-Specific Language for Protocol Stack Implementation in Embedded Systems*. Doctoral Dissertation.

50. Brax, Christoffer (2011) *Anomaly Detection in the Surveillance Domain*. Doctoral Dissertation.

51. Larsson, Johan (2011) *Unmanned Operation of Load-Haul-Dump Vehicles in Mining Environments*. Doctoral Dissertation.

52. Lidström, Kristoffer (2012) *Situation-Aware Vehicles: Supporting the Next Generation of Cooperative Traffic Systems*. Doctoral Dissertation.

53. Johansson, Daniel (2012) *Convergence in Mixed Reality-Virtuality Environments. Facilitating Natural User Behavior.* Doctoral Dissertation.

54. Stoyanov, Todor Dimitrov (2012) *Reliable Autonomous Navigation in Semi-Structured Environments using the Three-Dimensional Normal Distributions Transform (3D-NDT)*. Doctoral Dissertation.

55. Daoutis, Marios (2013) *Knowledge Based Perceptual Anchoring: Grounding percepts to concepts in cognitive robots*. Doctoral Dissertation.

56. Kristoffersson, Annica (2013) *Measuring the Quality of Interaction in Mobile Robotic Telepresence Systems using Presence, Spatial Formations and Sociometry*. Doctoral Dissertation.

57. Memedi, Mevludin (2014) *Mobile systems for monitoring Parkinson's disease*. Doctoral Dissertation.

58. König, Rikard (2014) *Enhancing Genetic Programming for Predictive Modeling*. Doctoral Dissertation.