

# **Genetic Programming and Decision Trees Applied to Medical Data Mining**

**A Thesis Presented to  
The Faculty of the Computer Science Program  
California State University Channel Islands**

**In (Partial) Fulfillment  
of the Requirements for the Degree  
Masters of Science in Computer Science**

**By  
Kathy Macropol  
May 2007**

**© 2007**

**Kathy Macropol**

**ALL RIGHTS RESERVED**

**APPROVED FOR THE COMPUTER SCIENCE PROGRAM**

---

**Advisor: Dr. William Wolfe** **Date**

---

**Advisor: Dr. Andrzej Bieszczad** **Date**

---

**Advisor: Dr. Peter Smith** **Date****APPROVED FOR THE UNIVERSITY**

---

**Advisor: Dr. Gary A. Berg** **Date**

# **Genetic Programming and Decision Trees Applied to Medical Data Mining**

by

Kathy Macropol

Computer Science Program  
California State University Channel Islands

## **Abstract**

The digital revolution has brought an explosion of stored data to our world, and data mining, especially on the rapidly expanding medical databases, has the capabilities to turn this information into new and useful medical knowledge. Data mining with cost sensitive decision trees created using Genetic Programming is focused upon, with an emphasis on their relevance and potential in medical data mining. An application that uses elitist multiobjective Genetic Programming to obtain a pareto front of univariate and linear decision trees was implemented, and various mutation operators were tested and compared on their performance in the development of the decision trees.

## **Acknowledgements**

Dr. William Wolfe has given me inspiration and help throughout this thesis. I was lost as to where to focus on my research, and Dr. Wolfe gave me direction and sound advice. Without his outstanding knowledge, understanding, and encouragement, this project would have been impossible for me. Throughout the two years I have been in Cal State Channel Islands, Dr. Andrzej Bieszczad and his wife, Professor Anna Bieszczad have been my professors, mentors, friends, and support. Their passion and dedication to education have been a model to me, and their enthusiasm motivated me to reach for a higher goal. I would also like to thank Cal State Channel Islands for offering the wonderful Master's of Science in Computer Science program that has allowed me to obtain advanced education and enabled me to continue on. Armed with the knowledge that CSUCI has imparted to me, and knowing my family will be behind me through thick and thin, I feel grateful and blessed.

## Table of Contents

<b>1. Introduction .....</b>	<b>7</b>
1.1 Why Data Mining?.....	7
1.2 Classification Data Mining.....	8
1.2.1 Overview of Classifiers .....	9
1.2.2 Types of Decision Trees .....	11
1.3 Overview of the Thesis Document .....	12
<b>2. Creating and Evaluating Decision Trees.....</b>	<b>14</b>
2.1 Evaluation Basics .....	14
2.2 Greedy Heuristics .....	16
2.3 Genetic Programming .....	16
2.4 Cost Sensitivity .....	18
2.5 MultiObjective Optimization and Pareto Fronts.....	20
2.6 MultiObjective Genetic Programming.....	22
<b>3. Implementation.....</b>	<b>24</b>
3.1 Experiments.....	24
3.1.1 Addition of elitism .....	25
3.1.2 Experimenting with mutation .....	25
3.1.3 Adding Linear Capabilities .....	26
3.2 Overview of Program and Parameters.....	26
3.2.1 Datasets .....	26
3.2.2 Some Program Implementation Details .....	26
<b>4. Results .....</b>	<b>30</b>
4.1 With and Without Elitism .....	30
4.2 Mutation .....	34
4.3 Linear Decision Trees .....	36
<b>5. Analysis .....</b>	<b>37</b>
5.1 The Basic Algorithm.....	37
5.2 Elitism.....	37
5.3 Mutation .....	37
5.4 Linear Decision Trees .....	38

<b>6. Future Work .....</b>	<b>39</b>
-----------------------------	-----------

<b>7. References.....</b>	<b>40</b>
---------------------------	-----------

# Chapter 1

## 1. Introduction

### 1.1 Why Data Mining?

The digital revolution has brought an explosion of stored data to our world, and it's increasing at an ever faster rate. A study in 2003 found that the world produces between 1 and 2 exabytes of unique information per year [1], and the size and number of databases are growing rapidly (43). The ease and convenience of computers means a rising number of companies, organizations, and individuals are choosing to go paperless...collecting, converting, and storing their information digitally.

These databases contain masses of interesting knowledge, and their analysis is currently a very exciting field for researchers. The digital format means the information can be accessed or shared easily, and scientists across the world have joined in the hunt to discover valuable knowledge from this digital information.

This search is especially important in the medical field. Any new pattern or technique discovered could possibly make a difference between life or death for thousands of individuals. The medical databases are already in place and growing at incredible rates. During every doctor's visit, every hospital stay, or even at the pharmacy, patient information is entered into databases. The Health Care Financing Administration (HCFA) databases alone contain Medicare records on some 37 million older Americans, and Medicaid data from 29 States on 22 million additional beneficiaries [3]. Managed-care data, imaging data (like X-rays and MRIs), pharmacy records, even financial data all contain a wealth of knowledge. Discovering patterns and trying to understand the models beneath the data leads to better quality healthcare and medicine.

However, the collection of data is quickly outpacing the ability for humans to understand or make use of it all. With databases like the GEHC Patient Record database, which contains over 6.3 million unique patient records, growing by ~25-30% a month, it would take a team of researchers an inordinate amount of time to go through just a tenth of it [4]. To make things even more difficult, traditional data analysis tools and techniques will often not even work on these datasets, because of their size and complexity [5].

Data mining is a field that was born to deal with these difficulties. Defined as “the process of automatically discovering useful information in large data repositories” [5], data mining techniques go through large volumes of data quickly and attempt to transform that data in a way that will enhance the discovery of new knowledge. In a sense, they “mine” the mountains of data, looking for new nuggets of knowledge. This frequently takes the form of discovered patterns, insights, rules, or predictive models that can quite often even outperform the best human domain experts (43).

Data mining uses a combination of many disciplines...from statistics and pattern recognition to artificial intelligence and machine learning. Like statistics, data mining is concerned with turning data into knowledge. However, in statistics the data sets are simple, static, numeric, and often collected specifically to validate or refute a particular hypothesis. Data mining, in contrast, utilizes numerous varying types of data, and can automatically extract patterns or connections between items without having a prior hypothesis [6].

Data mining techniques have already been applied to many disciplines, including medicine (33) [7] [8] [9]. Medical applications of data mining include prediction of the effectiveness of surgical procedures, medical tests of medications, and discovery of relationships among clinical and pathological data [10] (8) (100) (101) (102) (104) (105) to name just a few.

Though data mining has been successful in many of its applications, it is still a relatively new field. New techniques and innovations are found frequently, and there is much potential for growth.

## 1.2 Classification Data Mining

There are many different categories of data mining. Classification data mining is concerned with assigning an individual or object to one of several predefined categories[5]. Other types of data mining, such as clustering (which attempts to automatically group data into related clusters based on their similarities) are also possible. However, much of medical data mining has been concentrated on classification, and that is the technique focused on here.

An example of a classification problem would be to build a model that could predict whether or not a patient is diabetic, given their height, weight, and plasma glucose values. To do this, a classification algorithm will first build its model by using training data provided to it. The training data contains a set of records, each holding a collection of independent variables (*attributes*) and a category variable. So, for this example, the data set could consist of the medical records from multiple patients. For each patient, the age, weight, and blood pressure values would be the attributes, and the category variable would be whether that particular patient diabetes. For instance, two of the records might look like this:



	Height	Weight	Plasma Glucose	Diabetic?
Patient 1:	5'4"	120	94	No
Patient 2:	6'3"	200	130	Yes

After building the model, new records are given to it, and the model will try to predict the category this record would belong to (diabetic or not).

Building an effective model that could diagnose diseases like diabetes or cancer in their early stages could help avoid medical complications, or even save patient lives [11].

### 1.2.1 Overview of Classifiers

There are numerous methods used to build classifiers, and selecting the right method for the dataset can make a huge difference. A selected few are briefly described and analyzed below. [49] contains more information on classification techniques and their evaluations.

A naïve Bayes classifier calculates the statistical probability that an individual is in a class by using Bayes Theorem on the dataset. These have been used in many medical applications, from predicting the survival of patients with cirrhosis, to predicting loss of vision in patients with eye diseases [12, 13, 14]. Naïve Bayes classifiers are simple and have linear run-time. However, their greatest weakness is that they assume an individual's attributes are independent given the class.

Bayesian networks are also based on Bayes Theorem, but softens the requirement that attributes all must be independent. The attributes that are not conditionally independent are specified ahead of time, and a network based on the given dependencies is created. Bayesian networks are widely used and have been successful in data mining [40, 42, 43]. One drawback to them, though, is that the problem of calculate the probabilities of the branches in the network is NP hard [44].

Neural Networks (NNs) are loosely based on the architecture of a brain, and try to simulate the way a brain processes information. NNs are very powerful, and have been used in numerous applications [20] (21), (34 reference 4) (34 reference 7). Their application in the medical field is not as widespread, though, due to their "black box" type design. It is not possible to understand the concepts or knowledge stored within a neural net, since it is all in the form of obscure connection weights. In a medical setting, where doctors must make life critical decisions, if a model created is not comprehensible it frequently will not be accepted by the medical community. Clinical experts often first need to evaluate and validate the decision making process in the model before they will trust it enough to use it in practice. (8)

Rule based classifiers are defined as “a technique for classifying records using a collection of “if...then...” rules” [5]. The sets of rules are used to determine which class a given record belongs too. For example:

$r_1$ : (Blood Pressure = high)  $\wedge$  (Age > 50)  $\longrightarrow$  High Risk

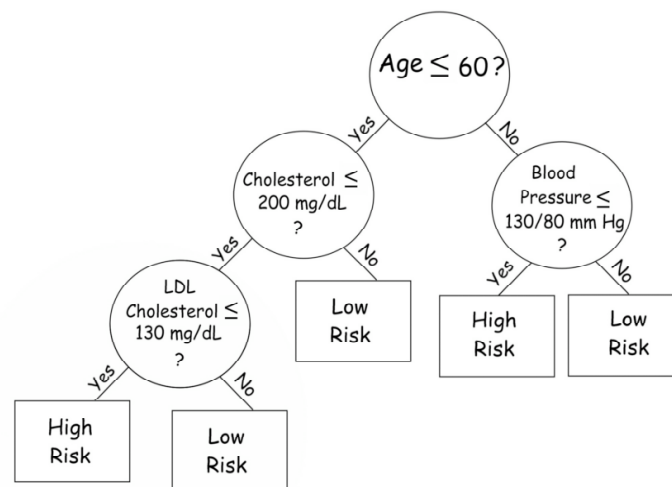
$r_2$ : (Cholesterol = low)  $\wedge$  (Exercise = high)  $\longrightarrow$  Low Risk

$r_3$ : (Weight = high)  $\wedge$  (Exercise = high)  $\longrightarrow$  Medium Risk

Association rules exhaustively look for hidden patterns, and the rules discovered are often very clear and easy to interpret. However, in data such as a medical dataset, a significant fraction of the discovered rules are irrelevant. In addition, many of the most relevant and useful rules only appear at low support values. But at these low support values, the increases in the number of discovered rules becomes too large for an expert to be able to evaluate them (32). Despite this, association rules have still been used successfully in medical data mining applications (32) (8).

Decision trees are often the method of choice when doing classification. Their efficiency and accuracy have at times even surprised experts (33). The main advantage of decision trees, however, is how easy to understand and interpret the decision model they create is.

A decision tree is a tree-like structure consisting of a set of decision nodes and leaf nodes (which indicate classes).



(figure 5. A sample decision tree)

Here, the decision nodes are represented by the circles, and the leafs are represented with squares.

To classify a record, the tree is traversed from the top down. At each node, the record's attributes are compared with the question asked at that node. Depending on the answer, the appropriate link to the next (child) node is followed. This continues until a leaf node is reached, which contains the predicted classification for the record.

Decision trees generated by data mining medical datasets are not only used for the classification they can perform. Often, the trees will be given to experts who look through the branches and decisions made, searching for possible new medical knowledge [24] (8).

Because of their high suitability and application in medicine, in addition to the fact that their structures are easily interpretable and one of the easiest to learn new knowledge from, decision trees were chosen as the focus of this thesis.

### 1.2.2 Types of Decision Trees

Because of their popularity and usefulness, decision trees are extensively researched, and have many options and variations.

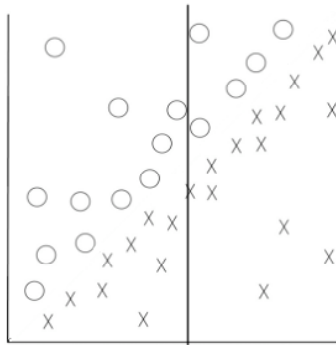
A binary decision tree is one in which there are at most two children for every node. Binary trees are frequently used, since the two branches are natural representations for a true / false answer to the question in the node. In this paper, we will only be considering binary decision trees.

There are also variations on what kind of questions are asked at every node. A univariate decision tree looks at just one attribute at every node, i.e. is  $AGE < 60$  ? Univariate decision trees are also known as axis parallel, because they are limited to using just one attribute to split the decision space with. This restricts them to a split orthogonal to the variable's axis (see figure 6). This may be inappropriate for problems where the input variables are related numerically (4). Unfortunately, medical datasets contain many items that are related numerically. For example, both glucose and HgbA1c levels are important for diabetics, and their values are very related (HgbA1c levels reflect the glucose level for an individual over the preceding 2-3 months)(36).

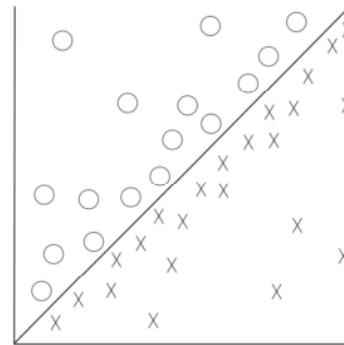
A multivariate decision tree considers a combination of more than one attribute at a single node. These attributes can be combined in an equation if their values are numeric. This allows the tree more flexibility and power in making the splits. It is possible for them to split some items that are related numerically, that a univariate decision tree could not. However, on some data sets, a univariate decision tree's axis parallel splits may be more appropriate. Although univariate is just a special case of multivariate trees, the algorithm chosen may not find a univariate test when it should (45).

For numeric attributes, there are two major types of multivariate decision trees

- Linear decision trees combines the attributes in a linear equation, i.e. is  $2 * \text{DAYS\_IN\_HOSPITAL} + \text{DAYS\_AT\_HOME} > 150$ . These trees are also known as oblique, because they can split along an oblique line from the axis. In some domains, they will result in much smaller and more accurate trees. (see figure 7b)
- Non-linear decision trees use quadratic or other types of non-linear equations. Non-linear decision trees are often harder to comprehend (trying to understand the meanings behind long, complicated equations at every node can be difficult), and they are not nearly as well studied as univariate or linear decision trees (1). However, for some problem spaces, they again have been shown to result in smaller and more accurate trees (1 reference 3).



(fig 6. Axis Parallel Split)



(fig 7. Oblique Split)

## 1.3 Overview of the Thesis Document

Chapter 2 discusses methods for creating and evaluating decision trees, with an emphasis on explaining genetic programming, cost sensitivity, multi-objective optimization, and the reasons behind how and why they are relevant and important to medical data mining. The central algorithms used in this thesis (based on Zhao's 2007 paper, "A multi-objective genetic programming approach to developing Pareto optimal decision trees"(3) ) to create and evolve the decision tree solutions are presented here.

Chapter 3 outlines the three experiments in extensions and changes to the basic algorithm, presented in the previous chapter, done for this thesis. In addition, details about the program implementation, parameters used, and output are shown and discussed.

Chapter 4 presents select charts, graphs, and samples of the data that was collected as the results of the experiments.

Chapter 5 analyses the results presented in chapter 4, and discusses the effectiveness of the algorithm and changes made to it.

Chapter 6 presents possibilities for future research, to both extend, improve upon, and further analyze the technique of building pareto fronts of decision trees using genetic programming.

# Chapter 2

## 2. Creating And Evaluating Decision Trees

### 2.1 Evaluation Basics

Different decision trees will often give different results, and be more or less effective. Knowing how to measure exactly how good (or the *fitness*) of a decision tree is important, since it allows you to determine if the tree created is functioning well or not.

To evaluate a decision tree, data from a dataset is run through it, and the number of correct and incorrect classifications it makes is kept in a confusion matrix.

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	$f_{11}$	$f_{10}$
	Class = 0	$f_{01}$	$f_{00}$

- The value of  $f_{11}$  is the number of True Positive (TP) classifications made, or the number of classification where the decision tree predicted TRUE / 1, and the test class was indeed TRUE / 1
- Similarly, the value of  $f_{00}$  is the number of True Negative (TN) classifications made (predicted FALSE / 0 when the actual class was indeed FALSE / 0).
- $f_{01}$  is the False Positive (FP) rate for the tree, the number of times the tree predicted 1, and the actual class was 0.
- $f_{10}$  is the False Negative (FN) rate for the tree, the number of times the tree predicted 0, and the actual class was 1.

Obviously, smaller FP and FN numbers mean better trees. However, other performance measures are also used to judge the fitness of decision trees. Here are four different measures that are often used to evaluate decision trees :

$$\text{Error Rate} = \text{FP} + \text{FN}$$

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FP} + \text{FN}}$$

- Error Rate counts the total number of errors a decision tree makes.
- Sensitivity is a measure of how well the tree does at catching positive classifications. Sensitivity ranges from 0 (worst) to 1 (best).
- Specificity, inversely, is a measure of how well the tree does at catching negative classifications. Specificity ranges from 0 (worst) to 1 (best).
- Recall is the same as specificity, though it is defined as being the proportion of relevant objects that are retrieved relative to the total number of relevant objects in the dataset.
- Precision measures the accuracy of the positive classifications the tree predicts. Precision ranges from 0 (worst) to 1 (best).
- Accuracy is a measure of how well the tree did overall (how many it got right out of how many there were total)

In addition to how accurately a tree is able to classify the data, a tree can also be measured by how large it is (number of nodes and height). Since larger trees are more confusing and therefore harder to interpret and gain knowledge from (not to mention that they take more processing power and memory), they would be ranked lower than smaller, more compact trees.

Just as there are numerous performance measures used on the trees, there are many different algorithms that can be used to create them. Finding an optimal binary decision tree is known to be NP-complete (1 reference 1), so

choosing the right method can make a big difference. Two methods for creating decision trees are discussed below.

## 2.2 Greedy Heuristics

Greedy heuristics are one of the most commonly used technique to build decision trees. They build the trees from top to bottom, starting at the very top (or root) node. At every node, they utilize a greedy heuristic to decide what attribute and value to pick for the comparison. This means, at every step, they select the attribute and threshold that contributes most to whatever measure they are using. ID3 is a greedy algorithm, used to create decision trees, that utilizes whatever attribute maximizes the splits on information gain (6). C4.5 is a popular algorithm, based on ID3, that uses information gain ratio to decide instead (19). CART-LC and OC1 create linear decision trees using greedy optimizers on the node coefficients (4).

These greedy techniques are fast, but suffer from a variety of problems. They are very sensitive to noise in the data, can not build multiple trees for the same dataset, and often assume that all attributes are conditionally independent. (7)

## 2.3 Genetic Programming

Genetic programming is a technique based on evolutionary computing, which is inspired by Darwin's theory of evolution. It is not limited to only building decision trees, but can and has been used to solve numerous types of optimization problems. [25]

Genetic Programming relies on the hope that, through some sort of natural selection and "genetic recombination", the potential for offspring to occasionally be 'better' than their parents, and pass these good traits on, exists.(50) A population of individuals are randomly created and maintained. These individuals are candidates to become the solution for whatever problem you wish to optimize for. So in this case, we would be creating and maintaining a population of decision trees.

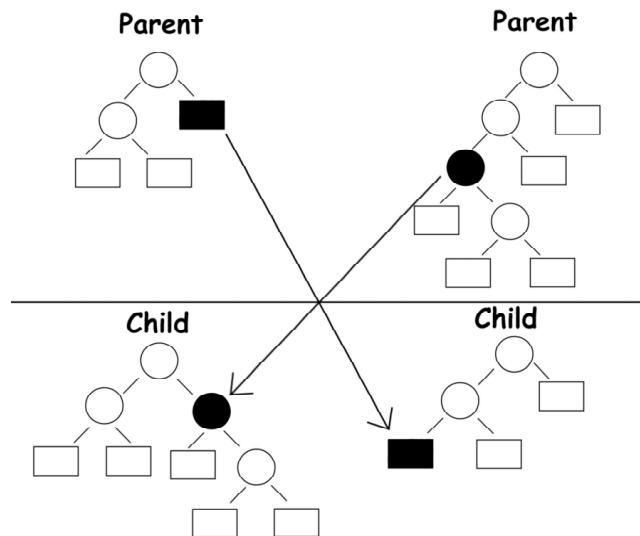
In the beginning, most of these individuals will probably be very poor solutions indeed. However, some will be better than others. A fitness function is used to evaluate numerically exactly how fit every candidate solution in the population is. Often a value such as the accuracy of the tree will be used as the fitness value. The individuals that are more fit have a better chance of being selected to generate "offspring"...new solutions for inclusion in the next generation, either by "genetic crossover" or "mutation". This set of new children, once created, are then used to replace the old generation and become the candidate solutions. (However, there is a random chance that fitter individuals



from the old generation will be able to survive and continue into the next generation). And the process starts over again until you arrive at an optimal solution, or close enough to it (i.e. a good enough decision tree).

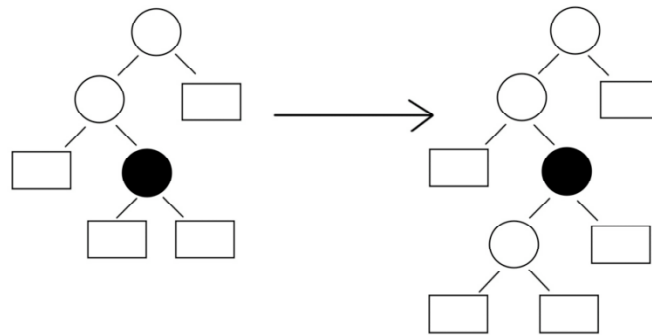
For decision trees, the process of crossover involves first choosing the two parent trees. This needs to be rather random (choosing the same two parents over and over again would cause the population to converge too quickly, limiting the search space, and produce “inbreeding” type effects), yet weighted towards those with better fitness. Tournament selection is an option that to fulfill this requirement, and is often used to choose each parent.[20] Tournament selection involves first randomly selecting a set number of individuals from the population to be in the tournament. These individuals then have their fitness values compared, and the one with the highest fitness value wins and becomes one of the parents for the child tree. If there is a tie, then that tie is broken by another tie breaker, such as the height of the tree or the number of nodes. A smaller tree that is able to classify just as well as a larger one will probably represent knowledge that is both more useful and easier to understand and interpret.

After the parents are chosen, crossover will be performed on them. In decision trees, this entails randomly choosing a node from either tree and swapping them. The resulting trees are the children, to be included in the next generation.

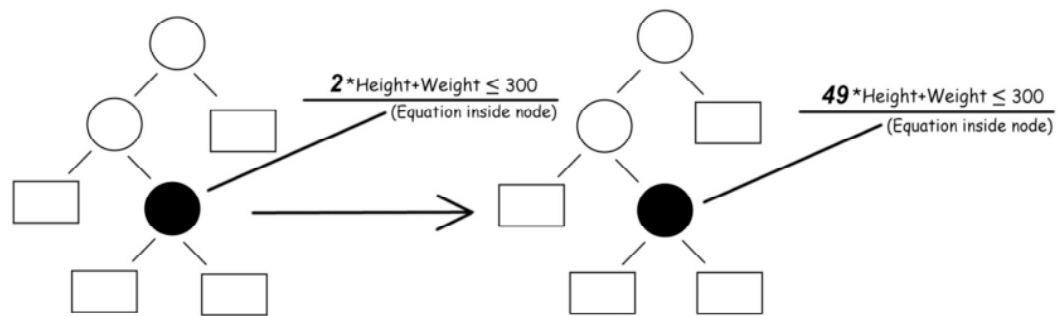


(figure 8. Crossover in Decision Trees)

In literature, the mutation operator in decision trees seems to usually be done by one of two different ways. A random node is chosen, and then either a completely new, random subtree replaces it, in what I refer to here in this thesis as *hard mutation* (see fig. 9) (3) (25) or a constant or attribute from the function inside the node is chosen and replaced with a new, random value, in what I refer to as *soft mutation* (see fig. 10) (7) (13) (30) (49).



(Fig. 9 Hard Mutation)



(Fig. 10 Soft Mutation)

Genetic programming to create decision trees has been used with success on real world medical datasets. In 2005, decision trees generated from a cardiovascular dataset using genetic programming pointed to a new medical rule that had been discovered just recently...by a comprehensive and expensive European Union funded research project (8). Moreover, an analysis of six different approaches of building decision trees on a hard real world medical problem (orthopedic fracture data) was conducted by researchers, and the genetic programming approach outperformed all the others (200).

## 2.4 Cost Sensitivity

Assessing the performance of trees by the accuracy values, FN, FP, etc. works well in theory. However, in the real world many classification problems are cost sensitive. This means that, for example, a false positive does not always incur the same cost as a false negative. In medicine, this is especially the case. A false positive could mean extra tests and mental stress for the patient. A false negative may mean a patient's disease going undiagnosed, and could result in death. In this case, the performance rating of the classifier should be decreased by much more when it predicts a false negative than when it predicts a false

positive, and the classifier should be trying to minimize the *expected misclassification cost*, rather than plain error rate (3).

One way of dealing with cost sensitive problems such as these, and to obtain this expected misclassification cost, is to use weights to compensate for the imbalance (i.e., the appropriate training samples are multiplied by a weight value to make errors on them more costly) (20). So, as an example, suppose a tree has a confusion matrix as shown below:

Tree A

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	5	2
	Class = 0	10	5

Originally, we would predict that this tree has an error rate of 12. However, if it turned out that false negatives were 3 times worse than false positives (which also can be expressed as: the weight of FP : FN = 1 : 3), the expected misclassification cost becomes  $1 \cdot 10 + 3 \cdot 2$ , or 16.

It turns out that, depending on what weights are chosen, different trees are learned. A tree that specializes in minimizing false negatives, for example, will not necessarily be the same tree that is learned when there is no cost function. Or when false positives need to be minimized. Or even when a different weight for false negatives is given. This means that there can be multiple trees, each of which is best when it comes to a different cost or cost range. As an example, let's say we have two trees, with their respective confusion matrices shown below.

Tree A

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	5	2
	Class = 0	10	5

Tree B

		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	6	9
	Class = 0	2	5

If there was no cost sensitivity (which is equivalent to (FP : FN = 1 : 1)) TreeA's expected misclassification rate would be the same as its error rate, which again is 12. TreeB's error rate is  $1 \cdot 2 + 1 \cdot 9 = 11$ . TreeB would be considered the better tree. However, if it turned out that the cost weights were again FP : FN = 1 : 3, then TreeA's error rate is again  $1 \cdot 10 + 3 \cdot 2 = 16$ , and

TreeB's error rate is now  $1 \cdot 2 + 3 \cdot 9 = 29$ . TreeA has now become more fit than TreeB...at this cost setting.

When the cost settings are known and fixed, it's always easy and clear to apply the weights and figure out exactly which trees are best and worst. However, it's not uncommon for decision makers to be unclear of what the exact costs of misclassification are, or to only have an idea of the range (i.e. false negatives are somewhere between two to four times worse than false positives. Or, the weights range from FP : FN = 1 : 2 to 1 : 4). Real world medical situations parallel this, with a precise cost of misdiagnosis or patient stress being difficult to pin down...as well as there being times where early diagnosis is more important for some categories of people than others[26]. This can lead to situations like the accuracy example above...where for a weighting of FP : FN = 1 : 1, TreeA is more fit. But for the weighting of 1 : 3, TreeB is more fit. When we create the trees, which tree should be decided upon to keep and use as the classifier?

Since the costs in this example are said to vary, keeping both trees would be the best solution. Then, when the weights are at or around 1 : 1, TreeA could be used. And when the weights are at or around 1 : 3, TreeB could be used instead. To take this idea a step further, an entire set of trees should be created, each specializing in a certain area of the desired range of weights. This will ensure a much smaller misclassification cost since, whenever a classifier is requested for a particular weighting, an appropriate tree specializing in (or very close to) that weighting will be used.

It does not have to be the false positive and false negative values that are being compared and weighted. In fact, the sensitivity (how well the tree catches false positives) to specificity (how well the tree catches false negatives) is often used in medicine. In this case, the same technique still applies. The sensitivities and specificities will be multiplied by the weights and added instead of the FPs and FNs. (Of course, in this way a larger value would actually mean a better fitness, considering the nature of the measures). From this point forward, it is assumed that sensitivity and specificity are the measures being used.

## 2.5 MultiObjective Optimization and Pareto Fronts

On closer inspection, it can be seen that this has actually become a case of *multi-objective optimization*. Both the sensitivity and specificity values are to be maximized. A problem is defined as multi-objective whenever it wishes to maximize two different, and often contrasting, objectives.

Multi-objective optimization algorithms attempt to find a set of the best solutions that maximize these multiple objectives. A "best solution" is an individual who is not *dominated* by another. The definition of dominance between two individuals, x and y, is given by:

Solution  $x$  *dominates* solution  $y$  if

$$f_i(x) \geq f_i(y) \quad \forall i = 1, 2, \dots, n, \text{ and } \exists j \in \{1, 2, \dots, n\} : f_j(x) > f_j(y)$$

Assuming the objectives we wish to maximize are  $f_1 \dots f_n$

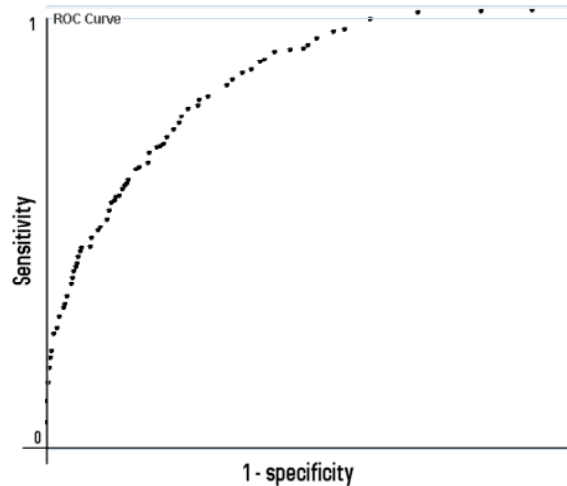
In other words,  $x$  dominates  $y$  if all of its objectives' values are at least the same as  $y$ 's...and it has at least one objective value that is greater than  $y$ 's.

There will often be multiple individuals who are not dominated and added to the set. A set of solutions like this, where no one solution is better than any other, is known as a non-dominated set (no tree dominates over another tree in the set). And if this non-dominated set dominates over every other set of solutions, it is known as the Pareto front. Multi-Objective Optimization algorithms attempt to find this Pareto front.

Pareto fronts can be visualized by Receiver Operating Characteristic (ROC) curves. These graphs are already widely used in the medical field to visualize the tradeoff between the sensitivity and specificity, depending on how the test is interpreted (An ROC curve is defined as “a graphical plot of the sensitivity vs. (1 - specificity) for a binary classifier system as its discrimination threshold is varied” [27] So an ROC curve shows the tradeoff between sensitivity and specificity. (Both Sensitivity and Specificity range from 0 to 1, with 1 being the highest ranking). The closer the curve gets to the upper and left borders of the graph, the better the model / set of trees are (because the x axis point gets smaller the better specificity gets, since it's 1 – specificity). A perfect model, that never misses a classification, will have just one point at (0,1).

To judge the accuracy of the model/s that are plotted, the area under the curve is taken. (An area of 1 corresponds to perfect (no false positives or negatives). And an area of .5 represents a worthless model.[28] ROC curves are accepted and widely applied in many domains (like medical diagnosis) (3)

To create an ROC curve using a Pareto front, the sensitivity and 1-specificity values of every tree in the front are obtained, and then graphed.



(fig11. Example of an ROC curve created from a pareto front)

## 2.6 Multi-Objective Genetic Programming

Finding the Pareto fronts, consisting of multiple solutions at varying intervals of cost, can be a time and computation heavy operation. Genetic programming techniques, however, with their evolving populations of many unique individuals at once, are very well suited for the task of finding multi-objective solutions. Many individuals can be evolving towards the different objectives all at the same time.

One problem with not having fixed cost weights in genetic programming, however, is that a numeric fitness value, like accuracy, cannot be determined. The weights of misclassification are not known. As shown earlier, the ranking values normally used to calculate a decision tree's fitness can change depending on what weights are chosen.

However, even though we cannot directly calculate a fitness number from one tree and use that to compare with all the other trees, we can still figure out if one tree dominates another. If one tree dominates another, then it is more fit. One tree dominates another if, for every objective that is to be maximized, its values are equal to or better than those of the other tree. (This definition of dominance will give us the entire pareto front. [45] contains a more generalized formula, that allows constrained portions of the front to be chosen.)

By using dominance, we can achieve a *relative* ranking across the population. Every tree in the population starts out with a rank of "1". Then it is compared with every other tree. If one tree dominates another, the dominated tree has its rank incremented. This means that trees that are not dominated throughout the population will still have a rank value of "1" by the end of the ranking. Those trees are in our current Pareto front

(they do not dominate each other, and they dominate all other trees in the population). Those trees are then considered the most fit, and are most likely to be selected as parents during the run of the genetic program.

# Chapter 3.

## 3. Implementation

### 3.1 Experiments

In this thesis, a multi-objective genetic programming algorithm for developing Pareto optimal decision trees was implemented (as discussed in (3)), extended, and experimented with.

The extensions and experiments were divided into three parts.

- 1) Addition of an elitism, which stores the best, non-dominated individuals found across all the generations, and randomly reintroduces them into the population at times. In genetic programming, and especially when it is multi-objective, the addition of an elitism can help make progress continue in the population. [29]
- 2) Experimentation with the two currently used methods of mutation on decision trees, to find the best mutation type and rate, and collect more information on the effect of mutation rates on populations such as these.
- 3) Modifying the tree structures and algorithm to allow for linear decision trees to be created and evolved. In the original paper, only univariate decision trees could be created. Adding linear capabilities to the trees allows them to cut through the space at oblique angles, rather than only axis-parallel.

#### 3.1.1 Addition of Elitism

Elitist strategy is a technique to make sure that the chromosomes of the fittest members are more likely to be passed on to the next generation, and past optimal solutions are not lost. Experimental results show that using the elitist strategy genetic algorithms converge faster toward the optimal strategy.[31].

In addition, though genetic programming has the advantage that it is a global search, and will not get stuck as easily in local optima the way greedy heuristics will, it also has the problem of being computation intensive. And when it is used in data mining on massive datasets, any improvement to allow for the option of greater speed is useful. The algorithm used here must compare every tree in the new generation to every other tree, giving it a complexity of  $N^2$  to the number of trees. Therefore, if there were any way we could decrease the



number of trees needed, yet still retain reasonable results, this would be of great benefit to the use and practicality of the algorithm.

In the program, the option to create and use an elitist set was made available. In the beginning, the current elitist set would be equivalent to the current global pareto front. For every generation after, the children would be compared with the individuals in the elitist set. If a child was dominated by one of the members of the set, it would be dropped out of competition to join the elite set. If the child dominated a member of the set, it would be added to the set, and the dominated member discarded. Otherwise, if they were compared to every member of the set and neither dominated nor was dominated, then they were part of the current pareto front and added to the set.

Because of the nature of the fitness ranking, the elitist set would grow steadily as the generations went on. Tight clusters of just slightly different trees would be added, and stay for the duration of the run, causing the set to grow unchecked and use excess amounts of memory and CPU time, without adding much new useful information into the population pool. There were several possible solutions to this problem, for example enforcing a minimum separation between the trees of the set, or including a maximum lifespan on the members.

An option to add a maximum lifespan to members of the elite set was implemented. This allowed the members to still have the chance to pass on their chromosomes to the future generations, and yet would not preclude another tree, who might possibly have similar confusion matrix stats but a different tree structure, from being added to the elite set as well.

### **3.1.2 Experimenting With the Mutations**

Two separate techniques for mutation on decision trees seem to be in use currently. Hard mutation replaces the entire substructure of a tree, and soft is used to change the coefficients of the equations. However, no studies seem to have been done on which mutation type gives better results, or if a combination between them might be optimal (some percentage of the time doing a hard mutation, the other a soft mutation).

Both mutation techniques were implemented, and were independently tested to compare their relative performance at various percentage levels of mutation. Then a 50% combination between them (where, during every mutation done, there is a 50% chance of doing a hard vs. a soft mutation) was evaluated.

### 3.1.3 Adding Linear Capabilities

The decision tree structures and strong typing (ensuring that only valid trees are able to be produced) were implemented similarly to the representation Bot and Langdon used in their paper on inducing linear classification trees (5). Each node consists of an equation composed of constants multiplied by an attribute and added together, from 1 to some user specified constant,  $n$ .

$$\text{Constant}_1 * \text{Attribute}_1 + \dots + \text{Constant}_n * \text{Attribute}_n \leq \text{Threshold}$$

The constants are chosen randomly from between -1 and 1. The attributes are also linearly normalized between -1 and 1. This means that threshold varies anywhere from  $-n$  to  $n$ , and a random number in that range is chosen for it.

## 3.2 Overview of Program and Parameters

### 3.2.1 Datasets

The dataset used for all experiment here except \_\_\_\_\_ was the Pima Indians dataset from the UCI machine learning repository[30]. Donated by Vincent Sigillito, the data was collected by the National Institute of Diabetes and Digestive and Kidney Diseases. It contains 8 numeric attributes and 768 instances (individuals). All attributes were linearly normalized between 0 and 1, and 5 fold cross-verification was applied to all runs with the data. For more details on this dataset, please see Appendix i.

The other dataset used was the Cleveland heart disease dataset, also from the UCI machine learning repository [30]. It was collected from the V.A. Medical Center, Long Beach and Cleveland Clinic Foundation by Robert Detrano, M.D., Ph.D. and donated by David W. Aha. It contains 13 numeric attributes and 303 instances. For more details on this dataset, please see Appendix ii.

### 3.2.2 Some Program Implementation Details

Default parameter values for the runs:

Generations	1000
populationSize	3000
maxTreeHeight	8
maxInitialTreeHeight	2
maxMutationHeight	2
elitistRate	.1
mutationRate	.1
copyRate	.1

crossoverRate	.7
tournamentSize	7

Though the values for most parameters were identical to Zhao's paper (3), the maximum tree height had to be decreased due to a lack of computer memory on the computer used to run the program.

To increase simplicity and comprehensibility, I controlled the tree sizes by pre-pruning (not allowing them to grow past a user specified point). Also, the trapezoidal method was used to estimate the area under the ROC curve.