An Innovative Application of a Constrained-Syntax Genetic Programming System to the Problem of Predicting Survival of Patients

Celia C. Bojarczuk[1], Heitor S. Lopes[2], and Alex A. Freitas[3]

[1] Departamento de Eletrotecnica, CEFET-PR
Av. 7 de setembro, 3165, Curitiba, 80230-901, Brazil
celia@cpgei.cefetpr.br
[2] CPGEI, CEFET-PR
Av. 7 de setembro, 3165, Curitiba, 80230-901, Brazil
hslopes@cpgei.cefetpr.br
[3] Computing Laboratory, University of Kent
Canterbury, CT2 7NF, UK
A.A.Freitas@ukc.ac.uk
www.cs.ukc.ac.uk/people/staff/aaf

Abstract. This paper proposes a constrained-syntax genetic programming (GP) algorithm for discovering classification rules in medical data sets. The proposed GP contains several syntactic constraints to be enforced by the system using a disjunctive normal form representation, so that individuals represent valid rule sets that are easy to interpret. The GP is compared with C4.5 in a real-world medical data set. This data set represents a difficult classification problem, and a new preprocessing method was devised for mining the data.

1 Introduction

Classification is an important problem extensively studied in several research areas, such as statistical pattern recognition, machine learning and data mining [Hand 1997]. The basic idea is to predict the class of an instance (a record of a given data set), based on the values of predictor attributes of that instance.

This paper proposes a genetic programming (GP) system for discovering simple classification rules in the following format: IF (a-certain-combination-of-attribute-values-is-satisfied) THEN (predict-a-certain-class). Each individual represents a set of these IF-THEN rules. This rule format has the advantage of being intuitively comprehensible for the user. Hence, he/she can combine the knowledge contained in the discovered rules with his/her own knowledge, in order to make intelligent decisions about the target classification problem – for instance, medical diagnosis.

The use of GP for discovering comprehensible IF-THEN classification rules is relatively little explored in the literature, by comparison with more traditional rule induction and decision-tree-induction methods [Witten and Frank 2000]. We believe such a use of GP is a promising research area, since GP has the advantage of performing a global search in the space of candidate rules. In the context of classification rule discovery, in general this makes it cope better with attribute interaction than conventional, greedy rule induction and decision-tree-building algorithms [Freitas 2002], [Dhar et al. 2000], [Papagelis and Kalles 2001].

The GP algorithm proposed in this paper is a constrained-syntax one. The idea of constrained-syntax GP is not new [Montana 1995]. However, we believe this paper has the contribution of proposing a constrained-syntax GP tailored for the discovery of simple classification rules. That is, it enforces several syntactic constraints, so that individuals represent rule sets that are valid and easy to interpret, due to the use of a disjunctive normal form representation.

The remainder of this paper is organized as follows. Section 2 describes the proposed constrained-syntax GP for discovering classification rules. Section 3 reports the results of computational experiments comparing the GP with C4.5. Finally, section 4 presents the conclusions and future research.

2 A Constrained-Syntax GP for Discovering Classification Rules

An individual can contain multiple classification rules, subject to the restriction that all its rules have the same consequent – i.e., they predict the same class. In other words, an individual consists of a set of rule antecedents and a single rule consequent. The rule antecedents are connected by a logical OR operator, and each rule antecedent consists of a set of conditions connected by a logical AND operator. Therefore, an individual is in disjunctive normal form (DNF) – i.e., an individual consists of a logical disjunction of rule antecedents, where each rule antecedent is a logical conjunction of conditions (attribute-value pairs). The rule consequent specifies the class to be predicted for an instance that satisfies all the conditions of any of the rule antecedents.

The terminal set consists of the attribute names and attribute values of the data set being mined. The function set consists of logical operators (AND, OR) and relational operators ("=", "≠", "≤", ">").

Figure 1 shows an example of the genetic material of an individual. Note that the rule consequent is not encoded into the genetic material of the individual. Rather, it is chosen by a deterministic procedure, as will be explained later. In the example of Figure 1 the individual contains two rules, since there is an OR node at the root of the tree. Indeed, the tree shown in that figure corresponds to the following two rule antecedents: IF ($A_1 \leq 2$) OR IF (($A_3 \neq 1$) AND ($A_5 > 1$)).

Once the genetic material (set of rule antecedents) of an individual is determined, the rule consequent (predicted class) associated with the individual is chosen in such a way that the fitness of the individual is maximized. More precisely, for each class, the system computes what would be the fitness of the individual if that class were chosen to be the class predicted by the individual. Then, the system chooses the class that leads to the best fitness value for the individual.

As mentioned above, all the rules of an individual have the same rule consequent – i.e., they predict the same class. This leaves us with the problem of how to discover

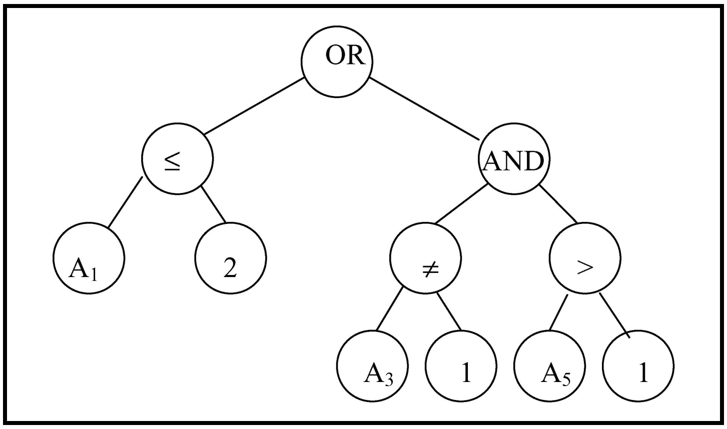

Fig. 1. Example of an individual

rules predicting different classes. The most common solution for this problem in the literature is to run the GP *k* times, where *k* is the number of classes [Kishore et al. 2000]. In the *i-th* (*i=1,...,k*) run, the GP discovers rules predicting the *i-th* class. Instead of using this conventional approach, our system works with a population of individuals where different individuals may have different rule consequents. Hence, in our approach an entire solution for the classification problem consists of *k* individuals, each of them predicting a different class. In other words, at the end of the evolution, the solution returned by GP will consist of *k* individuals, each of them being the best individual (the one with the best fitness value) for a different class.

To summarize, in our individual representation each individual consists of a set of rules predicting a given class, and an entire solution for the classification problem consists of *k* individuals, each of them predicting a different class.

One advantage of this approach, by comparison with the previously mentioned conventional approach of running the GP once for each class, is that in the former we need to run the GP just once to discover rules predicting different classes. Therefore, our approach is considerably more efficient, in terms of computational time.

2.1 Syntactic Constraints on the Individual Representation

Conventional GP systems must satisfy the property of closure, which means that the output of any function of the function set can be used as the input for any other function of that set. This property is satisfied, for instance, if the function set contains only mathematical operators (like +, -, /, *) and all terminal symbols are real-valued variables or constants. However, in a typical data mining scenario the situation is more complex, since we often want to mine a data set with a mixing of categorical (nominal) and continuous (real-valued) attributes. Hence, our individual representation includes several constraints useful for data mining applications, as follows.

First, we specify, for each function of the function set, what are the data types valid for the input arguments and the output of the function. The function set of our

GP consists of logical operators (AND, OR) and relational operators (“=”, “≠”, “≤”, “>”). The valid data types for the input arguments and output of these operators are shown in Table 1. Note that all operators of Table 1 take two input arguments, so that each GP individual is represented by a binary tree. Our GP can cope with attributes that are either categorical (nominal) or continuous (real-valued), which is a desirable flexibility in a data mining system. The data type restrictions specified in Table 1 naturally suggest an individual representation based on a hierarchy of operators, consisting of boolean operators (AND, OR) at the top of the tree, attributes and their values at the leaves, and relational operators (“=”, “≠”,“≤”, “>”) in the middle of the tree. An example of this hierarchical structure was previously shown in Figure 1. Note that the individual shown in that Figure satisfies all data type constraints specified in Table 1.

Table 1. Valid data types for each operator’s input arguments and output

Operator	**Input arguments**	**Output**
AND, OR	(boolean, boolean)	boolean
“=“, “≠“	(categorical, categorical)	boolean
“≤“, “>“	(real, real)	boolean

In addition to the data type constraints of Table 1, our GP system enforces two other constraints. First, an AND node cannot be an ancestor of an OR node. Although this is not essential for producing syntactically-valid individuals, it enforces the restriction that every individual represents a set of rule antecedents in (DNF). The DNF representation is not only intuitively simple, but also facilitates the enforcement of the second additional constraint, called “attribute-uniqueness constraint”. This constraint means that an attribute can occur at most once in a rule antecedent. This constraint avoids invalid rule antecedents like: IF (*Sex* = *male*) AND (*Sex* = *female*).

2.2 Genetic Operators

Our GP uses reproduction and crossover operators. The reproduction operator consists of passing a copy of an individual to the next generation. The crossover operator used here is a variant of the standard tree-crossover operator. In our system that crossover operator is adapted to our constrained-syntax individual representation, as follows.

First, a crossover point (a tree node) is randomly selected in one of the parent individuals, here called the first parent. Then the crossover point (tree node) of the other parent individual, here called the second parent, is randomly selected among the nodes that are compatible with the crossover point of the first parent, i.e., among the nodes that return the same data type as the data type returned by the crossover point of the first parent. Then the crossover is performed by swapping the subtrees rooted at the crossover points of the two parent individuals, as usual.

Our GP also uses a form of elitism that we call classwise elitism. The basic idea of elitism is that the best (or a small set of best) individual(s) of a generation is passed unchanged to the next generation, to prevent the stochastic process of evolution from losing that individual. Recall that the population contains individuals predicting different classes. In our classwise elitism the best individual of each of the *k* classes is chosen to be passed unchanged to the next generation. In other words, *k* elite individuals are passed unaltered to the next generation. The *i-th* elite individual ($i = 1,...,k$) is the best individual among all individuals predicting the *i-th* class. The motivation for this classwise elitism is to avoid that the population converges to a state where all individuals represent rule sets predicting the same class. Without classwise elitism this would tend to happen, because in general some classes are easier to predict than others, i.e., individuals predicting the easiest class would dominate the population.

2.3 Fitness Function

The fitness function used in this work is the same as the fitness function proposed in [Bojarczuk et al. 2000]. Note, however, that [Bojarczuk et al. 2000] used a simple individual representation, working only with boolean attribute values. This required all attributes to be booleanized in a preprocessing step, which significantly reduces the flexibility and autonomy of the algorithm. By contrast, this work uses a considerably more flexible and elaborate individual representation, as discussed earlier.

The fitness function evaluates the quality of each individual (a rule set where all rules predict the same class) according to two basic criteria, namely its predictive accuracy and its simplicity. Predictive accuracy is measured by the product $Se \cdot Sp$, where Se (the sensitivity) is given by $Se = tp / (tp + fn)$ and Sp (the specificity) is given by $Sp = tn / (tn + fp)$, where tp, fp, tn and fn denote respectively the number of true positives, false positives, true negatives and false negatives observed when a rule is used to classify a set of instances [Hand 1997].

The second criterion used in the fitness function is the simplicity (Sy) of the rule set represented by an individual, given by: $Sy = (maxnodes - 0.5 \cdot numnodes - 0.5) / (maxnodes - 1)$ where *numnodes* is the current number of nodes (functions and terminals) of an individual (tree), and *maxnodes* is the maximum allowed size of a tree (empirically set to 45). The inclusion of a simplicity term in the fitness function helps to produce simpler (shorter) rule sets to be shown to the user, and it also helps to avoid code bloat. Finally, the entire fitness function is given by the product of the indicators of predictive accuracy and simplicity, i.e.: $fitness = Se \cdot Sp \cdot Sy$. The motivation for this fitness function is explained in [Bojarczuk et al. 2000].

2.4 Classification of New Instances

Recall that, after the GP run is over, the result returned by GP consists of a set of *k* individuals, where *k* is the number of classes. The *i-th* returned individual (i=1,...,k) consists of a set of rules predicting the *i-th* class for a data instance (record) that satisfies the rule set associated with the individual. An instance is said to satisfy a rule set if it satisfies all the conditions of at least one of the rules contained in the rule set. Recall that an individual contains a rule set in disjunctive normal form.

When the set of returned individuals is used to classify a new instance (in the test set), the instance will be matched with all the *k* individuals, and one of the following three situations will occur:

(a) The instance satisfies the rule set of exactly one of the *k* individuals. In this case the instance is simply assigned the class predicted by that individual;

(b) The instance satisfies the rule set of two or more of the *k* individuals. In this case the instance is assigned the class predicted by the individual with the best fitness value (computed in the training set, of course);

(c) The instance does not satisfy the rule set of any of the *k* individuals. In this case the instance is assigned a default class, which is the majority class, that is the class of the majority of the instances in the training set.

3 Computational Results

In this section we compare the results of our GP with C4.5, a very well-known decision tree algorithm [Quinlan 1993], in a new data set, called Pediatric Adrenocortical Tumor, which has not been previously used in any computational classification experiment reported in the literature. We emphasize that preparing this data set for data mining purposes was a considerable challenge. We had to carry out a significant preprocessing of the available data, as described in the following. The data set used in our experiments consisted of 124 instances (records) and 10 attributes.

The first step was to decide which attribute would be used as the goal (or class) attribute, to be predicted. Discussing with the user, it was decided to predict how long a patient will survive after undergoing a surgery. The corresponding goal attribute is hereafter called *Survival*. The values of this attribute for the instances were not directly available in the original data set. It had to be computed in an elaborate way, as follows.

First the system computed, for each instance (patient), the number of days between the date of the surgery and the date of the last follow up of the patient. Then the system checked, for each instance, the value of another attribute called *Status*, whose domain contained four values. One of these values indicated that the patient was *dead*, whereas the other three values indicated that the patient was still *alive*. (The difference in the meaning of those three values indicating *alive* patient reflect different stages in the progress of the disease, but this difference is not relevant for our discussion here.)

A major problem in predicting *Survival* is that, if the *Status* of a patient (as recorded in the hospital's database) is different from *dead*, this does not necessarily means that patient is still *alive* in real life. Maybe the patient actually died, but this information was not yet included in the database, due to a loss of contact between the family of the patient and the hospital. On the other hand, if the value of *Status* recorded in the hospital's database is *dead*, this *Status* is presumably true. As a result, for many of the patients, one cannot be sure about the true value of the *Survival* attribute. One can be sure about this value only when the value of the *Status* attribute is *dead*. When *Status* is different from *dead*, the value of *Survival* computed as described above is just an underestimate of the true value of that attribute. Hence, any attempt to directly predict the value of *Survival* would be highly questionable.

To circumvent this problem, we transformed the original problem of predicting *Survival* for all patients into three separate problems, each of them carefully defined to lead, at least in principle, to more reliable results. We try to predict the value of *Survival* for each of three classes of this attribute separately. These three classes were defined by discretizing the *Survival* attribute (which was previously measured in number of days) into three ranges of values, namely less than one year, between one and two years, between two and five years. These intervals were determined by the user, a medical expert on Pediatric Adrenocortical Tumor. Hereafter these ranges are called class 1, class 2 and class 3, respectively, for short.

This leads to three classification experiments, each of them aiming at discriminating between two classes, a "positive" class and a "negative" class. In the *i-th* experiment, $i = 1,2,3$, the instances having class i are considered as positive-class instances, and all the other instances are considered as negative-class instances.

The reason why we need to perform three separate classification experiments is as follows. As mentioned above, when the patient's *Status* is different from *dead*, one cannot be sure about the true value of the *Survival* attribute. For instance, suppose that a patient underwent surgery one and a half year ago. One cannot be sure if the patient has class 2 or 3, since (s)he might or not live until (s)he completes two years of survival after surgery. However, one can be sure that this patient does not have class 1. So, its corresponding instance can be used as a negative-class instance in the first classification experiment, aiming at predicting whether or not a patient has class 1. On the other hand, that instance cannot be used in the second or third classification experiments, because in those experiments there would be no means to know if the instance had a positive class or a negative class.

The key idea is that an instance is used in a classification experiment only when one can be sure that it is either definitely a positive-class instance or definitely a negative-class instance, and for some instances (those having *Status* different from *dead*) this depends on the classification experiment being performed. Finally, we now precisely specify how we have defined which instances were used as positive-class or negative-class instances in each of the three classification experiments.

The first experiment consists of predicting class 1, i.e. *Survival* less than one year. In this experiment the positive-class instances are the patients whose *Status* is *dead* and whose *Survival* is less than or equal to one year. The negative-class instances are the patients whose *Survival* is greater than one year. After this instance-filtering process the data set contained 22 positive-class instances and 83 negative-class instances.

The second experiment consists of predicting class 2, i.e. *Survival* between one and two years. In this experiment the positive-class instances are the patients whose *Status* is *dead* and *Survival* is greater than one year and less than or equal to two years. The negative-class instances are the patients whose *Status* is *dead* and *Survival* is either less than one year or greater than two years. After this instance-filtering process the data set contained 8 positive-class instances and 86 negative-class instances.

The third experiment consists of predicting class 3, i.e. *Survival* between two years and five years. In this experiment the positive-class instances are the patients whose *Status* is *dead* and *Survival* is greater than two years and less than or equal to five years. The negative-class instances are the patients whose *Status* is *dead* and *Survival* is either less than two years or greater than five years. After this instance-filtering process the data set contained 6 positive-class instances and 62 negative-class instances.

Table 2 reports the accuracy rate obtained by C4.5 and the GP in each of the three classification experiments. The numbers after the "±" symbol denote standard deviations. In all the experiments we have used the default parameters of C4.5 and the GP, making no attempt to optimize the parameters of the two systems. The default parameters of the GP are: population size of 500 individuals, 50 generations, crossover probability of 95%, reproduction probability of 5%, initial population generated by the ramped half and half method, maximum tree size of 45 nodes. We used roulette wheel selection. All results were obtained by performing a 5-fold cross-validation procedure [Hand 1997], where each of the 5 iterations of the cross-validation procedure involved a single run of both the GP and C4.5.

Based on the results reported in Table 2, at first glance C4.5 seems to outperform our GP system in this data set. In two out of the three classes (namely, classes 2 and 3) the accuracy rate of C4.5 is significantly better than the one of the GP – since the corresponding accuracy rate intervals (taking into account the standard deviations) do not overlap. However, this conclusion would be premature, as we now show.

Table 2. Classification accuracy rate (%) on the test set

Class	C4.5	GP
1	75.7 ± 1.22	73.3 ± 2.43
2	88.2 ± 0.77	78.8 ± 2.81
3	87.3 ± 1.01	67.8 ± 6.82

Table 3. Sensitivity (*Se*) and Specificity (*Sp*) on the test set

Class	C4.5			GP		
	Se	*Sp*	*Se · Sp*	*Se*	*Sp*	*Se · Sp*
1	0.1	0.916	0.079	0.79	0.725	0.560
2	0	1	0	0.9	0.781	0.693
3	0	1	0	0.1	0.735	0.067

The problem with the results of Table 2 is that they are based on classification accuracy rate. Although this measure of predictive accuracy is still the most used in the literature, it has some drawbacks [Hand 1997]. The most important one is that it is relatively easy to achieve a high value of classification accuracy when one class (the majority class) has a high relative frequency in the data set, which is the case in our data set. In one extreme, suppose that 99% of the examples have a given class c_1. In this case one can trivially achieve a classification accuracy rate of 99% by "predicting" class c_1 for all examples. Does that mean that the classification algorithm (a trivial majority classifier) is doing a good job? Of course not. What this means is that the measure of classification accuracy rate is too weak in this case, in the sense that it is too easy to get a very high value of this measure. One needs a more demanding measure of predictive accuracy, which emphasizes the importance of correctly classifying examples of all classes, regardless of the relative frequency of each class.

Indeed, an analysis of the trees induced by C4.5 shows that the results of the last two rows of Table 2 (referring to classes 2 and 3) are very misleading. In particular, C4.5 is *not* discovering better rules for these classes. When predicting class 2 and class 3, C4.5 induces a degenerate, "empty" tree with no internal node; i.e., a tree containing only one leaf node, predicting the majority class. This has consistently occurred in all the five folds of the cross-validation procedure. Clearly, C4.5 opted for an "easy solution" for the classification problem, favoring the correct prediction of the majority of the examples at the expense of making an incorrect prediction of all the minority-class examples. Such an easy solution is useless for the user, since it provides no rules (i.e., no knowledge) for the user. Only when predicting class 1 C4.5 was able to induce a non-degenerate, non-empty tree on average. And even for this class an empty tree was induced in some folds of the cross-validation procedure.

By contrast, our GP system discovered, on average, rules with 2, 1.7 and 1.9 conditions, for rules predicting classes 1, 2 and 3, respectively, which constitute a simple rule set to be shown to the user. Overall, the rules were considered comprehensible by the user. We now need to evaluate these rules according to a more demanding measure of predictive accuracy, emphasizing the importance of correctly classifying examples of all classes, as mentioned above. Hence, we report in Table 3 the values of sensitivity (Se), specificity (Sp), and the product $Se \cdot Sp$ (see section 2) obtained by C4.5 and our GP system in the Pediatric adrenocortical tumor data set.

As can be observed in this table, both C4.5 and our GP failed to discover good rules predicting class 3 but, unlike C4.5, our GP succeeded in discovering good rules (with relatively good values of Se and Sp) predicting classes 1 and 2. In addition, in all the three classes, the value of the product $Se \cdot Sp$ obtained by our GP considerably outperforms the one obtained by C4.5.

4 Conclusions and Future Research

As mentioned in the introduction, the idea of constrained-syntax GP is not new. However, we believe this paper has the contribution of proposing a constrained-syntax GP tailored for the discovery of simple classification rules. This was achieved by incorporating into the GP the following mechanisms:

(a) An individual representation based on disjunctive normal form (DNF). As mentioned in section 2.1, the use of DNF has two advantages. First, it is an intuitively simple form of rule set presentation to the user. Second, it facilitates the enforcement of the attribute-uniqueness constraint – i.e., an attribute can occur at most once in a rule antecedent. (In passing note that, although most of the data type constraints enforced by our GP could alternatively be represented in a grammar-based GP, the attribute-uniqueness constraint cannot be directly represented in a grammar-based GP.)

(b) A result designation scheme where the solution for the classification problem consists of *k* individuals (where *k* is the number of classes), each of them predicting a different class, and all of them produced in the same run of the GP. This makes the GP more efficient, avoiding the need for k runs of the GP, as usual in the literature.

(c) Classwise elitism, an extension of the basic idea of elitism to the framework of classification. In this kind of elitism the best individual of each of the *k* classes is chosen to be passed unchanged to the next generation. This avoids that the population converges to a state where all individuals represent rule sets predicting the "easiest class", and guarantees that the result designation procedure works properly.

Although each of these ideas is perhaps relatively simple, their combination effectively produces a GP tailored for the discovery of classification rules.

In addition, this paper also offers a contribution from the data mining perspective. We have proposed a new way of preprocessing a medical data set for the purpose of predicting how long a patient will survive after a surgery.

The proposed preprocessing method was applied to the Pediatric Adrenocortical Tumor data set, but it is a relatively generic method, which could be also applied to other medical data sets where one wants to predict how long a patient will survive after a given event such as a major surgery. (Of course, the method is not generic enough to cover other kinds of prediction, such as medical diagnosis.)

Furthermore, the GP was compared with C4.5 in a difficult medical classification problem. The accuracy rate of C4.5 was found to be significantly better than the one of the GP at first glance. However, an analysis of the trees built by C4.5 showed that it was *not* discovering better classification rules. It was just building a degenerate, empty tree, predicting the majority class for all data instances. We then performed a more detailed analysis of the predictive accuracy of both systems, measuring the sensitivity and the specificity rates for each class separately, and showed that, according to this measure of predictive accuracy, overall the GP obtained considerably better results than C4.5.

We are currently applying our GP to other data sets. A future research direction might consist of performing experiments with other function sets and evaluate the influence of the function set of the GP in its performance, across several data sets.

References

[Bojarczuk et al. 2000] C.C. Bojarczuk, H.S. Lopes, A.A. Freitas. Genetic programming for knowledge discovery in chest pain diagnosis. IEEE Engineering in Medicine and Biology magazine - special issue on data mining and knowledge discovery, 19(4), 38-44, July/Aug. 2000.

[Dhar et al. 2000] V. Dhar, D. Chou and F. Provost. Discovering interesting patterns for investment decision making with GLOWER – a genetic learner overlaid with entropy reduction. *Data Mining and Knowledge Discovery Journal 4* (2000), 251-280.

[Freitas 2002] A.A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms.* Springer, 2002.

[Hand 1997] D.J. Hand. *Construction and Assessment of Classification Rules*. Chichester: John Wiley & Sons, 1997.

[Kishore et al. 2000] J.K. Kishore, L.M. Patnaik, V. Mani and V.K. Agrawal. Application of genetic programming for multicategory pattern classification. *IEEE Transactions on Evolutionary Computation 4(3)* (2000), 242-258.

[Montana 1995] D.J. Montana. Strongly typed genetic programming. *Evolutionary Computation 3* (1995), 199-230.

[Papagelis and Kalles 2001] A. Papagelis and D. Kalles. Breeding decision trees using evolutionary techniques. *Proc. 18th Int. Conf. on Machine Learning*, 393-400. San Mateo: Morgan Kaufmann, 2001.

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

[Witten and Frank 2000] I.H. Witten and E. Frank. *Data Mining: practical machine learning tools and techniques with Java implementations*. San Mateo: Morgan Kaufmann, 2000.