Creating Rule Ensembles From Automatically-Evolved Rule Induction Algorithms

Gisele L. Pappa and Alex A. Freitas

Abstract Ensembles are a set of classification models that, when combined, produce better predictions than when used by themselves. This chapter proposes a new evolutionary algorithm-based method for creating an ensemble of rule sets consisting of two stages. First, an evolutionary algorithm (more precisely, a genetic programming algorithm) is used to automatically create complete rule induction algorithms. Secondly, the automatically-evolved rule induction algorithms are used to produce rule sets that are then combined into an ensemble. Concerning this second stage, we investigate the effectiveness of two different approaches for combining the votes of all rule sets in the ensemble and two different approaches for selecting which subset of evolved rule induction algorithms (out of all evolved algorithms) should be used to produce the rule sets that will be combined into an ensemble.
Keywords: ensembles, evolutionary algorithms, rule induction algorithms, grammar-based genetic programming

1 Introduction

After many decades of research, classification is still one of the most studied data mining tasks. Classification problems can be solved by using a variety of types of algorithms, involving a really diverse set of representations to express classification models (the outputs of classification algorithms) [1]. Among these, rule induction algorithms produce classification models in the form of a set of IF-THEN classifi-

Gisele L. Pappa
Federal University of Minas Gerais, Computer Science Department, Av. Antônio Carlos, 6627, 31270-010, Belo Horizonte, MG, Brazil e-mail: glpappa@dcc.ufmg.br

Alex A. Freitas
University of Kent, Computing Laboratory, CT2 7NF, Canterbury, Kent, UK e-mail: A.A.Freitas@kent.ac.uk

cation rules (sometimes called decision rules), which have the advantage of being a knowledge representation intuitively comprehensible to the user.

Another type of algorithms that can be used to solve the classification task are evolutionary algorithms. Evolutionary algorithms (EAs) [2] are stochastic methods based on Darwin's theory of evolution and survival of the fittest. Since they are a very generic and flexible type of search algorithm, evolutionary algorithms can be used to build many different types of classification models. In particular, evolutionary algorithms became a common type of method to evolve decision trees [3], classification rules [4, 5] and neural networks [6], and are also often used as parameter optimizers for many other types of classifiers [7]. More recently, these methods have been often applied to build ensembles of classification models [8, 9, 10, 11, 12].

From a search perspective, a motivation for using evolutionary algorithms in data mining/machine learning is that they perform a population-based global search in the space of candidate solutions (normally candidate classification models, when the target problem is classification) [4]. This is in contrast with several other types of methods, such as the greedy (local search-based) methods typically used by more conventional types of rule induction algorithms. The global search performed by evolutionary algorithms makes them less likely to get trapped in local optima in the search space, by comparison with greedy methods.

Whatever the type of algorithm(s) used to solve a classification problem, in general an effective approach for increasing the predictive accuracy of that(ose) algorithm(s) consists of creating an ensemble of many classification models produced by that(ose) algorithm(s). An ensemble basically combines the predictions of many different classification models into a single prediction for each test example (unseen during training), and follows the principle that a "shared" decision will usually generate better results than an individual one. Actually, building an ensemble of classification models became a popular research area in classification and other supervised learning tasks (e.g. regression), since such ensembles have been shown to outperform – in terms of predictive accuracy – a single classification model [13].

In this chapter we are interested in evolutionary algorithms as a type of method for supporting the creation of an ensemble of classification models – in particular, an ensemble of rule sets. More precisely, this chapter proposes a new method for creating an ensemble of rule sets consisting of two stages. First, an evolutionary algorithm (a genetic programming algorithm) is used to *automatically* create *complete rule induction algorithms*. Secondly, the automatically-evolved rule induction algorithms are used to produce rule sets that are then combined into an ensemble. Concerning this second stage, we investigate the effectiveness of two different approaches for combining the votes of all rule sets in the ensemble (at prediction time) and two different approaches for selecting which subset of evolved rule induction algorithms (out of all evolved algorithms) should be used to produce the rule sets that will be combined into an ensemble.

The basic idea of using evolutionary algorithms – more specifically, genetic programming – for automatically creating complete rule induction algorithms (rather than manually designing rule induction algorithms as usual) was proposed in [14]. Note that by complete rule induction algorithm we mean algorithms with approximately the same level of complexity as well-known algorithms such as CN2 [15] or Ripper [16]. That work has shown that genetic programming can successfully take the automation of data mining/machine learning tasks one step further, by automatically creating rule induction algorithms competitive with manually-designed rule induction algorithms.

However, in [14] the evolved rule induction algorithms were not used to create an ensemble. That is, a set of 100 rule induction algorithms (individuals in the genetic programming algorithm's population) was evolved, but only the best one at the end of the evolution was returned as a solution to the user. As argued by Gagne et al. [8], this selection of the best individual out of many evolved individuals actually represents a missed opportunity for creating an ensemble "for free", since evolutionary algorithms naturally work with a (usually large) population of candidate solutions. At a high level of abstraction, this is the basic idea explored in this work, where we use the classification models (rule sets) produced by the rule induction algorithms automatically evolved by the genetic programming algorithm to produce an ensemble of rule sets.

Hence, the main contribution of this work is to propose a new evolutionary algorithm-based ensemble method where a genetic programming algorithm is used to evolve a population of complete rule induction algorithms, where the latter are then used to produce rule sets composing an ensemble. This is a new way of producing a diverse set of classification models. That is, instead of creating diverse classification models by injecting randomness into some *component* of a classification algorithm (e.g., injecting randomness in the attribute selection procedure in random forests – see Section 2), we inject a certain degree of randomness into the generation of the *complete* rule induction algorithms that will be used to produce the classification models that will form an ensemble. This degree of randomness is due to the stochastic nature of an evolutionary algorithm's search, which works with a diverse population of candidate solutions (rule induction algorithms in our case) spread across the search space. In any case, of course the evolutionary search for rule induction algorithms is not completely random, since it is actually a heuristic search guided by the fitness function (which favours rule induction algorithms with higher classification accuracy). Hence, it is hoped that the evolutionary search will do a good job of *automatically* creating a set of rule induction algorithms which are both accurate and diverse, which would tend to produce rule sets having those corresponding properties, which are desirable properties for an ensemble of classification models in general.

The chapter is organized as follows. Section 2 introduces some background knowledge on ensembles, and Section 3 introduces concepts of evolutionary algorithms and genetic programming. Section 4 shows how ensembles and evolutionary algorithms can be combined, and presents some related work. Section 5 gives an overview of the grammar-based genetic programming method used to evolve the rule induction algorithms, and describes how the ensemble is built. Finally, Section 6 reports some experimental results, and Section 7 draws some conclusions and presents some ideas of future work.

2 Ensembles of Classification Models

The concept of ensembles is based on the idea that combining a set of predictions coming from different classification models often generates better results than a single prediction produced by one of the models participating in the ensemble. Given a set of n classification models C_i, where i varies from 1 to n, an ensemble combines all the predictions of those classification models to predict a final class to an example.

According to [17], ensembles usually obtain better accuracy rates than single classification models because they solve three important problems that the single classification models have difficulties in dealing with: the statistical, the computational and the representational problems. The statistical problem refers to the size of the model's search space *versus* the number of training examples available. Many classification models' search spaces are simply too large to be explored considering the limited number of training examples available. Hence, many classification models with similar accuracy rates are found, and choosing one of them is a difficult task. The single classification model chosen might be one that does not generalize (on the test set) very well, by comparision with other models with similar classification accuracies in the training set. Ensembles avoid this problem, once a set of classification models is considered, and the chances of picking only bad models is reduced.

The computational problem relates to the fact that, as classification model search spaces are usually huge, one cannot guarantee the best model will be found in a feasible time. In order to cope with this problem, heuristics are used, and they can get trapped in local optima in the search space. Selecting a set of classification models instead of just one model reduces the chances of only getting classification models found around local optima.

The third problem, named representational, concerns classification model spaces that do not contain any good approximations to the "ideal" model. In this case, a set of weak classification models might guarantee better results than a single one.

Considering these problems, ensembles are usually built following one of two approaches. In the first approach, different classification models are built separately [18], and their predictions combined through some sort of voting scheme. In this kind of approach, the main difficult lies on how to choose how many and which classification models will be considered as part of the ensemble [11], as it is known that the success of an ensemble is highly dependent on how accurate and diverse its individual classification models are [19, 20]. In the second approach, this problem is avoided, as the classification models that compose the model are built incrementally, in a cooperative fashion [12].

Regardless of the approach used to build the ensembles, two important decisions have to be made: which method will be used to build the set of classification models, and which method will be used to to combine the predictions of each model into a single prediction. Concerning the methods used to build the ensembles, most of them work by manipulating the training set [21], the set of attributes [22] or the output labels of the training instances [23] before giving them to a classification algorithm. Some other methods work by injecting some randomness into the classification algorithms [24].

Among the most known methods for ensemble construction are bagging [21], boosting [25] and random forests [24]. Bagging works by creating multiple data samples, where each data sample is produced by sampling with replacement the original training set. These data samples typically have the same size the original training set has. Therefore, one data sample may have many copies of a specific example and no copies of others. The classification algorithm is then run on each of those data samples, producing a set of classification models constituting the ensemble.

Boosting, in contrast, works by setting weights to the training examples in the original training set. Initially, all the examples have the same weight. A sample of examples (also with the same size of the original training set) is probabilistically drawn from the training set. The probability of an example being chosen to be part of the sample is proportional to its weight. The classification algorithm is run multiple times, sequentially, in iteratively sampled data sets, and at each iteration the weights of the examples are updated according to the performance of the classification model in that particular iteration. In this way, examples difficult to be classified have their weights increased more frequently than other instances, which improves their probabilities of being selected for the next sample which will be used to train the next iteration of the classification algorithm.

Random forests are methods that combine bagging with randomly built decision trees. They produce a set of decision trees, each of them using a different sampled data set. In a popular approach to build a random forest, in order to build each decision tree, a number m of attributes are randomly selected from the total set of n attributes available in the training set – where m is typically much smaller than n. Out of the set of m randomly-selected attributes, the attribute that creates the best data split is used to label the current node of the decision tree, during the tree building process. Each decision tree is built to the largest possible extent, and there is no pruning. When classifying new examples, each tree gives its vote, and the votes of all decision trees are combined to classify a new example. Random forests are well-known for not overfitting.

After classification models are built using one of the methods described above or a related method, another method has to be applied to combine the predictions coming from different classification models. The most common of these combination methods is the majority voting, where the class assigned to a new example is the one predicted by the greatest number of classifiers, or some related weighted voting strategy.

At this point, it is important to note that, although most of the research in ensembles is concentrated on building ensembles using the same classification algorithm, ensembles can also be built using classification models produced by different classification algorithms (using different model representations). In this case, more sophisticated techniques to combine the results coming from different classifiers is needed. The most common of this type of methods is stacking [26].

Stacking works by creating a meta-model that learns to predict the final class to be assigned to a new example based on the classes predicted by the classification models that compose the ensemble. In order to learn a meta-model, a training set is built using a leave-one-out procedure. The most important decisions concerning stacking methods are the attributes to be used in the meta data set and the algorithm that will learn the meta model.

Ting and Witten [27], for instance, proposed a stacking method that predicts the probability distributions over the set of class values, rather than single class values. In this case, the meta-level attributes are the probabilities of each of the class values returned by each of the base level classification models in the ensemble. In this same work, the authors show there are several learning algorithms that are not suitable for learning meta-models, and they recommend a multi-response linear regression method for this task.

So far, we described some of the most used methods to create ensembles of classification models. Recently, AdaBoost [28] and random forests have been shown to be particularly powerful. At the same time, a lot of approaches using evolutionary algorithms for ensemble building were proposed [11, 8, 12]. In this chapter, we are particularly interested in what we call evolutionary ensembles. One of the main motivations to combine evolutionary algorithms and ensembles is to take advantage of the population of individuals (classification models) which is naturally evolved by the evolutionary algorithm. Section 3 gives an overview of evolutionary algorithms, and Section 4 describes methods using them to build ensembles of classification models.

3 Evolutionary Algorithms and Genetic Programming

Evolutionary algorithms [2] are stochastic methods based on Darwin's concepts of evolution and survival of the fittest, and they work by evolving a population of candidate solutions to a given target problem. Genetic Programming (GP) [29] is an area of evolutionary computation which aims to automatically evolve computer programs. Together with other types of evolutionary algorithms, its application is being successful because of its generality (applicability to potentially any problem domain), global search and associated implicit parallelism and noise tolerance [30, 31].

Essentially, a GP algorithm evolves a population of individuals, where each individual represents a "program" or "executable structure" that is a candidate solution to the target problem. These individuals are evaluated using a fitness function, which measures the goodness of the candidate solution (program) represented by the individual. The fittest individuals are usually selected to undergo reproduction, crossover and mutation operations. The new individuals produced by these operations are used to create a new population, which replaces the old one. This evolutionary process is carried out until an optimum solution or satisfactory solution is found, or a pre-defined number of generations (iterations) is reached.

The design of a GP algorithm has to consider five essential components:

1. A set of functions and terminals, used to create the first GP population.
2. The representation of the individuals.
3. The fitness function used to measure the quality of the individuals (candidate solutions).
4. A selection method.
5. Crossover and mutation operators, which produce new children individuals out of the selected parent individuals in the current generation (iteration).

The next subsections explain the main concepts involved in the design of these elements.

3.1 Functions and Terminal Sets

In a GP algorithm, the first population is randomly generated using a set of functions and terminals. The terminals are usually constants, variables and/or zero-argument functions. The most common functions are the boolean and arithmetical ones, but conditional and/or loop statements can also be used.

Although the designer of a GP algorithm has a lot of freedom to choose the function set, it should not have too many functions, since the more functions the greater the search space.

One constraint to be considered when choosing the sets of functions and terminals is that the closure property must be respected. This property states that every function in the function set has to be able to handle all the values it receives as input. Thus a division operator, for example, has to be modified to cope with division by zero. This is often implemented by making the operator return a given value, rather than an error, in case of division by zero.

3.2 Individual representation

Most GP algorithms work with either of two standard individual representations: a tree or a linear structure [31]. A linear representation is simply a sequence of commands that are executed from left to right. In a tree representation, the execution of the commands is usually made in postfix order (reading the leftmost node of the tree first). The majority of GP algorithms use the tree representation.

3.3 Fitness function and Selection Methods

GP algorithms evolve a population of individuals using the concepts of selection and survival of the fittest. Hence, after the initial GP population is initialized (usually randomly), individuals are evaluated using a fitness function. This function measures how well an individual solves the target problem. It is used to determine which individuals will have parts of their genetic material (i.e., parts of their candidate solution) passed onto the next generation via the action of some genetic operator (reproduction, crossover or mutation). The better the fitness of an individual, the higher the probability of that individual being selected for reproduction, crossover or mutation.

There are many selection methods, such as fitness-proportional selection, ranking selection and tournament selection. Tournament selection, for example, randomly gets a pre-defined number of individuals from the population and simulates a tournament among them. Typically, the individual with the best fitness is declared the winner of the tournament and is therefore selected for reproduction, crossover or mutation.

3.4 Crossover and Mutation Operators

Crossover swaps genetic material (parts of candidate solutions) between two individuals, whereas mutation replaces some part of the genetic material of an individual with a new randomly-generated genetic material. These two operations are applied with user-specified probabilities. The basic idea of these operators is as follows.

Crossover re-combines the genetic material of two parent individuals, in order to produce two new children. If the individuals are represented by trees, randomly-selected subtrees are swapped between the two parents. In the case of linear genomes, randomly-selected linear segments of code are swapped.

Unlike crossover, mutation acts on a single parent individual of the population. It randomly selects a subtree of the tree-based genome or a segment of code in linear genomes and replaces it by a new randomly-generated subtree or code segment. Both crossover and mutation operations can be implemented in many ways - see [31] for a detailed review of these operators.

It should be noted that, although the mechanisms of crossover and mutation are essentially a form of random moves in the search space (which makes them problem independent), a GP algorithm as a whole is a heuristic search method, because crossover and mutation are applied to parent individuals which are selected based on their (problem dependent) fitness, i.e. a measure of their ability in solving the target problem. Hence, by iteratively applying crossover and mutation operators to the best programs of the current gereration (iteration), the population of programs gradually evolves to better and better programs.

4 Evolutionary Methods for Creating Ensembles

Evolutionary algorithms and ensembles can be used together in two different broad approaches. In the first one, the evolutionary algorithm has the same role as a traditional classification algorithm: it builds a classification model from data. In this case, each individual in the EA represents a classification model, and a subset of them is selected and then combined using one of the traditional ensemble techniques, such as bagging or boosting [18]. The selection of the models (individuals) that will compose the final ensemble is the most difficult problem to be tackled in this approach.

In the second approach, evolutionary algorithms are used to optimize some components – or (broadly speaking) "parameters" – of an ensemble. These components or parameters might vary from the best set of attributes [10] to the number of classification models that should compose an ensemble [32]. Combinations of both strategies are also valid [9].

The evolutionary algorithm-based ensemble method proposed in this chapter can be considered as a new method belonging to the above second approach where, instead of optimizing a simple parameter - such as ensemble size, we "optimize" (evolve) a fundamental "macro-parameter": the classification algorithms that will produce the classification models that will compose the ensemble. In other words, a genetic programming algorithm is used to automatically create an entire set of classification (more precisely, rule induction) algorithms. After the evolved rule induction algorithms are applied to the data being mined, they create a set of classification models (rule sets), from which an ensemble of rule sets is produced. This method will be explained in detail in Section 5. Before moving to that section, however, this section reviews related work on evolutionary algorithm-based ensemble methods.

Let us now describe some methods following the first approach described above. One of the first ideas of combining ensembles and genetic programming (GP) was presented by Iba [18], who introduced the BagGP and BoostGP algorithms. In both algorithms, each individual in the population represents a classification model, and the population of a GP algorithm is divided into S sub-populations. In the case of BagGP, each subpopulation is trained with a data set of size N, sampled with replacement from the original training set (which also has size N). The best individuals of each population have the right to vote when classifying new examples in the test set.

BoostGP follows the same basic algorithmic procedure that AdaBoost does. At the beginning of the evolutionary process, each example of the training set is associated with the same weight w. The probability of an example being chosen to be part of the sampled training set, which has the same size as the original training set, is proportional to the weight it is associated with. Each sub-population is evolved sequentially with a different training set, and the weights are updated according to a loss function. After all the sub-populations are evolved, the best individual of each population votes to classify a new example.

In this same line of work, Folino *et al* [33] proposed a boosting system based on cellular genetic programming that they named ClustBoostCGPC. ClustBoostCGPC is a parallel GP algorithm, where each processing unit evolves a different population (where each individual represents a decision tree), which is locally evaluated. However, in contrast with the method proposed in [18], at the end of the evolution the well-known k-means clustering algorithm [34] is used to generate clusters of individuals for each population. The best individuals of each cluster in each population are chosen to build the ensemble. One drawback of this method is that it can generate really large ensembles, so the authors use some pruning strategies to remove individuals from a previously built ensemble.

Chen *et al* [11] proposed a multi-objective evolutionary algorithm that, combined with a Bayesian automatic relevance determination (ARD) method [35], designed and trained ensembles of neural networks using attribute selection. The neural nets were initially built using a subset of attributes from the training set, and their parameters optimized using the Bayesian ARD method. The evolutionary algorithm was used to evolve neural networks with few attributes and low error rate. At the end of the evolutionary process, a logistic regression method was used to choose the networks that would be part of the ensemble.

Gagne *et al* [8] proposed and compared two evolutionary algorithms for ensemble learning. The first, named Off-EEL (an “off-line” ensemble construction method), used an evolutionary algorithm to evolve neural networks and built the ensemble after evolution using a greedy approach. The second approach, named On-EEL (an “on-line” ensemble construction method), used the same type of evolutionary algorithm but selected the final classification models that would form the ensemble during the evolution process. The results showed that the first approach obtained better results than the second one. They claim this result can be explained by the On-EEL getting often trapped in local optima in the search space.

Kim *et al* [9] also proposed a local selection algorithm to evolve ensembles of neural nets combined with attribute selection. In this case, each neural net was trained with a different subset of attributes, but individual neural networks and ensembles were evolved simultaneously. Each individual was represented by a binary string that encoded the attributes that should be used to train the neural network and the list of ensembles the individual’s classification model belonged to. In this way, the ensembles being evolved competed directly to each other, and the individual neural networks also competed among themselves, and were allowed to move to the fittest ensembles.

In contrast to the methods described until now, Sirlantzis *et al* [32] used an evolutionary algorithm to select the best combination of a set of pre-defined classification algorithms and voting strategies, following the second approach previously described. Depending on the problem being tackled, they used 6 or 12 classification algorithms together with 4 voting strategies.

This was also the approach followed by Kim and Cho [12], where an evolutionary algorithm was used to find ensembles of pairs of attribute selection method and classification algorithm. In this case, they pre-defined 7 attribute selection methods and 6 classification algorithms. This generated 42 pairs of attribute selection method and classification algorithm, which were represented in an individual as a vector with 42 positions (“genes”). Each individual vector’s gene could be represented by a binary digit or a real-valued number. In the first case, a 1 indicated that the pair of attribute selection method and classification algorithm associated with that gene is part of the ensemble being evolved. In the second case, a number different from 0 meant the pair associated with the gene was being used in the ensemble, and its vote when classifying unseen examples was weighted by the value present in that gene of the individual. Hence, each individual represented an ensemble of classification models, which would be generated using the pairs of attribute selection method and classification algorithms indicated by its gene vector.

All the evolutionary algorithms described above aim to create very accurate and diverse ensembles. In order to achieve that, beside the ideas already discussed, other mechanisms were also used to improve the effectiveness of the evolutionary algorithm-based ensemble systems. For instance, fitness based on accuracy was combined with niching mechanisms or modified according to co-evolution principles [36], and speciation was applied [37]. Moreover, multi-objective EAs (MOEAs) became commonly used as a way to optimize both accuracy and diversity simultaneously [38].

5 Building an Ensemble of Rule Sets from Rule Induction Algorithms Automatically Created by Genetic Programming

As pointed out before, the main difference among the evolutionary method proposed in this paper and other evolutionary algorithm-based ensemble methods proposed in the literature is that the evolutionary method proposed here does not evolve classification models. Instead, it evolves complete rule induction algorithms. The evolved rule induction algorithms are then used to produce rule sets composing an ensemble.

The method proposed uses a grammar-based genetic programming (GGP) algorithm to evolve a population of rule induction algorithms [39]. GGP [40] is a special type of genetic programming (GP) algorithm where the individuals are generated following the production rules of a grammar (instead of sets of terminals and functions), which enforces the generation of syntactically correct individuals only. Besides, the grammar can incorporate background knowledge about how the problem being tackled is solved. In the case of the system used in this paper, the grammar incorporates knowledge about how humans design rule induction algorithms.

As explained before, there are some elements in GP that need to be defined according to the problem being tackled. In the case of GGP, the sets of terminals and functions of a standard GP is replaced by a grammar, which is described in detail in [14, 39]. Each individual represents a complete rule induction algorithm, such as CN2 or Ripper. Fig.1 shows an example of an individual generated by following the production rules of the grammar.

In order to extract from an individual's tree the pseudo-code of the corresponding rule induction algorithm, we read all the terminals (leaf-nodes) in the tree from left to right. The tree in Figure 1, for example, represents the pseudo-code of the CN2 algorithm [15] producing an ordered list of rules, with the beam-width (or star size, using the CN2 terminology) parameter set to 5 and the statistical significant test threshold set to 0.01.

Figure 2 shows the scheme of the proposed method. As observed, it represents a standard evolutionary algorithm execution, combined with some interesting points. First, as already explained, the individuals of the first population are generated by

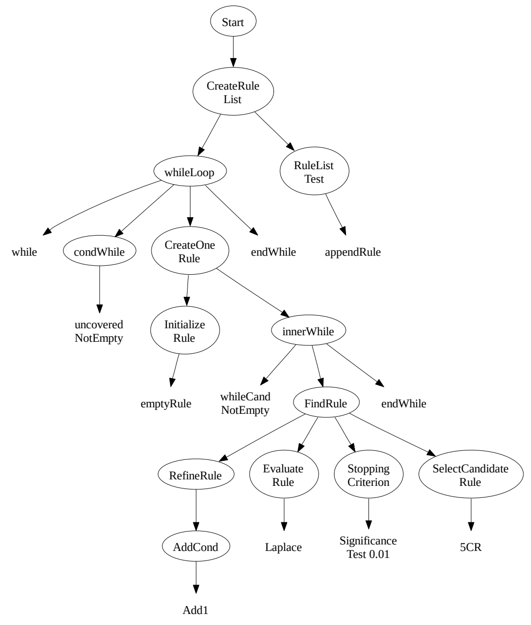

Fig. 1 Example of a GGP individual (candidate rule induction algorithm)

following the production rules of a grammar. Second, the individuals are evaluated using a "meta-training set", which is composed of a set of *n* complete data sets (i.e. a training set and a validation set for each data set, with no overlapping of examples between these two sets). After executing the candidate rule induction algorithm in each data set *i* of the meta-training set, a fitness function based on the average of the value of fit_i (given by Eq. 1) over all data sets is calculated. In Eq. 1, Acc_i represents the accuracy (on the validation set) obtained by the rules discovered by the rule induction algorithm in the training set of data set *i*. $DefAcc_i$ represents the default accuracy (the accuracy obtained when using the most frequent class in the training examples to classify all examples in the validation set).

$$fit_i = \begin{cases} \frac{Acc_i - DefAcc_i}{1 - DefAcc_i}, \text{ if Acc}_i > \text{DefAcc}_i \\ \frac{Acc_i - DefAcc_i}{DefAcc_i}, \text{ otherwise} \end{cases} \tag{1}$$

Note that, for each data set in the meta-training set, the validation set has the same role as the test set in conventional classification, consisting of examples unseen during training. However, we prefer to use the term validation set (rather than test set) in this context because the fitness values of individuals are iteratively computed by accessing the validation set many times during evolution, and so the term test set would be misleading.

By executing the candidate rule induction algorithms in a set of different classification problems, we aim at automatically evolving robust rule induction algorithms

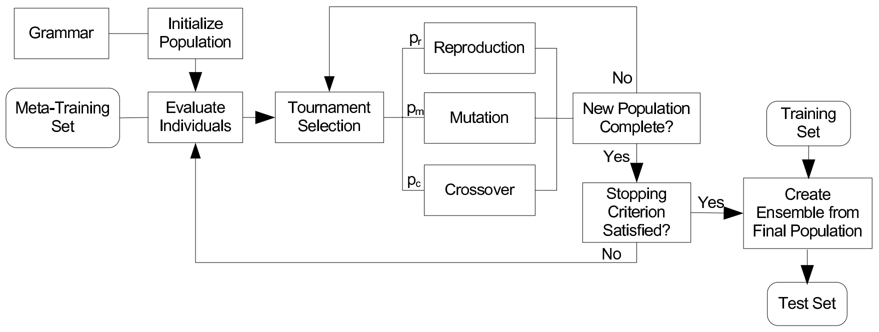

Fig. 2 Scheme of the GGP algorithm used to create the ensemble

which can generalize well on new data sets (application domains), different from the data sets (application domains) used in the meta-training set. During the evolution, individuals created by crossover and mutation operations also have to be valid according to the grammar. A detailed description of this system can be found in [14, 39].

At the end of the evolution, all the individuals in the last population (or a subset of them, depending on the version of the system used – see next Section) are used to create an ensemble of rule sets. That is, for each rule induction algorithm in the last population, we run it in each new target data set (unseen during the GGP algorithm's evolution), producing a corresponding rule set, and then we create an ensemble consisting of the rule sets produced by those rule induction algorithms.

Once such an ensemble of rule sets has been produced, two different voting strategies can be used to combine the rule sets' predictions, namely majority voting and fitness-weighted voting. The majority voting strategy, as its name suggests, classifies an example as belonging to the class predicted by the highest number of rule sets in the ensemble. The fitness-weighted voting strategy weighs the vote of each rule set according to the fitness value attributed to its corresponding individual (rule induction algorithm) in the GGP algorithm's last generation (see Eq. 1).

As pointed out before, the success of an ensemble is highly dependent on the diversity and accuracy of the individuals belonging to it. In terms of accuracy, it is expected that, at the end of the evolutionary process, all the individuals will present accuracies at least better than the ones obtained by the baseline classification model (which always predicts the most frequent class in the training set for all examples in the test set). In terms of diversity, an analysis of the components of the grammar present in the individuals of the last generation showed that usually a diverse set of rule induction algorithms is evolved.

Table 1 Data sets used to create ensembles

Data set	Examples	Attributes	Classes	Def. Acc.(%)
crx	690	5	2	67.7
heart-c	303	13	2	54.5
ionosphere	351	34	2	64
sonar	208	60	2	53
segment	2310	19	7	14.3

6 Computational Experiments

This section describes experiments performed to create ensembles of rule sets from the last population of individuals (rule induction algorithms) evolved by the GGP algorithm (with a population size of 100 individuals). Experiments were performed in 5 UCI data sets [41], listed in Table 1. The figures in the column *Examples* indicate the number of examples present in the data set, followed by the number of attributes and classes. The last column shows the default accuracy (the accuracy obtained when using the most frequent class in the training set to classify new examples in the test set). It is important to emphasize that these data sets are different from the ones used in the GGP algorithm's meta-training set during the evolutionary process. In the experiments reported in this work, the data sets used in the meta-training set were *monks-2*, *monks-3*, *balance-scale*, *lymph* and *zoo*.

The accuracies obtained by the ensemble of rule sets produced by the evolved rule induction algorithms (from now on referred to as Rule-Ens for short) using the two voting strategies explained in Section 5 were compared to the accuracies obtained by the rule set produced by the single best individual returned to the user as the best evolved rule induction algorithm (GGP-RI). Table 2 shows the results obtained by a 5-fold cross validation procedure (with numbers after the symbol $\pm$ representing standard deviations). The results were compared using a paired two tailed Student's t-test with significance level 0.05. Cells in dark gray represent significant wins of Rule-Ens against the GGP-RI. In Table 2 the column names "Rule-Ens Maj." and "Rule-Ens Fit." refer to Rule-Ens using the majority voting and fitness-weighted voting strategies, respectively.

The results in Table 2 show that, for the data set *segment*, both the ensemble using a majority voting and the ensemble using a fitness-weighted voting obtained better predictive accuracy rates than the GGP-RI. All the other results obtained are considered to be statistically the same as the ones obtained by the GGP-RIs.

The results in Table 2 were produced by using all individuals (rule induction algorithms) of the last population to produce rule sets and combining all those rule sets into an ensemble. In a second set of experiments, we tried to reduce the number of rule sets in the ensemble. We selected a subset of rule induction algorithms to be used to produce the rule sets for the ensemble according to the following two methods.

In the first method we selected the top 10 individuals of the last population according to a fitness-based ranking (where fitness is given by Eq. 1) and created the

Table 2 Predictive accuracy rates (%) obtained by the ensemble of rule sets built from the last population of evolved rule induction algorithms using two different voting strategies

Data Set	Rule-Ens Maj.	Rule-Ens Fit.	GGP-RIs
crx	83.2 ± 0.02	82.01 ± 0.02	79.4 ± 0.01
heart-c	80.54 ± 0.02	81.86 ± 0.02	76.6 ± 0.036
ionosphere	86.3 ± 0.01	89.91 ± 0.01	86.9 ± 0.024
segment	97.01 ± 0.002	96.45 ± 0.003	94.5 ± 0.005
sonar	74.38 ± 0.05	74.87 ± 0.04	73.1 ± 0.036

corresponding ensemble of rule sets produced by those individuals (denoted Ens-Top). In the second method we selected the top 5 and the bottom 5 individuals of the last population according to a fitness-based ranking and combined the rule sets produced by both groups of individuals in a single ensemble (denoted Ens-Mix). This last method intuitively tends to increase the diversity of the selected rule induction algorithms (which in turn tends to increase the diversity of their produced rule sets), once similar rule induction algorithms are likely to have similar fitness values. The results obtained are presented in Table 3. Again, a paired two-tailed Student's t-test with significance level of 0.05 was used to evaluate the statistical significance of the differences in the accuracies obtained by each of the two proposed ensemble methods and the GGP-RI baseline, and significant wins of one of the proposed ensemble methods are highlighted in dark grey in the table.

Table 3 shows that both types of ensembles obtain significantly better predictive accuracies than the GGP-RI baseline in the *segment* data set, and that the ensemble produced by using the mixed strategy (which in theory has more diverse rule sets) also obtains significantly better predictive accuracies than the GGP-RI baseline in the *crx* data set. Note that in the *crx* data set only the accuracy of Ens-Mix (and not the accuracy of Ens-Top) is significantly better than the accuracy of GGP-RI. This result shows that the weaker individuals in the last population actually contribute to generate a better ensemble (since the weaker individuals tend to increase the diversity of the rule sets in the ensemble) in the *crx* data set, leading to the generation of a better ensemble by comparison with the ensemble consisting only of rule sets produced by the strongest individuals in the last population. It is important to point out that the difference in fitness values from the best to the worst individual in the last population was significant in the case of the *crx* data set, but even the worst individual of the population could not be considered a bad individual, after being evolved for 100 generations.

In conclusion, overall the ensembles built with 10 individuals selected from the last population produced better results than the ensembles produced by taking all the 100 individuals of the last population into account. In addition, mixing the rule sets produced by the 5 best and the 5 worst individuals of the last population according to fitness values produced ensembles presenting good results (significantly better than the GGP-RI baseline) in 2 out of the 5 data sets used in the experiment.

Table 3 Predictive accuracy rates (%) obtained by an ensemble of rule sets built from a set of 10 rule induction algorithms selected from the last population of evolved rule induction algorithms

Data Set	Ens-Top	Ens-Mix	GGP-RIs
crx	80.99 ± 0.02	84.2 ± 0.02	79.4 ± 0.01
heart-c	80.55 ± 0.02	82.18 ± 0.01	76.6 ± 0.036
ionosphere	85.76 ± 0.012	85.76 ± 0.01	86.9 ± 0.024
segment	97.01 ± 0.002	96.71 ± 0.003	94.5 ± 0.005
sonar	74.38 ± 0.05	74.39 ± 0.05	73.1 ± 0.036

7 Conclusions and Future Research Directions

This paper proposed a new evolutionary algorithm-based method to produce an ensemble of rule sets. More precisely, the proposed method consists of two basic stages. First, it exploits the population-based search of a genetic programming algorithm to automatically evolve a diverse set of rule induction algorithms. We emphasize that what is being evolved by the genetic programming algorithm is a set of *complete rule induction algorithms*, with the same level of complexity as well-known manually-designed algorithms like CN2 or Ripper, rather than classification models as in many other evolutionary algorithms. Secondly, the set of evolved rule induction algorithms in the last population (or a selected subset of those algorithms) is used to produce a set of rule sets (one rule set for each rule induction algorithm), which are then combined into an ensemble of rule sets.

Experiments in 5 public-domain data sets showed that by using a simple majority voting scheme the ensembles of rule sets produced by using all the individuals or only 10% of the individuals in the last population were capable of obtaining predictive accuracy rates higher than the ones obtained by a single rule set (produced by the best individual in the last population) in 2 out of the 5 data sets.

Concerning future research directions, we believe these “proof of concept” results can be improved by adding some diversity measure to the genetic programming algorithm’s fitness function, or by using a more sophisticated technique to select the evolved individuals (rule induction algorithms) that will be used to produce the rule sets composing the ensemble.

In addition, other voting schemes, and even a meta-classification model, could also be used to compute the ensemble prediction. An interesting experiment would involve to actually leave up to the genetic programming algorithm the task of selecting how many and which individuals should be used and which voting scheme should be used to maximize the predictive accuracy of the resulting ensemble.

Yet another research direction to be explored is to use a multi-objective version of the genetic programming algorithm (based on the concept of Pareto dominance) which simultaneously optimizes both the diversity and the accuracy of the individuals being evolved.

Acknowledgments

The first author is financially supported by FAPEMIG, Brazil.

References

1. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. 2^{nd} edn. Morgan Kaufmann (2005)
2. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computation. Springer-Verlag (2003)
3. Wong, M.L., Leung, K.S.: Data Mining Using Grammar-Based Genetic Programming and Applications. Kluwer, Norwell, MA, USA (2000)
4. Freitas, A.A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer-Verlag (2002)
5. Tsakonas, A., Dounias, G., Jantzen, J., Axer, H., Bjerregaard, B., von Keyserlingk, D.G.: Evolving rule-based systems in two medical domains using genetic programming. Artificial Intelligence in Medicine **32**(3) (2004) 195–216
6. Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE **87**(9) (1999) 1423–1447
7. Rozsypal, A., Kubat, M.: Selecting representative examples and attributes by a genetic algorithm. Intelligent Data Analysis **7**(4) (2003) 291–304
8. Gagné, C., Sebag, M., Schoenauer, M., Tomassini, M.: Ensemble learning for free with evolutionary algorithms? In: GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, New York, NY, USA, ACM (2007) 1782–1789
9. Kim, Y., Street, W.N., Menczer, F.: Optimal ensemble construction via meta-evolutionary ensembles. Expert Systems With Applications **30**(4) (2006)
10. Oliveira, L.S., Morita, M., Sabourin, R., Bortolozzi, F.: Multi-objective genetic algorithms to create ensemble of classifiers. In: Proc. of the Third Int. Conf. on Evolutionary Multi-Criterion Optimization. (2005) 592–606
11. Chen, H., Yao, X.: Evolutionary multiobjective ensemble learning based on bayesian feature selection. In: CEC 2006. IEEE Congress on Evolutionary Computation. (2006) 267–274
12. Kim, K.J., Cho, S.B.: An evolutionary algorithm approach to optimal ensemble classifiers for dna microarray data analysis. IEEE Trans. Evolutionary Computation **12**(3) (2008) 377–388
13. Brown, G., Wyatt, J.L., Harris, R., Yao, X.: Diversity creation methods: a survey and categorisation. Information Fusion **6**(1) (2005) 5–20
14. Pappa, G.L., Freitas, A.A.: Automatically evolving rule induction algorithms. In Fürnkranz, J., Scheffer, T., Spiliopoulou, M., eds.: Proc. of the 17^{th} European Conf. on Machine Learning. Volume 4212., Berlin, Springer Verlag (2006) 341–352
15. Clark, P., Boswell, R.: Rule induction with cn2: some recent improvements. In Kodratoff, Y., ed.: EWSL-91: Proceedings of the European Working Session on Learning on Machine Learning, Springer-Verlag (1991) 151–163
16. Cohen, W.W.: Fast effective rule induction. In: Proc. of the 12th International Conference on Machine Learning, Morgan Kaufmann (1995) 115–123
17. Dietterich, T.G.: Ensemble learning. In Arbib, M., ed.: Handbook of Brain Theory and Neural Networks. MIT Press (2002) 405–409
18. Iba, H.: Bagging, boosting, and bloating in genetic programming. In Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E., eds.: Proceedings of the Genetic and Evolutionary Computation Conference. Volume 2., Orlando, Florida, USA, Morgan Kaufmann (1999) 1053–1060
19. Dietterich, T.G.: Machine-learning research: Four current directions. The AI Magazine **18**(4) (1998) 97–136
20. Kuncheva, L.I., Whitaker, C.J.: Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Machine Learning **51**(2) (2003) 181–207

21. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2) (1996) 123–140
22. Stefano, C.D., Marcelli, A.: Exploiting reliability for dynamic selection of classifiers by means of genetic algorithms. In: In Proc. ICDAR03. (2003) 671–675
23. Dietterich, T.G., Bakiri, G.: Solving multiclass learning problems via error-correcting output codes. Journal of Artificial Intelligence Research **2** (1995) 263–286
24. Breiman, L.: Random forests. Machine Learning **45**(1) (2001) 5–32
25. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: EuroCOLT '95: Proc. of the 2^{nd} European Conference on Computational Learning Theory, London, UK, Springer-Verlag (1995) 23–37
26. Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. Machine Learning **36**(1-2) (1999) 105–139
27. Ting, K.M., Witten, I.H.: Issues in stacked generalization. Journal of Artificial Intelligence Research **10** (1999) 271–289
28. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: EuroCOLT '95: Proc. of the 2nd European Conference on Computational Learning Theory, London, UK, Springer-Verlag (1995) 23–37
29. Koza, J.R.: Genetic Programming: On the Programming of Computers by the means of natural selection. The MIT Press, Cambridge, Massachusetts (1992)
30. Baeck, T., Fogel, D.B., Michalewicz, Z.: Evolutionary Computation 1 Basic Algorithms and Operators. Institute of Physics Publishing (2000)
31. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann (1998)
32. Sirlantzis, K., Fairhurst, M., Guest, R.: An evolutionary algorithm for classifier and combination rule selection in multiple classifier systems. Pattern Recognition, 2002. Proceedings. 16th International Conference on **2** (2002) 771–774 vol.2
33. Folino, G., Pizzuti, C., Spezzano, G.: Training distributed gp ensemble with a selective algorithm based on clustering and pruning for pattern classification. IEEE Trans. Evolutionary Computation **12**(4) (2008) 458–468
34. Tan, P., Steinbach, M., Kumar, V.: An Introduction to Data Mining. Addison-Wesley (2006)
35. Neal, R.: Bayesian Learning for Neural Networks. PhD thesis, University of Toronto (1994)
36. Liu, Y., Yao, X., Higuchi, T.: Evolutionary ensembles with negative correlation learning. IEEE-EC **4**(4) (2000) 380
37. Ando, S.: Heuristic speciation for evolving neural network ensemble. In: GECCO '07: Proc. of the 9th Conf. on Genetic and Evolutionary Computation, New York, NY, USA, ACM (2007) 1766–1773
38. Chandra, A., Yao, X.: Ensemble learning using multi-objective evolutionary algorithms. Journal of Mathematical Modeling and Algorithms **5**(4) (2006) 417–445
39. Pappa, G.L.: Automatically Evolving Rule Induction Algorithms with Grammar-based Genetic Programming. PhD thesis, Computing Laboratory, University of Kent, Canterbury, UK (2007)
40. Whigham, P.A.: Grammatical Bias for Evolutionary Learning. PhD thesis, School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy, Canberra, Australia (1996)
41. Newman, D.J., Hettich, S., Blake, C.L., Merz, C.J.: UCI Repository of machine learning databases. University of California, Irvine, http://www.ics.uci.edu/∼mlearn/MLRepository.html (1998)