An Evolutionary Learning Technique in Classification Study

Mrs. S. R. Swarna Latha ¹, Dr. G. M. Kadharnawaz ²
¹Research Scholar, Manonmaniam Sundaranar University, Tirunelveli, Tamil Nadu (India)
²Director, Department Of Computer Application, Sona College Of Technology, Salem Tamil Nadu (India)

Abstract
Classification is one of the most researchable ideas in machine learning and data mining. A wide range of real problems like credit scoring, bankruptcy prediction, medical diagnosis, pattern recognition, text categorization, software quality assessment, and many more have been used as classification problems. Genetic programming (GP) is a flexible and powerful evolutionary technique with some features that can be very valuable and suitable for the evolution of classifiers. This paper surveys existing literature about the application of genetic programming to classification, to show the different ways in which this evolutionary algorithm can help in the construction of accurate and reliable classifiers.

Key Words—Classification, Evolutionary Algorithm, Crossover, Mutation, Genetic Programming (GP)

I. Introduction

Two main approaches for learning are distinguished in machine learning: supervised and unsupervised learning. In supervised learning, attributes of data instances are divided into two types: inputs or independent variables and outputs or dependent variables. The goal of the learning process consists in predicting the value of the outputs from the value of the inputs. In order to accomplish this goal, a training set of data (data instances including the values of both input and output variables with known values) is employed to guide the learning process. Regression and classification are two types of supervised learning tasks. In regression, there is a continuous-valued output to be predicted while in classification the output is discrete. In unsupervised learning, there is no distinction of type between the variables of the data instances. The goal of unsupervised learning is to find the intrinsic structure, relations, or affinities present in data.

In supervised learning, attributes of data instances are divided into two types: inputs or independent variables and outputs or dependent variables. The goal of the learning process consists in predicting the value of the outputs from the value of the inputs. In order to accomplish this goal, a
training set of data (data instances including the values of both input and output variables with known values) is employed to guide the learning process. Regression and classification are two types of supervised learning tasks. In regression, there is a continuous-valued output to be predicted while in classification the output is discrete. In unsupervised learning, there is no distinction of type between the variables of the data instances. The goal of unsupervised learning is to find the intrinsic structure, relations, or affinities present in data.

Classification is one of the most studied problems in machine learning and data mining. It consists in predicting the value of a categorical attribute (the class) based on the values of other attributes (predicting attributes). A search algorithm is used to induce a classifier from a set of correctly classified data instances called the training set. Another set of correctly classified data instances, known as the testing set, is used to measure the quality of the classifier obtained. Different kinds of models, such as decision trees or rules, can be used to represent classifiers.

This paper reviews the works published in the literature, where GP is applied in some form to address classification tasks. The rest of the paper is organized as follows. Section II briefly presents the GP. Section III briefly presents the classification fields. Section IV gives a preliminary overview of the possible ways of applying GP for classification. The drawbacks and advantages of using GP for classification are analyzed in Section V. Section VI presents some concluding remarks.

II. GP (Genetic Programming)

Genetic programming (GP) is an evolutionary learning technique that offers a great potential for classification. GP is a very flexible heuristic technique that allows us to use complex pattern representations such as trees. GP and other evolutionary techniques has been successfully applied to different supervised learning tasks like regression and unsupervised learning tasks like clustering and association discovery. However, we focus our review on the application of GP to classification.

The distinctive features of GP make it a very convenient technique with regard to classification. The goal of having computers automatically solve problems is central to artificial intelligence, machine learning, and the broad area encompassed by what Turing called ‘machine intelligence’, “to get machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence.” Genetic programming (GP) is an evolutionary computation (EC) technique that automatically solves problems without having to tell the computer explicitly how to do it. At the most abstract level GP is a systematic, domain independent method for getting computers to automatically solve problems starting from a high-level statement of what needs to be done.

GP algorithm

1: Randomly create an initial population of programs from the available primitives.
2: repeat
3: Execute each program and ascertain its fitness.
4: Select one or two program(s) from the population with a probability based on fitness to participate in genetic operations.
5: Create new individual program(s) by applying genetic operations with specified probabilities.
6: until an acceptable solution is found or some other stopping condition is met.
7: return the best-so-far individual.

A. GP in a Nutshell

Technically, GP is a special evolutionary algorithm (EA) where the individuals in the population are computer programs. So, generation by generation GP iteratively transforms populations of programs into other populations of programs as illustrated in Fig. 1.

During the process, GP constructs new programs by applying genetic operations which are specialized to act on computer programs. Algorithmically, GP comprises the steps shown in Algorithm. The main genetic operations involved in GP (line 5 of Algorithm) are the following:
- Crossover
- Mutation

Some GP systems also support structured solutions, and some of these then include architecture-altering operations which randomly alter the architecture of a program to create a new offspring program. Also, often, in addition of crossover, mutation and the architecture-altering operations, an operation which simply copies selected individuals in the next generation is used. This operation, called reproduction, is typically applied only to produce a fractional value.

Genetic programming is a branch of genetic algorithms. The main difference between genetic programming and genetic algorithms is the representation of the solution. Genetic programming creates computer programs in the lisp or scheme computer languages as the solution. Genetic algorithms create a string of numbers that represent the solution. Genetic programming uses four steps to solve problems:
- Generate an initial population of random compositions of the functions and terminals of the problem (computer programs).
- Execute each program in the population and assign it a fitness value according to how well it solves the problem.
- Create a new population of computer programs.
  i) Copy the best existing programs.
ii) Create new computer programs by mutation.
iii) Create new computer programs by crossover. (Sexual reproduction)

- The best computer program that appeared in any generation, the best-so-far solution, is designated as the result of genetic programming.

**Flowchart for Genetic Programming**

The basic purpose of genetic algorithms (GAs) is optimization. Since optimization problems arise frequently, this makes GAs quite useful for a great variety of tasks. As in all optimization problems, we are faced with the problem of maximizing/minimizing an objective function \( f(x) \) over a given space \( X \) of arbitrary dimension. A brute force which would consist in examining every possible \( x \) in \( X \) in order to determine the element for which \( f \) is optimal is clearly infeasible. GAs give a heuristic way of searching the input space for optimal \( x \) that approximates brute force without enumerating all the elements and therefore bypasses performance issues specific to exhaustive search.

1. **Fitness function**

The most difficult and most important concept of genetic programming is the fitness function. The fitness function determines how well a program is able to solve the problem. It varies greatly from one type of program to the next. For example, if one were to create a genetic program to set the time of a clock, the fitness function would simply be the amount of time that the clock is wrong.
Unfortunately, few problems have such an easy fitness function. Most cases require a slight modification of the problem in order to find the fitness.

2. Crossover

The creation of one or two offspring programs by recombining randomly chosen parts from two selected programs. Two primary operations exist for modifying structures in genetic programming. The most important one is the crossover operation. In the crossover operation, two solutions are sexually combined to form two new solutions or offspring. The parents are chosen from the population by a function of the fitness of the solutions. Three methods exist for selecting the solutions for the crossover.

The first method uses probability based on the fitness of the solution. If \( f(S_i(t)) \) is the fitness of the solution \( S_i \) and

\[
\sum_{j=1}^{M} f(S_j(1))
\]

is the total sum of all the members of the population, then the probability that the solution \( S_i \) will be copied to the next generation is:

\[
\frac{f(S_i(1))}{\sum_{j=1}^{M} f(S_j(1))}
\]

Another method for selecting the solution to be copied is tournament selection. Typically the genetic program chooses two solutions random. The solution with the higher fitness will win. This method simulates biological mating patterns in which, two members of the same sex compete to mate with a third one of a different sex. Finally, the third method is done by rank. In rank selection, selection is based on the rank, (not the numerical value) of the fitness values of the solutions of the population.

3. Mutation

The creation of one new offspring program by randomly altering a randomly chosen part of one selected program. Mutation is another important feature of genetic programming. Two types of mutations are possible. In the first kind a function can only replace a function or a terminal can only replace a terminal. In the second kind an entire subtree can replace another subtree.

III. Classification

A classification problem deals with associating a given input pattern with one of the distinct classes. Patterns are specified by a number of features (representing some measurements made on the objects that are being classified) so it is natural to think of them as \( d \)-dimensional vectors, where \( d \) is the number of different features. This representation gives rise to a concept of feature space. Patterns are points in this \( d \)-dimensional space and classes are sub-spaces. Classification problem reduces to determining which of the regions a given pattern falls into. If classes do not overlap they are said to be separable and, in principle, one can design a decision rule which will successfully classify any input pattern. A decision rule determines a decision boundary which
partitions the feature space into regions associated with each class. It represents our best solution to the classification problem. Fig.3 illustrates a 2-dimensional feature space with three classes occupying regions of the space.

The goal is to design a decision rule which is easy to compute and yields the smallest possible probability of misclassification of input patterns from the feature space. Our information about the classes (which we use to design a decision rule) is usually derived from some finite sample of patterns with known class affiliations. This sample is called a training set. If we make a decision boundary complex enough every pattern in the training set will be properly classified using the underlying decision rule, even if the distributions of patterns overlap.

Classifiers are designed with a purpose of classifying novel patterns (not belonging to the training set) and it is unlikely that an overly complex decision boundary would provide good generalization (classify novel patterns with the same success rate that is achieved on the training set) as it as tuned to perform extremely well on the training set. This is known as over-fitting the training set. The picture shows a decision boundary over-fitting a training set distributed according to the classes of the previous figure. Perfect classification is achieved even though the classes overlap. Therefore, we might seek to simplify the shape of the decision boundary which will, by sacrificing performance on the training samples, improve the performance on new patterns. Different classifiers can be implemented by constructing an appropriate discriminant function $g_i(x)$, where $i$ is the class index. A pattern $x$ is associated with the class $j$ such that $g_j(x) > g_i(x)$ for every $i$ not equal to $j$. A simplest discriminant function is linear in pattern features. If a particular type of discriminant function is a priori chosen for a problem at hand, one is left with a task of finding the parameters yielding the optimal performance (minimal probability of error) of the classifier.

### IV. GP for Classification

Flexibility is one of the main advantages of GP, and this feature allows GP to be applied for classification in many different ways. Individuals are usually represented as trees, and trees are one of the most general data structures, they can be tailored to fit the representation requirements in a wide range of problems and domains. But GP is not limited to tree-like individuals, because it also allows the use of other representations, like linear or graph structures. In addition, the nature of GP individuals, which include terminals (variables and constants) and nonterminal (operators and
functions), gives them the ability not only to represent knowledge but also to perform computations, so that GP can be used in almost any classification-related task, not only in the core task of classifier induction, but also for preprocessing and post processing purposes. At the preprocessing stage, data can be transformed in order to increase the quality of the knowledge obtained, and GP can be used to perform this transformation.

The data transformation can consist in a selection of the attributes relevant to the classification problem, a weighting of the attributes, in order to give each attribute a credit proportional to its importance in the final prediction; or it can consist in the construction of new predicting attributes by combining some of the original ones. GP also offers a wide range of possibilities in the classifier induction task. GP can be readily fit to the main components of a model extraction algorithm. GP is a search algorithm, which can be employed to search in the space of classifiers to find the best one; the fitness function of GP is used as the preference criterion that drives the search process; and the flexibility of the GP representation allows it to employ many different kinds of models, with decision trees, classification rules, and discriminant functions being the most common choices.

A decision tree is composed of internal and leaf nodes. Since GP individuals are commonly encoded in a tree-like fashion, the application of GP to the evolution of decision trees is obvious. Each individual in the population can represent a decision tree classifier. A rule has two parts, the antecedent and the consequent. The antecedent contains a combination of conditions for the predicting attributes, and the consequent contains the predicted class. Usually, a condition is composed of a binary relational operator (=, <=, >, <, >=, <=) comparing the value of an attribute with a constant or another attribute, although more complex conditions can easily be incorporated since GP individuals can employ any kind of operator.

A discriminant function is a mathematical expression in which different kinds of operators and functions are applied to the attributes of a data instance that must be classified. The representation of a mathematical expression as a tree is again evident. The quality of classifiers must be measured by means of the fitness function. Usually, quality is based on accuracy, and it is often measured as the ratio between the number of correctly classified and the total number, but other possibilities exist. Several widely used metrics for measuring accuracy, like precision, support, confidence, recall, sensitivity, specificity, and others, are based on the confusion matrix in Table 1. The true positives (TP) are positive instances that are classified as positive, the false negatives (FN) are positive instances classified as negative, the false positives (FP) are negative instances classified as positive, and the true negatives (TN) are negative instances classified as negative.

<table>
<thead>
<tr>
<th>Predicted:</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real: Positive</td>
<td>True Positives (TP)</td>
<td>False Negatives (FN)</td>
</tr>
<tr>
<td>Negative</td>
<td>False Positives (FP)</td>
<td>True Negatives (TN)</td>
</tr>
</tbody>
</table>

Figure 4: Confusion matrix

Furthermore, other performance criteria can be taken in consideration in the fitness function, like novelty, interestingness, utility, or interpretability. Occam’s razor is a reasoning which is applied very often not only in classification, but virtually in every area of computer science. This principle
basically states that when choosing between models with the same accuracy, less complex models should be preferred.

This preference is due to several reasons, like enhanced interpretability and better generalization ability. Classification performance can be enhanced employing several classifiers instead of just one. This is the basic idea of ensemble classifiers. Ideally, different base classifiers in an ensemble capture different patterns or aspects of a pattern embedded in the whole range of data, and then through ensemble, these different patterns or aspects are incorporated into a final prediction. Since this technique consists in the combination of the classification output of several classifiers, it can be regarded as a form of post processing of the models extracted.

Two main issues, which are how to generate diverse base classifiers and how to combine base classifiers have to be addressed, and GP can be applied to both issues. The basic approach for using GP to generate diverse base classifiers consists in dividing the dataset into several subsets, where each of the subsets is used in an independent run of GP to construct each of the base classifiers. The application of GP to the combination of base classifiers is very similar to the construction of new features: the outputs of each of the base classifiers are at the leaf nodes of a tree GP individual, and these predictions are combined by means of the operations encoded in the nonterminal nodes of the tree.

![Diagram showing classification tasks where GP can be applied](image)

**Figure 5:** Shows the classification tasks where GP can be applied

In many problems, there are several goals which have to be optimized simultaneously. These goals are often conflicting, so that the optimization of one of the performance measures implies an unacceptably poor performance for other measures. For example, when feature selection is performed, we want to minimize the number of attributes employed and maximize the classification rate, but
higher accuracy can usually be achieved when more features are employed. A similar situation arises with instance selection, where a subset of the training set is selected in order to speed up the learning process, without compromising classification rates. Accuracy and interpretability of classifiers are also conflicting goals; generally, the most accurate classifiers are the most complex and difficult for human beings to understand. In these situations, a tradeoff solution must be sought, combining suboptimal but acceptable values for all performance measures. In this kind of problem, known as multi objective optimization (MO) problems, there is usually not a single solution, but instead a set of equivalent non dominated solutions, composed of all the solutions where it is not possible to enhance some objectives without degrading some others. EAs can be applied to MO problems easily and suitably, since different individuals in the population can search for different solutions in parallel. When EAs are applied to this kind of problem, the term employed is multi objective optimization evolutionary algorithm.

V. Advantages & Disadvantages
The application of GP to classification offers some interesting advantages, the main one being its flexibility, which allows the technique to be adapted to the needs of each particular problem. GP can be employed to construct classifiers using different kinds of representations, e.g., decision trees, classification rules, discriminant functions, and many more. GP can be useful not only for inducing classifiers, but also for other preprocessing and post processing tasks aimed at the enhancement of classifiers. In fact, GP usually performs an implicit process of feature selection and extraction.

Like genetic algorithm, genetic programming is domain-independent problem solving method. The fact that these stochastic genetically inspired algorithms perform a global search and are robust can be observed as both their advantage and their disadvantage, depending on the type of problem that is being solved. Some functions (e.g. non differentiable or with local minima) would be suitable for use of genetic programming. Since genetic programming is a week search method, if problem has specialized solving method, it may give better results than genetic programming. Genetic programming algorithm operates with populations of individuals that can encode simple computer programs, functions, solutions to our problem etc. Similarly as in the case of genetic algorithms, the efficiency of the evaluation function greatly impacts the efficiency of the whole algorithm and therefore as well application of genetic programming. For that reason, it is important to implement fast evaluation of individuals.

VI. Conclusion
This paper presents a survey of GP for classification. We begin by providing a brief analysis of the cardinal points in the two areas concerned: GP and classification. This provides us with the background context needed to understand the works reviewed, and serves as a guideline to categorize and sort relevant literature. Genetic programming is much more powerful than genetic algorithms. The output of the genetic algorithm is a quantity, while the output of the genetic programming is a computer program. In essence, this is the beginning of computer programs that program themselves. Genetic programming works best for several types of problems. The first type is where there is no ideal solution. In this case genetic programming will find a solution that attempts to compromise and be the most efficient solution from a large list of variables. Furthermore, genetic programming is
useful in finding solutions where the variables are constantly changing. The solutions found by GP would be different from one another when the variables change.

VII. References