Semi-supervised Genetic Programming for Classification

Filipe de L. Arcanjo*
Universidade Federal de Minas Gerais (UFMG), Brazil
filipe@dcc.ufmg.br

Gisele L. Pappa
Universidade Federal de Minas Gerais (UFMG), Brazil
glpappa@dcc.ufmg.br

Paulo V. Bicalho†
Universidade Federal de Minas Gerais (UFMG), Brazil
p.bicalho@dcc.ufmg.br

Wagner Meira Jr.
Universidade Federal de Minas Gerais (UFMG), Brazil
meira@dcc.ufmg.br

Altigran S. da Silva
Universidade Federal de Manaus (UFAM), Brazil
alti@dcc.ufam.edu.br

ABSTRACT

Learning from unlabeled data provides innumerable advantages to a wide range of applications where there is a huge amount of unlabeled data freely available. Semi-supervised learning, which builds models from a small set of labeled examples and a potential large set of unlabeled examples, is a paradigm that may effectively use those unlabeled data. Here we propose KGP, a semi-supervised transductive genetic programming algorithm for classification. Apart from being one of the first semi-supervised algorithms, it is transductive (instead of inductive), i.e., it requires only a training dataset with labeled and unlabeled examples, which should represent the complete data domain. The algorithm relies on the three main assumptions on which semi-supervised algorithms are built, and performs both global search on labeled instances and local search on unlabeled instances. Periodically, unlabeled examples are moved to the labeled set after a weighted voting process performed by a committee. Results on eight UCI datasets were compared with Self-Training and KNN, and showed KGP as a promising method for semi-supervised learning.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning Concept Learning; I.5.2 [**Design Methodology**]: Classifier design and evaluation

General Terms

Algorithms

*Undergraduate student in Computer Science

†Undergraduate student in Computer Science

GECCO'11, July 12–16, 2011, Dublin, Ireland.

Keywords

semi-supervised learning, genetic programming, transduction, classification

1. INTRODUCTION

Learning from unlabeled data provides innumerable advantages to a wide range of applications, including image and video processing [23], computational linguistics [11], and speech recognition [22], among others. Unsupervised learning often does the job, but it may be better to employ a semi-supervised learning strategy to effectively grasp the implicit models from the data. Semi-supervised learning is characterized by building models from a few labeled and a great number of unlabeled examples [5], and is becoming very attractive for scenarios such as the Web, where huge amounts of unlabeled data are available.

Semi-supervised learning methods may perform *inductive* or *transductive* learning [25]. When inductive learning takes place, the classifier learns a function f_i from the training data, and f_i is expected to be a good predictor for future data (test data). In transductive methods, in contrast, the function f_t — which is also learned from the training set — has as its main objective to be a good predictor of training *unlabeled* instances. Hence, the function f_t is not expected to generalize for future data.

In this sense, transductive learning could be considered as a simple case of inductive learning. However, there is no consensus in the literature, and the definition given above is based on [25]. Despite that, transductive methods are useful in contexts where the "whole universe" of data is known, and the main objective is to learn labels for unlabeled instances. Examples of these scenarios include record deduplication and web-page classification, among others.

Regardless of being inductive or transductive, there are three common assumptions on which semi-supervised methods rely on [5]. The first is the smoothness assumption, which states that dense regions of the problem search space may be modeled by smooth functions. An immediate consequence of this assumption is that two near points in a dense region should be associated with similar classes. The second assumption is that the boundaries between dense regions in the search space are not dense, helping to separate dense regions. The third assumption is that, despite the high dimensionality of the data, there are dense regions in the search

space that are also detectable considering less dimensions. It is worth noting that most of the existing methods rely on just one or at most two of these assumptions, which might explain their poor performance on some datasets.

Although evolutionary methods have been successfully applied to solve classification problems involving supervised learning [10, 18, 21], their use in semi-supervised learning is still understudied [8]. In this direction, this paper proposes a novel Genetic Programming method that takes into account the three aforementioned assumptions of semi-supervised learning.

More specifically, the main characteristics of the GP method proposed, and from now on referred as KGP (as some of its features implement a KNN-like strategy), are: (i) it is a semi-supervised method, which receives as input a set of labeled and a set of unlabeled examples, (ii) it performs transductive learning, and hence works only with a training set, (iii) the labels of unlabeled examples are not assigned by a single individual, but instead by a committee of classifiers, (iv) from time to time, unlabeled examples are classified and moved to the labeled set, and (v) the decisions of committee members are weighted according to how well they perform in the neighborhood of the unlabeled examples.

Another interesting feature of KGP is that, in contrast with conventional semi-supervised methods that create a new model from the updated training set, it evolves the model (in our case a committee of classifiers) together with new labeled instances. Hence, during the evolution process, every time a new unlabeled instance is labeled by the set of classifiers, this instance is used to improve the model.

A key issue for semi-supervised learning methods is how to evaluate them. Specifically, we did not find, by definition, a truly transductive method to be used as a baseline. Transductive SVM [14], which is commonly regarded as being transductive, does not exactly fit in the definition we adopt [25]. Indeed, as pointed out, this method creates models that should generalize to new data, and hence uses training and test sets. KGP, in turn, requires only a training set. Thus, we introduce a methodology to analyze transductive methods, and test KGP in a set of eight UCI datasets [19].

Results were compared with the well-known self-training algorithm [5], and also with a supervised algorithm, since we know the labels for all training examples, including unlabeled ones. In this case, KGP's evaluation may consider the number of examples correctly classified when compared to the ground-truth, as well as an analysis of the strengths and weaknesses the KGP semi-supervised learning strategy offers [24].

The remainder of this paper is organized as follows. Section 2 describes related works in the areas of GP for classification and semi-supervised learning. Section 3 introduces KGP, our semi-supervised transductive genetic programming algorithm, KGP. Section 4 reports experimental results, while Section 5 draws some conclusions and discusses future works.

2. RELATED WORK

Evolutionary computation has been used to generate a great variety of models using supervised learning (i.e., problems where the labels of training instances are known), including rules, trees and mathematical functions [1, 15, 21]. Here we follow this last approach, and generate mathematical functions to solve classification problems, but with one main difference: we work within a semi-supervised framework.

Semi-supervised learning techniques have become increasingly popular in the last years. In fact, since the communities of natural language processing and text classification showed their interest in automatically labeling data to improve learning, there has been a fast growth in the development of methods that work with both labeled and unlabeled data [17, 20]. The methods most disseminated in this area were initially the self-training and co-training algorithms [4]. Co-training has peculiarities, as it employs multi-view learning, and requires differentiated input data (i.e., data coming from different data views). Self-training, in contrast, was first proposed in the seventies, but it is still successfully used in many contexts [20, 24].

Self-training works as a wrapper, in the sense that it requires another learning algorithm. This approach has flexibility as its main strength. Performance is, however, limited by the fact that the models used in each iteration are obtained independently. The underlining method has no knowledge of how its being used and therefore cannot employ the models obtained in iterations $i_1, i_2, ...i_k$ in any way to obtain a new one in i_{k+1}. The method proposed here, in contrast, works with a population of classifiers that evolves over time. At each step, models are built not only from the data but also from the models of previous iterations.

Usually, semi-supervised algorithms, including self-training and co-training, work under one or more of the three aforementioned assumptions [5]: (i) the smoothness assumption, which states that label functions are smoother on high-density regions than in low-density regions, and hence points separated by a high-density path are likely to be close; (ii) the cluster or low-density separation, which states that decision boundaries should lie on low-density regions; and (iii) the manifold assumption, which says that high-dimensional data lie roughly on a low-dimensional manifold.

The Transductive SVM [14], for instance, is based on the assumption of low-density separation, and works by moving the decision boundary away from the unlabeled points. It does not fit, however, the definition of transductive learning adopted here [25], as the algorithm produces a model which is expected to generalize. The first assumption is followed by generative methods, including those that estimate the conditional density $p(x|y)$ of the data, where x is the attribute vector and y the class, and use the information coming from $p(x)$ to help this estimation. Most of these are also not transductive. Note that methods based on assumptions (i) and (ii) are local, while methods based on (iii) are global.

KGP addresses these three assumptions, as showed in Section 3. It also explores the powerful global search of GP to ensure that we have classifiers dealing with attributes dependencies, and that they are specialized into different regions of the example space. By using a committee to predict labels of examples we allow classifiers to be general enough when learning from labeled examples but also local enough to classify neighbour examples accordingly.

Concerning genetic programming algorithms, they are well-known for solving classification problems under a supervised context, but methods following semi-supervised approaches are still underexplored. Two recently published papers have presented contributions in this direction. In

[12] the authors propose a new semi-supervised clustering method, also known as constrained clustering. In the case of semi-supervised clustering, prior background knowledge is often introduced using constraints, which are based on instances relationships or spatial contiguity. [12] works with instance-level constraints, which restrict memberships of pairs of instances. They use a hybrid genetic-guided semi-supervised clustering algorithm, named Cop-HGA, to find clusters while obeying imposed constraints. Something similar was proposed in [9].

In [8], in contrast, a new inductive method called AGP was introduced for dealing with semi-supervised learning classification. It employs Active learning [6], a type of data sampling technique where, instead of selecting a subset of random examples to train a classifier, a subset of the most informative examples is selected. These examples are then given to a user to label, and added to the set of labeled training examples.

AGP is based on a committee of classifiers that chooses the best example to be labeled. However, it does not apply self-learning, i.e., the labeled set does not grow over time and remains static. It is also not transductive, as the result of the learning process is a single classifier that is expected to label instances of a training set. KGP does both transductive and self-learning, but not active learning. As shown later, the combination of KGP with an active learning approach may lead to even better results. It also employs committees in a rather different way, as the votes of each individual in the population are weighted.

3. THE KGP METHOD

This section describes KGP, a semi-supervised transductive genetic programming algorithm. One of the main strengths of KGP is to take into account the three basic assumptions of semi-supervised algorithms.

The smoothness assumption is addressed by the *weighted* voting process, which weights classifiers according to their performance on the neighborhood of the unlabeled example. This process guarantees that classifiers consider both the global (when learning from labeled data) and local contexts of data (when labeling unlabeled data). The low-density separation is covered by the *voting process* itself. Individuals with very high confidence for classifying an unlabeled instance have a high probability of lying on different dense regions (clusters) of the search space. Those with low probabilities are likely to be closer to the class boundaries, and that is why the voting process is important. By using the voting process, KGP almost ignores classifiers close to the class borders. Finally, the manifold assumption is indirectly addressed when the classifiers use only a subset of the attributes to generate classification functions, creating simpler functions in a reduced dimensional space.

Moreover, KGP evolves the model (in our case a committee of classifiers) together with new labeled instances. Hence, during the evolution process, every time a new unlabeled instance is labeled by the set of classifiers, this instance is used to improve the model previously created.

The next sections give an overview of the algorithm, detailing the individual representation, fitness function, genetic operators and the weighted voting process.

3.1 Overview

As a semi-supervised algorithm, KGP takes as inputs two sets of instances, namely U and L. The former is called *unlabeled set*, and consists of a series of vectors $\vec{x}_i \in \mathbb{R}^n$. The latter, which is the *labeled set*, contains pairs of the form $(\vec{x}_i, y_i)$, where $y_i \in \{-1, +1\}$ represents the class of the example. The current version of the method supports only binary classification, but can be easily extended to deal with multi-class problems following a one-against-al approach similar to that of SVM [13]. KGP's main goal is to correctly predict the classes of the instances in U and transfer them to L. This process is done incrementally.

Algorithm 1 shows an overview of the method, whose underlying process is the same as any supervised GP classification method. It begins by initializing a random population of n_i individuals, using the well-known techniques of *grow* and *full* [3]. Each individual consists of a mathematical expression and represents a classifier. In order to classify an individual as belonging to class $+1$ or -1, a threshold is applied to the output of the expression.

It then enters a loop that iterates for a maximum number of generations g. At each iteration, the individuals in the current population are evaluated using instances in L. That is, we use each individual i to obtain a map $m_i : L \rightarrow \{-1, +1\}$ that assigns a label to each of those instances. This *predicted* label may or not be equal to the one in L.

The success of an individual in correctly classifying labeled examples is taken into consideration to compute its fitness. The fitness value determines the chances of an individual being chosen by tournament selection [3]. Chosen individuals undergo crossover and mutation. Their offspring then becomes part of a new population, which will replace the current one. We also employ elitism, by copying the e most fit individuals to the new population.

Every τ generations, a labeling round takes place, and a committee of individuals votes to determine the class of each unlabeled instance. These votes are then weighted considering a confidence metric, which reflects how well each classifier performs in the labeled neighbourhood of the unlabeled example. Votes and weights are combined into a final prediction for the instance's class. At the end of this process, only the ℓ instances with higher votes' absolute values are moved to U. After g generations, the method guarantees that U is empty.

In the next subsections, the details of our method are presented by describing the steps of Algorithm 1.

3.2 Individual Representation

Each individual in KGP is a classifier, represented by a binary tree corresponding to mathematical expression. Four basic operations are allowed as function nodes: protected division [3], multiplication, addition, and subtraction. Terminals consist on integer constants from -9 to 9 and attributes from vectors that describe each instance. Figure 1 shows an example, where a and b are attributes from the dataset being considered. Individuals are bound to a maximum height h, and genetic operators are designed in such a way that this value is never exceeded.

When classifying an example, we go through the nodes recursively, replacing the attributes in the leaves by their values, and a value $z(\vec{x})$ is obtained. In general, we cannot assure that $z(\vec{x})$ will be bounded, which is a consequence of the fact that no restrictions are imposed as to what types of individuals can be generated.

Algorithm 1: KGP(L, U)

```
1: Build a random initial population;
2: for j = 1, 2, ..., g do
3:    Evaluate each individual i using the examples in L;
4:    Compute the fitness of each i according to its
      predictions for L's instances;
5:    Perform crossover and mutation according to user
      defined probabilities;
6:    Update the current population;
7:    if j ≡ 0 (mod τ) then
8:       Let S be an empty list;
9:       for x⃗ ∈ U do
10:         Evaluate each individual i in x⃗.
11:         Compute the set NN(k, x⃗) of the k nearest
            labeled neighbors of x⃗;
12:         Evaluate each i in N and compute its weight;
13:         Compute the final vote for v(x⃗) combining
            evaluation results and weights;
14:         Append v(x⃗) to S;
15:      end for
16:      Sort S by the absolute value of each element;
17:      Label the instances in U associated with the top ℓ
         values in S;
18:      Transfer these ℓ instances to L;
19:   end if
20: end for
```

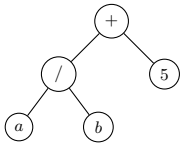


Figure 1: A simple individual tailored to classify instances of the form $\vec{x} = (a, b)$

The fact that no bounds can be obtained forbids any kind of normalization. To counter that, we apply a sigmoid to each z obtained, and consider its value instead. The sigmoid takes any real value and maps it on the open real interval $(0, 1)$:

$$f(\vec{x}) = \frac{1}{2}\tanh\left(\frac{z(\vec{x})}{10}\right) \tag{1}$$

A threshold t, whose value has to be obtained experimentally, is then applied to f. As the sigmoid returns a value within $(0, 1)$, t needs to be a value in this interval, instead of an arbitrary real number. Finally, we obtain the predicted class y' from $f(\vec{x})$:

$$y'(\vec{x}) = \begin{cases} +1, & \text{if } f(\vec{x}) \geq t \\ -1, & \text{if } f(\vec{x}) < t \end{cases} \tag{2}$$

3.3 Fitness Function

The evaluation of the individuals (classifiers) is based only on the set of labeled examples, and hence employs the traditional metrics used to evaluate supervised learning methods. Considering a problem with two classes, there are four possible combinations of real and predicted classes, as shown in Table 1. We take the harmonic mean of two rather common measurements used to assess classifier quality: *true positive rate* and *true negative rate*. For binary classification problems, they are defined as follows:

		Real	
		-1	+1
Predicted	-1	α	$\overline{\alpha}$
	+1	β	$\overline{\beta}$

Table 1: Confusion matrix and the four possible combinations for predicted and real class.

$$\text{true positive rate} = \frac{\alpha}{\alpha + \overline{\beta}} \tag{3}$$

$$\text{true negative rate} = \frac{\overline{\beta}}{\overline{\beta} + \overline{\alpha}} \tag{4}$$

in which α, $\overline{\alpha}$, β, $\overline{\beta}$ denote the number of instances in each cell of the aforementioned table. The obtained value is a real number between 0 and 1. It is used in conjunction with tournament selection [3] to choose individuals that will undergo crossover and mutation.

3.4 Genetic Operators

The classifiers are modified according to the traditional crossover and mutation operators. Crossover takes two individuals A and B as input and outputs a pair of new individuals A' and B'. A' and B' are created by swapping their respective selected subtrees. Following a procedure suggested by Koza [16], we introduce a bias to this operator, assuring that terminal nodes are selected 10% of the time.

Mutation chooses a node of an individual randomly, and replaces it by a subtree generated randomly.

3.5 Weighted Voting Process

As previously mentioned, every τ generations a labeling round takes place. During it, a committee of individuals composed by the whole population votes to determine the class of each unlabeled instance. The votes of all individuals are combined using vote weights. These weights are computed based on two main assumptions:

1. Individuals do not have equal performance over the whole feature space, that is, they tend to be better in certain regions and worse in others.

2. An instance $\vec{x}$ is more likely to belong to the class which is more common among its neighbors. Thus we should reward classifiers that predict the most common class in the neighborhood and penalize the ones which do not.

During the voting process, first the k nearest *labeled* neighbors $\text{NN}(k, \vec{x})$ of each *unlabeled* instance $\vec{x} \in U$ are identified. For that matter, any distance function will do. We currently use the Euclidean distance for reasons of simplicity and broad usage in a diverse range of applications. In scenarios such as text classification, other metrics such as the cosine distance may be employed [2].

Next, weights are computed taking into consideration the accuracy of the classifier in predicting the classes for the already labeled instances in $\text{NN}(k, \vec{x})$.

Table 2: Datasets from the UCI repository

Dataset	Instances	Attributes	Class Distribution
diabetes	768	9	268/500
ionosphere	351	35	126/225
magic	19020	11	6688/12332
musk	6598	167	1017/5581
spam	4601	58	1813/2788
spectf	267	45	212/55
survival	306	4	225/81
wdbc	569	31	357/212

The voting process also takes into account a number p, which is simply the percentage of neighbors belonging to class -1, as described in Equation 5. It is used to penalize classifiers that go against the most popular class in the neighbourhood, thus satisfying the second assumption we have adopted.

$$w_i(\vec{x}) = \text{accuracy}(i, \text{NN}(k, \vec{x})) \cdot \begin{cases} p, \text{ if } y'(\vec{x}) = -1 \\ 1-p, \text{ if } y'(\vec{x}) = +1 \end{cases} \quad (5)$$

The voting decision on the label to be assigned to an instance $\vec{x}$ is taken by computing the weighted average of the classes predicted by each individual i from the population P for $\vec{x}$:

$$v(\vec{x}) = \frac{\sum_{i \in P} w_i(\vec{x}) \cdot y'_i(\vec{x})}{\sum_{i \in P} w_i(\vec{x})} \quad (6)$$

Notice that $v(\vec{x}) \in [-1, +1]$ is a real number. We take its absolute value as a measurement of how confident the committee is of its choice. As shown in Algorithm 1, the obtained $v(\vec{x})$ values are ranked by confidence.

The top ℓ instances in this rank are labeled by rounding $v(\vec{x})$ to either $+1$ (if positive) or -1 (if negative) and transferred from U to L. The value of ℓ is chosen according to τ, in such a way that after g generations, all instances are labeled:

$$\ell = \left\lfloor \frac{|U_0|}{r} \right\rfloor \quad (7)$$

where r is the amount of labeling rounds, that is, $\lfloor g/\tau \rfloor$, and $|U_0|$ denotes the amount of unlabeled instances initially present on the unlabeled set. It is worth noticing that $|U_0|$ might not divide r. In that case, the last labeling round will handle more than ℓ instances.

4. EXPERIMENTS

Evaluating semi-supervised methods is still a challenge, as there is no consensus as to how it should be done. Here, in order to validate the proposed technique, we work with eight datasets from the UCI repository [19], described in Table 2. As observed, all of them have two classes and only numerical attributes. Experiments were performed varying the size of the labeled set L. For each size, 20 random samples were extracted from the complete dataset. The results were compared with two other algorithms: self-training and KNN itself. KNN was chosen because, apart from providing a general good accuracy, its rationale is partially similar to the one used by KGP. Self-training was chosen for being one of the most popular approaches to semi-supervised learning [5]. KNN works by assigning to each instance the most popular class amoung its K nearest neighbours. Self-training works as follows. It first creates an initial model from the labeled training set, and then uses this model to label examples from the unlabeled set. After that, according to a confidence metric provided by the enclosed classifier C , the algorithm adds the m best classified examples in terms of confidence to the labeled training set. This enhanced training set is then used to create an updated model. The process continues until all examples are labeled. Note that self-training has the advantage of working as a wrapper, i.e., it may employ any classification algorithm in its core. Here we have selected KNN for that matter, for the same reasons expressed above. This combination of self-training and KNN will be refered to as Self-KNN hereafter.

The GP parameters were set in a preliminary phase, and their values are reported in Table 3. For comparison purposes, we choose for KNN and Self-KNN the same k as used by KGP, that is, they look into neighbourhoods of the same size. We also assure Self-KNN performs as much iterations as KGP's labeling rounds.

Considering the parameters described in Table 3, it is interesting to analyze some statistics collected during the execution of KGP. Figure 2 shows six graphs obtained for the datasets ionosphere (10% of labeled instances) and spam (50%). All results are averages over 20 runs with different random seeds but the same data sample.

In the first column, we observe the behavior of the individuals' fitness during evolution. Note that the algorithm does not converge. Although the average fitness improves over time, there is still a fair diversity in the last generation. This is important considering the committee strategy adopted here.

The graphs in the second column report the confidences associated with each final vote $v(\vec{x})$ made by the classifiers. As evolution goes on, the average confidence decreases. This can be explained by an association of two factors. The first is a decay in the quality of the model, which is a consequence of incorrectly labeled instances moved to the labeled training set during execution. The other is the fact that hard to classify instances and outliers tend to accumulate over time. The behavior is consistent with the graphs in the third column, which contain the accuracy of labeled instances per labeling round. It also decreases with time.

Currently, a fixed number of examples is labeled at each round. One way to try to solve this problem is to add to the labeled training sets only examples labeled with a confidence higher than a threshold, which may be relaxed at the final generations, as at this point only the most difficult examples remain in the dataset. In some way, we have already tried to increase or reduce this number of examples when setting the parameters for the number of labeling rounds. However, we intend to study it more deeply in future work.

The results obtained by KGP, Self-KNN, and KNN using different sample sizes for labeled sets are showed in Table 4. Two metrics were used to evaluate the results: the macro-F_1 and the predictive accuracy. Macro-F_1 is a variation of f-measure, which calculates the harmonic mean of precision (fraction of examples correctly assigned to a class) and recall (true positive rate, which is the fraction of examples from a class correctly classified). Macro-F_1 measures the classification effectiveness taking into account the class distribution of the dataset. Table 4 presents the average ($\pm$) 95% confidence intervals of these metrics.

In order to ensure statistical significance, the results of KGP were compared to the two baselines using a statistical

Table 3: GP Parameters

Parameter	Value
Number of Generations (g)	50
Number of Individuals (n_i)	100
Maximum tree depth (h)	5
Tournament size	3
Mutation rate	0.05
Crossover rate	0.85
Elitism (e)	5
K	5
Labeling rounds (τ)	every 5 generations
Class threshold (t)	0.5
Pr. of using grow	0.5
Pr. of grow choosing a non-terminal	0.5

t-test with 95% confidence. The results of these tests are presented in Table 4 through three symbols: ▲ denotes a significant positive variation, ● a non significant variation and ▼ a significant negative variation. Note that these symbols only appear in the column of the baselines, and refer to how the method compares with KGP. For instance, the macro-F_1 of ionosphere in the column Self-KNN is followed by a ▼, meaning that Self-KNN obtains values of macroF_1 statistically worse than those obtained by KGP.

Considering the 24 experiments performed (8 datasets × three variations of data sample size) and the values of accuracy, KGP achieves statistically better results than Self-KNN in 8 cases, and statistically worse results in 7 cases. Note that in the dataset ionosphere these gains were the most expressive, being of 15% and 8.5% for samples of 10% and 30%, respectively. For spam, KGP is also better than Self-KNN in all three sample sizes, while for wdbc it is statistically significant for sample sizes of 50% and statistically the same as Self-KNN for the other cases. All other results obtained by the two methods are statistically equivalent.

There were three datasets in which statistically inferior results were obtained by KGP: diabetes, magic and musk. The reason why this happens might be related to the way noise builds up as the classification process takes place. In Self-KNN, for instance, new models are obtained at each iteration considering only the labeled data. KGP, on the other hand, uses as classifiers individuals from previous generations, i.e., while Self-KNN rebuilds the whole model, KGP updates it. We believe this may increase the rate at which noise builds up in the set of labeled instances over time for some datasets, and intend to investigate this issue further in future works. This hypothesis is consistent with the fact that the difference between KGP and the other approaches in thoses cases increases as the unlabeled set gets bigger. Furthermore, this hypothesis is also backed up by the fact that magic and musk are the largest datasets used in our experiments. Further evidence is provided by the graphs on the second column of Figure 2, which refer to datasets ionosphere and spam. Spam is much larger than ionosphere and has a much steeper curve when it comes to confidence even with a larger percentage of labeled instances.

Regarding the macro-F_1 results, they are also very similar to accuracy ones. The only difference is that now KGP is statistically better in 9 cases and statistically worse in 6. Analyzing the results of KGP compared to KNN, we notice that KGP achieves statistically better results in 12 cases, while KNN is better than Self-KNN in six. It is important to emphasize that the KNN run here is inductive, and learns a model from a sample of the training data and then applies it to the remaining example (test set).

After seeing the results and creating hypothesis that may improve the accuracy of KGP, we investigate the reasons behind classifier misclassifications. Is the problem the voting scheme or the models independence? Figure 3 shows the classification of four specific instances from the dataset spam, with 20% of labeled instances. The first two are correctly classified, and the last two are misclassified. Each line in the histogram represents a single classifier. Heights correspond to the product $w_i(\vec{x}) \cdot y'_i(\vec{x})$, that is, classifiers' contributions to the final vote. The decision of the class of the example is taken based on a weighted average over all classifiers, as previously shown in Equation 6. Note that in Figures 3 (a) and (b), it is not difficult to see that the confidences closer to +1 or -1 dominate the graph.

An interesting case is Figure 3 (c), where, although there is a majority of classifiers saying it should be -1, it is misclassified as +1. It happens because the weights associated with the classifiers that predicted the wrong class are higher. The given instance is probably located in a region of space rich in examples of class +1. One possibility to avoid such mistakes would be to consider new approaches to compute weights. Figure 3 (d) shows the case of a genuine error, which might have occurred because this instance is an outlier or is too close to the class boundaries.

KGP's computational performance is reasonable for most of the datasets employed in this study. For spectf, which is the smallest one, a single execution takes less than a second. For magic, the largest, running with 10% of labeled instances takes 1 minute and 45 seconds. These results were obtained on a dual quad-core machine with hyper-threading and 16GB of main memory. Most of the execution time is spent computing nearest neighbours and evaluating individuals. Performance improvements could be easily obtained by parallelizing individual's evaluations and making use of space partitioning techniques, such as KD-Trees [7].

5. CONCLUSION

This work introduced KGP, a semi-supervised genetic programming algorithm for classification. It was conceived based on the three assumptions on which semi-supervised learning relies on, and combines global information of labeled data with local information of unsupervised instances to evolve classification models.

KGP performs transductive learning, working only with a training set divided into labeled and unlabeled examples. As evolution goes on, labeling rounds take place, and a committee of classifiers performs a weighted voting to decide which unlabeled examples should be moved to the labeled training set. Finally, instead of recreating a whole classification model from scratch after new instances are added to the training set, it simply updates a model already being evolved. At the end of the evolutionary process, all examples are labeled.

Experiments with eight UCI datasets showed that KGP is competitive with Self-KNN and KNN. It presents significant gains in two datasets, and most of the time has a performance statistically equivalent to Self-KNN. During the analysis of the results, we identified a set of improvements which can lead KGP to better results. They involve, for instance, having a threshold to determine the number of unlabeled examples to be moved to the training set, instead of using the static approach Self-learning implements. Some active learning could also improve the GP performance at

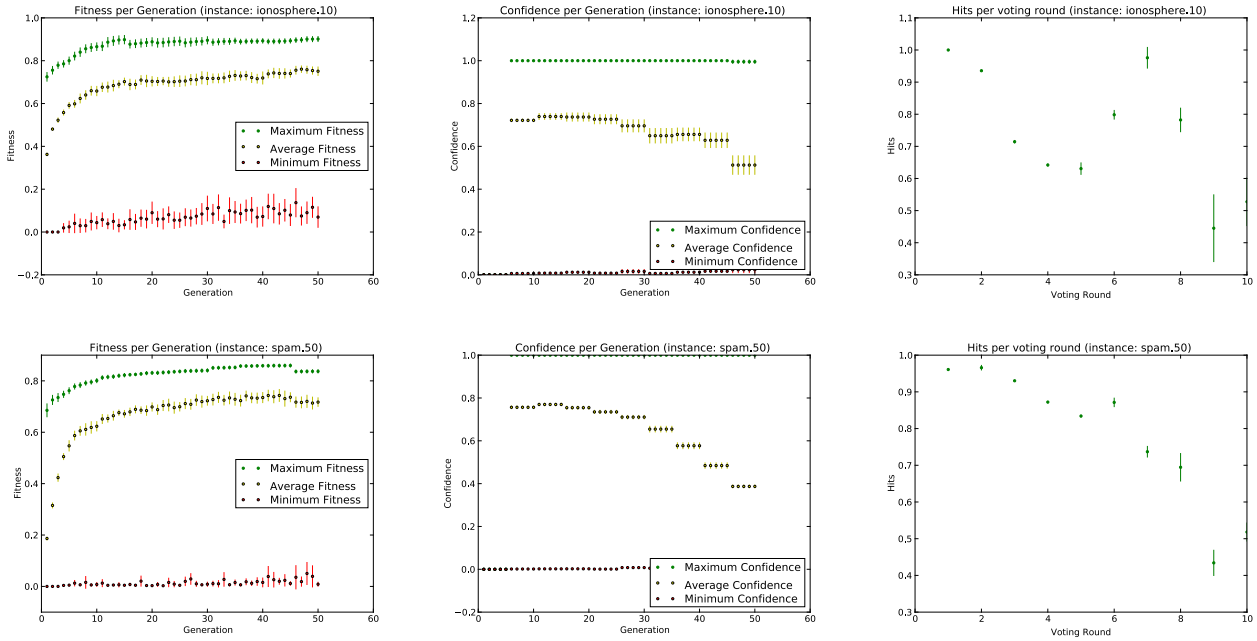
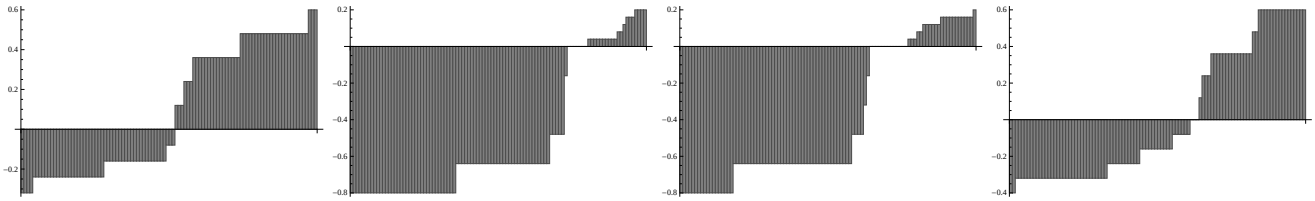


Figure 2: Some interesting GP statistics for datasets ionosphere with 10% of labeled examples and spam with 50% of labeled examples: first column shows values of fitness per generation, second column shows confidences during the voting rounds, and the third columns illustrates the percentage of examples in L correctly classified by the committee. Vertical dashes indicate 95% confidence intervals.



(a) Correctly classified +1 (b) Correct classified -1 (c) Misclassified +1 (d) Misclassified -1

Figure 3: The confidences of the committee in four possible cases: correctly classifying or missclassifying and example

the first generations, when the models are still not completely evolved.

The next step is to use KGP in a set of real world applications to corroborate the results obtained here. Potential applications include record deduplication and the creation of ground truth for Web databases, such as those coming from digital libraries or social networks. Finally, the algorithm can also be adapted to work within a inductive framework.

6. REFERENCES

[1] D. A. Augusto, H. J. C. Barbosa, and N. F. F. Ebecken. Coevolutionary multi-population genetic programming for data classification. In *GECCO*, pages 933–940, 2010.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology behind Search*. Addison-Wesley Professional, 2011.

[3] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, Jan. 1998.

[4] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. of the 11th Annual Conf. on Computational Learning Theory*, pages 92–100, 1998.

[5] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, 2010.

[6] D. A. COHN, L. ATLAS, and R. E. LADNER. Improving generalization with active learning. *Machine Learning*, 15(2):201Ű–221, 1994.

[7] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2010.

[8] J. de Freitas, G. L. Pappa, A. S. da Silva, M. A. Gonçalves, E. S. de Moura, A. Veloso, A. H. F. Laender, and M. G. de Carvalho. Active learning genetic programming for record deduplication. In

Table 4: Results of macro-F_1 and micro-F_1 obtained by KGP

Dataset	Labeled	MacroF$_1$			Accuracy		
	(%)	KGP	Self-KNN	KNN	KGP	Self-KNN	KNN
diabetes	10	0.6601 ± 0.0127	0.6616 ± 0.0134 ●	0.6777 ± 0.0086 ▲	0.7055 ± 0.0094	0.7074 ± 0.0101 ●	0.7177 ± 0.0074 ▲
	30	0.7693 ± 0.0042	0.7733 ± 0.0064 ●	0.7716 ± 0.0050 ●	0.7791 ± 0.0053	0.7999 ± 0.0059 ▲	0.7964 ± 0.0046 ▲
	50	0.8309 ± 0.0025	0.8441 ± 0.0048 ▲	0.8430 ± 0.0057 ▲	0.8298 ± 0.0043	0.8605 ± 0.0043 ▲	0.8590 ± 0.0051 ▲
ionosphere	10	0.7655 ± 0.0199	0.6620 ± 0.0012 ▼	0.7270 ± 0.0310 ▼	0.7766 ± 0.0195	0.6766 ± 0.0011 ▼	0.7457 ± 0.0293 ●
	30	0.8918 ± 0.0062	0.8280 ± 0.0237 ▼	0.8562 ± 0.0146 ▼	0.8990 ± 0.0061	0.8364 ± 0.0237 ▼	0.8645 ± 0.0146 ▼
	50	0.9410 ± 0.0038	0.9075 ± 0.0057 ▼	0.9097 ± 0.0045 ▼	0.9454 ± 0.0035	0.9135 ± 0.0055 ▼	0.9156 ± 0.0043 ▼
magic	10	0.7889 ± 0.0017	0.7830 ± 0.0019 ▼	0.7728 ± 0.0019 ▼	0.8122 ± 0.0013	0.8056 ± 0.0014 ▼	0.7995 ± 0.0016 ▼
	30	0.8402 ± 0.0010	0.8436 ± 0.0010 ▲	0.8389 ± 0.0008 ▼	0.8556 ± 0.0010	0.8584 ± 0.0009 ▲	0.8558 ± 0.0007 ●
	50	0.8807 ± 0.0013	0.8910 ± 0.0006 ▲	0.8889 ± 0.0007 ▲	0.8889 ± 0.0015	0.9011 ± 0.0006 ▲	0.8998 ± 0.0006 ▲
musk	10	0.8598 ± 0.0036	0.8735 ± 0.0039 ▲	0.8640 ± 0.0036 ●	0.9289 ± 0.0018	0.9382 ± 0.0017 ▲	0.9310 ± 0.0019 ●
	30	0.9019 ± 0.0020	0.9259 ± 0.0019 ▲	0.9262 ± 0.0027 ▲	0.9462 ± 0.0014	0.9624 ± 0.0010 ▲	0.9622 ± 0.0015 ▲
	50	0.9265 ± 0.0019	0.9529 ± 0.0019 ▲	0.9530 ± 0.0020 ▲	0.9585 ± 0.0013	0.9758 ± 0.0010 ▲	0.9758 ± 0.0010 ▲
spam	10	0.7594 ± 0.0049	0.7109 ± 0.0047 ▼	0.7047 ± 0.0042 ▼	0.7677 ± 0.0058	0.7251 ± 0.0047 ▼	0.7190 ± 0.0041 ▼
	30	0.8486 ± 0.0026	0.7908 ± 0.0026 ▼	0.7911 ± 0.0030 ▼	0.8500 ± 0.0028	0.8008 ± 0.0025 ▼	0.8004 ± 0.0029 ▼
	50	0.8914 ± 0.0015	0.8602 ± 0.0019 ▼	0.8582 ± 0.0016 ▼	0.8898 ± 0.0020	0.8663 ± 0.0018 ▼	0.8643 ± 0.0015 ▼
spectf	10	0.6585 ± 0.0101	0.6693 ± 0.0135 ●	0.6664 ± 0.0240 ●	0.7932 ± 0.0182	0.7893 ± 0.0247 ●	0.7812 ± 0.0122 ●
	30	0.7329 ± 0.0109	0.7281 ± 0.0160 ●	0.7276 ± 0.0121 ●	0.8387 ± 0.0090	0.8230 ± 0.0149 ●	0.8174 ± 0.0105 ▼
	50	0.8247 ± 0.0035	0.8114 ± 0.0092 ▼	0.8126 ± 0.0090 ▼	0.8923 ± 0.0017	0.8812 ± 0.0057 ▼	0.8801 ± 0.0066 ▼
survival	10	0.6544 ± 0.0135	0.6660 ± 0.0104 ●	0.6386 ± 0.0146 ●	0.7625 ± 0.0065	0.7620 ± 0.0037 ●	0.7482 ± 0.0111 ▼
	30	0.7476 ± 0.0123	0.7393 ± 0.0155 ●	0.7295 ± 0.0107 ▼	0.8171 ± 0.0078	0.8096 ± 0.0120 ●	0.8031 ± 0.0081 ▼
	50	0.8227 ± 0.0040	0.8137 ± 0.0072 ▼	0.8040 ± 0.0091 ▼	0.8676 ± 0.0027	0.8620 ± 0.0053 ●	0.8542 ± 0.0067 ▼
wdbc	10	0.9154 ± 0.0079	0.9074 ± 0.0078 ●	0.9016 ± 0.0072 ▼	0.9197 ± 0.0080	0.9120 ± 0.0078 ●	0.9075 ± 0.0069 ▼
	30	0.9439 ± 0.0037	0.9407 ± 0.0038 ●	0.9422 ± 0.0031 ●	0.9471 ± 0.0036	0.9445 ± 0.0036 ●	0.9460 ± 0.0029 ●
	50	0.9601 ± 0.0023	0.9593 ± 0.0027 ●	0.9590 ± 0.0026 ●	0.9625 ± 0.0022	0.9620 ± 0.0025 ●	0.9616 ± 0.0024 ●

IEEE Congress on Evolutionary Computation, pages 1–8, 2010.

[9] A. Demiriz, K. Bennett, K. P. Bennett, and M. J. Embrechts. Semi-supervised clustering using genetic algorithms ayhan demiriz. In *In Artificial Neural Networks in Engineering (ANNIE-99*, pages 809–814. ASME Press, 1999.

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

[11] C. Ginestet. Semisupervised learning for computational linguistics. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 172(3):694–694, 2009.

[12] Y. Hong, S. Kwong, H. Xiong, and Q. Ren. Genetic-guided semi-supervised clustering algorithm with instance-level constraints. In *GECCO '08: Proceedings of the 10th Annual Conf. on Genetic and Evolutionary Computation*, pages 1381–1388, 2008.

[13] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. In *IEEE Transactions on Neural Networks*, 2002.

[14] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 200–209, 1999.

[15] J. Kishore, L. Patnaik, V. Mani, and V. Agrawal. Application of genetic programming for multicategory pattern classification. *Evolutionary Computation, IEEE Transactions on*, 4(3):242 –258, Sept. 2000.

[16] J. R. Koza. *Genetic Programming: on the programming of computers by the means of natural selection*. The MIT Press, Massachusetts, 1992.

[17] B. Maeireizo, D. Litman, and R. Hwa. Co-training for predicting emotions with spoken dialogue data. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, ACLdemo '04, 2004.

[18] D. Muni, N. Pal, and J. Das. A novel approach to design classifier using genetic programming. *IEEE Transactions on Evolutionary Computation*, 8(2):183–196, Apr. 2004.

[19] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. *UCI Repository of machine learning databases*. University of California, Irvine, http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998.

[20] Z.-Y. Niu, D.-H. Ji, and C. L. Tan. Word sense disambiguation using label propagation based semi-supervised learning. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 395–402, 2005.

[21] M. Segond, C. Fonlupt, and D. Robilliard. Genetic programming for protein related text classification. In *GECCO*, pages 1099–1106, 2009.

[22] G. Tur, D. Hakkani-Tür, and R. E. Schapire. Combining active and semi-supervised learning for spoken language understanding. *Speech Communication*, 45(2):171 – 186, 2005.

[23] J. Wang, Y. Zhao, X. Wu, and X.-S. Hua. Transductive multi-label learning for video concept detection. In *Proceeding of the 1st ACM International Conference on Multimedia Information Retrieval*, MIR '08, pages 298–304, 2008.

[24] X. Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin Ű Madison, 2008.

[25] X. Zhu and A. B. Goldberg. *Introduction to Semi-supervised Learning*. Morgan and Claypool Publishers, 2009.