Simultaneous Generation of Prototypes and Features Through Genetic Programming

Mauricio García-Limón, Hugo Jair Escalante, Eduardo Morales, Alicia Morales-Reyes
Instituto Nacional de Astrofísica, Óptica y Electrónica
Computer Science Department
Luis Enrique Erro No. 1, Tonantzintla,
Puebla, 72840, Mexico
{mauricio.garcia,hugojair,emorales,a.morales}@inaoep.mx

ABSTRACT

Nearest-neighbor (NN) methods are highly effective and widely used pattern classification techniques. There are, however, some issues that hinder their application for large scale and noisy data sets; including, its high storage requirements, its sensitivity to noisy instances, and the fact that test cases must be compared to all of the training instances. Prototype (PG) and feature generation (FG) techniques aim at alleviating these issues to some extent; where, traditionally, both techniques have been implemented separately. This paper introduces a genetic programming approach to tackle the simultaneous generation of prototypes and features to be used for classification with a NN classifier. The proposed method learns to combine instances and attributes to produce a set of prototypes and a new feature space for each class of the classification problem via genetic programming. An heterogeneous representation is proposed together with ad-hoc genetic operators. The proposed approach overcomes some limitations of NN without degradation in its classification performance. Experimental results are reported and compared with several other techniques. The empirical assessment provides evidence of the effectiveness of the proposed approach in terms of classification accuracy and instance/feature reduction.

Categories and Subject Descriptors

I.5.2 [**Pattern recognition**]: Design Methodology—*Classifier design and evaluation, Feature evaluation and selection*

Keywords

Prototype Generation, Feature Extraction, NN Classifier, Genetic Programming, Pattern Recognition.

1. INTRODUCTION

Pattern classification is concerned with the construction of predictive models that can map instances to a predefined set of classes or labels. This is one of the main problems in machine learning and pattern recognition and it has been a widely studied field [9]. Among the variety of classification models proposed so far, nearest-neighbors (NN) methods are among the most popular ones [23]. This type of classifiers store all training instances and when classifying a new object, they compare it to all cases in the training set. The label for the test object is obtained by processing the labels associated to its nearest training instances.

Although NN methods are highly effective, and easy to implement/understand, they present some issues that hinder their application to certain types of problems. First, NN methods have interesting convergence properties as the size of the training set increases, however, storing large training sets consumes significantly more resources. Second, when classifying a new test object it has to be compared to every instance in the training set, which becomes computationally expensive for large data sets. Third, similar to other classification models, NN techniques are susceptible to the curse of dimensionality. Finally, another important issue of NN methods is that they are sensitive to noisy instances (e.g., misclassified instances and outliers).

These weaknesses of NN methods have been tackled with different approaches, mainly with two variants of instance reduction, namely, instance selection [7] and prototype generation [21]. Instance reduction techniques select a subset of instances to substitute the original training set. Prototype generation (PG) methods produce artificial instances and can represent better the input space, therefore are more general. Both methods diminish the storage requirements for NN classifiers and make more efficient the labeling process. However, instance reduction techniques do not deal with high-dimensionality and noisy data issues in NN methods. For dealing with these shortcomings, feature selection and construction can be applied; these methodologies aim at reducing the dimensionality of data and extracting informative data representations [8]. Although, instance and feature reduction approaches are beneficial for NN classifiers, both methodologies are commonly applied separately. Hence, not all NN's concerns are addressed by existing methods.

This paper introduces SGPFGP: *Simultaneous Generation of Prototypes and Features through Genetic Programming*, a novel approach to simultaneous feature and prototype generation for NN-based classification. SGPFGP learns how to combine instances and attributes to generate a set of prototypes and a new feature space for classification tasks via genetic programming. An heterogeneous repre-

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
Copyright 2014 ACM 978-1-4503-2662-9/14/07 ...$15.00.
http://dx.doi.org/10.1145/2576768.2598356.

sentation is proposed and appropriate genetic operators are introduced. To the best of our knowledge, this is the first approach that aims at learning simultaneously class-specific prototypes and features to be used for NN-based classification. The effectiveness of the proposed method is evaluated and its performance is compared to alternative techniques. Experimental results show that SGPFGP compares favorably to state-of-the-art PG methods in terms of accuracy and reduction, with the additional benefit of dimensionality reduction.

The rest of this paper is organized as follows. Next section reviews related work on instance and feature reduction. Section 3 describes in detail the proposed SGPFGP approach. Section 4 presents the experimental framework to assess SGPFGP. Section 5 reports experimental results obtained by the proposed method. Finally, Section 6 outlines conclusions and discusses future work directions.

2. RELATED WORK

The PG problem has been approached in several ways, for a comprehensive study we refer the reader to [21], where Triguero et al. review and classify most of the existing PG methods up to 2012, additionally, a taxonomy and experimental comparison of these methods is also reported.

Most PG methods are iterative, they start with a set of prototypes (either randomly selected from the data set or artificially generated) and these are refined trying to maximize the classification performance. A classical representative of iterative PG methods is learning vector quantization (LVQ) [9], although many alternative methods have been proposed. For instance, in [21] the best method for PG in terms of classification performance was GENN (Generalized Editing using NN) [11]. GENN is a detrimental method that iteratively removes and re-labels instances. GENN achieves substantially better results than any other PG method compared in [21], although it was among the worse in terms of reduction of instances. On the other hand, PSCSA (Prototype Selection Clonal Selection Algorithm) obtained the best performance in terms of reduction [6]. PSCSA models the PG problem as an optimization one and uses an artificial immune system, the clonal selection algorithm, to solve it. This method is able to exactly select a single example per class, achieving the best reduction performance among the other 24 methods considered in [21]. However, its performance in terms of accuracy was worse than many other strategies. Other effective methods are based on bio-inspired optimization as well, see e.g., [5, 15, 2, 4, 19]. All of these methods evolve a population of individuals (codifying prototypes) trying to maximize the classification performance.

Although very effective PG methods have been proposed so far, it is somewhat disappointing that these methods have been developed independently of feature reduction techniques. Nevertheless, a few efforts in this direction have been proposed. Cagné et al. propose co-evolutionary approaches to simultaneously learn prototypes and neighborhood distance measure [1]. Both, competitive and cooperative schemes, are evaluated. Hussein et al. propose a genetic algorithm for instance selection, and feature weighting [10]. The main conclusion from [10] is that NN performance increases when one combines both types of methods. Paredes et al. used gradient descend to tackle the same problem, resulting in a very effective model [16]. More recently, Triguero et al. proposed a differential evolution technique that combines PG with feature weighting [22]. The above methods perform instance selection and feature weighting, allowing a reduction in the number of instances of the training set; however, they do not reduce data dimensionality.

Kuncheva and Jain propose a genetic algorithm for instance and feature selection [14]. Pedrajas et al. propose a similar evolutionary algorithm but put more emphasis on efficiency [17]. More recently, Chen et al. propose a multi-objective evolutionary algorithm for instance and feature selection [3]. Although being effective, these methods are still limited in terms of not being able to learn or to generate new features. Besides, as stated above, instance selection methods are a special case of PG, and therefore, generating prototypes is advantageous over selection (see partial/early evidence in [13]). Finally, it should be noted that the simultaneous methods from [14, 3] have only been evaluated on very small data sets.

This paper proposes SGPFGP, a method for simultaneous prototype and feature generation based on genetic programming. Contrary to previous hybrid approaches (instance + feature reduction) for selection, the proposed approach is completely based on the generation paradigm. That is, the aim is to build artificial instances and new features to improve NN classification performance. Escalante et al. have recently proposed a PG method based on genetic programming that combines instances to produce prototypes [4]. The proposed method extends that work by also allowing features generation, and by proposing a modeling framework that selects class-specific prototypes and features.

3. SIMULTANEOUS GENERATION OF PROTOTYPES AND FEATURES

Genetic programming is one of the youngest evolutionary algorithms [12], it has the distinctive feature that programs, codified in elaborated data structures, are evolved to solve complex modeling problems. Herein, a method for Simultaneous Generation of Prototypes and Features through Genetic Programming (SGPFGP) is introduced. The goal is to learn class-specific prototypes and features for pattern classification by NN.

The underlying hypotheses of SGPFGP are that prototypes for a class are formed by combining training instances that belong to that class; likewise, the generated features for a particular class are formed by combining features of instances that belong to that class. By considering a tree-structure for codifying prototypes and features, and letting leaf nodes to denote instances (resp. features) while non-terminal nodes to denote operators between leaf nodes, it is quite natural to use genetic programming to evolve a population of tree-prototypes and tree-features trying to maximize the classification performance of an NN classifier. The rest of this section provides a detailed description of the proposed method.

3.1 Considered scenario

Let $\mathcal{T} = \{(\mathbf{x}_1, y_1) \ldots, (\mathbf{x}_N, y_N)\}$ be a training set of labeled instances, with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathcal{C} = \{1, \ldots, K\}$, where d is the problem dimensionality and K is the number of classes in the considered problem. The goal is to generate a set of sets of prototypes $\mathcal{P}^* = \{\mathcal{P}_1, \ldots, \mathcal{P}_K\}$ (one set of prototypes per class) where each set of prototypes is defined as follows: $\mathcal{P}_k = \{(\mathbf{w}_1^k, y_1^k), \ldots, (\mathbf{w}_{L_k}^k, y_{L_k}^k)\}$, for $k = 1, \ldots, K$

such that $L_k << N$, where L_k is the number of prototypes for class k, $\mathbf{w}_i^k \in \mathbb{R}^{d_k}$, and d_k is dimensionality of the feature space for prototypes in set $\mathcal{P}_k$. This means that each set of prototypes $\mathcal{P}_k$ is associated to a single class k and the prototypes of this class have the same dimensionality d_k, which is not necessarily the same for different classes.

3.2 Representation

For learning $\mathcal{P}^*$ a genetic program is proposed where each individual codifies a possible solution to this problem, see Figure 1. Specifically, individuals are represented by a set of trees pairs: $\{(\mathcal{S}_p^1, \mathcal{S}_f^1), \ldots, (\mathcal{S}_p^K, \mathcal{S}_f^K)\}$, where $\mathcal{S}_p^k = \{s_{p,1}^k, \ldots, s_{p,L_k}^k\}$ is a set of trees for generating class k prototypes and $\mathcal{S}_f^k = \{s_{f,1}^k, \ldots, s_{f,d_k}^k\}$ is a set of trees for generating class k features. Each tree is associated to a single prototype ($s_{p,i}^k$) or feature ($s_{f,j}^k$), and both sets of trees are dependent to each other: prototypes generated via $\mathcal{S}_p^k$ use features generated with $\mathcal{S}_f^k$. In this way, pairs ($\mathcal{S}_p^k, \mathcal{S}_f^k$) are associated to a set of prototypes: $\mathcal{P}_k = \{(\mathbf{w}_1^k, y_1^k), \ldots, (\mathbf{w}_{L_k}^k, y_{L_k}^k)\}$.

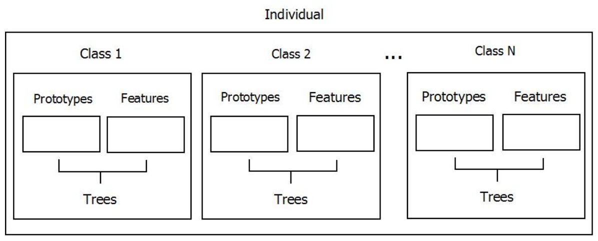

Figure 1: Representation of an individual: class-specific prototype-feature trees.

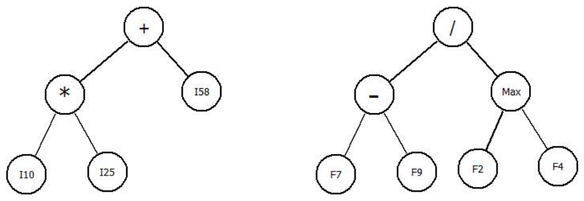

Figure 2: Prototype representation ($s_{p,i}^k$, left): instances are combined; and feature representation ($s_{p,i}^k$, right): features are combined.

Hence, in SGPFGP individuals are encoded as trees $s_{p,i}^k$ and $s_{f,j}^k$ in the genetic program (see Figure 2). Each tree $s_{p,i}^k$ (resp. $s_{f,j}^k$) can generate a prototype (resp. feature) by combining instances (resp. features) taken from the training set $\mathcal{T}$. That is, the terminal set for prototype-trees $s_{p,i}^k$ is the subset of instances from the training set $\mathcal{T}$ that are labeled with class k. Likewise, the terminal set for feature-trees $s_{f,j}^k$ is the set of $d-$dimensional feature vectors of instances in $\mathcal{T}$ that belong to class k.

The function set for combining instances and features is the following set of operators: $\{+, -, *, /, Min, Max\}$, all of them with arity two. In a preliminary study, other operators were evaluated (including operators of arity 1); however, better results were observed with the above mentioned function set.

3.3 Fitness function

The proposed representation is implemented into a standard genetic program [18], with the aim of maximizing the classification performance of a 1NN[1] classifier when using prototypes and features induced by different individuals. Accordingly, the fitness function for the genetic program is simply the classification performance obtained by a 1NN when classifying all instances in a validation set. Instances in the validation set are not considered in the terminal set of prototype trees (i.e., the validation set and $\mathcal{T}$ are disjoint). The motivation behind this hold-out split is to avoid, to some extent, generating overfitted prototypes/features.

More specifically, to assess the quality of every individual, the next procedure is followed: the prototypes induced by prototype-trees are obtained using the input space of feature-trees for each class. Next, every instance to be classified with the prototypes is first projected into the features learned by the genetic program for each class, and then it is compared to the prototypes of different classes (each comparison in the corresponding input space). The test instance is assigned to the class of the nearest prototype. One should note that by proceeding in this way, the 1NN classifier is allowed to learn specific features and prototypes for each of the considered classes.

3.4 Genetic program and considered operators

Algorithm 1 presents the details of the proposed SGPFGP method. The population was initialized with ramped-half-and-half considering a maximum tree deep of D_max. For initializing prototype-trees per class half of the trees are initialized with the full method, and the other half with the grow technique, with the restriction of using instances of the same class. Feature-trees were initialized similarly, but without the class-restriction, because features are initially common to all instances. For initialization, the proposed method takes as input PP and PF parameters, which specify the number of initial prototype-trees and feature-trees, respectively. One should notice that, although the user has to specify PP and PG, after initialization, the genetic program automatically determines/adjust the final number of prototypes and features.

Generic genetic operators and ad-hoc ones are considered for the SGPFGP method. An extensive preliminary experimentation was conducted to determine the best operators combination, here the ones that provided the best results are described (individuals are selected via tournament):

- **Crossover**: given two individuals, a prototype (resp. feature) tree from each parent is randomly selected, then a subtree of each selected tree is chosen and ex-

[1]One should note that the selection of 1NN (instead of any other KNN classifier) is not arbitrary. Most previous studies on PG for NN classification have used 1NN as well, see [21]. This is, in part, motivated by the fact that using 1NN allows PG methods to be able (at least theoretically) to reach the optimal number of prototypes (1 per class). Nevertheless, the proposed method can also work with other KNN classifiers.

Algorithm 1 Simultaneous Generation of Prototypes and Features through Genetic Programming.

```
Require: :
  $\mathcal{T}$ : Training Set;
  $\mathcal{V}$ : Validation Set;
  $Generations$ : Number of Generations
  $Population$ : Number of Individuals
  $PP$ : Proportion of initial prototypes
  $PF$ : Proportion of initial features
Ensure: $Best_x$ : The best individual, the solution that obtains the highest accuracy in $\mathcal{V}$.
  $P \Leftarrow Ramped\text{-}half\text{-}and\text{-}half(PP, PF, \mathcal{T})$
  $Best_x \Leftarrow \emptyset$
  $FBest \Leftarrow -Inf$
  $i = 0$
  while $i <= Generations$ do
    for $j = 1 \rightarrow Population$ do
      $f(j) \leftarrow fitness(P_j, \mathcal{V})$
      if $f(j) > FBest$ then
        $FBest \leftarrow f(j)$
        $Best_x \Leftarrow P_j$
      end if
    end for
    $Pool \Leftarrow Sampling(P)$
    $Offspring \Leftarrow MultiCrossover(Pool)$
    $Offspring \Leftarrow Crossover(Offspring)$
    $Offspring \Leftarrow Mutation(Offspring)$
    $P \Leftarrow Selection(P, Offspring)$
    $i \leftarrow i + 1$
  end while
  return $X_p$
```

changed. The outputs of this operator are two offspring. This process is repeated for each class associated to the classification problem. The operator is applied to all classes because the representation is complex and affecting a class at a time would not have a significant effect in the overall population.

- **Mutation**: given an individual, for each class, a randomly selected tree is eliminated and a new one is generated (using the same process as in the initialization). This procedure is similar for both prototype and feature trees.
- **Multi-Crossover**: a new crossover operator is introduced due to the representation complexity. Given two individuals, a new offspring is generated by combining trees from the parents. The offspring tree, for each class, is selected from the corresponding trees of the parents with uniform probability. This procedure is similar for both prototype and feature trees.

Genetic operators are applied with certain probabilities and in a predefined order (see Algorithm 1). A population of trees is evolved and the best individual found during the search process is returned. In the rest of the paper the performance of the proposed method is evaluated for simultaneous prototype and feature generation.

4. EXPERIMENTAL FRAMEWORK

For the experimental evaluation of the proposed method, the suite of data sets introduced by Triguero et al. [21] is considered. This benchmark consists of 59 data sets associated to different classification problems. Table 1 describes the considered data sets.

This suite comprises classification problems with varied characteristics in terms of number of classes, number/type of attributes (nominal and numeric), and number of instances. Table 2 summarizes the main features of the benchmark. This suite has been widely used for benchmarking GP methods [21, 4]. Commonly, results of PG methods are reported separately for **small** (data sets with less than 2,000 instances) and **large** (data sets with at least 2,000 instances) data sets. Accordingly, results for both types of data sets are reported. Different aspects of the proposed method are assessed through this benchmark and a performance comparison to other approaches using the same suite is carried out.

The evaluation methodology adopted in [21] consists of applying PG methods to each data set in a 10-fold cross validation scheme. That is, each data set is split into 10 subsets, and 10 rounds of training and testing are performed; in each round 9 subsets are used as training set and 1 subset is used for testing, the process is repeated 10 times using a different subset for testing each time. This means that for each data set the proposed PG method is applied 10 times, every time the method learns prototypes and features using the training subsets and the generated prototypes/features are evaluated in the test set.

For the evaluation process, a 1NN rule is used by the generated prototypes[2]. One should note that a single run of the proposed method over the whole benchmark represents 590 executions (10 times per data set). It is important to emphasize this because the same evaluation methodology as in [21] has been adopted (using exactly the same data partitions for training and testing in each of the 10-folds per data set), thus results are directly comparable to those reported by Triguero et al. and other authors that have used the same collection.

The following experimental settings are defined to assess the proposed approach: the number of generations is set to 50, the population size is set to 100, and the maximum tree depth is fixed to 3. The values of these parameters were determined after a brief preliminary study on parameter selection. Finally, to compare results obtained from different methods, a Wilcoxon signed-rank test is used to determine if the difference in performance is statistically significant, with a confidence level of 95%.

5. EXPERIMENTAL RESULTS

This section reports experimental results to show the effectiveness of the proposed method. The performance of SGPFGP is compared against 24 other PG techniques that have been evaluated with the same benchmark. The analysis starts by visually analyzing the positioning of the learned prototypes for a 2D data set. Next, the SGPFGP performance is evaluated in terms of classification accuracy, and instance/feature reduction rates.

[2]One should note that since there is a subset of prototypes and features per class. Every test-instance has to be represented and compared using the set of prototypes and features learned for each class. The test-instance is labeled with the class of the nearest prototype.

Table 1: Description of the data sets used for the experimental study [21]. The number of samples, the number of attributes and the number of classes are shown per data set.

Data set	Samples	Attributes	Classes
Abalone	4174	8	28
Appendicitis	106	7	2
Australian	690	14	2
Autos	205	25	6
Balance	624	4	3
Banana	5300	2	2
Bands	539	19	2
Breast-Cancer	286	9	2
Bupa	345	6	2
Car	1728	6	4
Chess	3196	36	2
Cleveland	297	13	5
Coil2000	9822	85	2
Contraceptive	1473	9	3
Crx	125	15	2
Dermatology	366	33	6
Ecoli	336	7	8
Flare-Solar	1066	9	2
German	1000	20	2
Glass	214	9	7
Haberman	306	3	2
Hayes-Roth	133	4	3
Heart	270	13	2
Hepatitis	155	19	2
Housevotes	435	16	2
Iris	150	4	3
Led7digit	500	7	10
Lymphography	148	18	2
Magic	19020	10	2
Mammographic	961	5	2
Marketing	8993	13	9
Monks	432	6	2
Movements-libras	360	90	15
Newthyroid	215	5	3
Nursey	12960	8	5
Pageblocks	5472	10	5
Penbased	10992	16	10
Phoneme	5404	5	2
Pima	768	8	2
Ring	7400	20	2
Saheart	462	9	2
Satimage	6435	36	7
Segment	2310	19	7
Sonar	208	60	2
Spambase	4597	57	2
Spectheart	267	44	2
Splice	3190	60	3
Tae	151	5	3
Texture	5500	40	11
Thyroid	7200	21	3
Tic-tac-toe	958	9	2
Titanic	2201	3	2
Twonorm	7400	20	2
Vehicle	846	18	4
Vowel	990	13	11
Wine	178	13	3
Wisconsin	683	9	2
Yeast	1484	8	10
Zoo	101	16	7

Table 2: Summary of features for the 59 data sets of the considered benchmark [21].

	Small	Large
# data sets	40	19
# Classes	[2, 15]	[2, 28]
# Instances	[101, 1,728]	[2,201, 19,020]
# Attributes	[3, 90]	[2, 85]
# BDs w. Nominal atts.	7	3
# BDs w. Numerical atts.	22	14
# BDs w. Mixed atts.	11	2

5.1 Visualization learned prototypes

In this section, the prototypes generated with SGPFGP using a 2D-data set are presented. The goal of this experiment is to analyze how the prototypes learned by SGPFGP are distributed in the input space for a complex classification problem. The banana data set is associated to a binary classification task (2-classes) that contains 5300 instances described by 2 attributes. Figure 3 shows a subsampling of the original data set; it can be seen that the decision surface for this problem is non linear and very complex to capture.

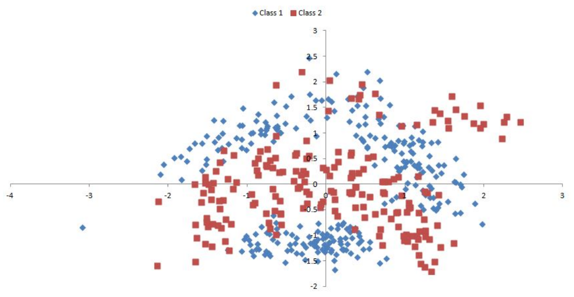

Figure 3: Banana data set.

Recall that the proposed method generates specific features for each class, hence, for this data set the SGPFGP output is two sets of prototypes/features. Figures 4 and 5 show a subsampling of the banana data set projected into the feature spaces of each of the classes. In each figure the corresponding prototypes are also plotted. From these figures, it can be seen that features for different classes result in very different projections of the training data. The projection induced by features from class 2 (Figure 5) somewhat preserves the structure of the original problem, while the projection for features from class 1 (Figure 4) is totally different. In both projections, one can see that the generated prototypes effectively capture the distribution of training examples of the respective class, i.e., prototypes of a class are closer to instances of its own class than to those of different classes.

For this particular example, the SGPFGP method selects 7 and 6 prototypes for classes 1 and 2, respectively; which represents a reduction performance of 0.9975. SGPFGP achieves an average classification performance of 0.8628 for this data set, which is very close to the performance of the best method for this data set in the comparative study of [21] (the DEPUR method obtained 0.896 [20]).

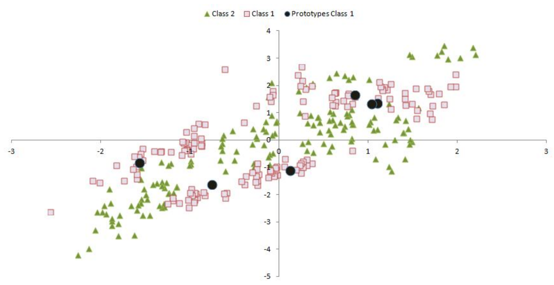

Figure 4: Feature space and prototypes of class 1 for the Banana data set.

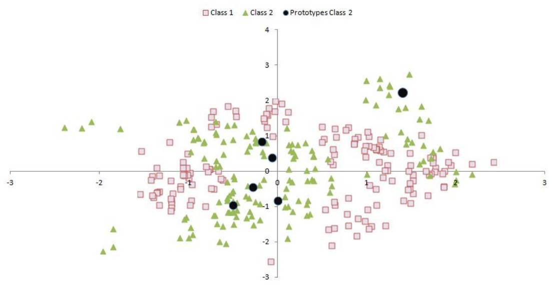

Figure 5: Feature space and prototypes of class 2 for the Banana data set.

5.2 Classification performance of prototypes

Table 3 shows the average classification performance obtained by the SGPFGP approach for small and large data sets, for comparison the corresponding results for 1NN using the whole training set are included as well as the performances (from [21]) obtained by the best methods in terms of accuracy (GENN [11]) and reduction (PSCSA [6]).

Table 3: Average classification accuracy for SGPFGP and reference methods.

	SGPFGP	GENN [11]	PSCSA [6]	1NN
Small	0.719 ± 0.06	***0.756*** ± 0.05	$0.668 \pm 0.07\star$	0.734 ± 0.05
Large	0.802 ± 0.01	***0.813*** ± 0.01	$0.670 \pm 0.02\star$	0.80 ± 0.01

$\star$ statistically significant difference.

From Table 3, it is observed that prototypes and features generated by SGPFGP do not degrade significantly the classification performance obtained with 1NN when using the whole data set. Therefore, data dimensionality and number of training instances are reduced without compromising the classification performance.

The proposed method outperforms PSCSA by a large margin over both small and large data sets, the differences are statistically significant. On the other hand, GENN outperforms SGPFGP by about 4% and 1% for small and large data sets, however these differences are not statistically significant. These results confirm that SGPFGP is a very competitive method in terms of accuracy, achieving similar performance as the best PG methods that were evaluated in [21]. Interestingly, better results were obtained by SGPFGP for large data sets; this is a very positive result because PG methods are precisely designed to deal with large data sets.

It is important to mention that SGPFGP classification performance is similar to that obtained by another PG method based on genetic programming [4]. However, SGPFGP obtains substantially better instance-reduction performance and is able to reduce data dimensionality.

Table 4 shows the average execution time (in seconds) for the classification phase of SGPFGP, compared against the execution time of 1NN over the large data sets. We can appreciate that the proposal is far more efficient, it reduces a 98.45% of the time in the classification. One should note that the execution time in SGPFGP includes the processes of projecting data into different feature spaces and estimating distances.

Table 4: Average execution time (seconds) in the classification step.

Dataset	SGPFGP	1NN
Large	0.0067	0.1695

5.3 Reduction performance of prototypes

Table 5 shows the average instance-reduction rates for SGPFGP, GENN, PSCSA and 1NN. From this table, it is observed that SGPFGP performance is very competitive as well. SGPFGP significantly outperforms GENN (improvements higher than 80%) and achieves virtually the same reduction performance as PSCSA, which is the best method in terms of reduction evaluated in [21]. In fact, SGPFGP offers a better tradeoff between accuracy (Table 3) and reduction (Table 5) than the referenced methods. These results can be better appreciated in Figures 6 and 7, where accuracy vs. reduction are plotted for the 24 methods considered in [21] for small and large data sets, respectively.

Table 5: Average reduction rates obtained by the considered methods.

	SGPFGP	GENN	PSCSA	1NN
Small	0.983 ± 0.0007	$0.186 \pm 0.0205\star$	***0.985*** ± 0.0001	0 ± 0
Large	0.994 ± 0.0005	$0.157 \pm 0.0049\star$	***0.998*** ± 0.0001	0 ± 0

$\star$ statistically significant difference

5.4 Accuracy vs. reduction tradeoff

It can be seen from Figure 6 that for small data sets, the performance in reduction of SGPFGP is among the best ones; although its accuracy is lower than several other techniques (e.g., PSO [15]). However, as depicted in Figure 7, for large data sets the proposed approach offers the best tradeoff between accuracy and reduction. Achieving similar accuracy as ENPC [5] and DEPUR [20] (two highly effective PG techniques), but with much better reduction performance. Also the fact that GENN obtains highly accurate prototypes at the expense of poor reduction performance is confirmed.

If the Pareto front is constructed using the reduction and accuracy performances for all of the considered PG methods, SGPFGP would be part of it (i.e., it *dominates* many other PG methods in both figures). SGPFGP would be located in the upper right corner of the Pareto front, making it an ideal solution that fulfills the accuracy vs. reduction tradeoff.

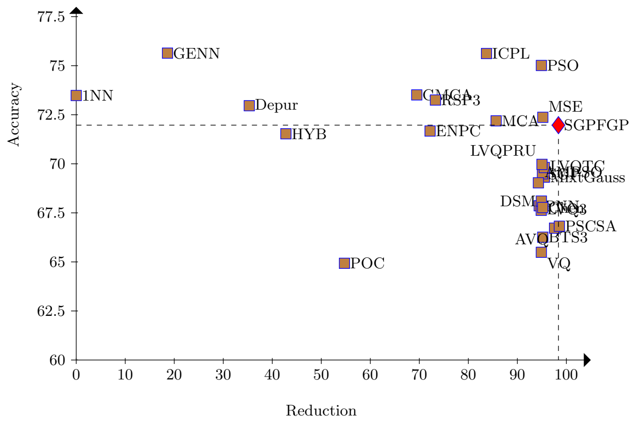

Figure 6: Average reduction (x−axis) vs. accuracy (y−axis) for small data sets.

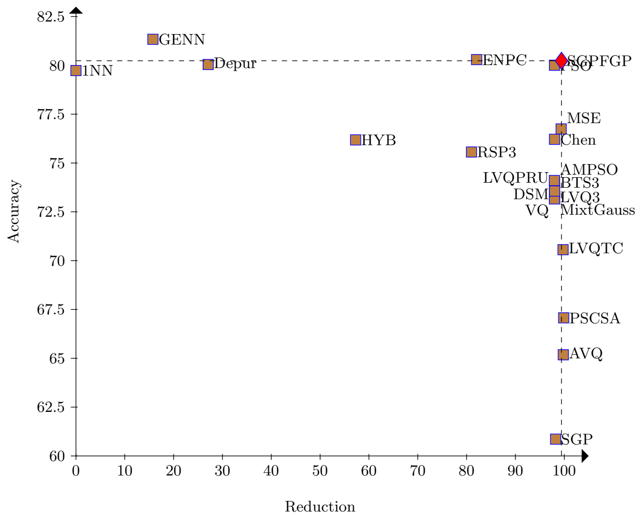

Figure 7: Average reduction (x−axis) vs. accuracy (y−axis) for large data sets.

Therefore, it is possible to conclude that SGPFGP is a very competitive method in terms of the accuracy and instance reduction tradeoff, mainly for large data sets.

5.5 Feature generation

The proposed PG method has demonstrated to be a competitive approach. Moreover, experimental results have shown the additional benefit of dimensionality reduction. In this regard, the average feature reduction rates were of 0.418 and 0.41 for small and large data sets, respectively. Thus, SGPFGP generates representations that account for only 60% of the original dimensionality and yet it is able to obtain state-of-the-art performance. The combined reduction of instances and features results in speeding up 1NN when classifying test instances. As future work we will compare the performance of SGPFGP with that of alternative feature selection/extraction techniques.

5.6 Performance over different attribute types

Because the considered data sets can have nominal, numeric or mixed attributes, it is worth analyzing SGPFGP performance for these different types of attributes. Table 6 shows the average accuracy obtained by the proposed method for different attribute types using small and large data sets. Nominal attributes are better exploited by the proposed approach for both small and large data sets (please note that there are only 10 data sets with only-nominal attributes, 7 small and 3 large ones). Although for large data sets the performance is virtually the same when using nominal or numeric attributes.

It is interesting that better performance is obtained by the proposed method when a single attribute type was considered (either nominal or numeric) and that SGPFGP has problems for data sets with mixed attributes. This result could be due to the nature of the proposed approach which combines attributes to generate new features: combining attributes of different nature does not necessarily result in a better feature.

Table 6: Average accuracy for different data types.

	Nominal	Numerical	Mixed
Small	$\boldsymbol{0.768} \pm 0.04$	0.724 ± 0.07	0.671 ± 0.06
Large	$\boldsymbol{0.828} \pm 0.02$	0.823 ± 0.01	0.613 ± 0.01

6. CONCLUSIONS AND FUTURE WORK

This paper introduced SGPFGP a novel approach to simultaneous feature and prototype generation for NN-based classification. SGPFGP learns to combine instances and attributes to generate class-specific prototypes and features via genetic programming. A heterogeneous representation is proposed and appropriate genetic operators are applied. To the best of our knowledge, this is the first approach that aims at learning simultaneously prototypes and features for every class used in NN-based classification. The performance of the proposed method is evaluated and compared with a wide variety of other PG methods. Experimental results show that SGPFGP compares favorably to state-of-the-art PG methods in terms of accuracy and reduction, with the additional benefit of reducing data sets dimensionality.

Conclusions of this work can be summarized as follows:

- SGPFGP is able to reduce data dimensionality and the number of training instances without compromising the classification performance. In fact, SGPFGP obtains similar performance as the most effective PG methods proposed so far.
- Better results are obtained by SGPFGP for large data sets, making it suitable for large scale classification problems.
- Instance-reduction performance of SGPFGP is very close to the optimal reduction rates (i.e., one instance per class). SGPFGP outperforms most of the reference methods considered in the study in terms of instance reduction.
- SGPFGP reduces about 40% data sets dimensionality. Building new features that are more informative for designing NN classifiers.
- SGPFGP obtains better results when data sets attributes are of the same type (either nominal or nu-

meric). This is possibly due to the fact that the current representation and operators are not designed to combine heterogeneous attributes.

- During the search SGPFGP tries to balance the number of instances per class. This is good for database problems with unbalanced data.

Current and future work include approaching the problem of simultaneous feature-prototype generation with a multi-objective genetic program that can result in solutions with better tradeoffs among the objectives involved in the problem (e.g., accuracy + instance-reduction + dimensionality-reduction). We would also like to compare the proposed approach with feature extraction/construction methods in the state-of-art. Finally, the proposed approach is currently being assessed on real and large scale data sets, including object recognition and image annotation data.

Acknowledgements. This work was partially supported by the LACCIR programm under project ID R1212LAC006. Mauricio García-Limón was supported by CONACyT with scholarship No. 345683.

7. REFERENCES

[1] C. Carné and M. Parizeau. Coevolution of nearest neighbot classifiers. *International Journal of Pattern Recognition and Artificial Intelligence*, 21(5):921–946, 2007.

[2] A. Cervantes, I. M. Galvan, and P. Isasi. AMPSO: a new particle swarm method for nearest neighborhood classification. *IEEE Transactions on Systems Man and Cybernetics part B*, 39(5):1082–1091, 2009.

[3] J. H. Chen, H. M. Chen, and S. Y. Ho. Design of nearest neighbor classifiers: multi-objective approach. *International Journal of Approximate Reasoning*, 40:3–22, 2005.

[4] H. J. Escalante, K. M. Mendoza, M. Graff, and A. Morales-Reyes. Genetic programming of prototypes for pattern classification. In *Proceedings of IbPRIA 2013*, volume 7887 of *LNCS*, pages 100–107. Springer, 2013.

[5] F. Fernandez and P. Isasi. Evolutionary design of nearest prototype classifiers. *Journal fo Heuristics*, 10:431–454, 2004.

[6] U. Garain. Prototype reduction using an artificial immune system. *Pattern Analysis and Applications*, 11(3–4):353–363, 2008.

[7] S. García, J. Derrac, J. R. Cano, and F. Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):417–435, 2012.

[8] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors. *Feature extraction, foundations and applications*, volume 207 of *Studies in fuzzines and soft computing*. Springer, 2006.

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer New York Inc., New York, NY, USA, 2001.

[10] F. Hussein, N. Kharma, and R. Ward. Genetic algorithms for feature selection and weighting a review and study. *Document Analysis and Recognition*, pages 1240–1244, 2001.

[11] J. Koplowitz and T. Brown. On the relation of performance to editing in nearest neighbor rules. *Pattern Recognition*, 13(3):251–255, 1981.

[12] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, 1992.

[13] L. Kuncheva and J. Bezdek. Nearest prototype classification: Clustering, genetic algorithms or random search. *IEEE Transations on Systems Man and Cybernetics part C*, 28(1):160–164, 1998.

[14] L. Kuncheva and L. C. Jain. Nearest neighbor classifier: Simultaneous editing and feature selection. *Pattern recognition letters*, 20:1149–1156, 1999.

[15] L. Nanni and A. Lumini. Particle swarm optimization for prototype reduction. *Neurocomputing*, 72(4–6):1092–1097, 2008.

[16] R. Paredes and E. Vidal. Learning prototypes and distances: A prototype reduction technique based on nearest neighbor error minimization. *Pattern Recognition*, 39:180–188, 2006.

[17] N. G. Pedrajas, A.H. García, and J. P. Rodríguez. A scalable approach to simultaneous evolutionary instance and feature selection. *Information Sciences*, 228:150–174, 2012.

[18] R. Poli, W. B. Langdon, and N. Freitag McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008.

[19] A. Rosales-Perez H. J. Escalante, C. A. Coello, J. A. Gonzalez and C. A. Reyes-Garcia. An Evolutionary Multi-Objective Approach for Prototype Generation. Proccedigns of the World Congress on Computational Intelligence, to appear, 2014.

[20] J. S. Sanchez, R. Barandela, A. I. Marques, R. Alejo, and J. Badenas. Analysis of new techniques to obtain quality training sets. *Pattern Recognition Letters*, 24(7):1015–1022, 2003.

[21] I. Triguero, J. Derrac, S. García, and F.Herrera. A taxonomy and experimental study on prototype generation for nearest neighbor classification. *IEEE Transations on Systems Man and Cybernetics part C*, 42(1):86–100, 2012.

[22] I. Triguero, J. Derrac, S. García, and F. Herrera. Integrating a differential evolution feature weighting scheme into prototype generation. *Neurocomputing*, 37:332–343, 2012.

[23] X. Wu, V. Kumar, R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge Information Systems*, 14(1):1–37, 2007.