Evolutionary Adaptive Self-Generating Prototypes for Imbalanced Datasets

Dayvid V. R. Oliveira, George D. C. Cavalcanti, Tsang Ing Ren and Ricardo M. A. Silva
Centro de Informática
Universidade Federal de Pernambuco, Brazil
http://www.cin.ufpe.br/~viisar
{dvro, gdcc, tir, rmas}@cin.ufpe.br

***Abstract*—The nearest neighbor (NN) is one of the most well known classifiers in pattern recognition. Despite the high classification accuracy, the NN has several drawbacks: high storage requirements, bad time of response, and high noise sensitivity. Prototype Generation (PG) is one of the most well-known solutions to tackle these shortcomings. In supervised classification, many real world datasets do not have an equitable distribution among the different classes, these are called imbalanced datasets. Many PG techniques that have a high classification accuracy in regular datasets, have a poor performance when dealing with imbalanced datasets. The *Self-Generating Prototypes* (SGP) is one of these techniques. The *Adaptive Self-Generating Prototypes* was proposed to tackle the SGP problem with imbalanced datasets, but, in doing so, the reduction rate is compromised. This paper proposes the *Evolutionary Adaptive Self-Generating Prototypes* (EASGP), a SGP based technique with iterative merging and evolutionary pruning to help find the optimal solution. An experimental analysis is performed with datasets of different levels of imbalance ratio and statistical tests are used to evaluate the proposed technique. The results obtained show that EASGP outperforms previous SGP based algorithms in classification accuracy and reduction.**

***Keywords*—Prototype Generation (PG), Self-Generating Prototypes (SGP), Adaptive Self-Generating Prototypes (ASGP), Imbalaced Datasets**

I. Introduction

The nearest neighbor (NN) rule [1] is one of the most well known supervised learning techniques. The general idea is to classify a new instance as being of the same class as its nearest instance from the training set. The K-nearest neighbor rule (KNN) [2] is a generalization of the NN rule that considers the label of the K nearest instances of an instance to make it's classification. This technique is very simple, yet is one of the most interesting and useful algorithms in Pattern Recognition [3].

Despite the high classification accuracy, the KNN has several drawbacks [4] [5] [6], the three most relevant are:

1) **high storage requirements**: it needs to store all training set, because the decision rule is defined by all training instances.
2) **bad response time**: for every new classification, the KNN needs to visit all instances from the training set ($\mathcal{O}(n)$ complexity, where n is the size of the traning set).
3) **low tolerance to noise**: it considers all data from the training set to be relevant to the classification task. Because of that, noisy data might compromise the classification accuracy.

In the literature, many approaches have been proposed to solve these issues [7]. One of these approaches is the use of instance reduction techniques [8]. Techniques of instance reduction aim to select only relevant instances from the training set, creating a smaller training set without compromising the classification accuracy [8]. In doing so, the storage requirements and time of response are reduced, because less instances are saved and visited in each classification, and also removes noisy data, improving classification and solving the noise sensitivity problem.

Prototype selection (PS) is a process of instance reduction that removes instances that are redundant or irrelevant to the classification, and selects the representative ones. Prototype generation (PG) is the process of instance reduction that generates artificial instances in order to achieve a higher generalization and create a more suitable training set. Because of the limitations of the search space of PS techniques, PG techniques have the potential of achieve a higher reduction rate than PS techniques.

Imbalanced datasets have many instances of one class, the majority class, and only a few of the other, the minority class. Learning from such datasets is a difficult task, being considered an important problem in data mining and pattern recognition [9] [10]. Despite the high performance in improving classification, instance reduction techniques do not cope with imbalanced datasets, for they cannot discriminate noisy instances from the minority class.

The *Self-Generating Prototypes* (SGP) [11] is a centroid-based prototype generation technique, it was even being combined with other techniques to improve classification [12], but does not work well with imbalanced datasets. The same thing happens with the *Self-Generating Prototypes 2* (SGP2).

Experiments have shown that, in some datasets, SGP and SGP2 consider all minority class instances as noise or outliers that need to be removed in order to improve generalization [13]. In order to solve this issue, in [13], the authors proposed the *Adaptive Self-Generating Prototypes* (ASGP) a SGP based technique that is adaptive to different levels of imbalance ratio. This technique achieved interesting results, and the empirical experiments have shown that ASGP is a better solution for imbalanced datasets [13].

ASGP outperforms SGP2 in classification accuracy, but

it is outperformed in reduction rate because ASGP generates more minority class prototypes than needed (increasing false positives in classification).

To solve the issues of SGP, SGP2 and ASGP, in this paper, we propose the *Evolutionary Adaptive Self-Generating Prototypes* (EASGP), a PG technique composed by a main loop and two new steps: incremental merging and evolutionary pruning. The incremental merging expands the search space inserting new generalized samples that might better represent the distribution of the classes. The evolutionary pruning is a memetic algorithm (MA) based on the *Steady-State Memetic Algorithm* (SSMA) [14] used to explore the search space and find optimal solution. The evolutionary pruning uses the original training set as a validation set, instead of using only the new generated samples.

The experiments showed that EASGP outperforms SGP, SGP2 and ASGP in classification accuracy and reduction rate for datasets with different levels of imbalance. This result was confirmed with the One-Sided Wilcoxon Signed Rank Test [15].

This paper is organized as follows: Section II gives a brief review of PG techniques and presents the SGP, SGP2 and ASGP techniques. Section III presents the EASGP technique. Section IV presents the experiments and results. Finally, Section V concludes the paper.

II. Background

This section presents the main concepts of prototype selection (PS) and prototype generation (PG), including the Self-Generating Prototypes (SGP), Self-Generating Prototypes 2 (SGP2) and Adaptive Self-Generating Prototypes (ASGP).

A. Prototype Selection and Generation

PS methods are instance selection methods that, in order to improve the NN rule, attempt to find the smallest subset of the training set that enable the KNN to correctly classify a test sample [16].

The PS problem can be defined as follows: Let TR be the training set, and $S \subseteq TR$ be the subset of instances selected by a PS technique, where S has less noise or redundant instances. The classification of a test sample x_i is made using the KNN rule over S instead of TR.

Since PS problems can be reduced to a combinatorial problem, it is possible to use Evolutionary Algorithms (EAs) to solve them. In fact, a high number of the PS algorithms recently proposed are based on EAs. Those methods were called evolutionary prototype selection (EPS) methods [17]. The Steady-State Memetic Algorithm (SSMA) [14] is a Memetic Algorithm (MA) applied to PS. Studies have shown that this is one of the most successful PS techniques [5]. The use of stratified PS based on SSMA is an alternative to solve the scalability issue, the problem of increasing the running time when the number of instances increase [18].

There are over 50 PS methods proposed in the literature. A complete study of PS, including taxonomy, can be found in [5].

PG methods are instance reduction methods that attempt to find the smallest set of artificial generated instances that improves the accuracy of the NN rule based on the training set.

A PG problem can be defined as follows: Let TR be the training set, and TG a set of prototypes generated or selected by a PG method based on TR, where *Size(*TG*) < Size(*TR*)*. The classification of a test sample x_i is made using the KNN rule over TG, instead of TR.

Most PG techniques that uses EAs are based on positioning adjustment, that means that they move the prototypes around the m-dimensional space, adjusting them until it finds an optimal solution [6]. Examples of interesting positioning adjustment PG techniques are the *Evolutionary Nearest Prototype Classifier* (ENPC) [19] and the *Particle Swarm Optimization* (PSO) [20]. Other techniques such as *Prototype Selection Clonal Selection Algorithm* (PSCSA) [21], and *Differential Evolution* (DE) [22] have also achieved interesting results.

For imbalanced datasets, the use of selection of evolutionary selection of generalized examples [23] has achieved interesting results.

There are over 25 PG methods proposed in the literature. A complete study of PG, including taxonomy, can be found in [6].

B. Self-Generating Prototypes Based Algorithms

The *Self-Generating Prototypes* (SGP) [11] is an interesting PG technique [12]. The SGP method generates prototypes using a combination of centroid based and space splitting mechanisms. The SGP creates groups of instances generates a single prototype (the representant) for each group.

In the beginning of the process, for each class, the SGP generates one group containing all instances of that class. Then, the following steps are performed the until the solution converge:

1) If, for all instances of a group, the closest prototype is the representant of the group itself, then no modification is performed.
2) If, for all instances of a group, the closest prototype is from a different class, then the group is divided in two new subgroups. The separation is made by a hyperplane that passes through the representant of the original group and is perpendicular to the first principal component of the instances in the original group.
3) If, for some instances of a group, the closest prototype is a representant of a different group of the same class, these instances are moved from the original group to the group of that closest prototype.
4) If, for some instances in a group, the closest prototype is the representant of a group with different class, the misclassified instances are removed from the original group and form a new group.

After any of these procedures, each group representant is updated.

In order to improve the generation capability, the SGP implements a trade-off between training error and the model

complexity using two parameters: R_{min} and R_{mis}. If the number of instances in a group divided by the number of instances of the largest group is less than a threshold R_{min}, the group is discarded. Also, along the SGP algorithm, if the number of misclassified instances in a group divided by the number of instances in that group is less than a threshold R_{mis}, no modification is performed. This step is called *generalization step*.

The SGP2 introduces two steps in order to reduce even more the number of prototypes: A merging step and a pruning step. In the merging step, two groups A and B are merged if both A and B are from the same class and the second closest prototype to all instances in A is the representant of B, and the second closest prototype to all instances in B is the representant of A. The pruning step removes redundant prototypes using the following rule: if all instances of a group is correctly classified by their second closest prototype, the group is discarded.

In [13], the authors analyzed the SGP behavior when trained with imbalanced datasets. Sometimes, the SGP returned no prototypes at all of the minority class. The authors concluded that one of the major issues happens in the generalization step, so they proposed a different approach in the use of the generalization factor R_{min}.

SGP eliminates the groups that have less than R_{min} times L instances, L being the size of the largest group in the dataset. The *Adaptive Self-Generating Prototypes* (ASGP) suggests that this elimination is not fair with the minority class groups. To solve this issue, the same R_{min} is used for all groups, but L is the size of the largest group of the same class, instead of the size of the largest group of all classes. This step is detailed in Algorithm 1.

Algorithm 1 Generalization Step

Require: GP: a set of groups of instances
Require: H: a hashtable
Require: CS: a list of classes of the groups
for all Class $C \in CS$ **do**
  $L \leftarrow -1$
  for all Group G in GS **do**
    if $SizeOf(G) > L$ **then**
      $L \leftarrow SizeOf(G)$
    end if
  end for
  $H(C) \leftarrow L$
end for
for all Group G in GS **do**
  $C \leftarrow ClassOf(G)$
  $L \leftarrow H(C)$
  if $\frac{SizeOf(G)}{L} \leq R_{min}$ **then**
    Remove G from GS
  end if
end for
return GS

ASGP also proposes that the merge and pruning steps should be performed as usual, but after both procedures, all prototypes of the minority class should be included again, in case they were lost in the generalization procedure.

The ASGP method achieved interesting results with imbalanced datasets, but there are other drawbacks to be tackled. When performing the merging step, the order in which the merging occurs affects the final solution, making possible the algorithm to find sub-optimal solutions. The same thing happens with the pruning step. Another issue with the pruning step is that it removes a group only if all instances in that group are well classified without the group representant. A higher generalization might be achieved if there was a threshold that evaluated if a removal is an advantage or not.

When inserting all prototypes of the minority class, after the pruning step, the ASGP method might also insert not needed prototypes, and even, prototypes that does not fit the new data, generating an overlap between the prototypes of the minority and majority classes.

The next section presents the proposed technique that handles all the mentioned issues.

III. Evolutionary Adaptive Self-Generating Prototypes

This section presents the Evolutionary Adaptive Self-Generating Prototypes (EASGP), a prototype generation (PG) that uses an iterative merge and evolutionary pruning.

A. Motivation

The Self-Generating Prototypes (SGP) has a poor performance when trained with imbalanced datasets. The Adaptive Self-Generating Prototypes (ASGP) is an improved SGP that handles imbalanced datasets. Despite the fact that ASGP achieved better results than SGP2, ASGP has a lower reduction rate than SGP2. This behavior is acceptable, because of high cost of misclassifying the minority class [24], but it is not desired.

The following three flaws were found in the SGP2 and ASGP algorithms:

1) In the merging step, the order in which the groups are merged affects the resulting prototypes. If two groups are merged, another important merge might not take place, and a sub-optimal solution might be returned.
2) In the pruning step, the already reduced search space is not fully explored. Also, when the order in which the groups are visited is changed, the resulting prototypes also change. Because of that, a sub-optimal solution might be returned.
3) After the merging and pruning steps, when introducing back the prototypes of the minority class, the ASGP algorithm does not consider that the new set of majority class prototypes were generated considering another group of the minority class prototypes. This might cause overlapping between prototypes of different classes.

In order to solve these issues, this paper proposes the EASGP method, an SGP based PG technique that implements an incremental merging and evolutionary pruning steps.

B. Architecture

Figure 1 shows the architecture of EASGP. First, ASGP is used to generate the initial prototypes. These prototypes are used by the iterative merging algorithm, generating new prototypes and expanding the search space. Finally, the evolutionary pruning is applied to find the optimal subset of prototypes, and the best subset is returned. The evolutionary pruning uses the original training set as a validation set to find the best solution.

The EASGP steps are explained in the next subsections.

C. EASGP Initial Prototype Generation

Following the same approach of the SGP algorithm, In the beginning of the process, a group containing all the instances within each class is created for all class labels, and the mean of each group is elected representant. The ASGP main loop is executed once the initial groups are formed until no changes in the groups occurs.

The generalization step follows the ASGP approach, removing a group only if it is considerably smaller than the larger group of the same class.

The major inovations of EASGP are the iterative merging and evolutionary pruning, presented in the next subsections.

D. Iterative Merging

Differently than the merging step of ASGP, the iterative merging uses only the generated prototypes to perform the merge between two groups, instead of using all instances within the groups. This approach has a high performance effect on the algorithm, since the number of representants is usually significantly smaller than the number of instances in the training set.

In order to understand the iterative merging, the *merging-links* concept must be defined. A pair of prototypes A and B is called a *merging-link* if they meet the 3 following conditions:

- A and B belong to the same class.
- The closest prototype of A is B.
- The closest prototype of B is A.

The iterative merging also uses two lists: *search list* and *final list*. The *search list* is the list of prototypes where the procedure searches for *merging-links*. The *final list* is the list of prototypes to be returned by the procedure.

First, the iterative merging insert all prototypes from the EASGP initial prototypes procedure in the *search list* and then finds all *merging-links* prototypes in that list. If at least one *merging-link* is found, than this is not the last iteration. Each link is merged, and a new prototype is generated (the mean of the each link). The *merging-links* are now removed from *search list* and saved into the *final list*. The process is repeated until the last generation (no *merging-link* is found), when all remaining prototypes are included in the *final list*. Finally, all prototypes in *final list* are returned.

The iterative merging is detailed in Algorithm 2.

Algorithm 2 Iterative Merging

Require: PS: a set of prototypes
Require: *search list*: a list to search for links
Require: *final list*: a list to save the prototypes
1: save all PS in *search list*
2: *merging-links* $\leftarrow$ all merging links in *search list*
3: **while** *merging-links* is not empty **do**
4: **for all** *link* $\in$ *merging-links* **do**
5: $P \leftarrow mean(link)$
6: insert P in the *search list*
7: remove the *link* prototypes from *search list*
8: insert the *link* prototypes in *final list*
9: **end for**
10: *merging-links* $\leftarrow$ all merging links in *search list*
11: **end while**
12: insert all prototypes in *search list* in *final list*
13: **return** the prototypes in *final list*

With this approach, the EASGP iterative merging expands the region of search that was reduced by the initial prototypes procedure. The result of the iterative merging is not affected by the order in which the prototypes are merged (differently than the ASGP merge step).

E. Evolutionary Pruning

The pruning step aims to remove redundant prototypes without compromising the classification accuracy. The problem of pruning can be considered a problem of prototype selection (PS), which is to find the best subset of instances that better represent a training set. As mentioned in the previous section, the problem of PS is a combinatorial problem, and the current best approaches to solve this problem is to use evolutionary prototype selection (EPS), especially memetic algorithms (MA).

The evolutionary pruning uses the Steady-State Memetic Algorithm (SSMA) [14] concept to find the best subset of generated prototypes. First, a population of solutions is created using the chromosome representation. For each interection, two parent chromosomes are selected and the genetic operators are applied to generate an offspring. A local search is used to optimize each solution with a given probability. The evolutionary pruning uses the fitness function detailed the Equation 1.

$$Fitness(S) = \alpha \times \text{ AUC}_{rate} + (1 - \alpha) \times \text{ reduction}_{rate} \quad (1)$$

The value of the parameter α is within the interval $[0.51, 0.99]$. In the evolutionary pruning, the classification rate used in the fitness when handling imbalanced dataset is the Area Under the ROC Curve (AUC), avoiding over generalization and the possibility of removing of all prototypes of the minority class.

Also, the fitness is estimated using the original training set as a validation set, not only the prototypes from the merging step. Using this approach, the evolutionary pruning finds better solutions and avoids the cost of misclassification of already removed samples in the previous steps of the algorithm.

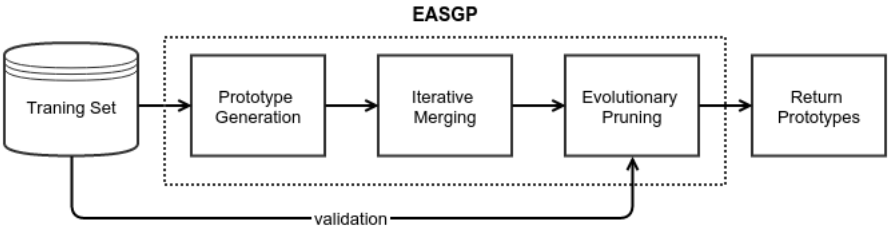

Fig. 1. Architecture of EASGP.

Algorithm 3 Evolutionary Pruning

```
Initialize Population.
while not termination-condition do
    parents ← Parent Selection (binary tournament)
    Off_1, Off_2 ← Crossover(parents)
    Mutation(Off_1, Off_2)
    for all Off_i do
        if local search decision then
            local-search(Off_i)
        end if
    end for
    Standard Replacement for Off_1 and Off_2
end while
return The best chromosome
```

The Algorithm 3 presents the evolutionary pruning algorithm.

The evolutionary pruning steps are detailed as follows:

- **Population Initialization**: Each chromosome represents a subset of prototypes, a gene is '1' when the prototype is in the subset and '0' when it is not. All chromosomes reference the same set of prototypes returned by the iterative merging. In the population inicialization, the chromosomes are inicialized randomly, and each chromosome is evaluated using the fitness function. Differently than other approaches, the evaluation is performed using all instances in the training set as a validation set, and not only the prototypes referenced by the chromosomes.
- **Parent Selection**: In order to select two parents, a binary tournament selection is employed. For each parent, two random candidates are selected from the population of chromosomes, and the best one (the one with the higher fitness) is elected parent.
- **Crossover**: The parents are combined to generate the offspring, two new individuals with half of the genes from each parent.
- **Mutation**: The mutation changes each gene of the offspring with a probability $P = 1/N$, where N is the size of the chromosome.
- **Offspring Evaluation**: The offspring is evaluated using the fitness function (Equation 1) and the validation set (the original training set).
- **Local Search Decision**: This step decides if a local search is applied to an individual in the offspring. The local search is performed with a probability P_{ls} which is detailed in the Equation 2.

$$P_{ls}(S) = \begin{cases} 1 & \text{if fitness}(S) > \text{fitness}(S_{worse}) \\ 0.0625 & \text{otherwise} \end{cases} \quad (2)$$

If the offspring is better than the worst solution of the population of chromosomes (has a higher fitness than the solution with lower fitness in the population), the local search is performed, otherwise, the local search is performed with a probability $P_{ls} = 0.0625$. This value was found empirically in [14].

- **Local Search**: For a given chromosome, it considers neighborhood solutions by removing an instance from the current solution. A change is maintained if it improves the classification accuracy of the current solution, otherwise, the change is reverted. If a removal becomes permanent, the whole neighborhood is considered again. To avoid local optimum, the local search also accepts solutions that decreases the classification accuracy but increase the fitness.
- **Standard Replacement**: If the offspring is better than the worst solution in the population (has a higher fitness than the solution with lower fitness in the population), the offspring replaces the worst solution.
- **Termination Condition**: The algorithm stops when a convergence of the solutions occurs, or when a number of evaluations NE (passed as parameter) is reached. In other evolutionary algorithms, usually $NE = 10000$. Because of the high reduction power of ASGP, $NE = 100$ is enough for EASGP.

The evolutionary pruning procedure works as an optimization algorithm. Like in the SSMA algorithm, the use of a local search makes it possible to find optimal solutions without reach the number of evaluations that a brute-force algorithm requires.

Compared to SSMA alone, one advantage of EASGP is that EASGP makes possible a higher generalization and requires fewer evaluations than other evolutionary algorithms, since the chromosomes only represent the previously generated prototypes and not the whole original training set. In classification accuracy, EASGP pruning also works as a fixer for the previous steps, removing any residual noisy data generated by the initial generation (ASGP main procedure) and the iterative merging.

IV. EXPERIMENTS

This section presents the methodology used in the experiments, and the results of the Evolutionary Adaptive Self-Generating Prototypes (EASGP).

A. Methodology

The EASGP method is evaluated using 15 imbalanced datasets from KEEL [25]. The datasets are binary (2 classes) and have different levels of imbalance. Table I summarises the datasets used in this study, detailing the number of samples, the number of attributes, the class distribution and the imbalance ratio (IR). The datasets are partitioned using the *five fold cross-validation* procedure, respecting the classes proportions.

TABLE I. DATASETS CHARACTERISTICS

Label	Dataset	#Attributes	#Instances	IR
1	pima	8	768	1.87
2	yeast1	8	1484	2.46
3	vehicle2	18	846	2.88
4	vehicle1	18	846	2.9
5	vehicle3	18	846	2.99
6	vehicle0	18	846	3.25
7	ecoli2	7	336	5.46
8	segment0	19	2308	6.02
9	ecoli3	7	336	8.6
10	yeast05679vs4	8	528	9.35
11	vowel0	13	988	9.98
12	glass016vs2	9	192	10.29
13	glass2	9	214	11.59
14	shuttlec0vsc4	9	1829	13.87
15	yeast1vs7	7	459	14.3

The evaluation metrics are: Area Under the ROC Curve (AUC) and reduction rate. To compare the results, we use the *One Sided Wilcoxon Rank Test* [15], with significance level $\alpha = 0.05$. Table II presents the techniques and parameters used in this experiment. The 1NN was the base classifier used for all techniques.

TABLE II. PARAMETERS USED IN THE EXPERIMENTS.

Algorithm	Parameters
SGP	$R_{min} = 0.20, R_{mis} = 0.20$
SGP2	$R_{min} = 0.20, R_{mis} = 0.20$
ASGP	$R_{min} = 0.20, R_{mis} = 0.20$
EASGP	$R_{min} = 0.20, R_{mis} = 0.20, \alpha = 0.97$

B. Results

Table III and Table IV are grouped in columns by algorithms, and the best result of each dataset is highlighted in bold. The last lines presents the results of the Wicoxon Test, considering $\alpha_{wilcoxon} = 0.05$, the symbol "+" is used when EASGP outperforms the algorithm in that column (NA means *not applicable*).

1) AUC: Table III shows the average and standard deviation of the classification accuracy (given by the AUC). The results show that EASGP outperformed all previous version of SGP with statistical confidence: SGP (*p-value* $= 0.0003275$), SGP2 (*p-value* $= 0.0003275$) and ASGP (*p-value* $= 0.007511$).

Figure 2 shows EASGP compared with the best of the other techniques. This figure shows that EASGP outperforms the best of SGP, SGP2 and ASGP in classification accuracy, being losing in only 2 datasets. Clearly, EASGP achieved the best performance in classification accuracy (AUC).

TABLE III. AVERAGE, STANDARD DEVIATION AND WILCOXON SIGNED RANK TEST P-VALUE AND RESULT OF THE SGP, SGP2, ASGP AND EASGP AUC RATE

Label	SGP	SGP2	ASGP	EASGP
1	0.6468(0.0372)	0.6508(0.0443)	0.6451(0.0713)	**0.6718(0.0607)**
2	0.6062(0.0347)	0.5950(0.0396)	0.6234(0.0168)	**0.6261(0.0084)**
3	0.7935(0.0456)	0.7874(0.0402)	**0.8400(0.0527)**	0.8146(0.0705)
4	0.5943(0.0573)	0.6070(0.0552)	**0.6425(0.0326)**	0.6393(0.0297)
5	0.6460(0.0299)	0.6437(0.0420)	0.6738(0.0366)	**0.6934(0.0471)**
6	0.7922(0.0217)	0.8020(0.0297)	0.8020(0.0255)	**0.8253(0.0373)**
7	0.5000(0.0000)	0.5000(0.0000)	**0.8986(0.0503)**	**0.8986(0.0503)**
8	0.6898(0.1323)	0.6850(0.1389)	0.8564(0.0657)	**0.8875(0.0620)**
9	0.5028(0.0063)	0.5187(0.0419)	0.8531(0.0681)	**0.8866(0.0610)**
10	0.5281(0.0629)	0.5281(0.0629)	0.7986(0.0485)	**0.8128(0.0444)**
11	0.5000(0.0000)	0.5000(0.0000)	**0.8588(0.0601)**	**0.8588(0.0601)**
12	0.5545(0.0792)	0.5069(0.0976)	0.6388(0.1886)	**0.6693(0.1050)**
13	0.6291(0.1284)	0.6215(0.1288)	0.6674(0.1109)	**0.7240(0.0742)**
14	0.5000(0.0000)	0.5000(0.0000)	**0.9960(0.0089)**	**0.9960(0.0089)**
15	0.5817(0.1328)	0.5677(0.1423)	0.6553(0.0524)	**0.7615(0.0945)**
Mean	0.6043(0.0512)	0.6009(0.0576)	0.7633(0.0593)	**0.7844(0.0543)**
p-value	0.0003275	0.0003275	0.007511	NA
Result	+	+	+	NA

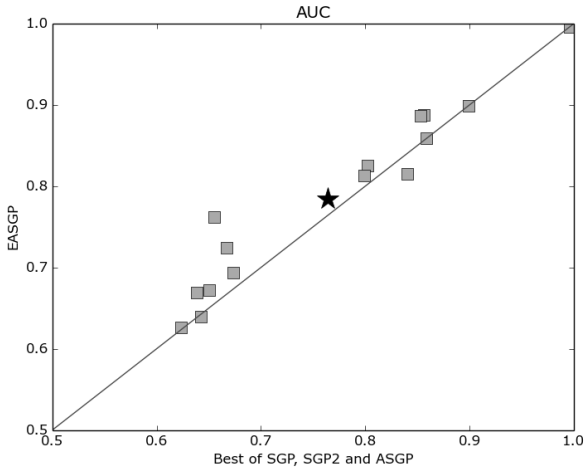

Fig. 2. Best of SGP, SGP2 and ASGP × EASGP AUC rate graph, where the squares are the datasets and the star is the average.

2) Reduction: Table IV shows the average and standard deviation of the reduction rate. The results show that EASGP outperformed all previous versions of SGP with statistical confidence: SGP (*p-value* $= 0.00143$), SGP2 (*p-value* $= 0.02341$), and ASGP (*p-value* $= 0.001109$).

The impressive result is that EASGP outperformed SGP2 in reduction rate. Based on previous studies [13], we concluded that SGP2 have a high reduction power, but sometimes it removes all samples of the minority class, which allows the algorithm to leave only a prototype of the majority class. Despite of that, EASGP was able to outperform SGP2 in reduction, without compromising classification accuracy.

Figure 3 shows EASGP compared with the best of the other techniques. This figure shows that EASGP outperforms the best of SGP, SGP2 and ASGP in reduction rate, losing in only 5 datasets. Even in the few datasets where EASGP was outperformed, we can see that the points are very close to the line, which means the difference is very small.

TABLE IV. AVERAGE, STANDARD DEVIATION AND WILCOXON SIGNED RANK TEST P-VALUE AND RESULT OF THE SGP, SGP2, ASGP AND EASGP REDUCTION RATE

Dataset	SGP	SGP2	ASGP	EASGP
1	0.7689(0.0125)	0.8333(0.0097)	0.7939(0.0108)	**0.9385(0.0234)**
2	0.7675(0.0073)	0.8491(0.0087)	0.8031(0.0053)	**0.9528(0.0227)**
3	0.9507(0.0076)	0.9663(0.0050)	0.9563(0.0085)	**0.9814(0.0039)**
4	0.8304(0.0042)	0.8785(0.0109)	0.8516(0.0205)	**0.9601(0.0134)**
5	0.8274(0.0064)	0.8738(0.0043)	0.8381(0.0055)	**0.9323(0.0316)**
6	0.9474(0.0053)	0.9607(0.0062)	0.9486(0.0132)	**0.9843(0.0049)**
7	**0.9963(0.0000)**	**0.9963(0.0000)**	0.9926(0.0000)	0.9926(0.0000)
8	0.9967(0.0009)	0.9977(0.0010)	0.9963(0.0012)	**0.9979(0.0003)**
9	0.9829(0.0301)	0.9866(0.0217)	0.9792(0.0301)	**0.9889(0.0084)**
10	0.9805(0.0381)	0.9829(0.0328)	0.9754(0.0445)	**0.9901(0.0117)**
11	**0.9987(0.0000)**	**0.9987(0.0000)**	0.9975(0.0000)	0.9975(0.0000)
12	0.8854(0.0193)	**0.9427(0.0266)**	0.8737(0.0117)	0.9349(0.0220)
13	0.8960(0.0128)	**0.9311(0.0192)**	0.8878(0.0097)	0.9275(0.0098)
14	**0.9993(0.0000)**	**0.9993(0.0000)**	0.9986(0.0000)	0.9986(0.0000)
15	0.8954(0.0184)	0.9466(0.0170)	0.9096(0.0136)	**0.9543(0.0121)**
Mean	0.9149(0.0109)	0.9429(0.0109)	0.9201(0.0116)	**0.9688(0.0109)**
p-value	0.00143	0.02341	0.001109	NA
Result	+	+	+	NA

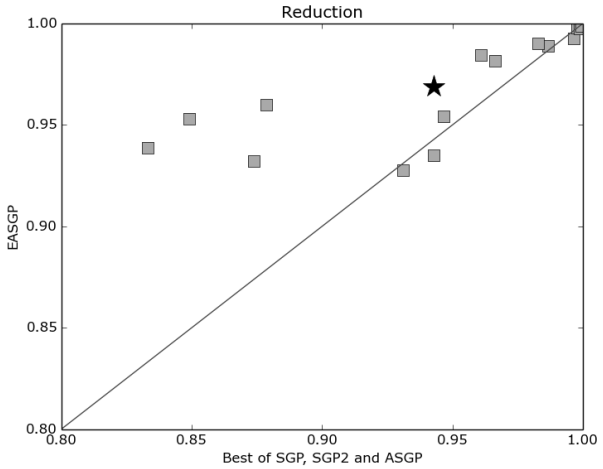

Fig. 3. Best of SGP, SGP2 and ASGP $\times$ EASGP reduction rate graph, the squares are the datasets and the star is the average.

We can conclude with confidence that EASGP achieved the best peformance in reduction rate.

3) Reduction vs. Classification: Figure 4 shows the dispersion graph (Reduction vs. AUC) of SGP, SGP2, ASGP and EASGP. This figure shows that EASGP achieved the best classification accuracy and reduction rate.

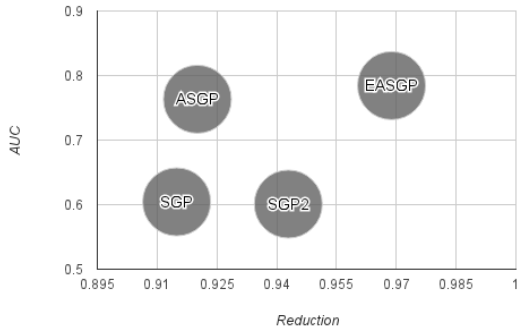

Fig. 4. Dispersion (Reduction vs. AUC) of EASGP, ASGP, SGP and SGP2.

We can state with confidence that EASGP outperformed all previous versions of SGP (SGP, SGP2 and ASGP) in both, classification accuracy and reduction rate.

V. CONCLUSION

This paper presented the Evolutionary Adaptive Self-Generating Prototypes (EASGP), a centroid based prototype generation (PG) algorithm for imbalanced datasets. EASGP uses an iterative merging to expand the search space, and evolutionary pruning to find the optimal solution.

An experimental study was carried out to compare EASGP and the previous versions of the Self-Generationg Prototupes (SGP). The main conclusions reached were:

1) EASGP outperformed all previous versions of SGP in classification accuracy on imbalanced datasets.
2) EASGP outperformed all previous versions of the SGP in reduction rate on imbalanced datasets.
3) Differently than previous versions of SGP, EASGP can be adjusted to give preference to the classification or reduction with the α parameter.

The use of an iterative merging and evolutionary pruning achieved excelent results. Future works include the use of these algorithms with other PG techniques.

ACKNOWLEDGMENT

The authors would like to thank CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, in portuguese), CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico, in portuguese) and FACEPE (Fundação de Amparo à Ciência e Tecnologia do Estado de Pernambuco, in portuguese).

REFERENCES

[1] T. Cover and P. Hart, "Nearest neighbor pattern classification," *Information Theory, IEEE Transactions on*, vol. 13, no. 1, pp. 21–27, 1967.

[2] E. A. Patrick and F. Fischer, "A generalization of the k-nearest neighbor rule," in *Proceedings of the 1st international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., 1969, pp. 63–63.

[3] G. Shakhnarovich, T. Darrell, and P. Indyk, *Nearest-neighbor methods in learning and vision: theory and practice*. MIT press Cambridge, MA, USA:, 2005, vol. 3.

[4] I. Kononenko and M. Kukar, *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing Limited, 2007.

[5] S. García, J. Derrac, J. R. Cano, and F. Herrera, "Prototype selection for nearest neighbor classification: Taxonomy and empirical study," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 3, pp. 417–435, 2012.

[6] I. Triguero, J. Derrac, S. García, and F. Herrera, "A taxonomy and experimental study on prototype generation for nearest neighbor classification," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 42, no. 1, pp. 86–100, 2012.

[7] F. Fernández and P. Isasi, "Local feature weighting in nearest prototype classification," *Neural Networks, IEEE Transactions on*, vol. 19, no. 1, pp. 40–53, 2008.

[8] D. R. Wilson and T. R. Martinez, "Reduction techniques for instance-based learning algorithms," *Machine learning*, vol. 38, no. 3, pp. 257–286, 2000.

[9] Q. Yang and X. Wu, "10 challenging problems in data mining research," *International Journal of Information Technology & Decision Making*, vol. 5, no. 04, pp. 597–604, 2006.

[10] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Information Sciences*, vol. 250, pp. 113–141, 2013.

[11] H. A. Fayed, S. R. Hashem, and A. F. Atiya, "Self-generating prototypes for pattern classification," *Pattern Recognition*, vol. 40, no. 5, pp. 1498–1509, 2007.

[12] C. de Santana Pereira and G. D. C. Cavalcanti, "Prototype selection: Combining self-generating prototypes and gaussian mixtures for pattern classification," in *Proceedings on International Joint Conference on Neural Networks*. IEEE, 2008, pp. 3505–3510.

[13] D. V. R. Oliveira, G. R. Magalhaes, G. D. C. Cavalcanti, and T. I. Ren, "Improved self-generating prototypes algorithm for imbalanced datasets," in *Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on*, vol. 1. IEEE, 2012, pp. 904–909.

[14] S. García, J. R. Cano, and F. Herrera, "A memetic algorithm for evolutionary prototype selection: A scaling up approach," *Pattern Recognition*, vol. 41, no. 8, pp. 2693–2709, 2008.

[15] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics*, vol. 1, no. 6, pp. 80–83, 1945.

[16] H. Liu and H. Motoda, "On issues of instance selection," *Data Mining and Knowledge Discovery*, vol. 6, no. 2, pp. 115–130, 2002.

[17] J. R. Cano, F. Herrera, and M. Lozano, "Using evolutionary algorithms as instance selection for data reduction in kdd: an experimental study," *Evolutionary Computation, IEEE Transactions on*, vol. 7, no. 6, pp. 561–575, 2003.

[18] J. Derrac, S. García, and F. Herrera, "Stratified prototype selection based on a steady-state memetic algorithm: a study of scalability," *Memetic Computing*, vol. 2, no. 3, pp. 183–199, 2010.

[19] F. Fernández and P. Isasi, "Evolutionary design of nearest prototype classifiers," *Journal of Heuristics*, vol. 10, no. 4, pp. 431–454, 2004.

[20] L. Nanni and A. Lumini, "Particle swarm optimization for prototype reduction," *Neurocomputing*, vol. 72, no. 4, pp. 1092–1097, 2009.

[21] U. Garain, "Prototype reduction using an artificial immune model," *Pattern analysis and applications*, vol. 11, no. 3-4, pp. 353–363, 2008.

[22] I. Triguero, S. García, and F. Herrera, "Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification," *Pattern Recognition*, vol. 44, no. 4, pp. 901–916, 2011.

[23] J. Derrac, I. Triguero, C. J. Carmona, F. Herrera *et al.*, "Evolutionary-based selection of generalized instances for imbalanced classification," *Knowledge-Based Systems*, vol. 25, no. 1, pp. 3–12, 2012.

[24] C. Elkan, "The foundations of cost-sensitive learning," in *International joint conference on artificial intelligence*, vol. 17, no. 1. Citeseer, 2001, pp. 973–978.

[25] J. Alcalá, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, no. 2-3, pp. 255–287, 2010.