Applied Soft Computing 44 (2016) 75–88

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Improving nearest neighbor classification using Ensembles of Evolutionary Generated Prototype Subsets

Nele Verbiest[a], Sarah Vluymans[a,b,*], Chris Cornelis[a,c], Nicolás García-Pedrajas[d], Yvan Saeys[b,e]

[a] Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Belgium
[b] VIB Inflammation Research Center, Ghent, Belgium
[c] Department of Computer Science and Artificial Intelligence, University of Granada, Spain
[d] Department of Computer Science and Numerical Analysis, University of Córdoba, Spain
[e] Department of Respiratory Medicine, Ghent University, Belgium

ARTICLE INFO

Article history:
Received 2 July 2015
Received in revised form 9 February 2016
Accepted 18 March 2016
Available online 6 April 2016

Keywords:
Classification
Evolutionary algorithms
K Nearest Neighbor
Ensembles

ABSTRACT

One of the most accurate types of prototype selection algorithms, preprocessing techniques that select a subset of instances from the data before applying nearest neighbor classification to it, are evolutionary approaches. These algorithms result in very high accuracy and reduction rates, but unfortunately come at a substantial computational cost. In this paper, we introduce a framework that allows to efficiently use the intermediary results of the prototype selection algorithms to further increase their accuracy performance. Instead of only using the fittest prototype subset generated by the evolutionary algorithm, we use multiple prototype subsets in an ensemble setting. Secondly, in order to classify a test instance, we only use prototype subsets that accurately classify training instances in the neighborhood of that test instance. In an experimental evaluation, we apply our new framework to four state-of-the-art prototype selection algorithms and show that, by using our framework, more accurate results are obtained after less evaluations of the prototype selection method. We also present a case study with a prototype generation algorithm, showing that our framework is easily extended to other preprocessing paradigms as well.

1. Introduction

Classification, the task of labeling instances described by features using already labeled training data, is an important field in data mining with numerous applications in research and industry. A widely used and easy to understand classification method is K Nearest Neighbors (KNN, [1]). In order to classify an unlabeled instance, KNN looks up the K instances in the training data closest to it and assigns the instance to the majority class among these nearest neighbors.

One of the main drawbacks associated with KNN classification is that the lazy learning nature of KNN implies that any instance can have an impact on the classification of new instances. As a result, KNN is highly susceptible to noise, which often occurs in datasets drawn from real-world situations due to, for instance, human annotation, measuring errors or data transmission. Furthermore, storing the entire training set can imply high storage needs and look-up times.

A successful solution to these problems is Prototype Selection (PS, [2]). This preprocessing technique selects a subset of the data in order to improve the accuracy of the KNN classification applied afterwards or to reduce the storage requirements significantly. Many PS methods have been proposed in the literature. An extensive overview and taxonomy can be found in [3]. They are most commonly combined with the 1NN classifier, which is most sensitive to noise. We therefore also focus on this classifier here. Evolutionary approaches to PS [4–7] have proven to be the best among state-of-the-art PS methods, both with respect to accuracy and reduction [3]. They are able to improve the classification accuracy of 1NN significantly and at the same time reduce the data by up to 90 percent. The combination of this type of methods and nearest neighbor classification has consequently been widely explored (e.g. [8–13]). The good performance of evolutionary PS algorithms is achieved by producing and evaluating many generations of candidate prototype subsets. Unfortunately, this means that they tend to be slow.

* Corresponding author at: Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Belgium.
E-mail address: Sarah.Vluymans@UGent.be (S. Vluymans).

http://dx.doi.org/10.1016/j.asoc.2016.03.015
1568-4946/

In this paper we propose a classification framework, called Ensembles of Evolutionary Generated Prototype Subsets (EEGPS), that draws more information out of the long calculations that evolutionary PS methods make. During the course of an evolutionary PS algorithm, many good prototype subsets are encountered, but only the overall best prototype subset is used. However, candidate prototype subsets that are globally not optimal might still be useful to classify instances in certain regions of the feature space. The PS algorithms solely recognize that these subsets are not the overall best and discard them without further question. The large computational cost of the algorithms can be more appropriately used by storing them for later use. The sub-optimal subsets have been constructed and evaluated anyway, there is little or no additional cost in saving them. EEGPS implements this idea by using multiple prototype subsets, that were generated during the execution of an evolutionary PS algorithm. In order to classify a test instance, the EEGPS classifier uses an ensemble of prototype subsets that perform well in the region of that test instance, as opposed to using a single prototype set combined with KNN. As the EEGPS framework uses prototype subsets already generated by the PS algorithm, the additional running time needed to apply EEGPS is small. Moreover, we will experimentally demonstrate that when using the EEGPS framework, less evaluations of the PS algorithm are needed to obtain similar and even better results than when using the traditional PS setting. This proposal takes a clear step away from traditional PS algorithms described in [3], both evolutionary and non-evolutionary. All such methods lead to a single reduced training set, which is considered optimal based on certain internal evaluation criteria. This set is used in the classification of new instances. None of these methods yield multiple subsets that can be set up in an ensemble. Naturally, applying them several times to the same training set (with varying parameter values) can yield multiple different subsets. The point of our development of EEGPS is that we do not need to apply an existing PS method multiple times with additional parameter tuning, but rather that we can readily use information already generated by one application of the best performing PS methods: the evolutionary algorithms. The main novelty and contributions of this work are the following:

1. We propose the recovery of information generated by evolutionary PS methods, which is traditionally entirely discarded. Instead of solely keeping track of the fittest individual in the population, we store every encountered candidate. Afterwards, we select a good-performing and diverse set from among them. This set is used rather than the single fittest individual, thereby maximally exploiting the large computational cost of the evolutionary method.
2. We develop a custom classification ensemble based on the selected group of prototype subsets. Within this ensemble, the subsets are dynamically weighted, based on the test instance to classify, in order to assign more weight to subsets performing well in the neighborhood of the test instance.
3. We experimentally show that the additional cost of the EEGPS framework compared to the traditional setting is negligible and that its application can lead to an increased classification accuracy.

The remainder of this paper is organized as follows. In Section 2 we provide the reader with the basics of evolutionary PS algorithms. In Section 3 we present the EEGPS framework and in Section 4 we experimentally demonstrate the good performance of EEGPS. We conclude our experimental study in Section 5 with an example application of how our framework can be extended to other evolutionary data reduction algorithms as well. We conclude and point out future research directions in Section 6.

2. Preliminaries: evolutionary prototype selection

In this section we provide the background on evolutionary PS algorithms necessary for the understanding of the remainder of the paper and for the interpretation of the results obtained in the experimental evaluation. We first discuss the general framework of evolutionary PS techniques [14] and afterwards the four state-of-the-art techniques used in our experiments.

The core component of an evolutionary algorithm is a population that changes over several generations. It consists of individuals represented by chromosomes. The evolution is guided by genetic operators: recombination (crossover) and mutation. Each chromosome represents a solution. How good a solution it is, is evaluated by the so-called fitness function. It assesses the quality of the individuals and is the main driver of the evolutionary algorithm.

In the remainder, we denote the training set by T. For evolutionary PS methods, chromosomes correspond to prototype subsets of T and are encoded as binary strings of length $|T|$. A 1 on the ith position means that the ith instance of T is included in the prototype subset, a 0 means that it is not. All evolutionary PS methods discussed in this paper randomly initialize the population, that is, random binary strings of length $|T|$ constitute the initial population.

The fitness function used in the four evolutionary PS algorithms recalled below consists of two components, the leave-one-out training accuracy $\mathrm{acc}(S)$ associated with the prototype subset $S \subseteq T$ and the reduction $\mathrm{red}(S)$ that compares the size of S to the size of the original training set T. The fitness function balances these two components using a parameter $\alpha \in [0, 1]$ as follows:

$$\mathrm{fitness}(S) = \alpha \cdot \mathrm{acc}(S) + (1 - \alpha) \cdot \mathrm{red}(S). \quad (1)$$

As the fitness function incorporates both the accuracy and the reduction, the evolutionary PS method finds small prototype subsets that result in accurate predictions.

All considered PS methods use the same mutation operator that flips the bit in the gene with a given probability. This probability is kept low, as this operator is based on the rarer phenomenon of mutation in nature. The algorithms halt when a predetermined number of fitness evaluations is reached. The other components of the evolutionary algorithm scheme are different for each specific evolutionary PS method; we discuss them in detail below.

We recall four state-of-the-art evolutionary-based PS methods. These algorithms perform very well on a broad selection of datasets, as experimentally demonstrated in [3]. We first discuss two basic algorithms, the Generational Genetic Algorithm (GGA, [5,6]) and the Steady State Genetic Algorithm (SSGA, [4]). These algorithms both follow the general scheme of evolutionary algorithms, but they differ by the fact that SSGA only generates two new individuals in each generation, whereas GGA replaces a more substantial part of its population. Next, we discuss the Steady State Memetic Algorithm (SSMA, [7]), that adds an optimization phase to the SSGA algorithm, and the Adaptive Search for Instance Selection (CHC, [4]) algorithm that extends GGA.

2.1. Generational Genetic Algorithm (GGA, [5,6])

Parent selection in the GGA algorithm uses a stochastic procedure where fitter individuals in the population have a higher chance of being selected. There are as many parents selected as there are individuals in the population, which means that some individuals can be selected multiple times. These parents are randomly matched in pairs. Crossover happens between these parents with a given probability. The two-point crossover operator is used, which exchanges one part of the chromosome between the two parents. The generated offspring consists of the children resulting from the crossover procedure and the parents that were selected but did not undergo crossover. Elitism is applied, meaning that

the best individual in the previous generation is copied to the new generation without any changes. The entire offspring, except the least fit individual, survives and goes to the next generation.

2.2. Steady State Genetic Algorithm (SSGA, [4])

The SSGA algorithm selects two parents in each generation. Each parent is selected by means of a binary tournament, which means that two individuals are randomly picked from the population and the fittest one is selected. These parents are recombined using the two-point crossover operator. The recombination does not depend on a given probability, but is always performed instead. The newly generated individuals replace the two least fit ones from the current population, provided the former attain a higher fitness value.

2.3. Steady State Memetic Algorithm (SSMA, [7])

SSMA proceeds like SSGA, except that after the recombination and mutation phase, each of the generated children undergoes a local optimization. If the fitness of the child is higher than a certain threshold, the optimization is performed. In the other case, the optimization only takes place with a low probability. In this way, the optimization is only carried out if the generated child is likely to be included in the next generations and superfluous calculations are avoided. The local optimization procedure considers each instance in the prototype subset corresponding to the child and removes it if the gain in training accuracy of the new prototype subset is higher than a certain threshold.

2.4. Adaptive Search for Instance Selection (CHC, [4])

In the same way as GGA, the CHC procedure selects all individuals in the population as parents. These parents are randomly paired and recombination takes place if the number of genes in which two parents differ is higher than a certain threshold. By promoting diversity, it is ensured that the evolutionary procedure does not get prematurely stuck in a local optimum. The threshold is fixed to a fourth of the size of the training data, but this number is decreased when no parents can be matched. The crossover operator randomly interchanges half of the bits in which the parents differ and always takes place. The generated children are merged with the original population to the offspring and the fittest p individuals survive, with p the size of the population. If the new generation does not differ enough from the previous one, the population is re-initialized by taking the fittest individual and copying it p times. In each copy a certain percentage of bits is set to 1 with a given probability.

3. Proposed framework: Ensembles of Evolutionary Generated Prototype Subsets (EEGPS)

During the application of an evolutionary PS algorithm many good and diverse prototype subsets are encountered, but only the final fittest one is used for the classification of test instances. The effort put in the construction and evaluation of the intermediate candidates is therefore not fully taken advantage of. The first idea of EEGPS is to use multiple good prototype subsets encountered during the PS algorithm to classify test instances. Even though these prototype subsets are globally not optimal, they may still have good properties and can be useful for classification. The second idea of EEGPS is that prototype subsets can be good to classify test instances in a particular region of the feature space, but that they are less suited to classify test instances in other regions. Using a single prototype subset neglects this idea. When using multiple prototype subsets, one can guide which prototype subsets to use to classify a test instance, depending on which of them perform well in the region of that test instance.

The EEGPS framework encapsulates both ideas. Below, we use the notation M for the evolutionary PS method at hand and set $E(S)$ to the value of the evaluation function of M for a prototype subset S. In our case, where evolutionary PS methods are used, $E(\cdot)$ corresponds to the fitness function. The general workflow of the EEGPS framework is depicted in Fig. 1.

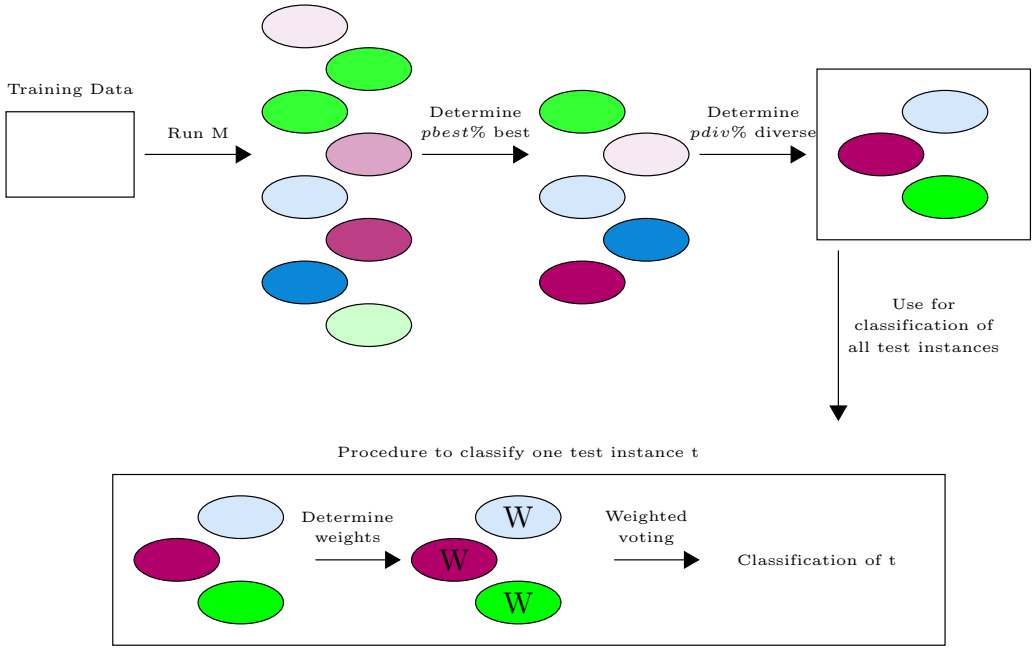

Fig. 1. Workflow of the EEGPS framework: the evolutionary PS algorithm M is carried out and the *pdiv* most diverse solutions are selected from among the *pbest%* fittest solutions that were generated by M. These *pdiv* solutions are used to classify test instances. In order to classify one test instance, the solutions get a weight that is used in the weighted voting strategy.

3.1. Description of EEGPS

The main steps of the EEGPS framework are summarized below. Before the classification of test instances takes place, the following steps are carried out:

1. **Determine *best*:** Execute M and store the *pbest%* prototype subsets with the highest values for E in *best*.
2. **Determine *div*:** Select the *pdiv%* most diverse prototype subsets among *best* and store them in *div*.

The parameters *pbest* and *pdiv* are user-defined. These two steps set up the prototype subsets used in the ensemble. Once they are carried out, the classification of test instances can start. For each individual test instance t, the selected prototype sets are assigned custom weights, to appropriately classify this particular instance. In order to classify t, the following steps are performed:

3. **Determine weights:** Assign a weight $W(S)$ to each prototype subset S in *div* that expresses to what extent S is suited for classification of instances in the neighborhood of size *nk* of t.
4. **Weighted voting:** Classify t using all the prototype subsets in *div* and use a weighted voting strategy to determine its final classification label.

The parameter *nk* is user-defined. Below, we discuss the separate steps in more detail.

Determine best. The evolutionary algorithm M is carried out as usual, except that all prototype subsets encountered are stored in a set during the execution. When M terminates, the *pbest%* prototype subsets with the highest fitness value are determined and stored in *best*.

Algorithm 1. Procedure to measure the diversity between two prototype subsets.

Input: Training set T, prototype subsets $S_1, S_2 \subseteq T$
Output: $diversity(S_1, S_2)$
1: $n_{00} \leftarrow 0$, $n_{01} \leftarrow 0$, $n_{10} \leftarrow 0$, $n_{11} \leftarrow 0$
2: **for each** $x \in T$ **do**
3: **if** x is classified correctly by 1NN, using S_1 **then**
4: **if** x is classified correctly by 1NN, using S_2 **then**
5: $n_{11} \leftarrow n_{11} + 1$
6: **else**
7: $n_{10} \leftarrow n_{10} + 1$
8: **else**
9: **if** x is classified correctly by 1NN, using S_2 **then**
10: $n_{01} \leftarrow n_{01} + 1$
11: **else**
12: $n_{00} \leftarrow n_{00} + 1$
13: $Q_{av}(S_1, S_2) \leftarrow \frac{n_{00}n_{11} - n_{01}n_{10}}{n_{00}n_{11} + n_{01}n_{10}}$
14: $diversity(S_1, S_2) \leftarrow 1 - Q_{av}(S_1, S_2)$

Determine div. The prototype subsets in *best* can contain many similar prototype subsets. It is important to select a diverse subset of prototype subsets from among *best*. As the prototype subsets are used for classification, we want to ensure that the classification using the different prototype subsets is diverse. In [15], an experimental study was carried out to compare measures of diversity in classifier ensembles. This study showed that the Q_{av} function, which measures the similarity between two classifiers, generally achieves good results. It is therefore used in this work. The procedure to measure the diversity between two prototype subsets S_1 and S_2 is given in Algorithm 1. The value n_{00} is the number of train instances misclassified by both S_1 and S_2, while n_{11} is the number of train instances correctly classified by S_1 and S_2. The remaining values n_{01} and n_{10} correspond to the number of instances correctly classified by one set and misclassified by the other. These values are calculated in lines 2–12. Once they are known, the Q_{av} measure can be calculated in line 13. Note that if S_1 and S_2 return exactly the same classification output, the value of $Q_{av}(S_1, S_2)$ equals one. As we are not interested in the similarity but in the diversity of the classification, we return one minus $Q_{av}(S_1, S_2)$ as the final diversity measure in line 14.

In order to obtain the most diverse prototype subsets among *best*, Algorithm 2 is used, setting the parameter **S** to *best*. Firstly, in line 1, the desired final number of prototype subsets is calculated based on the parameter *pdiv*. The set of most diverse prototype subsets *div* is initialized with the subset for which the evaluation function E is highest in line 2. This is the prototype subset that would have been used in the traditional PS setting. In lines 5–17, other prototype subsets are added until *div* has the desired size *ndiv*. The diversity between a prototype subset S and the set of prototype subsets already selected in *div* is defined as the sum of the diversities between S and the prototype subsets $P \in div$. This value is calculated in lines 10 and 11. In each iteration of the while loop, the prototype subset S_{best} for which the total diversity with *div* is maximal is added to it.

Algorithm 2. Procedure to select the most diverse prototype subsets among a set of prototype subsets.

Input: A set **S** of candidate prototype sets, parameter *pdiv*
Output: Set *div* of selected prototype sets
1: $ndiv \leftarrow pdiv \cdot |\mathbf{S}|$
2: $div \leftarrow \{S\}$, where S is the fittest individual in **S**
3: $\mathbf{S} \leftarrow \mathbf{S} \setminus \{S\}$
4: $ndiv \leftarrow ndiv - 1$
5: **while** $ndiv > 0$ **do**
6: $div_{max} \leftarrow 0$
7: $S_{best} \leftarrow null$
8: **for each** $S \in \mathbf{S}$ **do**
9: $div_{current} \leftarrow 0$
10: **for each** $P \in div$
11: $div_{current} \leftarrow div_{current} + diversity(S, P)$
12: **if** $div_{current} > div_{max}$ **do**
13: $div_{max} \leftarrow div_{current}$
14: $S_{best} \leftarrow S$
15: $\mathbf{S} \leftarrow \mathbf{S} \setminus S_{best}$
16: $div \leftarrow div \cup \{S_{best}\}$
17: $ndiv \leftarrow ndiv - 1$

Determine weights. Once the set *div* consisting of good and diverse prototype subsets is established, the classification of test instances can begin. When classifying a test instance t, we want to use prototype subsets in *div* that are good at classifying instances in the region of t. We assume that a prototype subset is good in the region of t if it classifies t's nearest neighbors in T correctly. The outline of the process that assigns weights to a prototype subset S is listed in Algorithm 3.

Algorithm 3. Procedure to assign weights to a prototype subset based on how well it classifies instances in the region of a test instance.

Input: Training set T, prototype subset $S \subseteq T$, test instance t, parameter *nk*
Output: weight $W(S)$
1: $N \leftarrow nk$ nearest neighbors of t in T
2: $W(S) \leftarrow 0$
3: **for each** $x \in N$ **do**
4: **if** $x \in S$ **then**
5: Determine the nearest neighbor y of x in $S \setminus \{x\}$
6: **else**
7: Determine the nearest neighbor y of x in S
8: **if** x and y belong to the same class **then**
9: $W(S) \leftarrow W(S) + 1$

In line 1, the *nk* nearest neighbors of t are determined within T and stored in the set N. Note that we use the entire training set T rather than a prototype subset to determine the neighbors, as T itself forms the most complete description of the problem

space that we have at our disposal. Determining neighbors in a reduced prototype set can result in more distant elements acting as neighbors in the weighting procedure and they will not be suitable representatives of the region around the target. The neighbors stored in N are classified in lines 4–9 using a leave-one-out procedure with S as pool of candidate nearest neighbors. Each time one of the neighbors is classified correctly, the weight of the prototype subset S is raised by one in line 9. This implies that the resulting weights are between 0 and nk. Using this approach, prototype subsets that classify instances near t well, are associated with high weights.

Weighted voting. The weights assigned to the prototype subsets are used in a weighted voting procedure. The procedure is outlined in Algorithm 4. Let C be the set of all available class labels. The score of every label in C is initialized at 0 at the beginning of the weighted voting process. Next, the test instance t is classified using each of the prototype subsets as pool of nearest neighbors in lines 3–5. The score of the predicted class c is augmented by the weight of the corresponding prototype subset. This implies that the classification based on prototype subsets that are well suited to classify instances in the region of t are taken more into account. In lines 6–11 the class label with the highest score is determined. This label is returned as the final prediction for t. We internally ensure that in case of ties, the final prediction is randomly drawn from the tied labels.

Algorithm 4. Weighted voting strategy used to classify test instances.

Input: Training set T, class labels C, set of prototype subsets div, weight $W(S)$ associated to each prototype subset S in div, test instance t
Output: Class label prediction c_{best}
1: **for each** $c \in C$ **do**
2: score$(c) \leftarrow 0$
3: **for each** prototype subset $S \in div$ **do**
4: $c \leftarrow$ class of the nearest neighbor of t in S
5: score$(c) \leftarrow$ score$(c) + W(S)$
6: score$_{\text{best}} \leftarrow -1$
7: $c_{\text{best}} \leftarrow$ null
8: **for each** $c \in C$ **do**
9: **if** score$(c) >$ score$_{\text{best}}$ **then**
10: score$_{\text{best}} \leftarrow$ score(c)
11: $c_{\text{best}} \leftarrow c$

3.2. Computational complexity of the proposed approach

In this section, we discuss the complexity of our framework. It is known that genetic approaches for PS impose a larger computational cost compared to other (simpler) PS algorithms (e.g. [3]). On the other hand, as stated above, they do perform best with respect to accuracy and reduction of the generated prototype set, which makes them still preferable over the alternative faster approaches. As will be clear from the experimental results presented in Section 4, the additional time needed by EEGPS compared to the traditional PS setting is negligible, combined with a significant increase in accuracy. This constitutes the fundamental motivation of our proposal.

The sole modification that has been made to the PS algorithms themselves, is the storing of all encountered prototype subsets. This has no influence on the runtime, but does imply additional storage requirements. However, this is no longer an insurmountable disadvantage nowadays, as storage has become cheap. The additional cost of EEGPS lies with the construction of the ensemble and the modified classification process. Let nev be the number of generations and p the size of the population in the genetic algorithm. The total number of candidate subsets constructed by an evolutionary PS method depends on its survival strategy. As described above, in GGA and CHC, almost the entire population is replaced in each generation. This implies that a total number of $p \cdot nev$ candidates will have been constructed in the end. For SSGA and SSMA, which are steady state genetic algorithms, at most two individuals are replaced every generation. This results in a total number of candidates of $p + 2 \cdot nev$. Note that these values are upper bounds, as the same prototype subset can be encountered multiple times. Below, we denote the total number of stored chromosomes as Ch. It is clear that, for all considered methods, this number is linear in both p and nev.

The construction of the ensemble in EEGPS consists of two steps: (a) the selection of the $pbest\%$ best subsets and (b) the selection of the $pdiv\%$ most diverse subsets among those. The first step can be handled in $\mathcal{O}(Ch)$ time, since it only requires one pass through the set of candidates. Their fitness has already been computed during the PS algorithm. Afterwards, Algorithm 2 is applied to the constructed set $best$. For the pairwise diversity calculations between two subsets S_1 and S_2, Algorithm 1 is used. During the PS algorithm, the fitness of both S_1 and S_2 was calculated by means of (1). This calculation involved computing the accuracy of 1NN on T using the elements in the subset as prototypes. This means that the classification step in lines 2–12 of Algorithm 1 has already been performed and, if stored, does not need to be repeated. The entire execution of this procedure therefore has complexity $\mathcal{O}(|T|)$. Moreover, we can ensure that every pairwise diversity computation is only computed once and stored, such that no duplicate calculations are performed in Algorithm 2. Calculating all pairwise diversity values can be performed in $\mathcal{O}(|T| \cdot |best|^2)$ time before Algorithm 2 is called. The largest cost of this algorithm lies with the while loop in lines 5-17. In every iteration, of which there are at most $|best|$, the prototype set S_{best} adding the most diversity to div is determined. Each iteration has therefore a cost of $\mathcal{O}(|best|^2)$, bringing the total cost of Algorithm 2 to $\mathcal{O}(|best|^3)$. The total time needed for the construction of the ensemble is

$$\begin{aligned}\mathcal{O}(Ch + |T| \cdot |best|^2 + |best|^3) &= \mathcal{O}(Ch + |T| \cdot Ch^2 + Ch^3) \\ &= \mathcal{O}(|T| \cdot Ch^2 + Ch^3).\end{aligned}$$

As a reminder, the value Ch is linear in both p and nev. The construction time of the ensemble is consequently at most cubic in p and nev and linear in $|T|$. The largest cost of the traditional genetic PS algorithms lies with the fitness calculations, which yield a cost quadratic in $|T|$ in each generation. As we will show in the experimental section, the additional time needed by EEGPS compared to the traditional PS setting will be negligible.

The classification of a test instance by EEGPS is performed by (a) computing the weights of the generated subsets and (b) determining the weighted vote. We compare this step to the complexity of 1NN, since this is the classifier normally combined with the PS procedure. To classify a test instance t with 1NN and a single prototype subset S, we only need to determine its nearest neighbor in S. This can be achieved in $\mathcal{O}(|S|) = \mathcal{O}(|T|)$ time. The first step in the classification of t by EEGPS is the calculation of the weights by means of Algorithm 3. For a subset S, this algorithm takes up $\mathcal{O}(nk)$ time. Indeed, the weights are based on the nearest nk training instances and, as noted above, the classification of these instances by S has already been determined and stored during the PS algorithm. As a result, the cost of finding the weights of all prototype sets in the ensemble is $\mathcal{O}(nk \cdot Ch)$. Finally, Algorithm 4 is applied. Its cost is determined by lines 3–5. In this case, since t is a test instance and not known at training time, the classification of this instance by S needs to be done explicitly. Line 4 therefore has a cost of $\mathcal{O}(|T|)$. In its totality, the for loop (and Algorithm 4) requires $\mathcal{O}(Ch \cdot |T|)$. We conclude that the classification of t by EEGPS costs $\mathcal{O}(Ch \cdot (|T| + nk))$ time, compared to the $\mathcal{O}(|T|)$ cost of 1NN. Both are linear in $|T|$, but, due to the extra factor Ch, the former can be expected to be somewhat slower than the latter.

Table 1
Datasets used in the experimental evaluation. We specify the number of instances (inst), features (feat) and classes (cl).

	inst	feat	cl		inst	feat	cl
appendicitis	106	7	2	housevotes	232	16	2
australian	690	14	2	iris	150	4	3
automobile	150	25	6	led7digit	500	7	10
balance	625	4	3	lymphography	148	18	4
bands	365	19	2	mammographic	830	5	2
breast	277	9	2	monk-2	432	6	2
bupa	345	6	2	movement_libras	360	90	15
car	1728	6	4	newthyroid	215	5	3
cleveland	297	13	5	pima	768	8	2
contraceptive	1473	9	3	saheart	462	9	2
crx	653	15	2	sonar	208	60	2
dermatology	358	34	6	spectfheart	267	44	2
ecoli	336	7	8	tae	151	5	3
flare	1066	11	6	tic-tac-toe	958	9	2
german	1000	20	2	vehicle	846	18	4
glass	214	9	7	vowel	990	13	11
haberman	306	3	2	wine	178	13	3
hayes-roth	160	4	3	wisconsin	683	9	2
heart	2270	13	2	yeast	1484	8	10
hepatitis	80	19	2	zoo	101	16	7
magic	19020	10	2	ring	7400	20	2
penbased	10992	16	10	twonorm	7400	20	2
phoneme	5404	5	2				

4. Experimental evaluation

In this section we assess the performance of the EEGPS framework. In Section 4.1 we present the outline of the experimental set-up and in Section 4.2 we analyze the influence of the EEGPS parameters on the performance of the EEGPS framework. In Sections 4.3 and 4.4, we evaluate and discuss how the accuracy of the EEGPS framework relates to the traditional evolutionary PS accuracy and provide some general guidelines for its use. In Sections 4.5 and 4.6 we discuss the relation of our approach to a weighted KNN algorithm and touch upon the matter of reduction related to PS.

4.1. Experimental set-up

We evaluate the proposed algorithms on the 40 datasets above the horizontal line described in Table 1 taken from the UCI [16] and KEEL [17] dataset repositories. The five larger datasets below the line are used in Section 4.3 to illustrate the efficiency of our approach. We use a 10 fold cross-validation procedure on each dataset, that is, we divide the data in 10 folds and use each fold as test data and the remaining folds as training data. We apply the EEGPS framework in conjunction with the GGA, SSGA, CHC and SSMA evolutionary PS algorithms discussed in Section 2. For each of these PS methods, we compare the performance of the algorithm within the EEGPS framework (that uses 1NN as classifier as described in Section 3) against the setting where the PS algorithm is used as preprocessing algorithm followed by 1NN classification.

As the proposed algorithms have a random component, we repeat each experiment 5 times. All results reported are the average results over the 10 folds and 5 runs. By repeating the experiments several times and reporting the average results, we account for the stochastic nature of the algorithms. Increasing this number of iterations further may render our conclusions more convincing, but we think the threat of invalidity should be low. Related studies (e.g. [3]) sometimes use fewer iterations. We report the classification accuracy, defined as the number of correctly classified objects divided by the total number of objects, and the running time. To assist fellow researchers in the replication of our experiments (see e.g. [18]), we list the parameters values used for the evolutionary PS algorithms in Table 2. These are the settings that were suggested in the original proposals [4–7].

We consider the number of evaluations used in the evolutionary PS algorithm as a parameter of the EEGPS framework. There are four parameters in total in the EEGPS framework:

- *nev*: Number of evaluations used in the evolutionary PS algorithm. We use *nev* = 1000, 2000, . . ., 10,000.
- *pbest*: Selected percentage of fittest prototype subsets. We use *pbest* = 1, 5, 10, 50, 100.

Table 2
Parameter settings of the evolutionary PS algorithms used in the experimental evaluation.

GGA	Mutation probability	0.01
	Crossover probability	0.6
	Population size	100
	α in the fitness function	0.5
SSGA	Mutation probability	0.01
	Population size	100
	α in the fitness function	0.5
CHC	Population size	100
	α in the fitness function	0.5
	Percentage of genes changed in restart (%)	35
SSMA	Mutation probability	0.01
	Population size	100
	α in the fitness function	0.5

- *pdiv*: Selected percentage of most diverse prototype subsets. We use *pdiv* = 1, 5, 10, 50, 100.
- *nk*: Number of neighbors used to assess the quality of the prototype subset. We use *nk* = 1, 3, 5.

4.2. Analysis of the EEGPS parameters

In this section, we consider the effect of the different parameters of our proposal on its performance. We note that the baseline accuracy of 1NN without any preprocessing is 0.7390, taken as average over the 40 datasets above the line in Table 1. The results presented below will show that all considered methods lead to a clear improvement over this value, providing a first indication of their importance.

4.2.1. Performance of the original PS algorithms

In order to interpret the results of EEGPS, we first need to know more about the performance of the original evolutionary PS algorithms. In Fig. 2, we show the accuracy values when the PS algorithms are applied in the traditional scheme, that is, the PS algorithm is applied to the training data and the test instances are classified with 1NN using the resulting prototype subset as pool of candidate nearest neighbors. A crucial first conclusion, which will be reflected in our further analysis, is that the more complex methods CHC and SSMA generally outperform their simpler relatives GGA and SSGA.

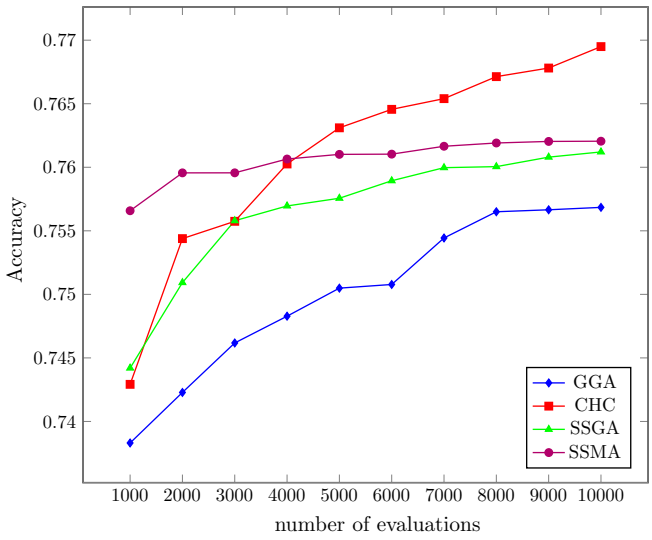

Fig. 2. Accuracy results for the PS algorithms in their traditional setting and for different values of *nev*.

4.2.2. Influence of the nk parameter

We first study the influence of *nk* on the accuracy of the EEGPS framework based on the average accuracy for the different parameter settings listed above. The results are listed in Appendix A. For each evolutionary method, we highlight which value for *nk* we observed to be optimal among the evaluated set:

GGA *nk* = 1 is optimal, independent from the values of *nev*, *pdiv* or *pbest*.

CHC *nk* = 3 is optimal in most cases. For some values of *pdiv* and *pbest* using *nk* = 5 is optimal, but in those cases the difference between *nk* = 3 and *nk* = 5 is not large (the maximum difference is 0.0006). For high values of all three parameters *pdiv*, *pbest* and *nev*, the parameter setting *nk* = 1 is optimal.

SSGA either *nk* = 1 or *nk* = 3 is optimal, depending on the other parameter values. For instance, for the lowest evaluated *nev*-value 1000, we find that, if *pbest* = 10, *nk* = 3 is optimal and *nk* = 1 otherwise. For the highest value *nev* = 10,000, the setting *nk* = 3 generally performs best.

SSMA *nk* = 3 is optimal.

Referring back to Fig. 2, we can conclude that the better the PS algorithm performs, the higher *nk* should be. For instance, the worst performing PS algorithm GGA always has *nk* = 1 as optimal parameter and the best performing algorithm CHC has mostly *nk* = 3 as optimal parameter value (sometimes even *nk* = 5). A possible explanation could be that when *nk* is higher, more prototype subsets are taken into account to classify the test instance, as non-zero weights are more likely to occur. Using more prototype subsets is beneficial when they are of high enough quality. The fact that a given PS algorithm performs better, implies that the population has a higher quality and that, in general, the generated prototype subsets are better. This could explain why the best performing PS algorithms work better when *nk* is higher. Another explanation can be found in the fact that when the prototype subsets are of high quality, it is not important that the neighbor on which the quality is evaluated is the closest to the test instance. The quality of prototype subsets that are a bit further than the nearest neighbor is still good enough to improve the classification of the test instance.

These two explanations also clarify why CHC has *nk* = 1 as optimal parameter when many prototype subsets are considered, that is, when the three values *pdiv*, *pbest* and *nev* are high. Observe in Fig. 2 that for a low number of evaluations, CHC does not perform well, which means that low quality subsets were generated. Even though the quality of the generated sets clearly improves over subsequent iterations, all encountered sets are stored. When *pdiv* and *pbest* are high, the earlier, low quality subsets represent a substantial part of the ensemble. In that case, it is better to use *nk* = 1, for similar reasons as given above.

4.2.3. Influence of the pbest and pdiv parameter

Based on our conclusions in the previous section, we use *nk* = 1 for GGA and SSGA and *nk* = 3 for SSMA and CHC in the remainder of the analysis. We visualize the effect of the parameters *pbest* and *pdiv* in Fig. 3. To condense the required space, we present the results for *nev* = 10,000 as a representative value. In each subfigure, the average accuracy is represented. We can draw the following conclusions:

GGA Fig. 3(a) shows that the performance improves when *pbest* is higher. For each value of *pbest*, the best value for *pdiv* is 50%.

CHC in Fig. 3(b), we observe that when 10,000 evaluations are carried out, the best combination is found for *pbest* = 10

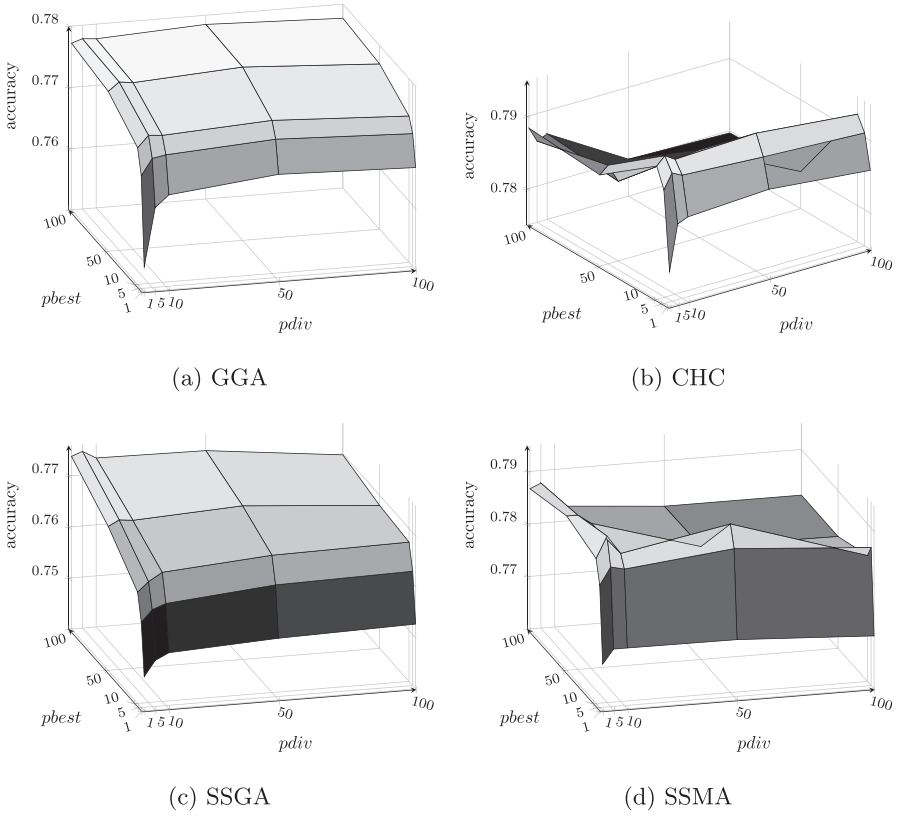

Fig. 3. Illustration of the *pbest* and *pdiv* sensitivity of the four PS methods within the EEGPS framework. For each methods, the parameter *nev* was set to 10,000. For GGA and SSGA, we set *nk* = 1. For CHC and SSMA, we set *nk* = 3.

and *pdiv* = 50. We briefly note that for lower values of *nev*, i.e. 1000 or 5000 evaluations, the best value for *pbest* is also 10, but it is best combined with *pdiv* = 5 in that case.

SSGA as for GGA, Fig. 3(c) shows a better performance of SSGA for increasing values of *pbest*. When *nev* = 10,000, the best *pdiv* value depends on the selected *pbest*. For lower values of *nev*, we found the combination *pbest* = 100 and *pdiv* = 50 to be optimal within the evaluated set.

SSMA for this method, Fig. 3(d) shows that the results are less clear-cut. Nevertheless, our results allow us to conclude that *pbest* = 10 performs well for *nev* = 10,000 and all other values of *nev*. It is best combined with *pdiv* = 50.

As in Section 4.2.2, we see a clear difference between the results for the more basic methods GGA and SSGA on the one hand and the results for CHC and SSMA on the other. For GGA and SSGA it is good to consider all prototype subsets (*pbest* = 100) and to remove half of the solutions that are similar to others (*pdiv* = 50), while for CHC and SSMA, it is in general better to select less solutions (*pbest* = 10 or *pbest* = 50) and to remove half of the solutions that are similar to others (*pdiv* = 50).

From these results we can conclude that it is useful to perform the crucial step of selecting the most diverse among the best prototype subsets, as we find values *pdiv* < 100 to outperform *pdiv* = 100. The differences between the methods can again be explained considering the results in Fig. 2. The best prototype subsets produced by SSMA and CHC are of higher quality than the best prototype subsets produced by SSGA or GGA. Therefore, it seems that for SSMA and CHC it is better to only work with these high quality prototype subsets and that the other generated prototype subsets deteriorate the performance of the ensemble. The difference between the best prototype subsets generated by SSGA and GGA and the remaining ones is smaller, meaning that these prototype subsets do not deteriorate the accuracy of the ensemble and can even improve it, by increasing the diversity.

We conclude that for GGA and SSGA, the parameter setting *pbest* = 100 and *pdiv* = 50 is a good choice and that for SSMA and CHC, *pbest* = 10 and *pdiv* = 50 is a good choice in most cases. We use these settings in the following paragraph, where we study the influence of the number of evaluations on the performance of EEGPS.

4.2.4. *Influence of the* nevnev *parameter*

In Fig. 4, we plot the results of the evolutionary PS algorithms in the EEGPS framework for different values of *nev*. For GGA and

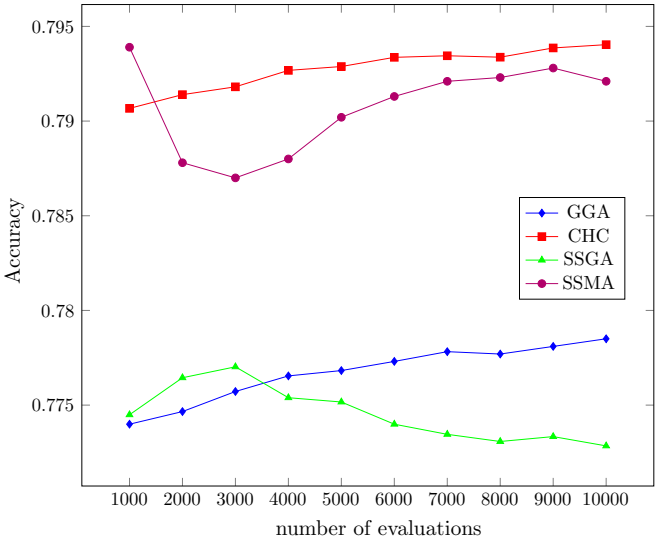

Fig. 4. Accuracy results for the PS algorithms within EEGPS for different values of *nev*.

SSGA, the remaining parameters were set to *pbest* = 100, *pdiv* = 50 and *nk* = 1. For CHC and SSMA, we used *pbest* = 10, *pdiv* = 50 and *nk* = 3. These correspond with our conclusions from Sections 4.2.2 and 4.2.3. The results for GGA and CHC are in line with our expectations: when more evaluations are taken, the quality of the prototype subsets improves and hence the accuracy is higher. However, it is interesting to see that the accuracy of SSGA increases until *nev* = 3000 and then decreases. Considering Fig. 2, this can be explained by the fact that SSGA fastly improves until *nev* = 3000 and afterwards only does so slightly. This means that more evaluations do not result in significantly better prototype subsets and many prototype subsets of similar quality are added to the ensemble, which apparently worsens the performance. This is most likely due to the unavoidable decrease in diversity within the selected group of prototype subsets that this phenomenon will result in.

The accuracy of SSMA first decreases and then slightly increases, but never reaches the accuracy that was obtained after 1000 evaluations. An explanation is once more found in Fig. 2, that shows that SSMA only very slowly improves its best prototype subset. Since we use *pbest* = 10 for SSMA, the prototype subsets that are used for SSMA in the EEGPS framework are all high-quality prototype subsets, but they might be very similar to each other. Taking more evaluations means that more of these similar prototype subsets are added, which possibly deteriorates the performance by decreasing the diversity of the ensemble. After 3000 evaluations the accuracy improves again, which might be due to the fact that the optimization step in the SSMA algorithm creates very high quality prototype subsets.

4.3. Comparison of EEGPS to the evolutionary PS methods

In the previous section, we studied the influence of the EEGPS parameters on the performance of the evolutionary PS methods within our framework. We concluded that for GGA and SSGA, *nk* = 1, *pbest* = 100 and *pdiv* = 50 were the best settings, while for CHC and SSMA *nk* = 3, *pbest* = 10 and *pdiv* = 50 is a good choice. We now verify whether the EEGPS framework improves the classification accuracy of the PS methods used in the traditional setting. Recall that the evolutionary PS methods under consideration have been shown to be the best performing PS techniques in the experimental study of [3].

In Fig. 5 we visualize the improvement in accuracy when using the EEGPS framework. EEGPS has on average a positive influence on all evolutionary PS algorithms and has the most influence on the CHC and SSMA algorithms. For all PS algorithms except SSMA, the improvement decreases when more evaluations are used. To test if the improvement is significant, we use the Wilcoxon statistical test [19] to compare the EEGPS variants of the PS algorithm against the PS algorithm in the traditional setting, as recommended in e.g. [20]. The sum of ranks in favor of EEGPS is given by R^+, while R^- is the sum of ranks in favor of the PS algorithm in the traditional setting. Additionally, the *p*-value is reported, which reflects the probability of obtaining a more extreme result than the one observed, when we assume the methods to perform equivalently (null hypothesis). We carry out the test at the 5 percent significance level. For each evolutionary PS algorithm, we test if the EEGPS variant significantly improves it, for increasing values of *nev*. The values of the statistics are listed in Table 3 for GGA and SSGA and in Table 4 for CHC and SSMA.

For GGA, the table shows that the EEGPS framework significantly improves GGA when in both settings less than 6000 evaluations are used, excepting the setting where PS performs 6000 evaluations and EEGPS only 1000. For 8000 or 10,000 evaluations, EEGPS does not significantly improve GGA, but the low *p*-values do suggest that there is some improvement. Additionally, we note that for each

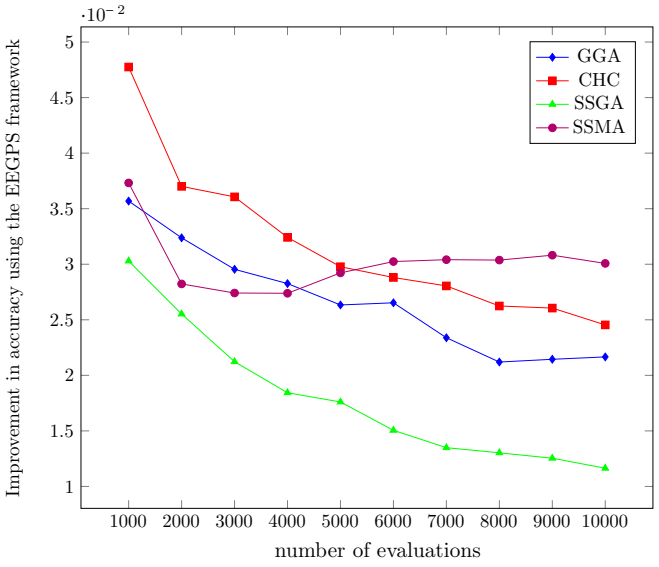

Fig. 5. Improvement in accuracy when using the EEGPS framework for the evolutionary PS algorithms.

comparison, R^+ is higher than R^-, which also suggests that using the EEGPS framework is at least as accurate as the evolutionary algorithm. Even when only 1000 evaluations are used in the EEGPS framework, the result does not seem to be worse than when GGA uses 10,000 evaluations.

The EEGPS framework only significantly improves the SSGA algorithm when the latter is allowed to perform less than 4000 evaluations. This result could be expected from Fig. 5: there is only a high average improvement for low values of *nev* and for higher values the benefits of the EEGPS framework are less clear. However, as for GGA, we see that in all comparisons, the R^+ value is higher than R^-, indicating that the EEGPS framework is at least as good as the traditional GGA, even when only a small number of evaluations are used.

The improvement achieved by the EEGPS framework is very clear when using CHC or SSMA. For all comparisons, the *p*-values are below 0.5, showing that the EEGPS framework significantly improves CHC and SSMA. It is remarkable that the results obtained using the EEGPS framework with 1000 evaluations are significantly better than when using CHC or SSMA with 10,000 evaluations. This shows that by taking maximum advantage of the work performed by an evolutionary PS method, we can outperform the traditional setting by even using considerably less evaluations.

To conclude this experimental evaluation, we study the additional time required to carry out the evolutionary PS algorithms within the EEGPS framework, which includes the time to select the best and most diverse prototype subsets and the time to classify the test instances. The results are presented in Fig. 6. We observe that the additional time is small, especially compared to the running time required by the evolutionary PS algorithms themselves, which take up several minutes on average. For CHC and SSMA, the additional time required is smaller, as the best parameter settings for these algorithms are *nk* = 3, *pbest* = 10 and *pdiv* = 50 as opposed to the other two algorithms that have *nk* = 1, *pbest* = 100 and *pdiv* = 50 as best parameter settings.

To further illustrate the efficiency of our approach, we conduct some additional experiments for CHC and SSMA on the five larger datasets below the horizontal line in Table 1. For CHC we use *nev* = 10,000, for SSMA we set this value to *nev* = 1000, since these values attained the best results in our analysis above. The

Table 3
Statistics of the Wilcoxon test comparing the GGA and SSGA PS algorithms in the EEGPS framework (R^+) with their traditional setting (R^-) for different values of *nev*. *p*-Values corresponding to significant differences are printed in bold.

	PS / EEGPS	1000	2000	3000	4000	5000	6000	8000	10000
GGA	R^+	705	728	727	732	734	737	743	738
	1000 R^-	115	92	93	88	86	83	77	82
	p	**0.00007**	**0.00002**	**0.00002**	**0.00002**	**0.00001**	**0.00001**	**0.00001**	**0.00001**
	R^+	669	683	661	702	675	707	706	714
	2000 R^-	151	137	119	118	105	113	114	106
	p	**0.00049**	**0.00024**	**0.00015**	**0.00008**	**0.00007**	**0.00006**	**0.00007**	**0.00004**
	R^+	615	626	637	647	653	660	659	664
	3000 R^-	205	194	183	173	167	160	161	156
	p	**0.00574**	**0.00361**	**0.00223**	**0.00141**	**0.00106**	**0.00076**	**0.00080**	**0.00062**
	R^+	585	576	606	617	619	621	629	634
	4000 R^-	235	204	214	203	201	199	191	186
	p	**0.01833**	**0.00925**	**0.00826**	**0.00529**	**0.00486**	**0.00447**	**0.00312**	**0.00255**
	R^+	562	568	578	585	589	595	604	615
	5000 R^-	258	252	242	235	231	225	216	205
	p	**0.04038**	**0.03314**	**0.02352**	**0.01833**	**0.01583**	**0.01265**	**0.00894**	**0.00574**
	R^+	542	530	565	551	555	558	567	601
	6000 R^-	278	250	255	229	225	222	213	219
	p	0.07492	**0.04992**	**0.03661**	**0.02421**	**0.02091**	**0.01870**	**0.01325**	**0.01005**
	R^+	478.5	491	499	501	507	513	516	525
	8000 R^-	341.5	329	321	319	313	307	304	295
	p	0.35288	0.27331	0.22898	0.21874	0.19002	0.16417	0.15229	0.12055
	R^+	477	486	487	493	495	502	507	494
	10000 R^-	343	334	333	327	325	318	313	286
	p	0.36426	0.30383	0.29755	0.26171	0.25046	0.21375	0.19002	0.14476
SSGA	R^+	676	677	694	683	648	666	624	635
	1000 R^-	144	143	126	137	132	154	156	185
	p	**0.00034**	**0.00032**	**0.00013**	**0.00024**	**0.00031**	**0.00057**	**0.00107**	**0.00244**
	R^+	590	602	640	607	613	601	589	581
	2000 R^-	230	218	180	213	207	219	231	239
	p	**0.01526**	**0.00967**	**0.00195**	**0.00794**	**0.00623**	**0.01005**	**0.01583**	**0.02116**
	R^+	554	560	591	567	549	555	526	544
	3000 R^-	266	260	229	253	231	265	254	276
	p	0.05211	**0.04308**	**0.01470**	**0.03426**	**0.02603**	0.05050	0.05680	0.07063
	R^+	525	530	557	544	520	534	507	522
	4000 R^-	295	290	263	276	260	286	273	298
	p	0.12055	0.10530	**0.04741**	0.07063	0.06859	0.0942	0.10107	0.13050
	R^+	518	523	546	527	511	525	495	520
	5000 R^-	302	297	274	293	269	295	285	300
	p	0.14474	0.12711	0.06655	0.11426	0.08997	0.12055	0.14095	0.13747
	R^+	500	502	523	504	486	500	480	502
	6000 R^-	320	318	297	316	294	320	300	318
	p	0.22382	0.21375	0.12711	0.20401	0.17809	0.22382	0.20662	0.21375
	R^+	485	486	506	490	471	477	472.5	468
	8000 R^-	335	334	314	330	309	343	347.5	352
	p	0.31019	0.30383	0.19460	0.27924	0.25540	0.36426	0.73325	0.43168
	R^+	474	475	501	478	459	468	438	436
	10000 R^-	346	345	319	342	321	352	342	344
	p	0.38596	0.37864	0.21874	0.35719	0.33211	0.43168	0.49852	0.51640

results of these experiments are presented in Table 5, which were again taken as averages over five runs of the algorithms. As before, we can observe the clear advantage on the classification accuracy of using the PS methods within the EEGPS framework. The table further reports the ratio of the runtime of the PS algorithm within EEGPS over the runtime of the method in its traditional setting. Since the ratios are all close to 1, it is clear that the additional time required by EEPGS time is minor. We conclude that we obtain an increase in accuracy with limited additional effort.

4.4. Guidelines

To summarize, for their use within the EEGPS framework in practice, we can recommend the following parameter values for the PS algorithms:

GGA	*nk* = 1, *pbest* = 100, *pdiv* = 50, *nev* = 10,000.
CHC	*nk* = 3, *pbest* = 10, *pdiv* = 50, *nev* = 10,000.
SSGA	*nk* = 1, *pbest* = 100, *pdiv* = 50, *nev* = 3000.
SSMA	*nk* = 3, *pbest* = 10, *pdiv* = 50, *nev* = 1000.

Table 4
Statistics of the Wilcoxon test comparing the CHC and SSMA PS algorithm in the EEGPS framework (R^+) with their traditional setting (R^-) for different values of *nev*. *p*-Values corresponding to significant differences are printed in bold.

	PS / EEGPS		1000	2000	3000	4000	5000	6000	8000	10000
CHC	1000	R^+	780	729	738	696	639	630	621	594
		R^-	40	91	82	124	141	150	199	226
		p	**< 10E-5**	**0.00002**	**0.00001**	**0.00012**	**0.00050**	**0.00079**	**0.00447**	**0.01314**
	2000	R^+	778	703	711	710	695	679	643	619
		R^-	42	77	69	110	125	141	177	201
		p	**< 10E-5**	**0.00001**	**0.00001**	**0.00005**	**0.00012**	**0.00029**	**0.00170**	**0.00486**
	3000	R^+	782	738.5	748	711	692	690	657	633
		R^-	38	81.5	72	109	128	130	163	187
		p	**< 10E-5**	**0.00003**	**0.00001**	**0.00005**	**0.00015**	**0.00016**	**0.00088**	**0.00262**
	4000	R^+	796	750	767	705	711	716	685	628
		R^-	24	70	53	75	109	104	135	152
		p	**< 10E-5**	**0.00001**	**< 10E-5**	**0.00001**	**0.00005**	**0.00004**	**0.00021**	**0.00087**
	5000	R^+	796	755	772	712	687	718	665	667
		R^-	24	65	48	68	93	102	115	153
		p	**< 10E-5**	**< 10E-5**	**< 10E-5**	**0.00001**	**0.00003**	**0.00003**	**0.00012**	**0.00054**
	6000	R^+	793	755	769	747	731	714	702	684
		R^-	27	65	51	73	89	106	118	136
		p	**< 10E-5**	**< 10E-5**	**< 10E-5**	**0.00001**	**0.00002**	**0.00004**	**0.00008**	**0.00023**
	8000	R^+	792	763	770	749	706	725	711	686
		R^-	28	57	50	71	74	95	109	134
		p	**< 10E-5**	**< 10E-5**	**< 10E-5**	**0.00001**	**0.00001**	**0.00002**	**0.00005**	**0.00020**
	10000	R^+	787	726	766	748	734	723	711	649
		R^-	33	54	54	72	86	97	109	131
		p	**< 10E-5**	**< 10E-5**	**< 10E-5**	**0.00001**	**0.00001**	**0.00003**	**0.00005**	**0.00029**
SSMA	1000	R^+	790	780	778	776	767	770	770	767
		R^-	30	40	42	44	53	50	50	53
		p	**< 10E-5**	**< 10E-5**	**< 10E-5**	**< 10E-5**	**< 10E-5**	**< 10E-5**	**< 10E-5**	**< 10E-5**
	2000	R^+	759	747	748	742	732	736	738	736
		R^-	61	73	72	78	88	84	82	84
		p	**< 10E-5**	**0.00001**	**0.00001**	**0.00001**	**0.00002**	**0.00001**	**0.00001**	**0.00001**
	3000	R^+	743	733	725	723	711	718	714	716
		R^-	77	87	95	97	109	102	106	104
		p	**0.00001**	**0.00001**	**0.00002**	**0.00003**	**0.00005**	**0.00003**	**0.00004**	**0.00004**
	4000	R^+	756	733	725	719	702	715	705	710
		R^-	64	87	95	101	118	105	115	110
		p	**< 10E-5**	**0.00001**	**0.00002**	**0.00003**	**0.00008**	**0.00004**	**0.00007**	**0.00005**
	5000	R^+	767	752	747	709	697	699	690	728
		R^-	53	68	73	71	83	81	90	92
		p	**< 10E-5**	**< 10E-5**	**0.00001**	**0.00001**	**0.00002**	**0.00002**	**0.00003**	**0.00002**
	6000	R^+	758	741	714	739	725	696	723	726
		R^-	62	79	66	81	95	84	97	94
		p	**< 10E-5**	**0.00001**	**0.00001**	**0.00001**	**0.00002**	**0.00002**	**0.00003**	**0.00002**
	8000	R^+	772	765	770	762	750	754	754	721
		R^-	48	55	50	58	70	66	66	59
		p	**< 10E-5**	**< 10E-5**	**< 10E-5**	**< 10E-5**	**0.00001**	**< 10E-5**	**< 10E-5**	**< 10E-5**
	10000	R^+	752	746	754	746	733	732	744	738
		R^-	28	34	26	34	47	48	36	42
		p	**< 10E-5**	**< 10E-5**	**< 10E-5**	**< 10E-5**	**< 10E-5**	**< 10E-5**	**< 10E-5**	**< 10E-5**

The overall highest accuracy values were obtained by SSMA and CHC within EEGPS, setting the parameters as specified above.

Table 5
Accuracy results of CHC and SSMA with and without EEGPS for five larger datasets. We also present the ratio (Rat.) of the execution times of the methods within the framework compared to the traditional setting.

Dataset	CHC			SSMA		
	PS	EEGPS	Rat.	PS	EEGPS	Rat.
magic	0.8159	0.8205	1.0083	0.8176	0.8210	1.0027
penbased	0.9562	0.9820	1.0026	0.9822	0.9832	1.0037
phoneme	0.8234	0.8407	1.0073	0.8533	0.8614	1.0144
ring	0.8764	0.8769	1.0070	0.9238	0.9331	1.0029
twonorm	0.9658	0.9671	1.0066	0.9603	0.9623	1.0037

4.5. Relation to KNN

As 1NN is applied with each prototype set, one could argue that the proposed classification ensemble is highly related to KNN, with *K* set equal to the number of prototype sets being used and its own prototype set taken as their union. Nevertheless, these two techniques are certainly different. For KNN, every prototype can act as a neighbor for a target instance a single time. In EEGPS, the same element can be selected multiple times, when it is present in several prototype sets and is determined as the nearest neighbor of a target instance within more than one of them. Consequently, EEGPS uses an adaptive rather than fixed value of K. Furthermore,

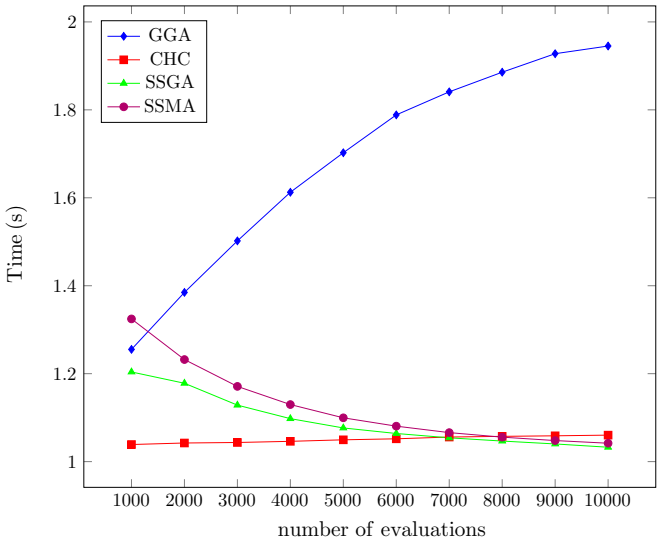

Fig. 6. Additional time required by EEGPS.

the weight of a neighbor builds up as it is selected more often, which cannot be modeled with a straightforward weighted KNN approach. We have conducted a set of experiments showing that the performance of EEGPS is superior to that of KNN using the union of all prototype sets as its own prototype sets. Using the settings advised in Section 4.4, we found respective accuracies of 0.4909 (GGA), 0.4765 (CHC), 0.4905 (SSGA) and 0.6544 (SSMA) in this set-up. Recall that the baseline average accuracy of 1NN without preprocessing is 0.7390. The performance of our EEGPS framework is clearly substantially better than this suggested related approach. It was evident that the considerable number of prototype sets used in the ensemble lead to a lower global reduction when combined, because a level of diversity has been guaranteed between them. This implies that their union more closely coincides with the original training set. The performance of KNN was therefore found to be low, as it could not benefit from a decent PS. Moreover, the number K can be high, namely the number of prototype sets used by EEGPS, which means that too many neighbors are used to classify test instances, deteriorating the prediction.

4.6. A note on reduction

Our clear aim in the development of the EEGPS framework lies with the improvement of the classification performance related with an evolutionary PS method with minimal additional effort. The other two issues of KNN for which PS was originally proposed, namely its possible high storage and runtime requirements, are less critical nowadays, as hard drives have become relatively cheap and the stored dataset can be searched efficiently by the use of appropriate data structures (e.g. [21,22]). Still, as a side-note we also report an indication of the achieved reduction, because a high reduction remains a goal of certain PS methods. Since we are working with a set of prototype sets, we choose to report the average reduction they individually attain compared to the original training set. For 10,000 evaluations, using the same values for the other parameters as above, we find average reductions of 80.25%, 79.67%, 97.04% and 95.37% for GGA, SSGA, CHC and SSMA respectively. When we consider the union of all these sets, we obtain an indication of the global reduction. Using these same settings, for GGA and SSGA, the global reduction was found to be 0%. This is understandable, since a very high number of prototype subsets will be used ($pbest = 100$, $pdiv = 50$) and a level of diversity has been ascertained between them. For CHC and SSMA we found average global reductions of 87.68% and 93.11% respectively.

5. Extension of the framework: application to prototype generation

In order to show that the EEGPS framework can easily incorporate other genetic preprocessing approaches as well, we present its application to a prototype generation (PG) method. In fact, any preprocessing method that encounters multiple candidate (reduced or modified) training sets during its execution can be integrated in our framework. The sole modification which needs to be made to the algorithm under consideration, is the implementation of a mechanism storing all encountered solutions. The final construction of the ensemble and the classification process are not method-specific.

PG is similar to PS in that both preprocessing paradigms construct a reduced training set. As described above, PS does so by selecting a subset of the instances in the training set. This implies that all elements in the resulting prototype set already appeared in the original, unreduced training set. PG differs from this set-up. Apart from selecting existing instances, these methods also allow the construction of artificial elements. The final prototype set is therefore not necessarily a subset of the training set. We refer the reader to [23] for a review, taxonomy and experimental comparison of PG methods. As for PS methods, the focus of PG lies on the 1NN classifier.

A group of PG methods is based on differential evolution (DE, [24]). Like the genetic algorithms discussed above, DE evolves a population of candidate solutions over a number of generations, guided by custom genetic operators. It is therefore perfectly suited to be integrated in the framework proposed in this paper. An experimental comparison between PG methods based on DE was conducted in [9]. This study concluded that Scale Factor Local Search in Differential Evolution (SFLSDE, [25]) is one of the top-performing methods within this group. We therefore select it as a representative PG method to be evaluated with our framework.

SFLSDE is a positioning adjustment algorithm. It initializes the candidate prototype subsets in the population by selecting a given percentage of the original training instances. In each iteration, it optimizes the positions of the instances in these sets by moving them across the feature space. The number of instances remains unchanged. SFLSDE is a memetic DE algorithm with custom mutation and crossover schemes, integrating local search components. We refer to [25] for further details.

Within EEGPS, every prototype subset constructed by SFLSDE is stored. When the algorithm terminates, the classification ensemble is constructed as described in Section 3. In these experiments, we have not varied the *nev* parameter, but have set it to 500, as was done in [9]. The remaining internal parameters of SFLSDE have also been set to the values used in that study. We do vary the remaining three parameters of the framework. For the percentages *pbest* and *pdiv*, we use the values 10, 50 and 100. The value *nk* is set to 1, 3 and 5. Table 6 presents the results of the experiments, that is, the average accuracy attained by SFLSDE within the corresponding EEGPS setting over the 40 datasets above the line in Table 1. As for the PS methods, the additional time required to construct the ensemble is negligible compared to the time needed for the execution of SFLSDE itself: the former never reached more than 0.3% of the latter. With respect to the accuracy, the benefits of the framework are less clear than they were for the genetic PS methods. Only for three settings do we observe a slight advantage of the framework over SFLSDE itself. In many other cases, the integration of SFLSDE in EEGPS leads to a decrease in accuracy. This is most likely due to the sensitivity of PG methods to overfitting, as noted in e.g. [8]. The combination of multiple overfitted subsets in the

Table 6
Accuracy results of SFLSDE within the EEGPS framework for several parameter settings. The baseline average accuracy of 1NN after SFLDE is 0.7464. Higher values are printed in bold.

pbest	*pdiv*	*nk*		
		1	3	5
10	10	0.7092	0.7141	0.7126
10	50	0.7194	0.7237	0.7215
10	100	0.7202	0.7253	0.7224
50	10	0.7266	0.7299	0.7280
50	50	0.7306	0.7347	0.7318
50	100	0.7322	0.7355	0.7326
100	10	0.7393	0.7427	0.7390
100	50	0.7444	**0.7468**	0.7425
100	100	**0.7463**	**0.7470**	0.7429

ensemble will accentuate this effect. Furthermore, since the number of instances in each prototype subset is the same and the subsets are optimized by moving the instances within them, we can expect them to be too related to guarantee a sufficient level of diversity. The ensemble therefore loses power. We conclude that, as opposed to the PS methods considered above, SFLSDE with its default parameters seems less suitable for integration within our framework. The lesson learned from this case study is that, although our framework allows for an easy integration of many different methods, the user should always consider whether the extension is appropriate and whether the encountered subsets are diverse enough. In any case, no objection can be made to the little additional time required by EEGPS, as was confirmed in this section.

6. Conclusion

In this paper we proposed a framework called Ensembles of Evolutionary Generated Prototype Subsets (EEGPS) that allows to use an evolutionary PS algorithm in a more efficient manner. Instead of only using the fittest prototype subset generated by the evolutionary PS algorithm, we use multiple fit prototype subsets in an ensemble framework. In order to classify a new test instance, it is determined which prototype subsets are well suited to classify instances in the neighborhood of that test instance. Those prototype subsets are used in a voting strategy to determine the class of the test instance. Our experimental study clearly shows the benefits of this EEGPS strategy. Using EEGPS, the results are more accurate and good results are already obtained after a small number of evaluations, whereas PS algorithms in the traditional setting require many evaluations to achieve a good performance. We have also shown that any preprocessing method that encounters multiple candidate training sets during its execution can be easily plugged into EEGPS. The results in this paper were obtained using the KNN classifier. This approach could be extended for other classifiers and moreover, some ideas could also be applied for other ensemble classification techniques, where elements of the ensemble can be selected based on their performance for neighboring instances of the instance to be classified.

Finally, as we live in the big data era where datasets can commonly contain millions of instances [26], an important next step will also be the extension of the proposed framework to that setting. The focus of this paper was on the development of the framework, the comparison of evolutionary PS methods with and without using the framework and the extensive evaluation of the internal parameters. Very large datasets were excluded from the current study, as one can follow earlier proposals (e.g. [27,28]) to obtain the desired extension to big data in a straightforward way.

Acknowledgments

The research of Sarah Vluymans is funded by the Special Research Fund (BOF) of Ghent University. Chris Cornelis was partially supported by the Spanish Ministry of Science and Technology under the project TIN2011-28488 and the Andalusian Research Plans P11-TIC-7765, P10-TIC-6858 and P12-TIC-2958.

Appendix A. Table of results

In Table A.1, we present an overview of experimental results of EEGPS, where we vary the parameters *nev*, *nk*, *pbest* and *pdiv*. All

Table A.1
Average accuracy of the EEGPS framework over 40 datasets for different parameter settings.

nev	pbest	pdiv	GGA			CHC			SGA			SSMA		
			nk = 1	*nk* = 3	*nk* = 5	*nk* = 1	*nk* = 3	*nk* = 5	*nk* = 1	*nk* = 3	*nk* = 5	*nk* = 1	*nk* = 3	*nk* = 5
1000	10	10	**0.7653**	0.7643	0.7610	0.7835	**0.7883**	0.7860	0.7585	**0.7613**	0.7585	0.7849	**0.7912**	0.7895
1000	10	50	**0.7683**	0.7659	0.7625	0.7854	**0.7907**	0.7895	0.7606	**0.7628**	0.7595	0.7876	**0.7939**	0.7925
1000	10	100	**0.7672**	0.7640	0.7596	0.7850	**0.7889**	0.7863	0.7600	**0.7605**	0.7569	0.7882	**0.7936**	0.7906
1000	50	10	**0.7725**	0.7700	0.7660	0.7822	**0.7906**	0.7903	**0.7700**	0.7661	0.7629	0.7824	**0.7889**	0.7863
1000	50	50	**0.7736**	0.7712	0.7675	0.7825	0.7902	**0.7907**	**0.7706**	0.7664	0.7632	0.7868	**0.7965**	0.7937
1000	50	100	**0.7728**	0.7692	0.7645	0.7828	**0.7905**	0.7893	**0.7702**	0.7647	0.7608	0.7780	**0.7851**	0.7809
1000	100	10	**0.7734**	0.7715	0.7675	0.7788	**0.7877**	0.7873	**0.7739**	0.7725	0.7690	0.7813	**0.7855**	0.7824
1000	100	50	**0.7740**	0.7718	0.7682	0.7806	0.7887	**0.7893**	**0.7745**	0.7741	0.7701	0.7780	**0.7851**	0.7809
1000	100	100	**0.7739**	0.7711	0.7665	0.7785	**0.7863**	0.7848	**0.7739**	0.7709	0.7671	0.7760	**0.7778**	0.7726
5000	10	10	**0.7707**	0.7680	0.7647	0.7892	**0.7932**	0.7922	0.7612	**0.7636**	0.7627	0.7808	**0.7880**	0.7863
5000	10	50	**0.7716**	0.7687	0.7654	0.7900	**0.7929**	0.7924	0.7630	**0.7635**	0.7626	0.7821	**0.7902**	0.7886
5000	10	100	**0.7707**	0.7664	0.7633	0.7888	**0.7918**	0.7893	**0.7637**	0.7634	0.7622	0.7795	**0.7875**	0.7849
5000	50	10	**0.7739**	0.7702	0.7666	0.7800	**0.7855**	0.7842	**0.7695**	0.7692	0.7662	0.7829	**0.7890**	0.7856
5000	50	50	**0.7747**	0.7706	0.7670	0.7820	0.7911	**0.7913**	**0.7700**	0.7696	0.7668	0.7783	**0.7825**	0.7777
5000	50	100	**0.7736**	0.7687	0.7645	0.7782	**0.7812**	0.7766	**0.7696**	0.7672	0.7643	0.7779	**0.7810**	0.7761
5000	100	10	**0.7760**	0.7737	0.7705	0.7798	**0.7833**	0.7807	0.7738	**0.7749**	0.7725	0.7813	**0.7874**	0.7832
5000	100	50	**0.7768**	0.7744	0.7714	0.7782	**0.7812**	0.7766	**0.7752**	0.7745	0.7706	0.7779	**0.7810**	0.7761
5000	100	100	**0.7761**	0.7720	0.7683	**0.7771**	0.7753	0.7691	**0.7716**	0.7706	0.7680	0.7779	**0.7809**	0.7760
10,000	10	10	**0.7740**	0.7723	0.7701	0.7881	**0.7936**	0.7935	0.7653	**0.7657**	0.7645	0.7818	**0.7882**	0.7850
10,000	10	50	**0.7750**	0.7729	0.7703	0.7891	0.7940	**0.7941**	**0.7669**	0.7654	0.7645	0.7832	**0.7921**	0.7889
10,000	10	100	**0.7739**	0.7705	0.7677	0.7875	**0.7926**	0.7914	**0.7673**	0.7653	0.7641	0.7783	**0.7840**	0.7790
10,000	50	10	**0.7772**	0.7750	0.7719	0.7818	**0.7860**	0.7842	0.7690	**0.7715**	0.7692	0.7815	**0.7876**	0.7836
10,000	50	50	**0.7783**	0.7757	0.7732	0.7828	**0.7871**	0.7840	0.7701	**0.7718**	0.7691	0.7782	**0.7815**	0.7768
10,000	50	100	**0.7770**	0.7735	0.7702	**0.7807**	0.7801	0.7757	0.7680	**0.7691**	0.7664	0.7782	**0.7813**	0.7765
10,000	100	10	**0.7776**	0.7768	0.7739	0.7820	**0.7868**	0.7843	0.7731	**0.7752**	0.7727	0.7786	**0.7830**	0.7780
10,000	100	50	**0.7785**	0.7774	0.7743	**0.7807**	0.7801	0.7757	0.7728	**0.7747**	0.7715	0.7782	**0.7813**	0.7765
10,000	100	100	**0.7777**	0.7758	0.7724	**0.7807**	0.7797	0.7751	**0.7698**	0.7691	0.7666	0.7782	**0.7813**	0.7765

values are average accuracies of EEGPS over the 40 datasets above the line in Table 1. These results were discussed in Section 4.2.2.

References

[1] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inf. Theory 13 (1) (1967) 21–27.
[2] S. García, J. Luengo, F. Herrera, Data Preprocessing in Data Mining, Springer, 2015.
[3] S. Garcia, J. Derrac, J. Cano, F. Herrera, Prototype selection for nearest neighbor classification: taxonomy and empirical study, IEEE Trans. Pattern Anal. Mach. Intell. 34 (3) (2012) 417–435.
[4] J. Cano, F. Herrera, M. Lozano, Using evolutionary algorithms as instance selection for data reduction in KDD: an experimental study, IEEE Trans. Evol. Comput. 7 (6) (2003) 561–575.
[5] L. Kuncheva, L. Jain, Nearest neighbor classifier: simultaneous editing and feature selection, Pattern Recognit. Lett. 20 (11) (1999) 1149–1156.
[6] L. Kuncheva, Editing for the k-nearest neighbors rule by a genetic algorithm, Pattern Recognit. Lett. 16 (8) (1995) 809–814.
[7] S. García, J. Cano, F. Herrera, A memetic algorithm for evolutionary prototype selection: a scaling up approach, Pattern Recognit. 41 (8) (2008) 2693–2709.
[8] I. Triguero, S. García, F. Herrera, IPADE: Iterative prototype adjustment for nearest neighbor classification, IEEE Trans. Neural Netw. 21 (12) (2010) 1984–1990.
[9] I. Triguero, S. García, F. Herrera, Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification, Pattern Recognit. 44 (4) (2011) 901–916.
[10] J. Derrac, I. Triguero, S. García, F. Herrera, Integrating instance selection, instance weighting, and feature weighting for nearest neighbor classifiers by coevolutionary algorithms, IEEE Trans. Syst. Man Cybern. B: Cybern. 42 (5) (2012) 1383–1397.
[11] C. Tsai, W. Eberle, C. Chu, Genetic algorithms in feature and instance selection, Knowl. Based Syst. 39 (2013) 240–247.
[12] J. Derrac, N. Verbiest, S. García, C. Cornelis, F. Herrera, On the use of evolutionary feature selection for improving fuzzy rough set based prototype selection, Soft Comput. 17 (2) (2013) 223–238.
[13] B. Krawczyk, I. Triguero, S. Garcia, M. Wozniak, F. Herrera, A first attempt on evolutionary prototype reduction for nearest neighbor one-class classification, in: 2014 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 747–753.
[14] T. Back, Evolutionary Algorithms in Theory and Practice, Oxford Univ. Press, 1996.
[15] L. Kuncheva, C. Whitaker, Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy, Mach. Learn. 51 (2) (2003) 181–207.
[16] K. Bache, M. Lichman, UCI machine learning repository. URL: http://archive.ics.uci.edu/ml19.
[17] J. Alcalá, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, KEEL data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, J. Mult. Valued Logic Soft Comput. 17 (2010) 255–287.
[18] M. Črepinšek, S. Liu, M. Mernik, Replication and comparison of computational experiments in applied evolutionary computing: common pitfalls and guidelines to avoid them, Appl. Soft Comput. 19 (2014) 161–170.
[19] F. Wilcoxon, Individual comparisons by ranking methods, Biom. Bull. (1945) 80–83.
[20] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evol. Comput. 1 (1) (2011) 3–18.
[21] N. Roussopoulos, S. Kelley, F. Vincent, Nearest neighbor queries, in: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD 95), ACM, 1995, pp. 71–79.
[22] M. Muja, D. Lowe, Fast approximate nearest neighbors with automatic algorithm configuration, VISAPP International Conference on Computer Vision Theory and Applications, 2009, pp. 331–340.
[23] I. Triguero, J. Derrac, S. Garcia, F. Herrera, A taxonomy and experimental study on prototype generation for nearest neighbor classification, IEEE Trans. Syst. Man Cybern. C: Appl. Rev. 42 (1) (2012) 86–100.
[24] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (4) (1997) 341–359.
[25] F. Neri, V. Tirronen, Scale factor local search in differential evolution, Memet. Comput. 1 (2) (2009) 153–171.
[26] V. Mayer-Schönberger, K. Cukier, Big Data: A Revolution That will Transform How We Live, Work, and Think, Houghton Mifflin Harcourt, 2013.
[27] I. Triguero, D. Peralta, J. Bacardit, S. García, F. Herrera, MRPR: A mapreduce solution for prototype reduction in big data classification, Neurocomputing 150 (2015) 331–345.
[28] I. Triguero, M. Galar, S. Vluymans, C. Cornelis, H. Bustince, F. Herrera, Y. Saeys, Evolutionary undersampling for imbalanced big data classification, in: Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC2015), IEEE, 2015, pp. 715–722.