Pattern Anal Applic (2010) 13:131–141
DOI 10.1007/s10044-008-0142-x

THEORETICAL ADVANCES

# A new fast prototype selection method based on clustering

**J. Arturo Olvera-López · J. Ariel Carrasco-Ochoa · J. Francisco Martínez-Trinidad**

Received: 15 February 2008 / Accepted: 11 September 2008 / Published online: 13 January 2009


**Abstract** In supervised classification, a training set $T$ is given to a classifier for classifying new prototypes. In practice, not all information in $T$ is useful for classifiers, therefore, it is convenient to discard irrelevant prototypes from $T$. This process is known as prototype selection, which is an important task for classifiers since through this process the time for classification or training could be reduced. In this work, we propose a new fast prototype selection method for large datasets, based on clustering, which selects border prototypes and some interior prototypes. Experimental results showing the performance of our method and comparing accuracy and runtimes against other prototype selection methods are reported.

**Keywords** Prototype selection · Supervised classification · Instance-based classifiers · Border prototypes · Data reduction · Clustering

## 1 Introduction

In Pattern Recognition, supervised classification is a process that assigns a class or label to new prototypes using a set of previously assessed prototypes called training set, denoted in this paper as $T$.

In practice, $T$ contains useless information for the classification task, that is, superfluous prototypes, which can be noisy or redundant therefore a process to discard them from $T$ is needed. This selection process is known as prototype selection. The main goal of a prototype selection method is to obtain a set $S \subset T$ such that $S$ does not contain superfluous prototypes.

Through prototype selection, the training set size is reduced, which could be useful for reducing the runtimes in the classification process, particularly for instance-based classifiers when they use large datasets.

There are two strategies [1, 2] for reducing the training set:

1. *Selection.* Some prototypes from $T$ are retained while ruling out those that do not contribute significantly to the classification accuracy.
2. *Replacement.* The original dataset is replaced by some labelled prototypes that do not necessarily coincide with some prototypes in $T$.

In particular, the prototype selection method proposed in this paper will follow the first strategy.

**Definition 1** The prototype $p_j \in T$ is a border prototype for the class $C_i$ if $p_j \in C_i$ and $p_j$ is the Nearest Neighbor of a prototype from another class $C_k \neq C_i$.

**Definition 2** The prototype $p_j \in T$ is an interior prototype for the class $C_i$ if $p_j$ is not a border prototype.

In a training set, the border prototypes of a class provide useful information to the classifier for preserving the class discrimination regions [3, 4]. On the other hand, interior prototypes of a class could be less useful. The aim of this

J. A. Olvera-López (✉) · J. A. Carrasco-Ochoa · J. F. Martínez-Trinidad
Computer Science Department,
National Institute of Astrophysics, Optics and Electronics,
Luis Enrrique Erro No. 1, Sta. María Tonantzintla,
CP 72000 Puebla, Mexico
e-mail: aolvera@ccc.inaoep.mx

J. A. Carrasco-Ochoa
e-mail: ariel@ccc.inaoep.mx

J. F. Martínez-Trinidad
e-mail: fmartine@ccc.inaoep.mx

paper is to present a new fast prototype selection method which finds and selects border and interior prototypes. Our method will use clustering for selecting border and interior prototypes.

In order to show the performance of our method, we present an experimental comparison between our method and some other prototype selection methods using the obtained prototype sets as training for the *k-NN* (*k-Nearest Neighbors*) [5], *Locally Weighted-Linear Regression* (*LWLR*) [6], *Support Vector Machines* (*SVM*) [7–9], decision trees (*C4.5*) [10] and *NB* (*Naive Bayes*) [11] classifiers. In addition, we report the runtimes for each method in order to show that our method is faster than the tested prototype selection methods, which confirms the advantage of the new method and its utility for large datasets, where prototype selection is more useful.

This paper is structured as follows: in Sect. 2, some works related to prototype selection are described. Section 3 introduces our method for prototype selection. Sect. 4 shows experimental results. Finally, Sect. 5 contains conclusions and directions for future work.

## 2 Related work

Several methods have been proposed for solving the prototype selection problem, in this section, some of the most relevant methods are briefly described.

The *Condensed Nearest Neighbor* (*CNN*) [12] and the *Edited Nearest Neighbor* (*ENN*) [13] rules are two of the first prototype selection methods. The *CNN* method starts with $S = \varnothing$ and its initial step consists in randomly including in $S$ one prototype belonging to each class. Then each prototype in $T$ is classified using only the prototypes in $S$. If a prototype is misclassified, it is added to $S$, to ensure that it will be correctly classified. This process is repeated until all prototypes in $T$ are correctly classified. This method ensures that $S$ classifies correctly all prototypes in $T$, this is, $S$ is consistent. *CNN* does not guarantee to find a minimal consistent subset. In [14], the concept of Mutual Nearest Neighbor (two prototypes do not belong to the mutual neighbourhood when they are not found in each other's *k-Nearest Neighborhood*) is used for selecting prototypes close to decision boundaries via a modified *CNN* algorithm.

Another variant of *CNN* is the *Generalized Condensed Nearest Neighbor Rule* (*GCNN*) [15], which is similar to *CNN* but *GCNN* includes in $S$ prototypes according to the *Absorption*($p$) criterion, which is calculated in terms of the nearest neighbor and nearest enemy (nearest prototype with different class) of $p$ in $S$. The selection process finishes when all prototypes in $T$ have been strongly absorbed, that is, when their *Absorption* satisfies a threshold value given by the user.

The ENN method discards from $T$ those prototypes that do not belong to their *k-Nearest Neighbors*' class. This method is used as noise filter because it deletes noisy prototypes, that is, prototypes with a different class in a neighborhood. A variant of this method is the *Repeated ENN* (*RENN*) where *ENN* is applied repeatedly until all prototypes in $S$ have the same class that the majority of their *k-Nearest Neighbors*. Another extension of *ENN* is the All *k-NN* prototype selection method [16]. This method works as follows: for $i = 1$ to $k$, flag as bad any prototype misclassified by its $i$ Nearest Neighbors. After completing the loop all $k$ times, remove any prototype flagged as bad.

Devijver and Kittler [17] proposed the *Multiedit* method for prototype selection, which makes $m$ random partitions ($P_1 \cdots P_m$) from $T$. After that, *ENN* (using 1-*NN*) is applied over each partition $P_i$ finding the neighbors of $P_i$ in $P_{(i+1) \bmod m}$. This process is repeated until there are not changes (eliminations) in $f$ successive iterations.

Wilson and Martinez [3] presented five methods *DROP1*,…,*DROP5* (*Decremental Reduction Optimization Procedure*) for prototype selection. These methods are based on the concept of *associate*. The *associates* of a prototype $p$ are those prototypes such that $p$ is one of their *k-Nearest Neighbors*. These methods discard the prototype $p$ if its associates can be correctly classified without $p$. *DROP1* uses the next selection rule: delete the prototype $p$ if the associates of $p$ (in $S$) are correctly classified without $p$. *DROP2* takes into account the effect in $T$ of eliminating a prototype in the partial subset, i.e., a prototype $p$ is deleted only if its associates in $T$ are correctly classified without $p$. *DROP3* uses a filter similar to *ENN* and after applies *DROP2*. *DROP4* is similar to *DROP3* but it uses a different filter. *DROP5* is based in *DROP2* but it starts deleting the prototypes that are nearest to their nearest enemies. According to the authors' results, the best methods are *DROP3* and *DROP5*.

The *Iterative Case Filtering algorithm* (*ICF*) was proposed in [4]. *ICF* is based on the Coverage and Reachable sets which are the neighborhood set and associates set, respectively. In this method, a prototype $p$ is flagged for removal if $|Reachable(p)| > |Coverage(p)|$, which means that more cases can solve $p$ than $p$ can solve itself. After, all prototypes flagged for removal are deleted.

Some authors [2, 18, 19] have suggested the idea of using clustering for prototype selection; this idea consists in: after splitting $T$ in $n$ clusters, the selected prototypes will be the centers of the $n$ clusters. In [20], the *CLU* prototype selection method is based on this idea and it was applied to the signature recognition problem. Another method that follows the same idea is *Nearest Sub-class Classifier* (*NSB*) [21] which allows selecting different number of prototypes (centers) per class via the *Maximum Variance Cluster algorithm* [22]. In [23], following the

same approach, the *Generalized-Modified Chang Algorithm* (*GCM*) method is proposed; this method merges the same-class nearest clusters and selects the centers from the new merged clusters.

The *Pair Opposite Class-Nearest Neighbor* (*POC-NN*) [24] selects border prototypes in $T$. The selection process in *POC-NN* is based on the mean of the prototypes in each class. For finding a border prototype $p_{b1}$ in the class $C_1$, *POC-NN* computes the mean $m_2$ of the prototypes in the opposite class $C_2$ and then $p_{b1}$ is the nearest prototype (belonging to class $C_1$) to $m_2$. Another method that finds border prototypes is proposed in [25] which, as *SVM*, applies the Structural Risk Minimization principle, but it is used to select border prototypes for nearest neighbor classification.

Most of the existing prototype selection methods are very expensive when large datasets are processed, and as consequence, these methods could be not applicable. In this paper we propose a prototype selection method, which is applicable for selecting prototypes in large datasets, where prototype selection is very useful.

## 3 Proposed method

In prototype selection, some authors (e.g. [3, 4]) argue that in a training set, interior prototypes can be removed with little effect on classification accuracy; and they also argue that border prototypes are critical for classification since they provide relevant information for preserving discrimination between classes. Many prototype selection methods, although they do not say it explicitly, select border prototypes. For instance, in Fig. 1a we show a synthetic bi-dimensional dataset with prototypes belonging to the classes "+" and "•". In Fig. 1b and e we depict the subsets selected by *DROP3*, *DROP5* (two of the most

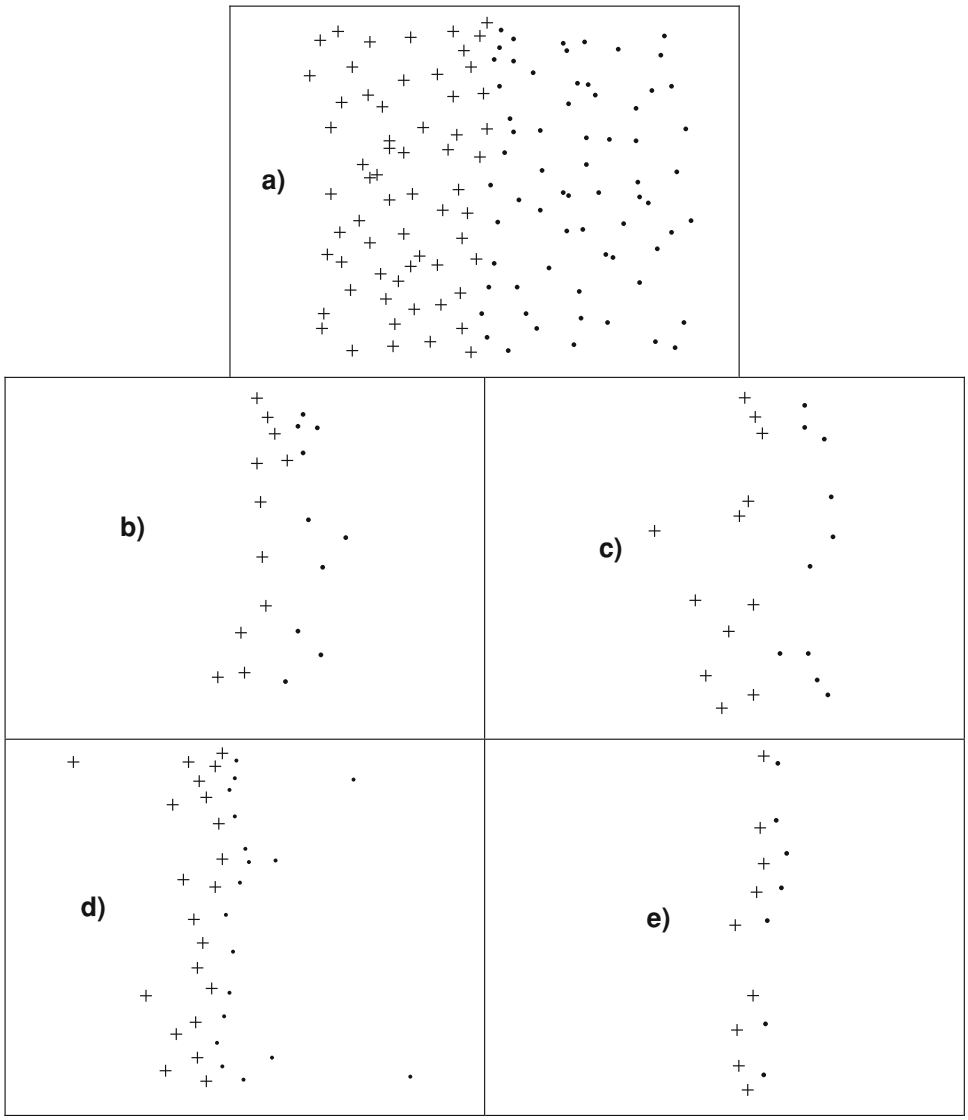


**Fig. 1** **a** Dataset with classes "+" and "•". **b** Prototype subset selected by *DROP3*. **c** Prototype subset selected by *DROP5*. **d** Prototype subset selected by *GCNN*. **e** Prototype subset selected by *POC-NN*

successful methods), *GCNN* (a recently proposed method) and *POC-NN* (a method proposed for selecting border prototypes). From Fig. 1, it can be seen that these methods indeed select prototypes near to the border between classes. However, it is important to notice that selecting border prototypes using the *k-NN* rule can be expensive mainly in large datasets. Motivated by these facts we propose a new fast prototype selection method that selects border prototypes and some of the interior ones.

Now the problem is how to select the border and interior prototypes. For solving this problem we propose to divide the training set in regions in order to find the border prototypes into small regions instead of finding them over the whole training set, which could be very expensive. The idea is to analyze those regions which contain prototypes belonging to different classes since they contain border prototypes. For regions containing prototypes belonging to a single class, our method will preserve only one representative prototype.

For dividing the training data in regions we use clustering, thus we called our method *Prototype Selection by Clustering* (*PSC*). We are interested in dividing the training data in regions (even if they contain prototypes from different classes), independently of the distribution of the data. Therefore, any clustering algorithm could be used, however, we decided to use *C-means* because of its simplicity, and because it allows us to specify the number of clusters to be created.

The border selection criterion in *PSC* is based on non homogeneous clusters. An homogeneous cluster is a prototype set such that all prototypes belong to the same class whereas in a non homogeneous cluster there are prototypes belonging to different classes. In order to find border prototypes, *PSC* generates clusters and analyzes non homogeneous clusters.

The *PSC* method for selecting prototypes is shown in Table 1. *PSC* starts creating $C$ clusters from $T$ (step 2). After that, for each cluster $A_j$ it is necessary to decide whether or not $A_j$ is homogeneous.

If $A_j$ is homogeneous (steps 4–7) then the prototypes in $A_j$ are interior prototypes, that is, they do not lie in the border between classes. *PSC* finds the nearest prototype $p_i$ to the mean (mean prototype) $m$ of cluster $A_j$ and discards the remaining prototypes from $A_j$ so that $A_j$ is reduced to $p_i$. *PSC* retains this kind of prototypes in order to preserve representative prototypes of homogeneous regions.

If $A_j$ is non homogeneous (steps 8–15) then there are in $A_j$ some prototypes located near to prototypes from other classes, that is, border prototypes. In order to find the border prototypes, *PSC* finds the majority class $C_M$ in $A_j$. Once these class has been found, the border prototypes in $C_M$ are those nearest prototypes in $A_j$ belonging to each class $C_k$, $C_k \neq C_M$ (steps 11–13), and in the same way, the border prototypes of $A_j$ in the other classes $C_k$ are those prototypes nearest to each border prototype in $C_M$ (steps 14–15).

The prototype set selected by *PSC* is the set of the mean prototypes from each homogeneous cluster and the border prototypes from each non homogeneous cluster.

To illustrate, in a graphical way, how *PSC* finds border prototypes let us consider the dataset shown in Fig. 2a (the same bi-dimensional dataset shown in Fig. 1a). The clusters ($C_1 \ldots C_{12}$) created by *C-means* from the dataset are depicted in Fig. 2b in which the non-homogeneous clusters

**Table 1** *PSC* method for prototype selection

**PSC** (Training set $T$, number of clusters $C$): prototype set $S$
1 $S = \varnothing$
2 *Clust*=*Gen_clust*($T$,$C$)
3 For each cluster $A_j$ in *Clust*
4 If $A_j$ is homogeneous then
5 $m$ = mean of cluster $A_j$
6 Let $p_i$ be the nearest prototype to $m$
7 $S = S \cup \{p_i\}$
8 Else // $A_j$ is non homogeneous
9 Let $C_M$ be the majority class in cluster $A_j$
10 For each class $C_k$ in $A_j$ ($C_k \neq C_M$)
11 For each prototype $p_j$ belonging to class $C_k$
12 Let $p_c \in C_M$ be the nearest prototype $p_j$
13 $S = S \cup \{p_c\}$
14 Let $p_M \in C_k$ be the nearest prototype to $p_c$
15 $S = S \cup \{p_M\}$
16 Return $S$

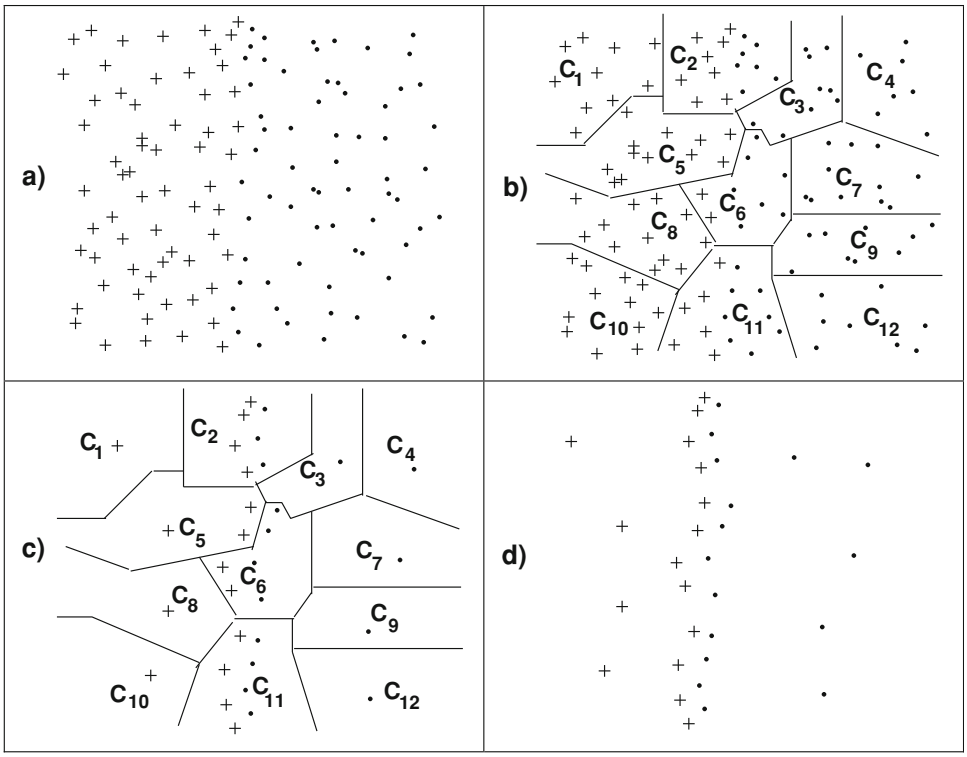


**Fig. 2** **a** Dataset with classes "+" and "•". **b** Clusters created from the dataset depicted in **a**. **c** Selected prototypes for each cluster. **d** Prototype subset selected by *PSC*

are $C_2$, $C_6$ and $C_{11}$ whereas the remaining clusters are homogeneous.

In the clusters $C_6$ and $C_{11}$, the minority class is "+", then the border prototypes in the most frequent class (•) are the nearest prototypes to each minority class prototype (+). On the other hand, the border prototypes in class "+" are the nearest prototypes (belonging to class "+") to each border in class "•".

The same process described before is applied to the cluster $C_2$ where the minority class is "•". The selected prototypes, for each cluster, are depicted in Fig. 2c and the prototype set, obtained by *PSC*, is depicted in Fig. 2d.

We can observe that *PSC* finds border prototypes and some interior prototypes (mean prototypes in the homogeneous clusters).

## 4 Experimental results

All the experiments of this section were done over 12 datasets taken from the UCI dataset repository [26]. For all the experiments, tenfold cross validation was used, that is, each dataset was divided into ten mutually exclusive blocks and each prototype selection method was applied over a training set $T$ built with nine of the ten blocks and the left block was used as testing set. Each block was used as testing set, and the average of the ten tests was reported.

In order to show the performance of the proposed method, we compared it against other selection methods. Since one of the methods compared against *PSC* was CLU, which divides the training set in clusters and selects only the centers (means), we carried out an experiment using different $C$ values over the 12 datasets, in order to choose the best $C$ value for each method, this is, the value where each method had the best performance in the average case. In Table 2, we show the accuracy obtained by *PSC* and *CLU* using $C = r, 4r, 6r, 8r$ and $10r$, where $r$ is the number of classes in the dataset. For testing the prototype sets selected by *PSC* and *CLU*, the *k-NN* classifier (using $k = 3$ and the Euclidean distance) was used. Additionally, we show the accuracy obtained by the original training set *Orig*. Based on the results shown in Table 2, we can observe that the best $C$ value for *PSC* was $C = 6r$ and the best one for *CLU* was $C = 8r$, therefore in the next experiments these values were used.

The second experiment consisted in comparing *PSC* against *CLU*, *POC-NN*, *GCNN*, *DROP3*, and *DROP5*, using *k-NN* with $k = 3$ (the best value for *DROP* methods [3]) and the Euclidean distance. We chose these methods because according to the results reported in [3, 4], *DROP3* and *DROP5* were the best methods among the *DROP*s and they outperformed to other relevant prototype selection methods such as ENN, RENN and *ICF*. *GCNN* was also considered in our comparison, since according to results reported by its authors; this method is competitive against the *DROP*s. *CLU* was compared because it is another prototype selection method based on clustering. *POC-NN* was included because it was designed to find border prototypes.

**Table 2** Accuracy obtained by *CLU* and *PSC* creating different number of clusters

| Dataset | *Orig.* | Number of clusters | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $C = 2r$ | | $C = 4r$ | | $C = 6r$ | | $C = 8r$ | | $C = 10r$ | |
| | | *CLU* | *PSC* | *CLU* | *PSC* | *CLU* | *PSC* | *CLU* | *PSC* | *CLU* | *PSC* |
| Glass | 71.42 | 43.41 | 58.62 | 53.26 | 56.21 | 52.83 | 59.09 | 55.58 | 56.16 | 56.08 | 61.77 |
| Iris | 94.66 | 64.66 | 79.88 | 84.00 | 84.66 | 86.66 | 94.66 | 87.33 | 88.88 | 84.00 | 83.33 |
| Letter | 95.00 | 43.20 | 70.10 | 44.51 | 74.08 | 41.22 | 79.35 | 48.35 | 79.45 | 54.95 | 81.88 |
| Liver | 65.22 | 49.79 | 55.01 | 51.57 | 55.32 | 52.18 | 55.36 | 52.58 | 54.54 | 51.87 | 54.78 |
| Pendigits | 99.43 | 56.80 | 92.60 | 79.71 | 92.90 | 85.74 | 94.32 | 85.74 | 94.15 | 86.61 | 94.34 |
| Segmentation | 95.10 | 51.95 | 83.19 | 70.71 | 90.52 | 81.00 | 91.28 | 83.47 | 90.80 | 84.85 | 90.23 |
| Sonar | 82.71 | 60.35 | 72.52 | 71.88 | 76.14 | 68.09 | 79.76 | 56.73 | 77.11 | 54.78 | 68.59 |
| Spambase | 80.55 | 52.53 | 67.65 | 60.33 | 69.37 | 66.95 | 71.95 | 66.95 | 69.44 | 66.68 | 69.52 |
| Thyroid | 95.45 | 82.35 | 87.07 | 87.38 | 89.39 | 85.10 | 90.75 | 88.88 | 89.35 | 85.54 | 89.26 |
| UPS | 96.48 | 54.30 | 91.36 | 72.18 | 91.30 | 78.71 | 90.35 | 82.30 | 90.68 | 81.80 | 90.16 |
| Wine | 94.44 | 75.16 | 82.51 | 85.91 | 88.30 | 88.11 | 92.67 | 87.37 | 88.41 | 88.30 | 88.19 |
| Yeast | 54.64 | 44.06 | 46.83 | 43.92 | 46.49 | 45.68 | 53.04 | 46.96 | 51.04 | 46.69 | 50.67 |
| Average | 85.43 | 56.55 | 73.95 | 67.11 | 76.22 | 69.36 | 79.38 | 70.19 | 77.50 | 70.18 | 76.89 |

In Table 3, we report the results obtained applying *DROP3*, *DROP5*, *GCNN*, *POC-NN*, *CLU* and *PSC* over the 12 datasets. For each method, the classification accuracy (*Acc*) obtained by *k-NN* with the selected subsets, and the retention (*Str*), that is, the percentage of the original training set that was retained by each method, are shown; the retention is computed as $Str = 100|S|/|T|$, where $S$ is the prototype subset selected by each method, from the training set $T$. In the tables shown in this document, the symbol "*" next to the accuracy results indicates that there is a statistically significant difference with respect to our method (according to the $k$-fold cross-validated paired $t$ test [27] with 9 degrees of freedom and a confidence of 99%).

In Fig. 3, the scatter graphic of classification accuracy obtained by *k-NN* with the selected subsets (horizontal axis) versus retention (vertical axis), from average results shown in Table 3, is depicted. In this kind of graphic, the most located at right the best accuracy and the most located at bottom the best retention percentage.

In Table 4, the dataset sizes and runtimes[1] spent by each method are shown. The symbols "+", "++", and "+++" are used to indicate that the runtime was more than 10, 20, and 30 h, respectively.

According to the results shown in Table 3 and Fig. 3, in the average case, the best prototype selection method in accuracy was *DROP3*. The accuracy results obtained by *PSC* were better than *CLU*'s results, but lower than those obtained by *POC-NN*, the *DROP*s and *GCNN*. It was expected, since the *k-NN* classifier was used and these methods are strongly based on the *k-NN* rule. However, it is important to highlight that according to Table 4, for the larger datasets, *CLU* and *PSC* were much faster than *POC-NN*, the *DROP*s and *GCNN*.

In the previous experiment, the classifier was *k-NN*, but in order to know how good the selected subsets are, as training sets, for other classifiers, as third experiment the prototype sets obtained by the *DROP*s, *GCNN*, *POC-NN*, *CLU* and *PSC* were used as training for: *Locally Weighted-Linear Regression* (*LWLR*), *Support Vector Machines* (*SVM*), *C4.5* (*Decision trees*) and *Naive Bayes* (*NB*).[2] Some of these classifiers, in particular *SVM* and *LWLR* require long times for training or classifying on large datasets, then for these classifiers, prototype selection is required for reducing the training and/or classification runtimes. The results obtained in this experiment are shown in Tables 5, 6, 7, 8 and Figs. 4, 5, 6 and 7. The retention results in these tables are identical to those in Table 3 because the same prototype subsets obtained in the previous experiment were tested, but now, using other classifiers.

According to the results reported in Table 5 and Fig. 4, we can observe that, for *LWLR*, the best method in most of the cases was *GCNN* but in the majority of these cases there is not significant difference between *GCNN* and *PSC*, which was the best method in the average case.

Based on the results obtained using *SVM* (Table 6; Fig. 5), in accuracy, the best method was *GCNN* which

[1] These runtimes were obtained using an Intel Celeron CPU 2.4 GHz, 512 MB RAM.

[2] For *SVM*, we used the software from [28], for *C4.5* and *Naive Bayes* WEKA [29] was used, *LWLR* and *k-NN* were implemented in MATLAB [30].

**Table 3** Accuracy (*Acc*) and retention (*Str*) results using the original training set (*Orig*) and the prototype sets obtained by *DROPs*, *GCNN*, *POC-NN*, *CLU* and *PSC* as training for *k-NN*

| Dataset | *Orig.* | | *DROP3* | | *DROP5* | | *GCNN* | | *POC-NN* | | *CLU* | | *PSC* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* |
| Glass | 71.42 | 100 | 66.28* | 24.35 | 62.16* | 25.91 | 69.61* | 61.62 | 71.47* | 57.42 | 55.58* | 25.13 | 59.09 | 38.45 |
| Iris | 94.66 | 100 | 95.33* | 15.33 | 94.00* | 12.44 | 96.00* | 38.00 | 77.33* | 12.88 | 87.33* | 18.21 | 94.66 | 20.45 |
| Letter | 95.00 | 100 | 92.68* | 16.33 | 92.17* | 13.63 | 95.29* | 34.08 | 94.62* | 42.41 | 48.35* | 1.04 | 79.35 | 18.33 |
| Liver | 65.22 | 100 | 67.82* | 26.83 | 63.46 | 30.59 | 66.09* | 83.70 | 55.68 | 58.09 | 52.58* | 6.36 | 55.36 | 45.87 |
| Pendigits | 99.43 | 100 | 98.89* | 5.67 | 98.16* | 4.00 | 97.31* | 11.10 | 97.86* | 9.61 | 85.74* | 0.88 | 94.32 | 6.23 |
| Segmentation | 95.10 | 100 | 92.19* | 15.94 | 91.86* | 14.30 | 92.71* | 13.82 | 87.38* | 19.54 | 83.47* | 2.22 | 91.28 | 15.14 |
| Sonar | 82.71 | 100 | 74.60* | 30.13 | 75.95* | 29.86 | 83.88* | 95.61 | 80.95* | 54.38 | 56.73* | 8.54 | 79.76 | 35.41 |
| Spambase | 80.55 | 100 | 78.44* | 15.71 | 78.72* | 20.97 | 73.54 | 1.33 | 75.37* | 34.04 | 66.95* | 0.28 | 71.95 | 24.74 |
| Thyroid | 95.45 | 100 | 93.98 | 9.77 | 94.46 | 8.84 | 93.96 | 30.33 | 93.05 | 15.34 | 88.88* | 11.65 | 90.75 | 13.47 |
| UPS | 96.48 | 100 | 94.59* | 10.27 | 93.99* | 7.96 | 94.78* | 34.53 | 95.63* | 29.80 | 82.30* | 0.95 | 90.35 | 15.47 |
| Wine | 94.44 | 100 | 94.41* | 15.04 | 93.86* | 10.55 | 94.44* | 78.89 | 94.93* | 40.82 | 87.37* | 12.30 | 92.67 | 37.15 |
| Yeast | 54.64 | 100 | 56.26* | 19.54 | 54.58* | 25.30 | 46.56 | 31.40 | 53.63* | 78.22 | 46.96* | 4.49 | 53.04 | 43.20 |
| Average | 85.43 | 100 | 83.79 | 17.08 | 82.78 | 17.03 | 83.68 | 42.87 | 81.49 | 37.71 | 70.19 | 7.67 | 79.38 | 26.16 |

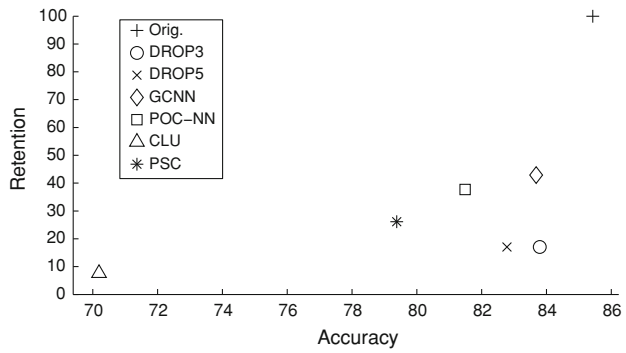


**Fig. 3** Scatter graphic of the average results shown in table 3

obtained the best results for five of the twelve datasets, but in three of them *GCNN* does not have significant difference with respect to *PSC*. In the average case *PSC* was the best method again.

Using *C4.5* and *NB* (Tables 7, 8; Figs. 6, 7), the best methods in accuracy were *POC-NN* and *GCNN* which had the better performance in five and four datasets, respectively. These methods were followed by *PSC* which had the best performance for three datasets. Once again, in the average case, the best method in accuracy was *PSC*.

Based on the results, we can see that for *LWLR*, *SVM*, *C4.5*, and *NB*, all the tested prototype selection methods

**Table 4** Datasets sizes and runtimes (in seconds) spent by the tested methods. "+", "++" and "+++" indicates that the runtime was more than 10, 20 and 30 hours respectively

| Dataset | Size | | | Runtimes | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Prototypes | Features | Classes | DROP3 | DROP5 | GCNN | POC-NN | CLU | PSC |
| Iris | 150 | 4 | 3 | 0.19 | 0.16 | 1.25 | 1.02 | 0.74 | 0.79 |
| Wine | 178 | 13 | 3 | 0.84 | 0.59 | 3.03 | 2.32 | 1.62 | 1.82 |
| Sonar | 208 | 60 | 2 | 1.77 | 2.57 | 20.15 | 19.33 | 1.80 | 1.94 |
| Glass | 214 | 9 | 6 | 0.46 | 0.47 | 4.14 | 3.02 | 0.43 | 0.54 |
| Thyroid | 215 | 5 | 3 | 0.37 | 0.40 | 2.10 | 1.60 | 0.67 | 0.69 |
| Liver | 345 | 6 | 2 | 0.76 | 0.76 | 19.45 | 7.21 | 1.62 | 1.70 |
| Yeast | 1,484 | 8 | 10 | 25.60 | 23.80 | 280.75 | 250.19 | 3.97 | 4.35 |
| Segmentation | 2,100 | 19 | 7 | 208.39 | 208.91 | 684.78 | 689.21 | 15.84 | 16.52 |
| Spambase | 4,601 | 57 | 2 | 3782.57 | 2226.42 | 348.56 | 735.08 | 181.73 | 189.57 |
| Pendigits | 7,494 | 16 | 10 | 3180.28 | 2460.05 | 18642.30 | 12600.87 | 93.13 | 94.64 |
| Letter | 20,000 | 16 | 26 | 15213.13 | 14339.61 | + | ++ | 596.64 | 613.58 |
| UPS | 9,000 | 255 | 10 | + | ++ | +++ | +++ | 783.65 | 793.94 |

**Table 5** Accuracy and retention results obtained using the original training set (*Orig.*) and the prototype sets obtained by: *DROPs, GCNN, POC-NN, CLU* and *PSC* as training for *LWLR*

| Dataset | *Orig.* | | *DROP3* | | *DROP5* | | *GCNN* | | *POC-NN* | | *CLU* | | *PSC* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* |
| Glass | 57.85 | 100 | 50.09* | 24.35 | 52.38 | 25.91 | 56.88 | 61.62 | 50.41 | 57.42 | 46.70* | 25.13 | 56.06 | 38.45 |
| Iris | 98.00 | 100 | 92.00* | 15.33 | 92.00* | 12.44 | 95.33 | 38.00 | 85.33 | 12.88 | 88.00* | 18.21 | 96.00 | 20.45 |
| Letter | 41.42 | 100 | 24.38 | 16.33 | 26.13 | 13.63 | 25.45 | 34.08 | 28.27 | 42.41 | 22.70* | 1.04 | 27.08 | 18.33 |
| Liver | 70.13 | 100 | 68.26 | 26.83 | 69.54 | 30.59 | 70.13 | 83.70 | 68.68 | 58.09 | 48.39* | 6.36 | 68.57 | 45.87 |
| Pendigits | 59.63 | 100 | 51.66 | 5.67 | 54.06 | 4.00 | 49.25* | 11.10 | 44.60 | 9.61 | 42.40* | 0.88 | 48.83 | 6.23 |
| Segmentation | 78.28 | 100 | 55.14 | 15.94 | 49.80 | 14.30 | 55.90 | 13.82 | 54.57 | 19.54 | 34.76* | 2.22 | 53.33 | 15.14 |
| Sonar | 64.40 | 100 | 64.50 | 30.13 | 59.66 | 29.86 | 67.73 | 95.61 | 62.47 | 54.38 | 52.23* | 8.54 | 64.47 | 35.41 |
| Spambase | 92.67 | 100 | 88.08 | 15.71 | 89.76* | 20.97 | 62.09* | 1.33 | 91.16* | 34.04 | 62.60* | 0.28 | 89.34 | 24.74 |
| Thyroid | 91.16 | 100 | 75.43* | 9.77 | 75.93 | 8.84 | 76.19* | 30.33 | 64.02* | 15.34 | 66.51* | 11.65 | 71.42 | 13.47 |
| UPS | 49.85 | 100 | 32.55* | 10.27 | 34.79* | 7.96 | 41.94* | 34.53 | 40.60* | 29.80 | 48.84* | 0.95 | 36.72 | 15.47 |
| Wine | 92.15 | 100 | 62.75* | 15.04 | 60.78* | 10.55 | 92.15 | 78.89 | 86.40 | 40.82 | 53.88* | 12.30 | 88.72 | 37.15 |
| Yeast | 37.12 | 100 | 36.65 | 19.54 | 36.86* | 25.30 | 35.42* | 31.40 | 35.71 | 78.22 | 31.80* | 4.49 | 35.23 | 43.20 |
| Average | 69.39 | 100 | 58.46 | 17.08 | 58.47 | 17.03 | 60.71 | 42.87 | 59.35 | 37.71 | 49.90 | 7.67 | 61.31 | 26.16 |

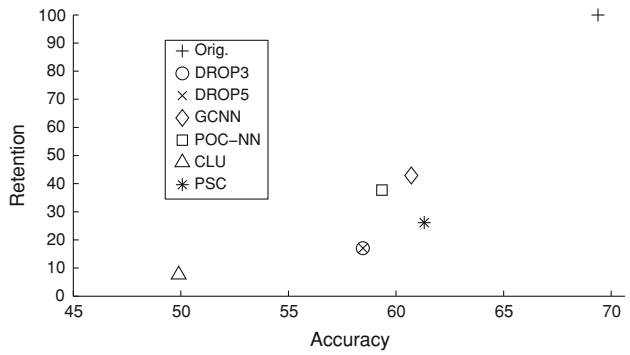


**Fig. 4** Scatter graphic of the average accuracy results using *LWLR* (Table 5)

are competitive in accuracy, excluding *CLU* which obtained poor accuracy results for all the classifiers.

The last experiment, related to the runtimes, was done using the *Statlog-Shuttle* dataset (also taken from UCI) which is larger than the 12 datasets used before. In this experiment we measured the runtimes spent by each method over different training set sizes constructed from the prototypes in the *Statlog-Shuttle* dataset. The results are shown in Fig. 8. In this experiment, *POC-NN* was not able to run over training sets larger than 10,000 prototypes (we used the source code provided by its authors). In Fig. 8, some points are not depicted because they are out of the vertical axis scale.

**Table 6** Accuracy and retention results obtained using the original training set (*Orig.*) and the prototype sets obtained by: *DROPs, GCNN, POC-NN, CLU* and *PSC* as training for *SVM*

| Dataset | *Orig.* | | *DROP3* | | *DROP5* | | *GCNN* | | *POC-NN* | | *CLU* | | *PSC* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* |
| Glass | 72.29 | 100 | 59.74 | 24.35 | 63.50 | 25.91 | 66.82* | 61.62 | 57.46* | 57.42 | 56.03* | 25.13 | 65.84 | 38.45 |
| Iris | 96.66 | 100 | 91.33 | 15.33 | 93.33 | 12.44 | 94.66 | 38.00 | 94.00 | 12.88 | 85.33* | 18.21 | 93.33 | 20.45 |
| Letter | 82.34 | 100 | 73.38* | 16.33 | 71.47* | 13.63 | 74.54* | 34.08 | 78.43* | 42.41 | 69.26* | 1.04 | 72.90 | 18.33 |
| Liver | 70.72 | 100 | 58.26 | 26.83 | 56.78 | 30.59 | 69.73 | 83.70 | 58.56 | 58.09 | 52.50* | 6.36 | 64.07 | 45.87 |
| Pendigits | 99.65 | 100 | 98.89 | 5.67 | 98.42* | 4.00 | 94.98* | 11.10 | 95.87* | 9.61 | 92.06* | 0.88 | 98.49 | 6.23 |
| Segmentation | 91.58 | 100 | 86.80* | 15.94 | 85.38* | 14.30 | 87.23* | 13.82 | 83.85* | 19.54 | 70.95* | 2.22 | 83.95 | 15.14 |
| Sonar | 85.14 | 100 | 65.35 | 30.13 | 65.90 | 29.86 | 78.09 | 95.61 | 77.47 | 54.38 | 61.02* | 8.54 | 75.21 | 35.41 |
| Spambase | 93.30 | 100 | 90.56* | 15.71 | 92.02* | 20.97 | 78.67 | 1.33 | 89.78* | 34.04 | 57.17* | 0.28 | 88.43 | 24.74 |
| Thyroid | 93.54 | 100 | 90.45 | 9.77 | 93.03 | 8.84 | 89.78 | 30.33 | 92.61* | 15.34 | 87.01* | 11.65 | 90.78 | 13.47 |
| UPS | 94.92 | 100 | 93.47* | 10.27 | 92.66* | 7.96 | 85.35* | 34.53 | 93.22* | 29.80 | 92.11* | 0.95 | 94.08 | 15.47 |
| Wine | 97.18 | 100 | 94.93 | 15.04 | 92.71* | 10.55 | 94.83* | 78.89 | 95.52 | 40.82 | 74.21* | 12.30 | 95.52 | 37.15 |
| Yeast | 57.81 | 100 | 50.60* | 19.54 | 56.81* | 25.30 | 52.96 | 31.40 | 52.57 | 78.22 | 47.29* | 4.49 | 54.55 | 43.20 |
| Average | 86.26 | 100 | 79.48 | 17.08 | 80.17 | 17.03 | 81.20 | 42.87 | 80.85 | 37.71 | 69.85 | 7.67 | 81.36 | 26.16 |

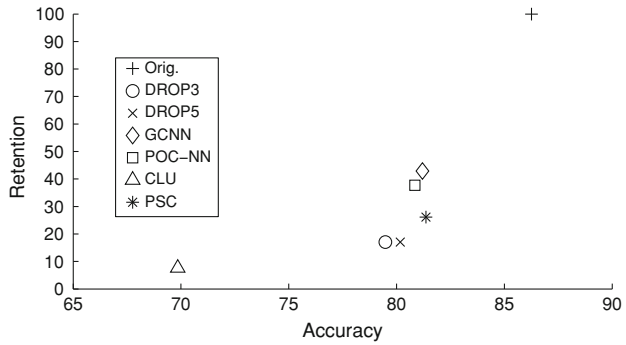


**Fig. 5** Scatter graphic of the average accuracy results using *SVM* (Table 6)

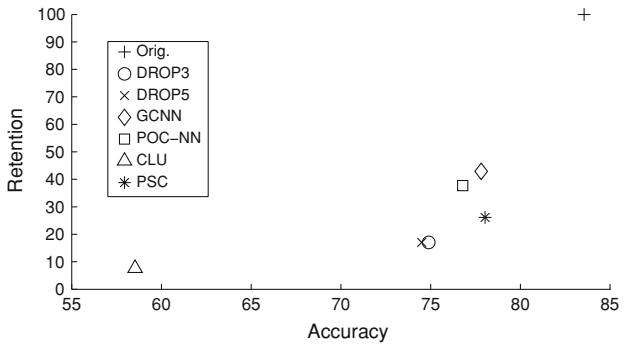


**Fig. 6** Scatter graphic of the average accuracy results using *C4.5* (Table 7)

**Table 7** Accuracy and retention results obtained using the original training set (*Orig.*) and the prototype sets obtained by: *DROPs, GCNN, POC-NN, CLU* and *PSC* as training for *C4.5*

| Dataset | *Orig.* | | *DROP3* | | *DROP5* | | *GCNN* | | *POC-NN* | | *CLU* | | *PSC* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* |
| Glass | 67.29 | 100 | 60.19 | 24.35 | 53.76 | 25.91 | 60.75 | 61.62 | 63.56* | 57.42 | 49.67* | 25.13 | 60.58 | 38.45 |
| Iris | 93.99 | 100 | 92.66* | 15.33 | 90.66 | 12.44 | 90.66 | 38.00 | 73.99 | 12.88 | 82.66* | 18.21 | 90.66 | 20.45 |
| Letter | 88.29 | 100 | 72.96* | 16.33 | 73.00* | 13.63 | 81.21* | 34.08 | 83.35* | 42.41 | 48.55* | 1.04 | 74.80 | 18.33 |
| Liver | 63.67 | 100 | 59.48 | 26.83 | 63.67 | 30.59 | 61.76* | 83.70 | 54.46 | 58.09 | 54.19* | 6.36 | 63.67 | 45.87 |
| Pendigits | 96.09 | 100 | 75.78 | 5.67 | 76.33 | 4.00 | 82.68* | 11.10 | 88.53* | 9.61 | 61.80* | 0.88 | 79.44 | 6.23 |
| Segmentation | 96.02 | 100 | 81.61* | 15.94 | 88.75 | 14.30 | 85.71 | 13.82 | 87.80 | 19.54 | 58.28* | 2.22 | 89.00 | 15.14 |
| Sonar | 72.57 | 100 | 73.45 | 30.13 | 70.73 | 29.86 | 73.00 | 95.61 | 70.19 | 54.38 | 54.14* | 8.54 | 77.45 | 35.41 |
| Spambase | 92.52 | 100 | 89.15 | 15.71 | 90.52 | 20.97 | 79.32* | 1.33 | 89.34* | 34.04 | 50.74* | 0.28 | 89.61 | 24.74 |
| Thyroid | 94.89 | 100 | 81.79 | 9.77 | 82.64 | 8.84 | 90.12 | 30.33 | 80.88 | 15.34 | 74.84* | 11.65 | 88.83 | 13.47 |
| UPS | 87.79 | 100 | 74.42 | 10.27 | 74.35 | 7.96 | 85.77* | 34.53 | 84.27* | 29.80 | 48.49* | 0.95 | 80.19 | 15.47 |
| Wine | 94.44 | 100 | 84.43 | 15.04 | 78.88 | 10.55 | 95.55 | 78.89 | 90.32 | 40.82 | 75.55* | 12.30 | 90.77 | 37.15 |
| Yeast | 55.04 | 100 | 52.70* | 19.54 | 50.67 | 25.30 | 47.30 | 31.40 | 54.76* | 78.22 | 43.39* | 4.49 | 51.35 | 43.20 |
| Average | 83.55 | 100 | 74.89 | 17.08 | 74.50 | 17.03 | 77.82 | 42.87 | 76.79 | 37.71 | 58.53 | 7.67 | 78.03 | 26.16 |

**Table 8** Accuracy and retention results obtained using the original training set (*Orig.*) and the prototype sets obtained by: *DROPs, GCNN, POC-NN, CLU* and *PSC* as training for *NB*

| Dataset | *Orig.* | | *DROP3* | | *DROP5* | | *GCNN* | | *POC-NN* | | *CLU* | | *PSC* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* | *Acc* | *Str* |
| Glass | 48.05 | 100 | 49.56 | 24.35 | 47.74 | 25.91 | 47.57 | 61.62 | 46.66 | 57.42 | 55.60* | 25.13 | 47.14 | 38.45 |
| Iris | 95.33 | 100 | 91.99 | 15.33 | 93.99 | 12.44 | 95.33 | 38.00 | 76.66 | 12.88 | 92.66* | 18.21 | 93.33 | 20.45 |
| Letter | 64.00 | 100 | 53.88* | 16.33 | 56.09* | 13.63 | 45.50 | 34.08 | 57.83 | 42.41 | 46.97* | 1.04 | 51.06 | 18.33 |
| Liver | 56.02 | 100 | 61.50 | 26.83 | 61.77 | 30.59 | 56.88 | 83.70 | 56.57 | 58.09 | 47.79* | 6.36 | 56.88 | 45.87 |
| Pendigits | 87.97 | 100 | 80.57 | 5.67 | 81.30 | 4.00 | 84.45 | 11.10 | 79.46 | 9.61 | 80.49* | 0.88 | 82.60 | 6.23 |
| Segmentation | 80.19 | 100 | 75.23 | 15.94 | 71.76 | 14.30 | 76.66 | 13.82 | 71.71 | 19.54 | 67.57* | 2.22 | 74.76 | 15.14 |
| Sonar | 66.26 | 100 | 66.88 | 30.13 | 58.71 | 29.86 | 64.78 | 95.61 | 67.92 | 54.38 | 54.16* | 8.54 | 69.35 | 35.41 |
| Spambase | 79.41 | 100 | 76.22* | 15.71 | 76.89* | 20.97 | 81.61* | 1.33 | 73.14 | 34.04 | 41.25* | 0.28 | 77.39 | 24.74 |
| Thyroid | 97.22 | 100 | 95.82 | 9.77 | 95.77 | 8.84 | 95.36 | 30.33 | 95.82 | 15.34 | 93.05* | 11.65 | 95.45 | 13.47 |
| UPS | 77.21 | 100 | 71.58 | 10.27 | 70.51 | 7.96 | 72.27 | 34.53 | 69.21 | 29.80 | 70.58* | 0.95 | 73.54 | 15.47 |
| Wine | 98.81 | 100 | 61.11* | 15.04 | 66.66* | 10.55 | 96.66 | 78.89 | 91.11 | 40.82 | 85.55* | 12.30 | 97.77 | 37.15 |
| Yeast | 57.47 | 100 | 55.58 | 19.54 | 53.63 | 25.30 | 49.39* | 31.40 | 55.85 | 78.22 | 41.10* | 4.49 | 54.63 | 43.20 |
| Average | 75.66 | 100 | 69.99 | 17.08 | 69.57 | 17.03 | 72.21 | 42.87 | 70.16 | 37.71 | 64.73 | 7.67 | 72.83 | 26.16 |

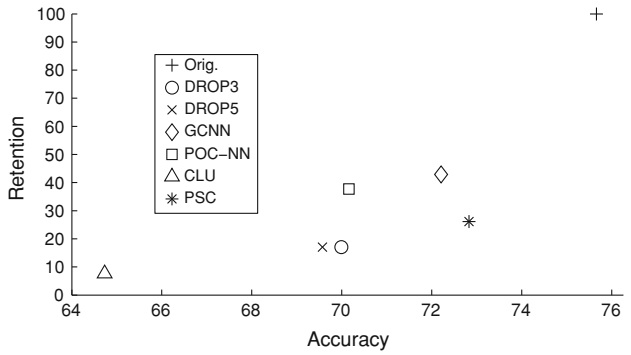


**Fig. 7** Scatter graphic of the average accuracy results using *NB* (Table 8)

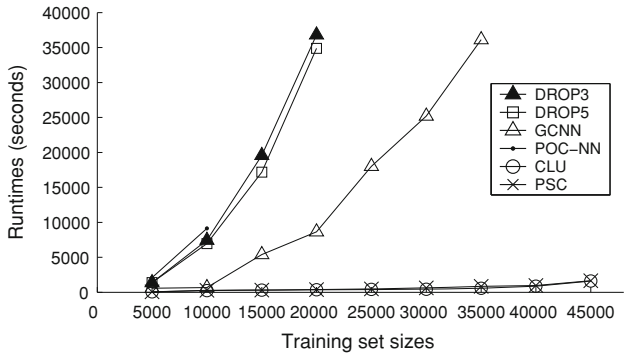


**Fig. 8** Runtimes spent by each method using different training set sizes

Based on Fig. 8, we can notice that *CLU* and *PSC* clearly are the fastest methods, and therefore, they are the only methods that can be applied for large training sets, but *PSC* is much better in accuracy than *CLU* (Tables 3, 5, 6, 7, 8; Figs. 3, 4, 5, 6, 7).

## 5 Conclusions

In supervised classification, prototype selection is an important task, specially for instance-based classifiers, because through this task, the runtimes in the classification stage could be reduced. In this paper, we introduced and compared the *PSC* method for prototype selection. Our method selects border prototypes (in order to preserve the class discrimination regions) and some interior prototypes. The experimental results using different classifiers showed that *PSC* had a competitive performance against other methods.

According to our results for large datasets (datasets where prototype selection is more useful), *CLU* and *PSC* were the fastest methods but *PSC* was much better than *CLU* in accuracy. Therefore, we can conclude that for large datasets, *PSC* is the best choice for selecting prototypes. This characteristic of our method is very important because other successful prototype selection methods spent a lot of time (or could be inapplicable) for processing large datasets.

As we described in Sect. 3, *PSC* requires, as parameter, the value of *C* for the *C-means* algorithm, then, as future direction, we are interested in automatically fixing this parameter.

## References

1. Kuncheva LI, Bezdek JC (1998) Nearest prototype classification, clustering, genetic algorithms, or random search? IEEE Trans Syst Man Cybern C28(1):160–164
2. Bezdek JC, Kuncheva LI (2001) Nearest prototype classifier designs: an experimental study. Int J Intell Syst 16(12):1445–1473
3. Wilson DR, Martínez TR (2000) Reduction techniques for instance-based learning algorithms. Mach Learn 38:257–286
4. Brighton H, Mellish C (2002) Advances in instance selection for instance-based learning algorithms. Data Min Knowl Disc 6(2):153–172
5. Cover T, Hart P (1967) Nearest neighbor pattern classification. IEEE Trans Inf Theory 13:21–27
6. Atkeson CG, Moorel AW, Schaal S (1997) Locally weighted learning. Artif Intell Rev 11(1–5):11–73
7. Vapnik V (1995) The nature of statistical learning theory. Springer, New York
8. Vapnik VN (1998) Statistical learning theory. Wiley, New York
9. Cristanni N, Shawe-Taylor J (2000) An introduction to support vector machines and other kernel-based learning methods. Cambridge University Press, Cambridge
10. Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Mateo
11. Duda RO, Hart PE, Stork DG (2000) Pattern classification, 2nd edn. Wiley, New York
12. Hart PE (1968) The condensed nearest neighbor rule. IEEE Trans Inf Theory 14:515–516
13. Wilson DL (1972) Asymptotic properties of nearest neighbor rules using edited data. IEEE Trans Syst Man Cybern 2:408–421
14. Chidananda GK, Krishna G (1979) The condensed nearest neighbor rule using the concept of mutual nearest neighborhood. IEEE Trans Inf Theory 25:488–490
15. Chien-Hsing C, Bo-Han K, Fu C (2006) The generalized condensed nearest neighbor rule as a data reduction method. In: Proceedings of the 18th international conference on pattern recognition. IEEE Computer Society, Hong-Kong, pp 556–559
16. Tomek I (1976) An experiment with the edited nearest-neighbor rule. IEEE Trans Syst Man Cybern 6–6:448–452
17. Devijver PA, Kittler J (1980) On the edited nearest neighbor rule. In: Proceedings of the fifth international conference on pattern recognition, Los Alamitos, CA, pp 72–80
18. Liu H, Motoda H (2002) On issues of instance selection. Data Min Knowl Disc 6:115–130
19. Spillmann B, Neuhaus M, Bunke H, Pękalska E, Duin RPW (2006) Transforming strings to vector spaces using prototype selection. In: Yeung D-Y et al (eds) SSPR & SPR 2006, Lecture Notes in Computer Science, vol 4109, Hong-Kong, pp 287–296
20. Lumini A, Nanni L (2006) A clustering method for automatic biometric template selection. Pattern Recogn 39:495–497

21. Venmann CJ, Reinders MJT (2005) The nearest sub-class classifier: a compromise between the nearest mean and nearest neighbor classifier. IEEE Trans Pattern Anal Mach Intell 27(9):1417–1429
22. Venmann CJ, Reinders MJT, Backer E (2002) A maximum variance clustering algorithm. IEEE Trans Pattern Anal Mach Intell 24(9):1273–1280
23. Mollineda RA, Ferri FJ, Vidal E (2002) An efficient prototype merging strategy for the condensed 1-NN rule through class-conditional hierarchical clustering. Pattern Recogn 35:2771–2782
24. Raicharoen T, Lursinsap C (2005) A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (POC-NN) algorithm. Pattern Recognit Lett 26(10):1554–1567
25. Karaçali B, Krim H (2002) Fast minimization of structural risk by nearest neighbor rule. IEEE Trans Neural Netw 14:127–137
26. Asuncion A, Newman DJ (2007) UCI machine learning repository. In: University of California, School of Information and Computer Science, Irvine, CA. http://www.ics.uci.edu/~mlearn/MLRepository.html
27. Dietterich TG (1998) Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput 10(7):1895–1924
28. Vojtech F, Václav H (2004) Statistical pattern recognition toolbox for Matlab. Research report, Center for Machine Perception Department of Cybernetic, Faculty of Electrical Engineering, Czech Technical University
29. Witten IH, Frank E (2005) Data mining: practical machine learning tools techniques, 2nd edn. Morgan Kaufmann, San Francisco
30. The MathWorks Inc. (1994–2008) Natick. [http://www.mathworks.com]