Extensions to rank-based prototype selection in k-Nearest Neighbour classification

Juan Ramón Rico-Juan[a], Jose J. Valero-Mas[b], Jorge Calvo-Zaragoza[a]

[a] *Department of Software and Computing Systems, University of Alicante, Carretera San Vicente del Raspeig s/n, 03690 Alicante, Spain*
[b] *Independent scientist, Carretera San Vicente del Raspeig s/n, 03690 Alicante, Spain*

Abstract

The k-nearest neighbour rule is commonly considered for classification tasks given its straightforward implementation and good performance in many applications. However, its efficiency represents an obstacle in real-case scenarios because the classification requires computing a distance to every single prototype of the training set. Prototype Selection (PS) is a typical approach to alleviate this problem, which focuses on reducing the size of the training set by selecting the most interesting prototypes. In this context, rank methods have been postulated as a good solution: following some heuristics, these methods perform an ordering of the prototypes according to their relevance in the classification task, which is then used to select the most relevant ones. This work presents a significant improvement of existing rank methods by proposing two extensions: i) a greater robustness against noise at label level by considering the parameter 'k' of the classification in the selection process; and ii) a new parameter-free rule to select the prototypes once they have been ordered. The experiments performed in different scenarios and datasets demonstrate the goodness of these extensions. Also, it is empirically proved that the new full approach is competitive with respect to existing PS algorithms.

Keywords: k-Nearest Neighbour, Prototype Selection, Rank methods, Condensing techniques

Email addresses: juanramonrico@ua.es (Juan Ramón Rico-Juan), jose.jvmas@gmail.com (Jose J. Valero-Mas), jcalvo@dlsi.ua.es (Jorge Calvo-Zaragoza)

1. Introduction

The k-Nearest Neighbor (kNN) rule is one of the well-known algorithms in the supervised classification field [1]. Its wide popularity comes from both its conceptual simplicity as well as its good results categorizing a prototype[1] with respect to its k nearest neighbour prototypes of the training set [2]. In spite of its longevity, it is still subject of ongoing research [3, 4, 5]. However, since no classification model is generated out of the training data, this algorithm generally exhibits a low efficiency in both memory consumption and computational cost.

These shortcomings have been widely analyzed in the literature, where three different families of solutions have been proposed:

(i) Fast Similarity Search (FSS) methods, which base its performance on the creation of search models for fast prototype retrieval in the training set [6] as, for example, the Approximating and Eliminating Search Algorithm (AESA) family of algorithms [7].

(ii) Approximate Similarity Search (ASS) algorithms which work on the premise of searching sufficiently similar prototypes to a given query in the training set at the cost of slightly decreasing the classification accuracy [8], as for instance the methods in [9, 10].

(iii) Data Reduction (DR) techniques, which consist of pre-processing techniques that aim at reducing the size of the training set without affecting the quality of the classification [11].

In this work we shall focus on the latter family of methods, i.e., the ones which aim at reducing the size of the training set by means of pre-processing it.

DR can be broadly divided into two different approaches: Prototype Generation (PG) [12] and Prototype Selection (PS) [13]. The former builds a new

[1] Following previous works' terminology in this field, we will use *prototype* as a synonym of *sample* or *instance*; *i.e.*, an input point from the classification domain.

training set with artificial prototypes that represent more efficiently the same information, while the latter simply selects the most interesting prototypes of the initial training set. PS strategies are more general as regards data representation because it is not necessary to know how the feature space is codified [14] but only the distance values among the prototypes in the set. We therefore focus on this set of strategies.

Over the last decades, there have been a number of proposals for performing PS, which will be reviewed in detail in the next section. Recently, rank-based approaches has been proposed, which are based on ordering the prototypes of the training set according to their relevance in the success of the classification task. That is, prototypes are ranked following some criteria, after which they are selected according to the established order [15].

Among the current rank methods we identify two main drawbacks. The first is that, so far, the process does not take into account the possible noise at the label level. It is true that the kNN classification is robust to this type of phenomenon because of the parameter 'k', which tends to soften the impact of this noise by taking into account more neighbours when classifying. However, PS methods are performed before the classification process, and so this robustness to noise might be mitigated if the PS algorithm does not take into account the value of the parameter 'k' that will be eventually considered. Thus, we extend in this work the current rank methods so that they also consider the 'k' during the selection of the prototypes. On the other hand, these methods require an extra parameter to be fixed, which regulates how many prototypes are finally selected. Given this, we also extend these rank methods to avoid the need for tuning this parameter so that the selection criterion depends exclusively on the data itself. As will be seen in the experiments, these extensions provide higher robustness to noise, as well as optimal results in the trade-off between accuracy and efficiency, thus establishing the new procedures as successful alternative to the PS methods proposed to date.

The rest of the article is structured as follows: Section 2 introduces previous attempts to PS, including those concerning rank methods. Section 3

describes our new strategy to extend previous rank methods. Section 4 presents the different data collections, evaluation metrics, and alternative PS strategies to benchmark with. Experimental evidence of the goodness of the proposed approach is given in Section 5 through a series of experiments and analyses. Finally, Section 6 outlines the main conclusions as well as promising lines for future work.

2. Background

Given that the work is framed in the context of PS, this section provides some background in this regard.

PS techniques aim at reducing the size of a given training set while maintaining (or increasing) as much as possible the accuracy of the classifier. To achieve this goal, these techniques select those prototypes of the training set that are most promising, discarding the rest of them. Formally, let $\mathcal{T}$ denote an initial training set. PS seek for a reduced set $\mathcal{S} \subset \mathcal{T}$.

Typically, the accuracy obtained with $\mathcal{S}$ is lower than that obtained with $\mathcal{T}$. This is why PS methods are evaluated based on two opposing criteria: accuracy of the eventual classification and number of selected prototypes. When the relevance of these two criteria is equal, it leads to formulate PS as a multi-objective optimization problem [16].

The different selection criteria lead to different techniques. Traditionally, PS algorithms have been broadly divided into three groups: condensing, editing, and hybrid approaches.

Condensing techniques focus on selecting only relevant prototypes for the classification rate, usually leading to remarkable size reduction compared to the initial training set. The *Condensing Nearest Neighbour* [17] was the first representative of this type. It focuses on keeping those prototypes that are close to boundaries, and discarding the rest. The reduction starts with an empty set $\mathcal{S}$, and then every prototype of $\mathcal{T}$ is queried. If the prototype at issue is misclassified using $\mathcal{S}$ with 1-NN, then the prototype is included in $\mathcal{S}$. Other-

wise, the prototype is discarded. At the end, $\mathcal{S}$ is returned as a representative reduced version of $\mathcal{T}$. Extensions to this technique include: *Reduced Nearest Neighbour* [18], which performs the condensing algorithm and then revisits each maintained prototype to assure whether it is actually necessary for the classification; *Selective Nearest Neighbour* [19], which ensures that the nearest neighbour of each prototype of $\mathcal{T}$ belongs to $\mathcal{S}$; and *Fast Condensing Nearest Neighbour* [20], which provides a fast, order-independent variant of the classical algorithm.

Editing methods try to minimize the overlapping degree among the different classes of the task, which is generally caused by *outlier* prototypes (*i.e.*, atypical prototypes of the classes). The *Editing Nearest Neighbour* [21] was the first proposal to reduce the training set by removing outliers and noisy instances. It starts with a $\mathcal{S}$ equal to $\mathcal{T}$. Then, the process applies the 1-NN rule to each single prototype in $\mathcal{S}$. If the element is misclassified, it is removed from $\mathcal{S}$. Common extensions to this technique are the *Repeated-Editing Nearest Neighbour* [22], which repeatedly applies editing until homogeneity is reached, and the *Multi-Editing Nearest Neighbour* [23], which repeatedly performs editing over distributed blocks of the training set. The reader is referred to [24] where a recent study and benchmark on this particular family of PS methods can be found.

Both condensing and editing approaches present some complementary disadvantages. Although condensing strategies manage to reduce the training set drastically in a normal situation by emphasizing decision boundaries between classes, it is extremely vulnerable to noise. On the other hand, although editing allows removing noisy data, it is prone to discarding relevant prototypes. As an alternative to these options, there have been a number of *hybrid* approaches that tried to pursue combined objectives. For instance, *Multi-Editing Condensing Nearest Neighbor* [25] first applies the Multi-Editing technique to reduce the amount of noise and then uses Condensing, mostly keeping those relevant prototypes. The *Decremental Reduction Optimization Procedure* [26] orders the prototypes according to the distance to their nearest neighbours and

then, starting from the furthest ones, prototypes are removed as long as they do not affect the accuracy; the *Iterative Case Filtering* [27] bases its performance on the coverage and reachability premises to select a prototype subset that is able to maximize the accuracy; it has been recently extended to deal with multi-label classification [28]. In addition, Evolutionary Algorithms have also been adapted to perform PS [29, 30]. For instance, the *Cross-generational elitist selection, Heterogeneous recombination and Cataclysmic mutation* search [31], whose name indicates the behaviour of its genetic operators, is considered one of the most successful applications of EA for this task [13].

Recently, a new family of hybrid algorithms for PS has been proposed, which is referred to as *rank methods* [15, 32]. Rank methods are devoted to sorting the prototypes of the training set according to its expected relevance for the 1-NN classifier. Then, prototypes are selected following this relevance order, being the size of the selected training set governed by a parameter that must be manually tweaked by the user. It has been demonstrated that these methods are competitive against state-of-the-art PS algorithms [33], while being very easy to understand and implement.

The contribution of this work is to extend current rank methods to improve their performance in two specific aspects: on the one hand, as these rank methods were developed assuming 1-NN classification, they show some limitations when the dataset is noisy, and thus we propose some extensions to the sorting strategies so that the prototypes are actually ordered according to the k parameter that will be later used for the eventual classification; on the other hand, in order to avoid the manual tuning of the selection parameter, we introduce a strategy which automatically selects the optimal number of prototypes according to their relevance in the rank for the specific training set at issue.

3. Extensions to Rank Methods for Prototype Selection

This section describes the proposed extensions to improve the rank methods for PS. For the sake of clarity, we first introduce the basic notions of the afore-

mentioned rank methods on which the modifications will be performed. After that, we present the different proposals to improve their robustness against noisy instances. Finally, we explain the selection rule proposed to avoid the need for the manual tuning.

3.1. Classical Rank Methods

This section introduces the gist of rank methods for PS as well as those strategies already proposed under this paradigm.

The main idea behind rank methods is that prototypes of the training set are not selected but ordered. Following some heuristics, prototypes are given a score that indicates their individual relevance with respect to classification accuracy. Eventually, prototypes are selected starting from the highest score until a certain criterion is accomplished.

A particular approach for rank methods is to follow a voting heuristic, *ie.* each prototype of the training set votes for the other elements which lead to its correct classification. The rank methods considered in this paper focus on the two voting heuristics proposed so far, to our best knowledge: Farthest Neighbour (FN) and Nearest to Enemy (NE) [15]. Both strategies are based on the idea that a prototype can only vote for another element, and the point is to decide which prototype the vote is given to.

For the sake of clarity, some notation is presented first. We will use $d(\cdot, \cdot)$ to denote the distance between prototypes used for the kNN rule. Let $\zeta(p)$ denote the class label of prototype p. Let f_p denote the *friends of* p which are the set of prototypes that share class label with p, *ie.* $f_p = \{p' : \zeta(p') = \zeta(p)\}$. In a similar manner, let e_p be the *enemies of* p, which are the remaining prototypes not included in f_p. As both strategies loop over each prototype of the training set, we will use letter a to denote the prototype issuing the vote. Then, we will use letter b to denote the nearest enemy of a : $\arg\min_{p \in e_p} d(a, p)$.

3.1.1. Farthest Neighbour Voting

The *one vote to the Farthest Neighbour* (FN) strategy searches for a prototype c, which is the farthest friend of a but still closer than its nearest enemy b. That is, a will give its vote to prototype c such that

$$c = \arg\max_{p} d(a,p) : p \in f_a \wedge d(a,p) < d(a,b). \tag{1}$$

3.1.2. Nearest to Enemy Voting

The *one vote to the Nearest to Enemy* (NE) strategy makes a vote for the *friend* that is the closest to its nearest enemy b. This friend must also be within the area centred at a and radius $d(a,b)$. Formally, a will give its vote to prototype c such that

$$c = \arg\min_{p} d(p,b) : p \in f_a \wedge d(a,p) < d(a,b). \tag{2}$$

3.2. Dealing with Noisy Data

As commented, the introduced rank methods are meant to perform the selection considering just a single neighbour. However, note that the classification process with the kNN rule can be then applied on the reduced set with any k value. This first contribution of the paper aims at extending the aforementioned method so that both the prototype selection and classification stages are aligned in the use of the same k value. For that, each prototype should now be capable of emitting $k \geq 1$ votes. In principle, such premise should report a superior robustness against noise in the data.

3.2.1. Extension to Farthest Neighbour (FN)

As commented, the FN rule implements the strategy of prototype a voting for the farthest element of the same class c which is still closer than its nearest enemy b. The proposed extension states that, instead of casting one vote to a single element c (as in Eq. 1), a number k of votes shall be emitted to a set of prototypes $\mathbf{c} = \{c_1, c_2, \ldots, c_k\}$ such that

$$\mathbf{c} = \arg\max_{\mathbf{P}} \left\{ \sum_{i=1}^{k} d(a,p_i) : p_i \in f_a \wedge d(a,p_i) < d(a,b) \right\}. \tag{3}$$

The idea is to vote for the prototypes that contribute to classify prototype a correctly using the kNN rule, as well as to reduce the density of prototypes over a definite area. Also note that, by considering $\mathbf{c}$ a set of prototypes, we are forcing that the selected elements are different among them and thus each of the k votes goes to a different element of f_a. This strategy is graphically shown in Fig. 1.

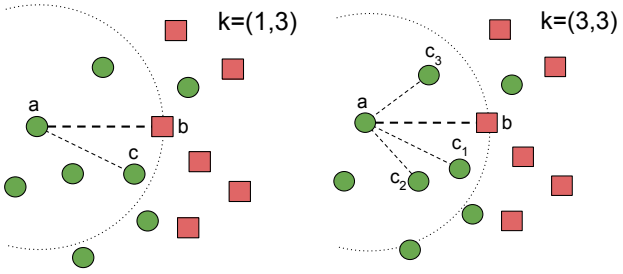

Figure 1: Examples of FN rank algorithm: (left) example of classic FN in which a value of $k = 1$ is used for the PS stage and a value of $k = 3$ is considered for the classification; (right) proposed new approach in with both selection and classification stages are done with $k = 3$. In both graphics, a represents selected prototype, b is the nearest enemy to a, and $c_1, \ldots, c_3$ are prototypes voted by a. Label $k = (x, y)$ means value x is used for selection and y for classification in kNN technique.

As it may be noted from Fig. 1, the main difference between both implementations of the FN method is that prototype a now seeks for k elements to vote for rather than restricting always to a value of $k = 1$ elements.

In addition to the previous explanation, Algorithm 1 provides a formal description of the proposed extension. The idea exposed in this code is the following one:

(i) all instances of the training set start with one single vote

(ii) for a given instance $a \in T$ we obtain its i-th nearest enemy b with method *ithNearestEnemy* (i stands for either first or second nearest enemy, which is a parameter used for avoiding possible outliers)

(iii) after that, we obtain the c_1, c_2... farthest nearest neighbours of a (done with method *kFarthestNearestNeighbour*) using element b as a reference;

(iv) each farthest nearest neighbour is given a vote;

(v) finally, a class-wise normalization of the votes is performed so as to resemble a probability mass, i.e., for each prototype p, we divide the received number of votes by the sum of the votes in the set f_p.

Let ρ_p denote the probability mass associated to a prototype p. It can be mathematically computed as:

$$\rho_p = \frac{\nu\left(p\right)}{\sum_{\forall q \in f_p} \nu\left(q\right)} \tag{4}$$

where $\nu\left(\cdot\right)$ stands for the function that retrieves the number of votes received by the prototype passed as argument.

Note that steps *(iii)* and *(iv)* are the ones which actually differ from the original Farthest Neighbour algorithm: instead of only considering a single c farthest neighbour, we now seek for k elements in order to encompass the necessary information from the neighbourhood that the kNN might need during the classification stage. Let T be the number of prototypes of the training set, the computational cost of the original algorithm is $\mathcal{O}(T^2+T)$ [34], whereas the extension is $\mathcal{O}(T^2 + kT)$. Given that the parameter k of the kNN is typically negligible with respect to T, the asymptotic computational cost of both the original and the extended algorithms belong to $\mathcal{O}(T^2)$.

3.2.2. Extension to Nearest to Enemy (NE)

In a similar way to the FN rank algorithm, the NE algorithm can be also adapted to consider several candidates to improve its robustness against noise.

As a reminder, the NE rule implements a strategy in which prototype a votes for the element of the same class c which is the closest one to its nearest enemy b (check Eq. 2). Thus, the proposed extension allows increasing the number of votes k cast by prototype a to a set of prototypes $\mathbf{c} = \{c_1, c_2, \ldots, c_k\}$ such that

$$\mathbf{c} = \arg\min_{\mathbf{p}} \left\{ \sum_{i=1}^{k} d(p_i, b) : p_i \in f_a \wedge d(a, p_i) < d(a, b) \right\}. \tag{5}$$

The idea is to try to avoid any misclassification produced by b using kNN rule in an area with prototypes of other classes. As in the FN case note that, by

Algorithm 1 k Farthest Nearest Neighbour

1: **function** i-FN(T, k, α) ▷ i^{th} nearest enemy; T training set; k for classify; α probability of density
2: **for** $a \in T$ **do**
3: a.votes $\leftarrow 1$ ▷ Initialization vote
4: **end for**
5: **for** $a \in T$ **do**
6: $b \leftarrow$ ithNearestEnemy(i, a, T)
7: $C \leftarrow$ kFarthestNearestNeighbour(k, b, a, T)
8: **for** $p \in C$ **do**
9: p.votes $\leftarrow p$.votes $+ 1$
10: **end for**
11: **if** $|C| < k$ **then** ▷ Less than k candidates, a vote for himself
12: a.votes $\leftarrow a$.votes $+ 1$
13: **end if**
14: **end for**
15: **for** $a \in T$ **do** ▷ Compute prototype probability
16: $a.prob \leftarrow a$.votes/SumVotes(class$(a), T)$
17: **end for**
return density_selection(T, α)
18: **end function**

considering **c** a set of prototypes, we are forcing that the selected elements are different among them and thus each of the k votes goes to a different element of f_a. This strategy is graphically shown in Fig. 2.

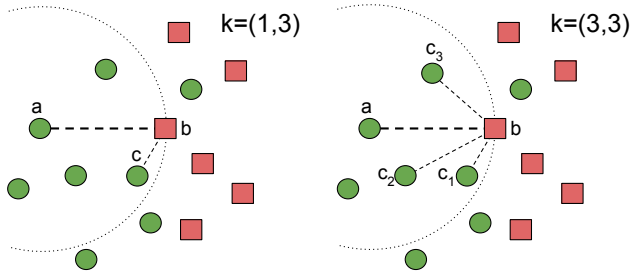

Figure 2: Examples of NE rank algorithm: (left) example of classic NE in which a value of $k = 1$ is used for the PS stage and a value of $k = 3$ is considered for the classification; (right) proposed new approach in with both selection and classification stages are done with $k = 3$. In both graphics, a represents selected prototype, b is the nearest enemy to a, and $c_1, \ldots, c_3$ are prototypes voted by a. Label $k = (x, y)$ means value x is used for selection and y for classification in kNN technique.

As Fig. 2 shows, the only difference between both NE methods is that prototype a casts k votes instead of restricting to a single vote for one prototype.

In addition to the previous explanation, Algorithm 2 provides a formal description of the proposed extension. The idea in this case is the following one: *(i)* all instances of the training set start with one single vote; *(ii)* for a given instance $a \in T$ we obtain its i-th nearest enemy b with method *ithNearestEnemy* (i stands for either first or second nearest enemy, which is a parameter used for avoiding possible outliers); *(iii)* after that, we obtain the nearest elements $c_1, c_2, \ldots$ to enemy b we with the same class as a (done with method *kNearestNeighbourToEnemy*); *(iv)* each element c_1, c_2... is then given a vote; *(v)* finally, a class-wise normalization of the votes is performed so as to resemble a probability mass, as done for the FN strategy (see Eq. 4).

As in the previous extension, note that steps *(iii)* and *(iv)* are the ones which actually differ from the original Nearest To Enemy algorithm: instead of only considering a single c element, we now seek for k different instances to avoid

Algorithm 2 k Nearest Enemy

function i-NE(T, k, α) ▷ i^{th} nearest enemy; T training set; k for classify; α probability of density
  for $a \in T$ **do**
    a.votes $\leftarrow 1$ ▷ Initialization vote
  end for
  for $a \in T$ **do**
    $b \leftarrow$ ithNearestEnemy(i, a, T)
    $C \leftarrow$ kNearestNeighbourToEnemy(k, b, a, T)
    for $p \in C$ **do**
      p.votes $\leftarrow p$.votes $+ 1$
    end for
    if $|C| < k$ **then** ▷ Less than k candidates, a vote for himself
      a.votes $\leftarrow a$.votes $+ 1$
    end if
  end for
  for $a \in T$ **do** ▷ Compute prototype probability
    a.prob $\leftarrow a$.votes/SumVotes(class$(a), T)$
  end for
  return density_selection(T, α)
end function

noise issues. The computational cost of this extension also remains in $\mathcal{O}(T^2)$.

3.3. Automatic Prototype Selection Rule

As previously commented, after the voting stage of the methods, the received votes are normalised to produce a *relevance ratio*, i.e., the sum over these values for all the prototypes of a given class equals the unit. Then, the training set is sorted according to those relevance values and, from top to bottom, the candidates are selected from the rank until their accumulated score exceeds an external parameter $\alpha \in (0, 1]$ that allows the performance of the rank method to be tuned by the user. Low values of this parameter will lead to a higher reduction of the size of the training set, while high values will remove only the most irrelevant prototypes. Although tuning parameters may be considered an inconvenient, in this case this is specially interesting because the parameter allows the user to enhance a particular objective (either reduction or accuracy) depending on the requirements of the system.

Following this idea, in this paper we provide two extensions to this selection process which should improve the performance of both the reduction and the classification rate.

The first of them focuses on performing a class-wise selection of prototypes: instead of performing a global selection of the instances, the idea is to carry out the process considering the different classes involved in the problem; thus, it can be guaranteed that all classes are represented in the resulting reduced set. It must be pointed out that the ordering process is still the same as in the classic rank methods (from top to bottom considering the *relevance ratio* or *probability mass*) but it is now done individually for each of the classes involved in the task instead of considering them altogether. Such basic modification is expected to report some improvement in the classification rate since the selection is now done taking into consideration the different classes of the problem and not in such a *blind* approach as previously. This idea can be easily implemented following Algorithm 3. As aforementioned, the *probability* term refers to the result of the normalization process described in Eq. 4.

Algorithm 3 Prototype selection with density of probability per class.

```
1: function density_selection(T, α)    ▷ T voted training set; α probability of density
2:     S ← ∅
3:     for x ∈ classes(T) do
4:         T' ← sort_probability_class(T, x)
5:         prob ← 0
6:         for p ∈ T' do                                   ▷ Prototype class x
7:             if prob < α then                            ▷ This prototype is needed
8:                 prob ← prob + p.prob
9:                 S ← S ∪ {p}
10:            end if
11:        end for
12:    end for
       return S
13: end function
```

The second extension proposed deals with the tuning parameter used for selecting the density of probability per class to maintain. While it has been previously discussed that such parameter does not constitute a disadvantage by itself, we propose a new rule to automatically select the most suitable prototypes according to their representativeness and effectiveness for the classification. This new method is described in Algorithm 4: the approach orders the prototypes according to its relevance for their own class, computed from the received votes, and initializes the reduced training set S as an empty set; prototypes are then selected by simply checking whether they are misclassified using S as a training set, being then included in it if there is a misclassification. This is similar to the *Condensing Nearest Neighbour* rule (see Sect. 2); however, instead of consulting the prototypes in a random order, in our proposal such ordering is done taking into account both the class and the classification relevance of the prototype: let $|C|$ be the number of different classes in the dataset, we first check the $|C|$ prototypes (one per class) with the highest relevance according to the votes received following any of the previous (FN or NE) rules; the next $|C|$ prototypes constitute the prototypes which obtain the second highest relevance for their class; and so on. The process is repeated until no more prototypes can be consulted. As previously commented, the term *probability* in this case refers to the result of the normalization process described in Eq. 4.

It must be mentioned that, to discard possible outliers, we introduce a heuristic filter which avoids selecting instances with just one vote (presumably, its own one), as they typically represent outliers and/or noisy elements.

Note that the advantage of this new selection rule is twofold: on the one hand, it relieves the user from having to select an appropriate parameter for this operation; on the other hand, the rule follows a self-selection operation, which is dynamically adjusted according to the data itself and thus providing a more generalisable behaviour. The computational cost of this rule mostly comes from both the prototype ordering, which is $\mathcal{O}(T\log(T))$, and the selection stage, which can be expressed as $\mathcal{O}(\frac{T^2}{2})$; thus, the final cost of the rule is $\mathcal{O}(T\log(T)+T^2) \in \mathcal{O}(T^2)$.

Algorithm 4 Automatic prototype selection (APS)

```
1: function APS(T, k, [min_votes = 2])
2:     T' ← Sort_ith_probability_per_class(T)
3:     S ← ∅
4:     for a ∈ T' do
5:         if a.votes ≥ min_votes then          ▷ Minimum no. of votes required
6:             if kNN(k, a, S) ≠ ζ(a) then                    ▷ a is needed
7:                 S ← S ∪ {a}
8:             end if
9:         end if
10:    end for
11: return S
12: end function
```

Finally, once the PS process has been applied to the initial training set T and a reduced set S has been retrieved, the kNN rule is used to classify new instances using S. In formal terms, given an unlabelled query prototype z and the set $\mathbf{p}_z^{(k)}$ representing the k-nearest prototypes of query z within the reduced set S according to some dissimilarity function, the kNN rules assigns z the label $\hat{l}$ such that

$$\hat{l} = \arg\max_l |\{p : p \in \mathbf{p}_z^{(k)} \wedge \zeta(p) = l\}|$$

that is, $\hat{l}$ is the most frequent label among $p_z^{(k)}$.

4. Experimental Setup

In this section we present the configuration of the experiments carried out to evaluate the proposed improvements, such as the considered datasets, the set of PS algorithms to comparatively assess the performance of the proposed algorithms, and the evaluation protocol.

4.1. Datasets

Our experiments are conducted with seven corpora: the *SPECIAL DATABASE 3* (NIST3) of the National Institute of Standards and Technology, from which a subset of the upper case characters was randomly selected; the Mixed National Institute of Standards and Technology dataset (MNIST) [35] of handwritten digits; the *United States Postal Office* (USPS) handwritten digit dataset [36]; the MPEG-7 shape silhouette dataset [37], and the Handwritten Online Musical Symbol (HOMUS) dataset [38]; and two additional corpora of the UCI [39] (Penbased [40] and Letter [41]).

For the first four cases, contour descriptions with Freeman Chain Codes (FCC) [42] are extracted from the symbol shapes, and the string Edit Distance (ED) [43] is used as dissimilarity measure. In the fifth case, and due to its good results in the baseline experimentation offered with these data, Dynamic Time Warping (DTW) [44] is used. Since datasets from the UCI may contain missing values in the samples, the Heterogeneous Value Difference Metric (HVDM) [45] is used for the last two datasets. Table 1 shows a summary of the main features of these datasets.

Table 1: Description of the datasets used in the experimentation.

Name	Instances	Classes	Features	Distances
NIST3	6500	26	FCC contour	ED
MNIST	70000	10	FCC contour	ED
USPS	9298	10	FCC contour	ED
MPEG-7	1400	70	FCC contour	ED
HOMUS	15200	32	sequence of 2D points	DTW
Penbased	10992	10	numerical features	HVDM
Letter	20000	26	numerical features	HVDM

For each corpus, a 10-fold cross-validation process is applied in order to

provide more robust results. That is, at each fold, 90 % of the whole corpus is used for training and 10 % for performance evaluation.

Also in the experimentation we will add synthetic noise to evaluate the robustness of the PS methods considered in this type of scenario. The noise will be induced by the exchange of labels between pairs of randomly selected prototypes of different classes. The noise rates (percentage of prototypes that change labels) considered were 0 %, 20 % and 40 %, as they are common values in this type of benchmarking [46].

4.2. Prototype Selection Strategies

The main goal of the current work is to provide a comprehensive comparative experiment to evaluate the performance and competitiveness of the new rank methods as PS strategies. To cover the different families of approaches presented in Section 2, we shall consider the comparison amongst the following particular strategies:

- No selection: all the prototypes of the initial training set (kNN).
- Classical: Condensing Nearest Neighbour (CNN), Editing Nearest Neighbour (ENN), Fast Condensing Nearest Neighbour (FCNN) and Editing Condensing Nearest Neighbour (E-CNN).
- Hybrid: Iterative Case Filtering (ICF), Decremental Reduction Optimization Procedure (DROP3) and the Cross-generational elitist selection, Heterogeneous recombination and Cataclysmic mutation (CHC) evolutionary scheme.
- Classical rank methods: 1-FN k=(1,x), 2-FN k=(1,x), 1-NE k=(1,x), 2-NE k=(1,x) and IRB; value x represents the k in classification task. Each of them considers values of α within the range (0, 1) with a granularity of 0.1. The extreme values have been discarded since $\alpha = 0$ would mean an empty set and $\alpha = 1$ is equivalent to ALL.

- New rank methods: 1-FN k=(x,x), 2-FN k=(x,x), 1-NE k=(x,x), 2-NE k=(x,x), 1-FN-APS-new k=(x,x), NE-APS-new k=(x,x); values x represents the k in selection and classification tasks. To avoid outliers, for each of these methods we impose a minimum number of *votes* per prototype (ex.: 2, 4, 6), thus filtering out prototypes with fewer votes. As this value grows, the selection algorithm has greater confidence in the selected prototypes and the size of the final set is reduced.

It can be seen that the set of PS algorithms is quite complete, as it includes classical methods, modern strategies, as well as the classic and newly proposed rank methods.

4.3. Evaluation Protocol

We describe in this section the evaluation protocol followed to evaluate our proposals. The experiments consist of using the PS strategies to reduce an initial training set, and then checking the performance of the kNN rule from this set over the validation data. In this regard, values of $k \in \{1, 3, 5, 7\}$ will be considered during the classification.

In order to analyse the impact of a given PS strategy, we must take into account two metrics of interest: accuracy of the final classification and the size of the reduced set (with respect to the original one). The last metric provides a theoretical efficiency measure of each strategy. Note that the performance of the classification and the size of the data set can be considered conflicting objectives, as improving one of them generally implies a deterioration of the other — in the context of PS for kNN classification. In this sense, additional information can be obtained by evaluating this proposal from the perspective of a *Multi-Objective Optimization Problem* (MOP) in which both the size of the selected set and the accuracy of the classification are intended to be optimized at the same time. This assessment is usually carried out through the concept of non-dominance. This means that one solution (in this case, a tuple with the size of the training set and the precision of the classification) dominates another, if and only if, it

is better or equal in each target function and, at least, strictly better in one of them. The Pareto frontier is composed by all the non-dominated elements and represents the different optimal solutions to the MOP. Each of these solutions, known as the Pareto-Optimal configuration, are considered the best solutions to the problem without any particular priority among them.

In some cases, instead of measuring the problem as an MOP, it may be interesting to encompass the performance in a single metric. Thus, based on the fact that for a given dataset we can compute an efficiency measure (size of the dataset) and a measure of effectiveness (accuracy), we consider the *profit* metric of the reduced dataset as defined in [47], which relates classification accuracy and size of the dataset as:

$$\text{profit} = \frac{\text{accuracy}}{\text{size of the dataset}} \ . \tag{6}$$

5. Results

In order to comprehensively evaluate our proposals, the experimental results are presented in two different ways. First of all, we compare the classical rank methods with those including the proposal to improve the process in noise environments. That is, we will compare the classical rank-based PS algorithms that always assumed $k = 1$, with the new voting approach that considers the same k for both the selection and the classification processes. Then, we also carry out an exhaustive comparison of the PS methods mentioned in Section 4.2, among which the rank methods with the proposed improvements can be found, as well as the classical ones.

5.1. Evaluating the Effect of $k = 1$ and $k > 1$ in the Selection Process

This experiment aims at showing the differences between classic 1-FN, 2-FN, 1-NE, and 2-NE strategies with $k = 1$ against $k > 1$ for selection and classification, more precisely, with the values $k = \{3, 5, 7\}$. Since we are not yet considering the new strategy to perform the automatic selection of the number

of prototypes to maintain, we still need to set the selection parameter of the classical methods manually. Specifically, we consider values of $\alpha = \{0.1, 0.2, 0.3\}$.

The results of this experiment are reported in Table 2, including the aforementioned levels of induced label noise. It can be observed that, with hardly noisy datasets, the profit values of the selection methods with $k > 1$ are slightly worse than those with $k = 1$. However, as the noise figure increases, this evidence changes radically. With 20 % and 40 % of noise value, selection and classification algorithms with $k > 1$ obtain better profit rates in most of the results. This shows, as expected, that the new voting strategies, unlike what happened with the classic ones, are adaptable to noise environments as long as a good selection of the parameter k is made. Note that this parameter is what makes the kNN classifier robust in noisy situations, so it is important that PS methods can deal with different values of the parameter as well.

5.2. Comparative Results

Given the large number of experimental results obtained (up to 33, 600 taking into account all combinations of algorithms, datasets, and folds), it is difficult to report all the results in a compact way. Thus, Fig. 3 graphically shows these results with respect to the size of the training set and the accuracy achieved by the kNN classification after each PS strategy, averaged over all folds and datasets. Three different scenarios are considered as regards the level of induced noise in the labels of the original training set. For the sake of visualization, different colours are used to group the families of PS algorithms.

An initial remark about this figure is that the strategy that provides the best accuracy is that in which there is no selection, that is, the original kNN (size 100 %, and accuracy around 90 %). This happens regardless of the induced noise level, although as the noise level rises, the performance of the different strategies with low values of k are penalized. It can also be observed that PS algorithms generally achieve a remarkable reduction rate, since most results are concentrated between 0 and 25 % of the original sizes. The interesting results, therefore, focus on the points whose reduction is between 0 and 25 %

Table 2: Comparison of accuracies, sizes and profits (accuracy/size) of results between FN and NE prototype selection algorithms with $k = (1, x)$ and $k = (x, x)$. (+) means $k = (x, x)$ has better profit than $k = (1, x)$ and (-) the opposite.

		k=(1,x)						k=(x,x)						Profit		
		k=3		k=5		k=7		k=3		k=5		k=7		k=3	k=5	k=7
Noise	Name	Acc	Size	Acc	Size	Acc	Size	Acc	Size	Acc	Size	Acc	Size			
0 %	1-FN$_{0.1}$	78.1	3.9	76.4	3.9	74.8	3.9	78.2	4.1	75.5	4.4	72.2	4.6	-	-	-
	1-FN$_{0.2}$	83.5	8.3	81.4	8.3	79.6	8.3	82.5	8.8	81.6	9.4	79.8	9.7	-	-	-
	1-FN$_{0.3}$	85.8	13.9	84.3	13.9	83.3	13.9	85.6	14.5	83.9	15.1	82.8	15.6	-	-	-
	1-NE$_{0.1}$	69.2	2.1	64.3	2.1	60.5	2.1	70.0	2.2	68.2	2.5	66.5	2.6	-	-	-
	1-NE$_{0.2}$	78.3	4.3	75.8	4.3	73.3	4.3	76.9	4.7	76.0	5.3	75.1	5.6	-	-	-
	1-NE$_{0.3}$	83.6	7.6	81.8	7.6	80.3	7.6	81.6	7.9	80.1	8.6	79.0	9.3	-	-	-
	2-FN$_{0.1}$	77.8	3.8	75.2	3.8	73.6	3.8	78.4	4.1	75.6	4.4	72.5	4.6	-	-	-
	2-FN$_{0.2}$	82.3	8.1	80.3	8.1	78.7	8.1	82.3	8.8	81.5	9.4	80.3	9.8	-	-	-
	2-FN$_{0.3}$	85.7	13.5	84.3	13.5	83.3	13.5	85.6	14.3	84.0	15.1	83.0	15.7	-	-	-
	2-NE$_{0.1}$	70.9	2.1	66.7	2.1	63.9	2.1	70.7	2.2	69.4	2.5	66.7	2.6	-	-	-
	2-NE$_{0.2}$	79.0	4.4	77.3	4.4	75.6	4.4	76.9	4.7	76.3	5.3	75.2	5.6	-	-	-
	2-NE$_{0.3}$	83.4	7.6	82.0	7.6	80.9	7.6	82.0	7.9	80.4	8.6	79.4	9.2	-	-	-
20 %	1-FN$_{0.1}$	80.7	4.8	78.9	4.8	77.4	4.8	79.6	4.4	76.3	4.3	73.2	4.3	+	+	+
	1-FN$_{0.2}$	84.9	10.5	83.6	10.5	82.5	10.5	83.7	9.7	81.5	9.5	79.5	9.3	+	+	+
	1-FN$_{0.3}$	87.1	17.3	85.9	17.3	85.0	17.3	86.0	15.9	84.0	15.4	82.6	15.1	+	+	+
	1-NE$_{0.1}$	79.8	3.9	77.8	3.9	76.2	3.9	78.4	3.9	75.2	4.0	72.7	4.0	-	-	-
	1-NE$_{0.2}$	84.4	8.8	83.0	8.8	81.7	8.8	83.3	8.7	81.0	8.8	79.1	8.8	+	-	-
	1-NE$_{0.3}$	86.9	15.2	85.8	15.2	84.6	15.2	85.6	14.4	83.7	14.4	82.2	14.4	+	+	+
	2-FN$_{0.1}$	79.3	4.3	77.4	4.3	75.5	4.3	79.4	4.0	76.9	4.1	73.8	4.1	+	+	+
	2-FN$_{0.2}$	84.4	9.5	82.9	9.5	81.7	9.5	83.7	8.9	81.8	8.8	79.9	8.8	+	+	+
	2-FN$_{0.3}$	86.9	16.0	85.7	16.0	84.6	16.0	86.2	14.5	84.2	14.4	82.6	14.3	+	+	+
	2-NE$_{0.1}$	78.9	3.4	77.0	3.4	75.3	3.4	78.1	3.4	75.2	3.5	73.0	3.7	-	-	-
	2-NE$_{0.2}$	84.1	7.6	82.1	7.6	81.0	7.6	82.7	7.3	81.1	7.7	79.2	7.9	+	-	-
	2-NE$_{0.3}$	86.9	13.2	85.6	13.2	84.4	13.2	85.3	12.2	83.6	12.6	82.0	13.0	+	+	-
40 %	1-FN$_{0.1}$	80.9	5.5	79.1	5.5	77.4	5.5	79.4	4.8	75.5	4.7	72.5	4.5	+	+	+
	1-FN$_{0.2}$	85.2	12.1	84.2	12.1	83.1	12.1	83.5	10.6	81.2	10.1	79.2	9.9	+	+	+
	1-FN$_{0.3}$	84.7	20.0	85.4	20.0	84.9	20.0	85.6	17.4	83.9	16.6	82.3	16.2	+	+	+
	1-NE$_{0.1}$	80.4	5.0	78.6	5.0	76.8	5.0	78.9	4.6	75.2	4.6	72.2	4.5	+	+	+
	1-NE$_{0.2}$	85.0	11.4	83.8	11.4	82.7	11.4	83.4	10.2	81.0	9.9	79.1	9.8	+	+	+
	1-NE$_{0.3}$	85.0	19.0	85.6	19.0	85.0	19.0	85.5	16.9	83.8	16.4	82.2	16.1	+	+	+
	2-FN$_{0.1}$	80.7	4.9	78.9	4.9	76.9	4.9	79.2	4.2	76.4	4.1	73.0	4.0	+	+	+
	2-FN$_{0.2}$	84.6	10.9	83.6	10.9	82.7	10.9	83.7	9.3	81.5	9.0	79.4	8.8	+	+	+
	2-FN$_{0.3}$	86.0	17.8	85.9	17.8	85.1	17.8	86.0	15.4	83.7	14.7	82.2	14.4	+	+	+
	2-NE$_{0.1}$	80.5	4.3	78.3	4.3	76.5	4.3	78.7	3.9	75.5	3.9	72.8	3.9	+	+	+
	2-NE$_{0.2}$	84.7	9.7	83.3	9.7	82.1	9.7	83.3	8.6	81.1	8.6	79.2	8.5	+	+	+
	2-NE$_{0.3}$	86.3	16.5	85.7	16.5	84.8	16.5	85.6	14.3	83.6	14.1	82.0	14.0	+	+	+

and attain an accuracy higher than 80 %. It can be seen that in this area there are objectively better results than others: for example those whose reduction is equal but have better precision than others or, vice versa, those whose reduction is greater with the same accuracy. However, to determine the actual optimal results, we will resort to the concept of non-dominance introduced above.

The set of results that belong to the non-dominated frontier are reported in Table 3 where the elements in the tables correspond to the average of the individual results for all datasets and folds considered so as to report general trends. In addition, Table 4 depicts the configurations that report, individually for each dataset, the smallest set size with the minimum classification loss for the case in which no noise is induced in the data. We have only included the information without induced noise as all the results can be found in Table 2 and the presented Table 4 simply constitutes a summary for a particular case.

By checking these tables, one may optimize the PS process for any of the data collections considered. For instance, focusing on the HOMUS set which is meant for Optical Music Recognition, if the application is required to be as precise as possible, one should resort to the FCNN solution as it provides the least accuracy loss (only 2 %) with a remarkable set size reduction (which is around 22.5 %). Oppositely, in the event that memory is the main issue to tackle, one of the proposed 1-NE algorithms with the APS self-stop criterion provides a proper set size reduction (around 12.6 % of the initial size) with only an 8 % of accuracy loss.

The important conclusion to draw from these detailed results is that the proposed extensions to classical rank methods practically cover the entire non-dominance front. That is, except for the results with the original set (kNN) and the CNN or E-CNN at 0 % and 40 % noise, respectively — which obtain higher accuracy in the zone of lower efficiency — both classical rank methods and the rest of PS algorithms are dominated by the approaches proposed in this paper. Obviously, we find different parametrization of the proposed rank methods along the front, as regards the voting strategy, the minimum number of votes needed to consider a prototype, or the fact of whether including or not a

previous editing process. However, what these results show is that the proposed
465 methodology outperforms, in general, the state of the art in the bi-objective of accuracy and efficiency, thus postulating itself as a promising alternative for PS in kNN classification.

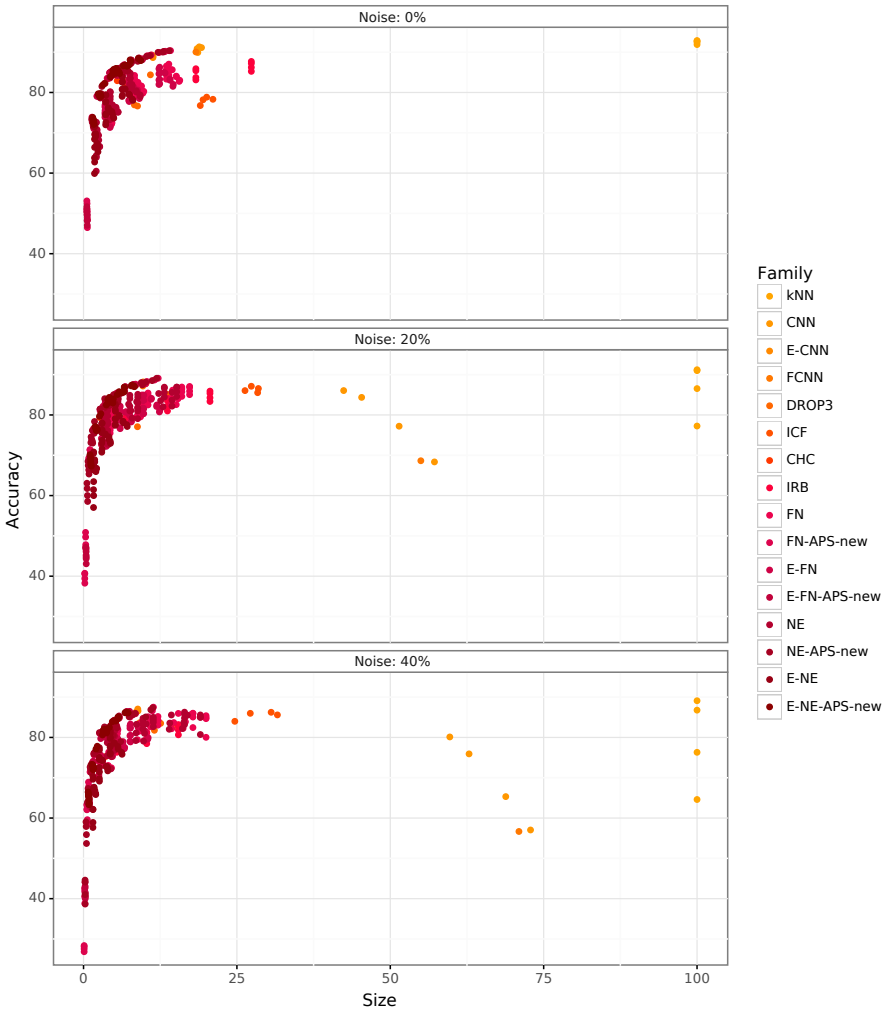

Figure 3: Averages of the experiments (10-CV) of data set size versus accuracy by family of algorithms, according to the level of induced noise.

Table 3: Results of the experiments belonging to the non-dominated frontier sorted by the resulting set size and grouped according to the induced noise.

Noise	Family	Algorithm	Size (%)	Accuracy%)
	FN-APS-new	2-FN-APS-new (votes>6) k=(1, 1)	0.5	51.1
	FN-APS-new	1-FN-APS-new (votes>6) k=(1, 1)	0.5	53.1
	E-FN-APS-new	1-E-FN-APS-new (votes>4) k=(1, 1)	1.3	73.4
	FN-APS-new	1-FN-APS-new (votes>4) k=(1, 1)	1.4	73.9
	FN-APS-new	1-FN-APS-new (votes>6) k=(3, 3)	2.2	79.1
	NE-APS-new	1-NE-APS-new (votes>6) k=(3, 3)	3.0	81.5
0 %	E-FN-APS-new	1-E-FN-APS-new (votes>2) k=(1, 1)	4.3	84.9
	E-NE-APS-new	1-E-NE-APS-new (votes>2) k=(1, 1)	4.9	85.8
	E-NE-APS-new	2-E-NE-APS-new (votes>2) k=(3, 3)	7.6	88.1
	NE-APS-new	1-NE-APS-new (votes>2) k=(5, 5)	12.2	90.0
	NE-APS-new	1-NE-APS-new (votes>2) k=(7, 7)	13.1	90.2
	CNN	CNN k=5	18.9	91.3
	kNN	kNN k=3	100.0	92.9
	FN-APS-new	1-FN-APS-new (votes>6) k=(1, 1)	0.2	40.7
	FN-APS-new	2-FN-APS-new (votes>6) k=(1, 1)	0.3	50.9
	NE-APS-new	1-NE-APS-new (votes>6) k=(1, 1)	0.6	63.1
	NE-APS-new	2-NE-APS-new (votes>6) k=(1, 1)	0.7	68.5
	NE-APS-new	1-NE-APS-new (votes>4) k=(1, 1)	1.2	74.6
	NE-APS-new	2-NE-APS-new (votes>4) k=(1, 1)	1.4	76.2
	NE-APS-new	2-NE-APS-new (votes>6) k=(3, 3)	1.8	78.4
20 %	FN-APS-new	2-FN-APS-new (votes>6) k=(5, 5)	2.7	81.4
	NE-APS-new	2-NE-APS-new (votes>4) k=(3, 3)	3.5	84.0
	E-NE-APS-new	2-E-NE-APS-new (votes>2) k=(3, 3)	6.6	86.9
	E-FN-APS-new	1-E-FN-APS-new (votes>2) k=(3, 3)	6.8	87.0
	E-FN-APS-new	2-E-FN-APS-new (votes>2) k=(5, 5)	8.1	87.6
	NE-APS-new	2-NE-APS-new (votes>2) k=(7, 7)	11.9	89.1
	FN-APS-new	2-FN-APS-new (votes>2) k=(7, 7)	12.2	89.1
	kNN	kNN k=7	100.0	91.2
	FN-APS-new	1-FN-APS-new (votes>6) k=(1, 1)	0.1	28.4
	FN-APS-new	2-FN-APS-new (votes>6) k=(1, 1)	0.2	42.6
	NE-APS-new	2-NE-APS-new (votes>6) k=(1, 1)	0.4	59.0
	FN-APS-new	1-FN-APS-new (votes>4) k=(1, 1)	0.5	63.3
	FN-APS-new	2-FN-APS-new (votes>4) k=(1, 1)	0.8	68.9
	NE-APS-new	2-NE-APS-new (votes>4) k=(1, 1)	1.1	72.6
40 %	NE-APS-new	2-NE-APS-new (votes>4) k=(3, 3)	2.6	81.2
	NE-APS-new	2-NE-APS-new (votes>4) k=(5, 5)	3.4	82.6
	E-FN-APS-new	1-E-FN-APS-new (votes>2) k=(3, 3)	5.8	85.3
	E-FN-APS-new	1-E-FN-APS-new (votes>2) k=(5, 5)	7.0	86.5
	E-CNN	E-CNN k=5	8.8	87.1
	FN-APS-new	2-FN-APS-new (votes>2) k=(7, 7)	11.4	87.5
	kNN	kNN k=7	100.0	89.1

Table 4: Summary of the methods achieving the highest dataset size reduction with the lowest accuracy loss for each of the data collections studied. The data in this table relates to the case in which no noise has been induced to the datasets.

Dataset	Acceptable accuracy sacrifice (%)	Lowest dataset size (%)	Method
NIST	2	20.7	CNN k(7)
	3	9.3	2-FN-APS-new (votes>4) k(7,7)
	4	7.5	1-NE-APS-new (votes>4) k(7,7)
	5	5.8	1-NE-APS-new (votes>6) k(7,7)
	6	5.2	1-NE-APS-new (votes>4) k(3,3)
MNIST	1	11.4	CNN k(7)
	2	4.1	2-E-NE-APS-new (votes>4) k(7,7)
	3	2.8	1-E-NE-APS-new (votes>6) k(5,5)
	4	1.6	CHC k(7)
	5	0.8	CHC k(1)
USPS	2	4.9	1-NE-APS-new (votes>4) k(7,7
	3	2.6	1-NE-APS-new (votes>6) k(5,5
	4	2.6	1-FN-APS-new (votes>6) k(7,7
	5	1.0	CHC k(1)
	6	1.0	CHC k(1)
MPEG7	2	29.5	FCNN
	3	28.9	CNN k(1)
	4	28.9	CNN k(1)
	5	28.9	CNN k(1)
	6	28.9	CNN k(1)
HOMUS	4	22.5	FCNN
	5	16.2	2-FN-APS-new (votes>2) k(5,5)
	6	14.1	2-FN-APS-new (votes>2) k(3,3)
	7	12.6	1-NE-APS-new (votes>2) k(3,3)
	8	12.6	1-NE-APS-new (votes>2) k(3,3)
Penbased	1	3.4	1-E-FN-APS-new (votes>2) k(5,5)
	2	2.3	2-FN-APS-new (votes>2) k(1,5)
	3	1.5	1-E-FN-APS-new (votes>2) k(1,1)
	4	1.2	CHC k(1)
	5	1.2	CHC k(1)
Letter	3	18.3	CNN k(5)
	4	14.2	1-NE-APS-new (votes>2) k(5,5)
	5	12.2	1-NE-APS-new (votes>2) k(3,3)
	6	10.4	2-E-NE-APS-new (votes>2) k(3,3)
	7	9.7	2-E-FN-APS-new (votes>4) k(5,5)

6. Conclusions and Future Work

In this paper we present extensions to some classical rank methods for PS based on voting heuristics. The first extension focuses on improving the tolerance of the reduced set to noisy data by considering the parameter 'k' of the classifier in the voting strategies. Additionally, a self-guided criterion is pro-

posed for the actual selection, which eliminates the need for tuning an external user parameter that the classical methods hold.

We conduct experiments with several datasets and report the performance according to both the size of the reduced set and the accuracy of the eventual classification. Our first experiment confirms that the first extension increases the robustness against label-level noise of the algorithms, especially when such level is high. Furthermore, comparative experiments with state-of-the-art algorithms report that the combination of both extensions leads to a wide range of optimal results, covering most of the efficiency-accuracy space of results over the rest of the prototype selection methods. Thus, this new methodology clearly improves the classic rank methods for PS, and is postulated as an alternative to other state-of-the-art methods.

As future work we plan to extend this proposal to carry out the process more efficiently. The kNN classifier does not have a training stage, which makes it suitable to adapt to new training prototypes dynamically. Therefore, we want to include this advantage in our approach as well. On the other hand, PS is especially useful when the training set is huge, and so the PS process itself must be efficient in order to deal with such a large amount of data. Therefore, we are interested in studying how to improve the efficiency of the voting and selection processes taking into account the proposed extensions. Finally, we are also interested in studying the possibility of letting each prototype cast more than a single vote taking into consideration, for instance, the distance to the selected prototype.

Acknowledgments

This work is supported by the Spanish Ministry HISPAMUS project TIN2017-86576-R, partially funded by the EU.

References

[1] R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification, John Wiley & Sons, 2001.

[2] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE transactions on information theory 13 (1) (1967) 21–27.

[3] N. Verbiest, S. Vluymans, C. Cornelis, N. García-Pedrajas, Y. Saeys, Improving nearest neighbor classification using ensembles of evolutionary generated prototype subsets, Applied Soft Computing 44 (2016) 75–88.

[4] Ömer Faruk Ertuğrul, M. E. Tağluk, A novel version of k nearest neighbor: Dependent nearest neighbor, Applied Soft Computing 55 (2017) 480 – 490.

[5] S. Zhang, X. Li, M. Zong, X. Zhu, R. Wang, Efficient kNN Classification With Different Numbers of Nearest Neighbors, IEEE Transactions on Neural Networks and Learning Systems 29 (5) (2017) 1774–1785.

[6] P. Jain, B. Kulis, I. S. Dhillon, K. Grauman, Online metric learning and fast similarity search, in: Advances in neural information processing systems, 2009, pp. 761–768.

[7] E. V. Ruiz, An algorithm for finding nearest neighbours in (approximately) constant average time, Pattern Recognition Letters 4 (3) (1986) 145–157.

[8] J. Wang, H. T. Shen, J. Song, J. Ji, Hashing for similarity search: A survey, arXiv preprint arXiv:1408.2927 (2014) 1–29.

[9] S. Ougiaroglou, G. Evangelidis, Fast and accurate k-nearest neighbor classification using prototype selection by clustering, in: 2012 16th Panhellenic Conference on Informatics, IEEE, 2012, pp. 168–173.

[10] A.-J. Gallego, J. Calvo-Zaragoza, J. J. Valero-Mas, J. R. Rico-Juan, Clustering-based k-nearest neighbor classification for large-scale data with neural codes representation, Pattern Recognition 74 (2018) 531–543.

[11] S. García, J. Luengo, F. Herrera, Data Preprocessing in Data Mining, Vol. 72 of Intelligent Systems Reference Library, Springer, 2015.

[12] I. Triguero, J. Derrac, S. Garcia, F. Herrera, A taxonomy and experimental study on prototype generation for nearest neighbor classification, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42 (1) (2012) 86–100.

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

[14] J. Calvo-Zaragoza, J. J. Valero-Mas, J. R. Rico-Juan, Prototype generation on structural data using dissimilarity space representation, Neural Computing and Applications 28 (9) (2017) 2415–2424.

[15] J. R. Rico-Juan, J. M. Iñesta, New rank methods for reducing the size of the training set using the nearest neighbor rule, Pattern Recognition Letters 33 (5) (2012) 654–660.

[16] K. Deb, Multi-objective optimization, in: Search methodologies, Springer, 2014, pp. 403–449.

[17] P. Hart, The condensed nearest neighbor rule (corresp.), IEEE Transactions on Information Theory 14 (3) (1968) 515–516.

[18] G. Gates, The reduced nearest neighbor rule (corresp.), IEEE Transactions on Information Theory 18 (3) (1972) 431–433.

[19] G. Ritter, H. Woodruff, S. Lowry, T. Isenhour, An algorithm for a selective nearest neighbor decision rule (corresp.), IEEE Transactions on Information Theory 21 (6) (1975) 665–669.

[20] F. Angiulli, Fast nearest neighbor condensation for large data sets classification, IEEE Transactions on Knowledge and Data Engineering 19 (11) (2007) 1450–1464.

[21] D. L. Wilson, Asymptotic properties of nearest neighbor rules using edited data, IEEE Transactions on Systems, Man, and Cybernetics 2 (3) (1972) 408–421.

[22] I. Tomek, An experiment with the edited nearest-neighbor rule, IEEE Transactions on Systems, Man, and Cybernetics (6) (1976) 448–452.

[23] P. A. Devijver, J. Kittler, Pattern recognition: A statistical approach, Prentice Hall, 1982.

[24] N. Mukahar, B. A. Rosdi, Performance Comparison of Prototype Selection Based on Edition Search for Nearest Neighbor Classification, in: Proceedings of the 7th International Conference on Software and Computer Applications, ACM, New York, NY, USA, 2018, pp. 143–146.

[25] B. V. Dasarathy, J. S. Sánchez, S. Townsend, Nearest neighbour editing and condensing tools–synergy exploitation, Pattern Analysis & Applications 3 (1) (2000) 19–30.

[26] D. R. Wilson, T. R. Martinez, Instance pruning techniques, in: Proceedings of the Fourteenth International Conference on Machine Learning, ICML '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, pp. 403–411.

[27] H. Brighton, C. Mellish, On the Consistency of Information Filters for Lazy Learning Algorithms, in: J. Zytkow, J. Rauch (Eds.), Principles of Data Mining and Knowledge Discovery, Vol. 1704 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1999, pp. 283–288.

[28] Á. Arnaiz-González, J. Díez-Pastor, J. J. R. Diez, C. García-Osorio, Local sets for multi-label instance selection, Applied Soft Computing 68 (2018) 651–666.

[29] J. Cano, F. Herrera, M. Lozano, Using evolutionary algorithms as instance selection for data reduction in kdd: an experimental study, IEEE

Transactions on Evolutionary Computation 7 (6) (2003) 561–575. doi:10.1109/TEVC.2003.819265.

[30] N. García-Pedrajas, J. Pérez-Rodríguez, Multi-selection of instances: A straightforward way to improve evolutionary instance selection, Applied Soft Computing 12 (11) (2012) 3590–3602.

[31] L. J. Eshelman, The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination, in: Proceedings of the First Workshop on Foundations of Genetic Algorithms, Bloomington Campus, Indiana, USA, 1990, pp. 265–283.

[32] P. Hernandez-Leal, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, J. A. Olvera-Lopez, InstanceRank based on borders for instance selection, Pattern Recognition 46 (1) (2013) 365–375.

[33] J. J. Valero-Mas, J. Calvo-Zaragoza, J. R. Rico-Juan, J. M. Iñesta, An experimental study on rank methods for prototype selection, Soft Computing 21 (19) (2017) 5703–5715.

[34] J. R. Rico-Juan, J. M. Iñesta, New rank methods for reducing the size of the training set using the nearest neighbor rule, Pattern Recognition Letters 33 (5) (2012) 654–660.

[35] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-Based Learning Applied to Document Recognition, in: Intelligent Signal Processing, IEEE Press, 2001, pp. 306–351.

[36] J. Hull, A database for handwritten text recognition research, IEEE T. Pattern Anal. 16 (5) (1994) 550–554.

[37] L. J. Latecki, R. Lakämper, U. Eckhardt, Shape descriptors for non-rigid shapes with a single closed contour, in: Conference on Computer Vision and Pattern Recognition (CVPR 2000), 13–15 June 2000, Hilton Head, SC, USA, IEEE Computer Society, 2000, pp. 1424–1429.

[38] J. Calvo-Zaragoza, J. Oncina, Recognition of pen-based music notation: the homus dataset, in: 22nd International Conference on Pattern Recognition (ICPR), IEEE, 2014, pp. 3038–3043.

[39] D. Dheeru, E. Karra Taniskidou, UCI machine learning repository (2017). URL `http://archive.ics.uci.edu/ml`

[40] F. Alimoglu, E. Alpaydin, Methods of combining multiple classifiers based on different representations for pen-based handwritten digit recognition, in: Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAINN 96), Istambul, Turkey, 1996.

[41] P. W. Frey, D. J. Slate, Letter recognition using holland-style adaptive classifiers, Machine Learning 6 (2) (1991) 161–182.

[42] H. Freeman, On the encoding of arbitrary geometric configurations, IRE Transactions on Electronic Computers (2) (1961) 260–268.

[43] R. A. Wagner, M. J. Fischer, The string-to-string correction problem, Journal of the ACM (JACM) 21 (1) (1974) 168–173.

[44] H. Sakoe, S. Chiba, Dynamic programming algorithm optimization for spoken word recognition, in: Readings in speech recognition, Elsevier, 1990, pp. 159–165.

[45] D. R. Wilson, T. R. Martinez, Improved heterogeneous distance functions, Journal of Artificial Intelligence Research 6 (1997) 1–34.

[46] N. Natarajan, I. Dhillon, P. Ravikumar, A. Tewari, Learning with noisy labels, in: Advances in Neural Information Processing Systems, Lake Tahoe, Canada, 2013, pp. 1196–1204.

[47] J. J. Valero-Mas, J. Calvo-Zaragoza, J. R. Rico-Juan, On the suitability of prototype selection methods for knn classification with distributed data, Neurocomputing 203 (2016) 150–160.