Adaptive training set reduction for nearest neighbor classification

Juan Ramón Rico-Juan, José M. Iñesta

www.elsevier.com/locate/neucom

PII: S0925-2312(14)00228-8
DOI: <http://dx.doi.org/10.1016/j.neucom.2014.01.033>
Reference: NEUCOM13954

To appear in: *Neurocomputing*

Received date: 26 April 2013
Revised date: 20 November 2013
Accepted date: 21 January 2014

Cite this article as: Juan Ramón Rico-Juan, José M. Iñesta, Adaptive training set reduction for nearest neighbor classification, *Neurocomputing*, <http://dx.doi.org/10.1016/j.neucom.2014.01.033>

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting galley proof before it is published in its final citable form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Adaptive training set reduction for nearest neighbor classification

Juan Ramón Rico-Juan^{a,*}, José M. Iñesta^{a,**}

^a*Dpto. Lenguajes y Sistemas Informáticos, Universidad de Alicante, E-03081 Alicante, Spain*

Abstract

The research community related to the human-interaction framework is becoming increasingly more interested in interactive pattern recognition, taking direct advantage of the feedback information provided by the user in each interaction step in order to improve raw performance. The application of this scheme requires learning techniques that are able to adaptively re-train the system and tune it to user behavior and the specific task considered. Traditional static editing methods filter the training set by applying certain rules in order to eliminate outliers or maintain those prototypes that can be beneficial in classification. This paper presents two new adaptive rank methods for selecting the best prototypes from a training set in order to establish its size according to an external parameter that controls the adaptation process, while maintaining the classification accuracy. These methods estimate the probability of each prototype of correctly classifying a new sample. This probability is used to sort the training set by relevance in classification. The results show that the proposed methods are able to maintain the error rate while reducing the size of the training set, thus allowing new examples to be learned with a few extra computations.

Keywords: editing, condensing, rank methods, sorted prototypes selection, adaptive pattern recognition, incremental algorithms

*Tel. +34 96 5903400 ext. 2738; Fax. +34 96 5909326

**Tel. +34 96 5903698; Fax. +34 96 5909326

Email addresses: juanra@dlsi.ua.es (Juan Ramón Rico-Juan), inesta@dlsi.ua.es (José M. Iñesta)

8 1. Introduction

9 The research community related to the human-interaction framework is becoming
 10 increasingly more interested in interactive pattern recognition (IPR), taking direct
 11 advantage of the feedback information provided by the user in each interaction step
 12 in order to improve raw performance. Placing pattern recognition techniques in this
 13 framework requires changes be made to the algorithms used for training.

14 The application of these ideas to the IPR framework implies that adequate training
 15 criteria must be established (Vinciarelli and Bengio, 2002; Ye et al., 2007). These cri-
 16 teria should allow the development of adaptive training algorithms that take the maxi-
 17 mum advantage of the interaction-derived data, which provides new (*data, class*) pairs,
 18 to re-train the system and tune it to user behavior and the specific task considered.

19 The k -nearest neighbors rule has been widely used in practice in non-parametric
 20 methods, particularly when a statistical knowledge of the conditional density functions
 21 of each class is not available, which is often the case in real classification problems.

22 The combination of simplicity and the fact that the asymptotic error is fixed by
 23 attending to the optimal Bayes error (Cover and Hart, 1967) are important qualities
 24 that characterize the nearest neighbor rule. However, one of the main problems of
 25 this technique is that it requires all the prototypes to be stored in memory in order
 26 to compute the distances needed to apply the k -nearest rule. Its sensitivity to noisy
 27 instances is also known. Many works (Hart, 1968; Wilson, 1972; Caballero et al., 2006;
 28 Gil-Pita and Yao, 2008; Alejo et al., 2011) have attempted to reduce the size of the
 29 training set, aiming both to reduce complexity and avoid outliers, whilst maintaining
 30 the same classification accuracy as when using the entire training set.

31 The problem of reducing instances can be dealt with two different approaches (Li
 32 et al., 2005), one of them is the generation of new prototypes that replace and repre-
 33 sent the previous ones (Chang, 1974), whilst the other consist of selecting a subset of
 34 prototypes of the original training set, either by using a *condensing* algorithm that ob-

tains a minimal subset of prototypes that lead to the same performance as when using the whole training set, or by using an *editing* algorithm, removing atypical prototypes from the original set, thus avoiding overlapping among classes. Other kind of algorithms are hybrid, since they combine the properties of the previous ones (Grochowski and Jankowski, 2004). These algorithms try to eliminate noisy examples and reduce the number of prototypes selected.

The main problems when using condensing algorithms are to decide which examples should remain in the set and how the typicality of the training examples should be evaluated. Condensing algorithms place more emphasis on minimizing the size of the training set and its consistence, but outlier samples that harm the accuracy of the classification are often selected for the prototype set. Identifying these noisy training examples is therefore the most important challenge for editing algorithms.

Further information can be found in a review (García et al., 2012) about prototype selection algorithms based on the nearest neighbor technique for classification. That publication also presents a taxonomy and an extended empirical study.

With respect to the algorithms proposed in this paper, all the prototypes in the set will be rated with a score that is used to establish a priority for them to be selected. These algorithms can therefore be used for editing or condensing, depending on an external parameter η representing an a posteriori accumulated probability that controls their performance in the range $[0, 1]$, as will be explained later. For values close to one, their behavior is similar to that of an editing algorithm, while when η is close to zero the algorithms perform like a condensing algorithm.

These methods are incremental, since they adapt the learned rank values to new examples with little extra computations, thus avoiding the need to recompute all the previous examples. These new incremental methods are based on the best of the static (classical) methods proposed by the authors in a previous article (Rico-Juan and Iñesta, 2012). The basic idea of this rank is based on the estimation of the probability of an

instance to participate in a correct classification by using the nearest neighbor rule. This methodology allows the user to control the size of the resulting training set.

The most important points of the methodology are described in the following sections. The second section provides an overview of the ideas used in the classical and static methodologies to reduce the training set. The two new incremental methods are introduced in the third section. In the fourth section, the results obtained when applying different algorithms to reduce the size of the training set, their classification error rates, and the amount of additional computations needed for the incremental model are shown applied to some widely used data collections. Finally, some conclusions and future lines of work are presented.

2. Classical static methodologies

The *Condensed Nearest Neighbor Rule* (CNN) (Hart, 1968) was one of the first techniques used to reduce the size of the training set. This algorithm selects a subset S of the original training set T such that every member of T is closer to a member of S of the same class than to a member of a different class. The main issue with this algorithm is its sensitivity to noisy prototypes.

Multiedit Condensing (MCNN) (Dasarathy et al., 2000) solves this particular CNN problem. The goal of MCNN is to remove the outliers by using an editing algorithm (Wilson, 1972) and then applying CNN. The editing algorithm starts with a subset $S = T$, and each instance in S is removed if its class does not agree with the majority of its k -NN. This procedure edits out both noisy instances and close to border prototypes. The MCNN applies the algorithm repeatedly until all remaining instances have the majority of their neighbors in the same class.

Fast Condensed Nearest Neighbor (FCNN) (Angiulli, 2007) extends the classic CNN, improving the performance in time cost and accuracy of the results. In the original publication, this algorithm was compared to hybrid methods, and it was shown

to be about three orders of magnitude faster than them, with comparable accuracy results. The basic steps begin with a subset with one centroid per class and follow the CNN criterion, selecting the best candidates using nearest neighbor and nearest enemy concepts (Voronoi cell).

2.1. Static antecedent of the proposed algorithm

Now, the basic ideas of the static algorithm previously published by the authors (Rico-Juan and Iñesta, 2012) are briefly introduced. The intuitive concepts are represented graphically in Figure 1, where each prototype of the training set, a , is evaluated searching its nearest enemy (the nearest to a from a different class), b , and its best candidate, c . This *best candidate* is the prototype that receives the vote from a , because if c is in the final selection set of prototypes, it will classify a correctly with the 1-NN technique.

Figure 1: Illustrations of the static voting methods used: (a) 1_FN and (b) 1_FE.

In the *one vote per prototype farther* algorithm (1_FN, illustrated in Fig. 1a), only

one vote per prototype, a , is considered in the training set. The algorithm is focused on finding the best candidate, c , to vote for that it is the farthest prototype from a , belonging to the same class as a . It will, therefore, be just nearer than the nearest enemy, $b = \operatorname{argmin}_{x \in T \setminus \{a\}} \{d(a, x) : \operatorname{class}(a) \neq \operatorname{class}(x)\}$:

$$c = \operatorname{argmax}_{x \in T \setminus \{a\}} \{ d(a, x) : \operatorname{class}(a) = \operatorname{class}(x) \wedge d(a, x) < d(a, b) \} \quad (1)$$

In the case of the *one vote per prototype near the enemy* algorithm (1 NE, illustrated in Fig. 1b), the scheme is similar to that of the method shown above, but now the vote for the best candidate, c , is for the prototype that would correctly classify a using the NN rule, which is simultaneously the nearest enemy to b and $d(a, c) < d(a, b)$:

$$c = \operatorname{argmin}_{x \in T \setminus \{a\}} \{ d(x, b) : \operatorname{class}(x) = \operatorname{class}(a) \wedge d(a, x) < d(a, b) \} \quad (2)$$

99 In these two methods, the process of voting is repeated for each element of the
100 training set T in order to obtain one score per prototype. An estimated posteriori prob-
101 ability per prototype is obtained by normalizing the scores using the sum of all the
102 votes for a given class. This probability is used as an estimate of how many times one
103 prototype could be used as the NN in order to classify another one correctly.

104 3. The proposed adaptive methodology

105 The main idea is that the prototypes in the training set, T , should vote for the rest
106 of the prototypes that assist them in a correct classification. The methods estimate a
107 probability for each prototype that indicates its relevance in a particular classification
108 task. These probabilities are normalized for each class, so the sum of the probabilities
109 for the training samples for each class will be one. This permits us to define an exter-
110 nal parameter, η in $[0, 1]$, as the probability mass of the best prototypes relevance for
111 classification.

112 The next step is to sort the prototypes in the training set according to their relevance
 113 and select the best candidates before their accumulated probability exceeds η . This
 114 parameter allows the performance of this method to be controlled and adapted to the
 115 particular needs in each task. Since the problem is analyzed in an incremental manner,
 116 it is assumed that the input of new prototypes in the training set leads to a review of the
 117 individual estimates. These issues are discussed below.

118 In order to illustrate the methodology proposed in this paper, a 2-dimensional bi-
 119 nary distribution of training samples is depicted in Figures 2 and 3 below in this section,
 120 for a k -NN classification task.

121 3.1. Incremental one vote per prototype farther

122 The *incremental one vote per prototype farther* (1_FN_j), is based on static 1_FN
 123 (Rico-Juan and Iñesta, 2012). This adaptive version receives the training set T and a
 124 new prototype x as parameters. The 1_FN_j algorithm is focused on finding the best
 125 candidate to vote for it. This situation is shown in Figure 2. The new prototype x is
 126 added to T , thus computing its vote using T (finding the nearest enemy and voting for
 127 the farthest example in its class; for example, in Figure 2, prototype a votes for c).
 128 However, four different situations may occur for the new prototype, x . The situations
 129 in Figure 2 represented by $x = 1$ and $x = 2$ do not generate additional updates in T ,
 130 because it does not replace c (voted prototype) or b (nearest enemy). In the situation of
 131 $x = 3$, x replaces c as the prototype to be voted for, and the algorithm undoes the vote
 132 for c and votes for the new x . In the case of $x = 4$, the new prototype replaces b , and
 133 the algorithm will update all calculations related to a in order to preserve the one vote
 134 per prototype farther criterion (Eq. (1)).

135 The adaptive algorithm used to implement this method is detailed below for a train-
 136 ing set T and a new prototype x . The algorithm can be applied to an already available
 137 data set or by starting from scratch, with the only constraint that T can not be the empty
 138 set, but must have at least one representative prototype per class.

Figure 2: The incremental voting method for two classes in the one vote per prototype farther approach, and the four cases to be considered when a new prototype is included in the training set.

```

139 function vote_1_FN_i ( $T, a$ ) _____
140     votes( $a$ )  $\leftarrow$  1 {Initialising vote}
141      $b \leftarrow$  findNE( $T, a$ )
142      $c \leftarrow$  findFN( $T, a, b$ )
143     updateVotesNE( $T, a$ ) {Case 4 in Fig. 2}
144     updateVotesFN( $T, a$ ) {Case 3 in Fig. 2}
145     if  $c = \text{null}$  then {if  $c$  does not exist, the vote from  $a$  goes to itself}
146         votes( $a$ )++
147     else
148         votes( $c$ )++
149     end if
150 end function
151 function updateVotesNE( $T, a$ ) _____
152     listNE  $\leftarrow$  {  $x \in T$  : findNE( $T, x$ )  $\neq$  findNE( $T \cup \{a\}, x$ ) }
153     for all  $x \in \text{listNE}$  do
154         votes ( findNE( $T, x$ ) ) --
155         votes ( findNE( $T \cup \{a\}, x$ ) ) ++
156     end for
157 end function

```

```

158 function updateVotesFN( $T, a$ )
159    $listFN \leftarrow \{ x \in T :$ 
160      $findFN(T, x, findNE(T, x)) \neq findFN(T \cup \{a\}, x, findNE(T \cup \{a\}, x)) \}$ 
161   for all  $x \in listFN$  do
162     votes (  $findFN(T, x, findNE(T, x))$  ) --
163     votes (  $findFN(T \cup \{a\}, x, findNE(T \cup \{a\}, x))$  ) ++
164   end for
165 end function

```

The function $updateVotesNE(T, a)$ updates the votes for the prototypes whose nearest enemy in T is different from that in $T \cup \{a\}$, and $updateVotesFN(T, a)$ updates the votes of the prototypes whose farthest neighbor in T is different from that in $T \cup \{a\}$. $findNE(T, a)$ returns the nearest prototype to a in T that is from a different class, and $findFN(T, a, b)$ returns the farthest prototype to a in T from the same class whose distance is lower than $d(a, b)$.

Most of the functions have been optimized using dynamic programming. For example, if the nearest neighbor of a prototype is searched in T , the first time the computational cost is $O(|T|)$ but next time it will be $O(1)$. So the costs of the previous functions are: $O(|T|)$ for finding functions, $O(|T||listNE|)$ for $updateVotesNE$, and $O(|T||listFN|)$ for $updateVotesFN$. The final cost of the main function $vote_1_FN_i(T, a)$ is $O(|T|, \max\{|listNE|, |listFN|\})$ compared to the static version, where $vote_1_FN$ that runs in $O(|T|^2)$. When $\max\{|listNE|, |listFN|\}$ is much smaller than $|T|$, as it is in practice (see the experiments section), we can consider an actual performance time in $O(|T|)$ for the adaptive method.

3.2. Incremental one vote per prototype near the enemy

The *Incremental one vote per prototype near the enemy* (1_NE_i) is based on static 1_NE (Rico-Juan and Iñesta, 2012). This algorithm receives the same parameters as the previous 1_FN_i : the training set T and a new prototype x . The 1_FE static method

185 attempts to find the best candidate to vote for it, while the adaptive method has to
 186 preserve the rules of this method (Eq. (2)) while computing as little as possible. The
 187 situation for a new prototype is shown in Figure 3. When the new x is added, its vote is
 188 computed using T , finding the nearest enemy and voting for its nearest prototype in the
 189 same class as x . In Figure 3, the prototype a votes for c . Four different situations may
 190 also occur for the new prototype, depending on the proximity relations to its neighbors.
 191 The situations represented by $x = 1$ and $x = 2$ do not generate additional updates in T ,
 192 because c (the prototype to be voted for) and b (the nearest enemy) do not need to be
 193 replaced. In the situation $x = 3$, c has to be replaced with x as the prototype to be voted
 194 for, and the algorithm must then undo the vote for c . In the case of $x = 4$, the new
 195 prototype replaces b , and the algorithm must therefore update all calculations related
 196 to a in order to preserve the criterion in equation (2).

Figure 3: The incremental voting method for two classes for one candidate near to the enemy class, and the four cases to be considered with a new prototype.

197 The adaptive algorithm used to implement this method is detailed below for T and
 198 the new prototype x . As in the previous algorithm, the only constraint is that T can not
 199 be the empty set, but must have at least one representative prototype per class.

200 **function** vote_1_NE_i(T, a) _____
 201 $\text{votes}(a) \leftarrow 1$ {Initialising vote}
 202 $b \leftarrow \text{findNE}(T, a)$
 203 $c \leftarrow \text{findNNE}(T, a, b)$

```

204   updateVotesNE( $T, a$ ) {Case 4 in Fig. 3}
205   updateVotesNNE( $T, a$ ) {Case 3 in Fig. 3}
206   if  $c = null$  then {if  $c$  does not exist, the vote from  $a$  goes to itself}
207       votes( $a$ )++
208   else
209       votes( $c$ )++
210   end if
211 end function
212 function updateVotesNE( $T, a$ ) _____
213      $listNE \leftarrow \{ x \in T : findNE(T, x) \neq findNE(T \cup \{a\}, x) \}$ 
214     for all  $x \in listNE$  do
215         votes (  $findNE(T, x)$  ) --
216         votes (  $findNE(T \cup \{a\}, x)$  ) ++
217     end for
218 end function
219 function updateVotesNNE( $T, a$ ) _____
220      $listNNE \leftarrow \{ x \in T :$ 
221          $findNNE(T, x, findNE(T, x)) \neq findNNE(T \cup \{a\}, x, findNE(T \cup \{a\}, x)) \}$ 
222     for all  $x \in listNNE$  do
223         votes (  $findNNE(T, x, findNE(T, x))$  ) --
224         votes (  $findNNE(T \cup \{a\}, x, findNE(T \cup \{a\}, x))$  ) ++
225     end for
226 end function

```

227 The methods that have the same name as those in the 1.FN.i algorithm play the
228 same role as it was described for this procedure above. The new methods here are the
229 method $findNNE(T, a, b)$ returns the nearest neighbor to enemy, b , in T of the same
230 class as a which distance is lower than $d(a, b)$. Then function $updateVotesNNE(T, a)$

updates the votes of the prototypes whose nearest neighbor enemy, in T is different from that in $T \cup \{a\}$.

In a similar way to previous algorithm, the computational cost of functions are: $O(|T||listNNE|)$ for `updateVotesNNE`, and $O(|T|)$ for `findNNE( $T, a, b$ )`. So the final cost of the main function `vote_1_NE_i( $T, a$ )` is $O(|T|, \max\{|listNE|, |listNNE|\})$ compared to the static version, where `vote_1_NE` that runs in $O(|T|^2)$. When $\max\{|listNE|, |listNNE|\} \ll |T|$, as can be seen in the experiments section, we can consider an actual performance time in $O(|T|)$ for the adaptive method.

4. Results

The experiments were performed using three well known databases. The first is a database of uppercase characters (the NIST Special Database 3 of the National Institute of Standards and Technology) and the second contains digits (the US Post Service digit recognition corpus, USPS). In both cases, the classification task was performed using contour descriptions with Freeman codes (Freeman, 1961), and the string edit distance (Wagner and Fischer, 1974) was used as the similarity measure. The last database tested was the UCI Machine Learning Repository (Asuncion and Newman, 2007), and some of the main collection sets were used. The prototypes are vectors of numbers and some of their components may have missing values. In order to deal with this problem, a normalized heterogeneous distance, such as HVDM (Wilson and Martinez, 1997) was used.

The proposed algorithms were tested using different values for the accumulated probability η , covering most of the available range ($\eta \in \{0.10, 0.25, 0.50, 0.75, 0.90\}$) in order to test their performances with different training sets.

It is important to note that, when the static and incremental methods are compared, i.e., NE vs. NE_i, and FN vs. FN_i, they are expected to perform the same in terms of their accuracies. The most important difference between them is that the latter follow

an incremental adaptive processing of the training set. This requires only a few additional calculations when new samples are added in order to adapt the ranking values for all prototypes.

All the experiments with the incremental methods started with a set composed of just one prototype per class, selected randomly from the complete set of prototypes.

In the early stages of testing we have discovered that if the algorithms vote for the second nearest enemy prototype, some outliers are avoided (Fig. 4), thus making their performance more robust in these situations. Additionally, in a usual behavior where nearest enemy is not an outlier, the difference estimations between the first and second enemy, do not change the evaluations of the proposed algorithms, while the distribution of the classes are well represented in the training set. Using a higher value for the order of best nearest enemy damaged the performance notably.

In order to represent the performance values of the algorithms, we have therefore added a number prefix to the name of the algorithms, denoting what position in the rank the vote goes to. 1_FN, 1_NE, 2_FN, and 2_NE algorithms were therefore tested.

Figure 4: An outlier case example. The first nearest enemy from x (1-NE) is rounded by enemy prototypes class, whilst the second (2-NE) avoid this problem.

Experiments with the NIST database

The subset of the 26 uppercase handwritten characters was used. The experiments were constructed by taking 500 writers and selecting the samples randomly. 4 sets

276 were extracted with 1300 examples each (26 classes and 50 examples per class) and
 277 the 4-fold cross-validation technique was used. 4 experiments were therefore evaluated
 278 with 1300 prototypes as the size of the test set and 3900 (the rest) prototypes for the
 279 training set.

280 As it is shown in Table 1, the best classification results were obtained for the origi-
 281 nal set, with 1_NE_i and 2_NE_i and $\eta = 0.75$, since they achieve better accuracy with
 282 fewer training prototypes. Figure 5 also shows the results grouped by method and pa-
 283 rameter value. The 1_FN_i and 2_FN_i methods obtained the best classification rates
 284 for small values of the external parameter. In the case of 0.1, the value dramatically
 285 reduces the training set size to 4.20% with regard to the original set and achieves an
 286 accuracy of 80.08%. For higher values of the external parameter η , the accuracy is
 287 similar for both algorithms (NE_i and FN_i), although in the case of NE_i the number
 288 of prototypes selected is smaller.

289 With regard to the CNN, MCNN and FCNN methods, better classification accura-
 290 cies were obtained by the proposed methods for similar training sizes. The 1_NE_i(0.75)
 291 and 2_NE_i(0.75) methods give a training set size of 51.06% and 51.76%, respectively,
 292 with an accuracy of around 90%. This performance is comparable to that of the edit-
 293 ing method. When roughly 25% of the training set is maintained, 1_FN_i(0.50) and
 294 2_FN_i(0.50) have the best average classification rates with a significant improvement
 295 over the CNN and slightly better than FCNN. If the size of the training set is set at
 296 approximately 13%, MCNN, 1_FN_i(0.25) and 2_FN_i(0.25), then the methods obtain
 297 similar results.

298 The table 2 shows the times¹ for the experiments performed by the online incre-
 299 mental learning system. Initially, the training set had just an example per class and the
 300 new examples were arriving one by one. The incremental algorithms learnt the new
 301 example whilst the static algorithms retrained all the system. The order of the mag-

¹The machine used was an Intel Core i7-2600k at 3.4GHz with 8 GB of RAM.

nititude of 1_FN, 2_FN, 1_NE and 2_NE algorithms were comparable, about $2500 \cdot 10^3$ seconds. For the static algorithms, the MCNN and CNN were faster than the former, and the fastest was FCNN, with about $1800 \cdot 10^3$. Nevertheless, the new incremental algorithms performed three orders of magnitude faster, $1.4 \cdot 10^3$ for the NIST database, showing dramatically the improvement on the efficiency of incremental/on-line learning.

Table 1: Comparison result rates over the NIST database (uppercase characters) with the number of training size prototypes and their percentage of the original (3900 examples, 150 per 26 classes) and the adaptive computation updates. Means and deviations for the 4-fold-cross-validation experiments are presented.

	accuracy (%)	training size (%)	prototypes updated (%)
original	91.3 $\pm$ 0.8	100.0 $\pm$ 0.0	
1_NE_i(0.10)	72.8 $\pm$ 1.5	2.21 $\pm$ 0.05	0.103 $\pm$ 0.002
1_NE_i(0.25)	83.8 $\pm$ 1.0	7.38 $\pm$ 0.12	
1_NE_i(0.50)	88.9 $\pm$ 1.0	23.44 $\pm$ 0.11	
1_NE_i(0.75)	90.6 $\pm$ 0.9	51.06 $\pm$ 0.08	
1_NE_i(0.90)	91.2 $\pm$ 0.8	80.67 $\pm$ 0.00	
2_NE_i(0.10)	71.2 $\pm$ 0.8	2.32 $\pm$ 0.07	0.201 $\pm$ 0.004
2_NE_i(0.25)	82.4 $\pm$ 1.2	7.44 $\pm$ 0.07	
2_NE_i(0.50)	88.7 $\pm$ 1.1	22.60 $\pm$ 0.06	
2_NE_i(0.75)	90.6 $\pm$ 1.0	50.76 $\pm$ 0.07	
2_NE_i(0.90)	91.2 $\pm$ 0.9	80.67 $\pm$ 0.00	
1_FN_i(0.10)	80.6 $\pm$ 0.8	4.33 $\pm$ 0.08	0.104 $\pm$ 0.005
1_FN_i(0.25)	85.9 $\pm$ 0.6	12.58 $\pm$ 0.16	
1_FN_i(0.50)	89.4 $\pm$ 0.7	31.46 $\pm$ 0.10	
1_FN_i(0.75)	90.5 $\pm$ 1.0	56.41 $\pm$ 0.13	
1_FN_i(0.90)	91.1 $\pm$ 0.8	80.67 $\pm$ 0.00	
2_FN_i(0.10)	80.0 $\pm$ 1.3	4.20 $\pm$ 0.05	0.202 $\pm$ 0.006
2_FN_i(0.25)	86.4 $\pm$ 1.0	12.42 $\pm$ 0.09	
2_FN_i(0.50)	89.2 $\pm$ 1.0	30.96 $\pm$ 0.13	
2_FN_i(0.75)	90.2 $\pm$ 0.8	55.99 $\pm$ 0.14	
2_FN_i(0.90)	91.0 $\pm$ 0.7	80.67 $\pm$ 0.00	
CNN	86.2 $\pm$ 1.4	25.8 $\pm$ 0.6	-
FCNN	88.2 $\pm$ 1.3	25.5 $\pm$ 0.5	-
MCNN	85.6 $\pm$ 1.3	13.6 $\pm$ 0.4	-

As is shown in Table 1 the prototypes updated by the adaptive methods have very low values of less than 0.202%.

It is noticeable that any reduction in the original training set size decreases the average accuracy. This may result from the high intrinsic dimensionality of the features, which causes all examples to be near to the same and a different class. Previous studies

Table 2: Average times for the experiments in 10^3 seconds (static vs. incremental methods) over the NIST (uppercase characters) and USPS dataset.

	NIST		USPS	
	static	incremental	static	incremental
CNN	1922.5	-	1289.8	-
FCNN	1844.8	-	755.7	-
MCNN	2102.0	-	2642.7	-
1_NE	2559.7	1.4	6388.8	0.3
1_FN	2524.3	1.4	5655.7	0.3
2_NE	2560.6	1.4	6396.4	0.3
2_FN	2525.3	1.4	5701.2	0.3

as that of Ferri et al. (1999) support this hypothesis, and the authors examine the sensitivity of the relation between edit methods based on the nearest neighbor rules and accuracy. Nevertheless, the $\{1,2\}_{NE}(0.75)$ methods have a classification rate that is similar to the original one with only 50% of the size of the training set, which is significantly better than those of CNN, MCNN and FCNN.

The USPS database

The original digit database was divided into two sets: The training set with 7291 examples and the test set with 2007 examples. No cross-validation was therefore performed.

Table 3 shows that the best classification rates were obtained by the original set, $1_{NE}(0.90)$ and $2_{FN}(0.90)$. In this case, the deviations are not available because only one test was performed. Again, the high dimensionality of the feature vectors, implies that any reduction in the size of the original training set decreases the accuracy. The $1_{NE}(0.90)$ and $2_{FN}(0.90)$ methods utilized 62.64% and 80.06% of the training set examples, respectively. It is worth noting that $1_{NE}(0.75)$ appears in the fourth position with only 37.77% of the training set. When approximately 25% of the training set is considered, $1_{NE}(0.50)$, $1_{FN}(0.50)$, and $2_{FN}(0.50)$ have the best classification rates, which are lower than those of CNN, MCNN and FCNN.

In this case, Table 3 shows that the prototype update obtained with the adaptive

(a) Classification rate

(b) Training set remaining size (%)

Figure 5: Comparison of the proposed methods grouped by method and value for the parameter η with the NIST database (uppercase characters). Classification rates and training set reduction rates are shown.

332 methods has low values - less than 0.1%.

333 Similar time results to those with the NIST were obtained for USPS (see Table
 334 2). The order of magnitude of 1_FN, 2_FN algorithms were comparable with about
 335 $5600 \cdot 10^3$ and 1_NE, 2_NE with $6300 \cdot 10^3$ seconds. The MCNN, CNN, and FCNN
 336 were faster than the static FN and NE, being FCNN the fastest with around $700 \cdot 10^3$.
 337 Now, the proposed incremental algorithms were four orders of magnitude faster, with
 338 $0.3 \cdot 10^3$ for this database.

Table 3: Comparison of results for the USPS digit database (7291 training and 2007 test examples). Only one fold was performed, signifying that no deviations are available.

	accuracy (%)	training size (%)	prototypes updated (%)
original	89.9	100.00	
1_NE_i(0.10)	78.0	1.12	0.06
1_NE_i(0.25)	87.0	4.65	
1_NE_i(0.50)	88.7	17.79	
1_NE_i(0.75)	89.3	50.35	
1_NE_i(0.90)	89.9	80.06	
2_NE_i(0.10)	73.1	1.14	0.1
2_NE_i(0.25)	86.1	4.47	
2_NE_i(0.50)	88.3	16.31	
2_NE_i(0.75)	89.2	50.06	
2_NE_i(0.90)	89.7	80.06	
1_FN_i(0.10)	82.7	1.34	0.06
1_FN_i(0.25)	85.4	4.97	
1_FN_i(0.50)	87.9	19.33	
1_FN_i(0.75)	89.3	50.84	
1_FN_i(0.90)	89.9	80.06	
2_FN_i(0.10)	82.5	1.10	0.09
2_FN_i(0.25)	87.0	4.17	
2_FN_i(0.50)	88.2	16.24	
2_FN_i(0.75)	89.4	50.40	
2_FN_i(0.90)	89.5	80.06	
CNN	85.0	19.50	
FCNN	86.0	19.10	
MCNN	86.9	8.40	

339 *UCI Machine Learning Repository*

340 Some of the main data collections from this repository have been employed in this
 341 study. A 10-fold-cross-validation method was used. The sizes of the training and test
 342 set were different, according to the total size of each particular collection.

343 The adaptive methods with $\eta = 0.1$ obtained similar classification rates to those ob-
 344 tained when using the whole set, but with a dramatic decrease in the size of the training
 345 set, as in the case of *bcw* (see Table 4(a)), with an average of 0.5% of the original
 346 size and only 0.3% of prototypes needed to be updated. These reductions were signif-
 347 icantly better than those obtained with the CNN, MCNN and FCNN methods. In the
 348 worst cases, such as *glass* and *io*, the results were comparable to those of MCNN and
 349 FCNN, but when η was set to 0.25, which is not presented in the table. These figures
 350 are exactly the same as those obtained by the static versions of the algorithms, and the
 351 whole evaluation of the accuracy can be seen in (Rico-Juan and Iñesta, 2012) Table 4,
 352 in which an exhaustive study was presented for the static algorithms. This shows that
 353 the incrementality of the proposed methods does not affect their performance.

354 The prototype updates (see Table 4(b)) were in an interval between 0.3% for the
 355 *bcw* database and 2.9% for the *hepatitis* database.

356 The time costs of the algorithms are shown in Table 4(c). The experiments was
 357 performed as detailed before (for NIST and USPS). As the size of datasets are fewer
 358 than NIST and USPS the differences with respect the times are lower but the order
 359 between the algorithms is the same and the incremental versions a clearly better.

360 Note that in the case of the experiments performed with the USPS database (Ta-
 361 ble 3), 1_FN_i and 1_NE_i show that very few prototypes were updated by the adaptive
 362 methods: less than 0.06% with regard to the MNIST and the UCI databases tested, this
 363 being the best result in terms of the number of updated prototypes.

364 5. Conclusions and future work

365 This paper presents two new adaptive algorithms for prototype selection with appli-
 366 cation in classification tasks in which the training sets can be incrementally built. This
 367 situation occurs in interactive applications, in which the user feedback can be taken into
 368 account as new labeled samples with which to improve the system performance in the

next runs. These algorithms provide a different estimation of the probability that a new sample can be correctly classified by another prototype, in an incremental manner. In most results, the accuracy obtained in the same conditions was better than that obtained using classical algorithms such as CNN, MCNN, and FCNN. The classification accuracy of these incremental algorithms was the same as that of the static algorithms from which the new ones evolved. The 1_FN_j, 2_FN_j and 1_NE_j algorithms showed good behavior in terms of trade off between the classification accuracy and the reduction in the number of prototypes in the training set.

The static algorithms from which the new incremental ones are derived had a computational time complexity of $O(|T|^2)$ (the same as CNN, MCNN and FCNN). However, the proposed algorithms run in $O(|T||U|)$ where U are the number of prototypes to be updated adaptively. Therefore, it requires very few computations, since $|U| \ll |T|$ and the actual performance is linear, $O(|T|)$. This way, the adaptive algorithms provide a substantial reduction of the computational cost, while maintaining the good quality of the results of the static algorithms. In particular, the methods that use the nearest prototype (1_FN_j and 1_NE_j) have less updates than those using the second nearest prototype (2_FN_j and 2_NE_j). While the updates in the first case range between 0.3% and 1.5%, in the second case they range between 0.7% and 2.9%, which is almost double. It is clear that the latter methods changes more ranking values in the training set. However, although the percentage of updates depends on the database, the results are always quite good, with a value of less than 2.9%. Keep in mind that the static model needs to update the 100% of the prototypes in the same conditions.

The new proposed algorithms FN_j and NE_j are much faster and rank all training set prototypes, not only the condensed resulting subset.

As future works, these adaptive methods could also be combined with techniques such as Bagging or Boosting to obtain a more stable list of ranking values. Also, more attention should be paid to the outliers problem, preserving the actual computational

cost of the new proposed algorithms. Finally, due to the way of prototype selection of NE and FN (using probability instead of number of prototypes) it would be interesting to test/adapt the algorithms on datasets with imbalanced classes.

Acknowledgments

This work is partially supported by the Spanish CICYT under project DPI2006-15542-C04-01, the Spanish MICINN through project TIN2009-14205-CO4-01 and by the Spanish research program Consolider Ingenio 2010: MIPRCV (CSD2007-00018).

6. References

- Alejo, R., Sotoca, J. M., García, V., Valdovinos, R. M., 2011. Back Propagation with Balanced MSE Cost Function and Nearest Neighbor Editing for Handling Class Overlap and Class Imbalance. In: Advances in Computational Intelligence, LNCS 6691. pp. 199–206.
- Angiulli, F., 2007. Fast Nearest Neighbor Condensation for Large Data Sets Classification. IEEE Trans. Knowl. Data Eng. 19 (11), 1450–1464.
- Asuncion, A., Newman, D., 2007. UCI Machine Learning Repository.
URL <http://www.ics.uci.edu/~mllearn/MLRepository.html>
- Caballero, Y., Bello, R., Alvarez, D., Garcia, M. M., Pizano, Y., 2006. Improving the k-NN method: Rough Set in edit training set. In: IFIP 19th World Computer Congress PPAI'06. pp. 21–30.
- Chang, C., 1974. Finding Prototypes for Nearest Neighbour Classifiers. IEEE Transactions on Computers 23 (11), 1179–1184.
- Cover, T., Hart, P., Jan. 1967. Nearest neighbor pattern classification. IEEE Transactions on Information Theory 13 (1), 21–27.

- 419 Dasarathy, B. V., Sánchez, J. S., Townsend, S., 2000. Nearest Neighbour Editing and
420 Condensing Tools-Synergy Exploitation. *Pattern Anal. Appl.* 3 (1), 19–30.
- 421 Ferri, F. J., Albert, J. V., Vidal, E., 1999. Considerations about sample-size sensitivity
422 of a family of edited nearest-neighbor rules. *IEEE Transactions on Systems, Man,
423 and Cybernetics, Part B* 29 (5), 667–672.
- 424 Freeman, H., Jun. 1961. On the encoding of arbitrary geometric configurations. *IRE
425 Transactions on Electronic Computer* 10, 260–268.
- 426 García, S., Derrac, J., Cano, J. R., Herrera, F., 2012. Prototype Selection for Nearest
427 Neighbor Classification: Taxonomy and Empirical Study. *IEEE Trans. Pattern Anal.
428 Mach. Intell.* 34 (3), 417–435.
- 429 Gil-Pita, R., Yao, X., 2008. Evolving edited k-Nearest Neighbor Classifiers. *Int. J.
430 Neural Syst.*, 459–467.
- 431 Grochowski, M., Jankowski, N., 2004. Comparison of Instance Selection Algorithms I.
432 Results and Comments. In: *IN ARTIFICIAL INTELLIGENCE AND SOFT COM-
433 PUTING*. Springer, pp. 598–603.
- 434 Hart, P., 1968. The condensed nearest neighbor rule. *IEEE Transactions on Information
435 Theory* 14 (3), 515–516.
- 436 Li, Y., Huang, J., Zhang, W., Zhang, X., Dec. 2005. New prototype selection rule
437 integrated condensing with editing process for the nearest neighbor rules. In: *IEEE
438 International Conference on Industrial Technology*. pp. 950–954.
- 439 Rico-Juan, J. R., Iñesta, J. M., Feb. 2012. New rank methods for reducing the size of
440 the training set using the nearest neighbor rule. *Pattern Recognition Letters* 33 (5),
441 654–660.
- 442 Vinciarelli, A., Bengio, S., 2002. Writer adaptation techniques in HMM based off-line
443 cursive script recognition. *Pattern Recognition Letters* 23 (8), 905–916.

- 444 Wagner, R. A., Fischer, M. J., 1974. The string-to-string correction problem. J. ACM
445 21, 168–173.
- 446 Wilson, D., 1972. Asymptotic properties of nearest neighbor rules using edited data.
447 IEEE Transactions on Systems, Man and Cybernetics 2 (3), 408–421.
- 448 Wilson, D. R., Martinez, T. R., 1997. Improved Heterogeneous Distance Functions.
449 Journal of Artificial Intelligence Research, 1–34.
- 450 Ye, J., Zhao, Z., Liu, H., 2007. Adaptive distance metric learning for clustering. In:
451 IEEE Conference on Computer Vision and Pattern Recognition (CVPR’07). pp. 1–
452 7.

Table 4: Comparison results of some of the UCI Machine Repository databases.

	bcw		wdbc		glass		hc		hh	
	acc.	%	acc.	%	acc.	%	acc.	%	acc.	%
original	95.6 ± 1.4	100 ± 0	94.9 ± 1.5	100 ± 0	88 ± 6	100 ± 0	53 ± 6	100 ± 0	78 ± 9	100 ± 0
CNN	93 ± 2	10.9 ± 0.7	94 ± 3	14 ± 1	89 ± 7	30 ± 2	47 ± 4	64.3 ± 1.5	75 ± 8	42 ± 3
FCNN	93 ± 3	9.4 ± 0.1	93 ± 4	13.3 ± 0.1	90 ± 5	23.7 ± 0.2	51 ± 11	60.1 ± 0.2	70 ± 11	37.6 ± 0.2
MCNN	95 ± 2	2.8 ± 0.5	95 ± 3	7.2 ± 0.9	80 ± 10	17 ± 1	50 ± 7	13.9 ± 1.6	80 ± 10	15.1 ± 1.6
1_NE _i (0.1)	96.1 ± 1.8	0.5 ± 0.1	84 ± 8	0.8 ± 0.1	60 ± 10	3.8 ± 0.3	45.8 ± 8.9	4.5 ± 0.4	73 ± 9	2.4 ± 0.4
2_NE _i (0.1)	96.1 ± 2.9	0.7 ± 0.1	85.6 ± 8.2	0.9 ± 0.1	63.2 ± 11.2	3.9 ± 0.3	44.1 ± 10.7	4.1 ± 0.3	79.9 ± 9.1	2 ± 0.3
1_FN _i (0.1)	96 ± 1.5	2 ± 0	90 ± 5	3.9 ± 0.2	50 ± 2	5.3 ± 0.4	52 ± 7	6 ± 0.3	80 ± 10	4.0 ± 0.2
2_FN _i (0.1)	96 ± 2	1.8 ± 0.1	93 ± 3	4 ± 0	51.2 ± 9.2	5 ± 0.4	46 ± 8	5.5 ± 0.2	81 ± 8	4.0 ± 0.3

bcw (breast cancer wisconsin); hc (heart cleveland); hh (heart hungarian)

	hepatitis		io		iris		pd		zoo	
	acc.	%	acc.	%	acc.	%	acc.	%	acc.	%
original	81 ± 15	100 ± 0	87 ± 4	100 ± 0	94 ± 5	100 ± 0	70 ± 6	100 ± 0	95 ± 6	100 ± 0
CNN	74 ± 17	36.3 ± 0.2	87.1 ± 0.5	23.3 ± 1.6	95 ± 4	18.3 ± 0.2	66 ± 5	48.2 ± 0.2	97 ± 6	16.8 ± 0.3
FCNN	79 ± 9	27.3 ± 0.3	85.5 ± 0.5	11.1 ± 0.1	96 ± 5	15.8 ± 0.2	66 ± 4	46.8 ± 0.2	96 ± 10	13.8 ± 0.3
MCNN	84 ± 10	13.5 ± 1.1	84.9 ± 0.6	11.1 ± 1.1	95 ± 5	8.7 ± 1.7	73 ± 5	15 ± 1	94 ± 7	13 ± 2
1_NE _i (0.1)	67 ± 12	2.2 ± 0.3	55 ± 12	0.9 ± 0.2	817 ± 11	2.7 ± 0.4	69 ± 7	3.0 ± 0.1	90 ± 12	7.7 ± 0.1
2_NE _i (0.1)	65.2 ± 22.7	2.2 ± 0.5	52.8 ± 9.7	0.8 ± 0.2	81.3 ± 10.3	2.3 ± 0.2	68.5 ± 5.9	3 ± 0.2	92.9 ± 9.6	7.7 ± 0
1_FN _i (0.1)	86 ± 4	4.4 ± 0.3	76 ± 5	3.4 ± 0.2	83 ± 13	4.3 ± 0.5	70 ± 5	4.6 ± 0.2	90 ± 11	7.7 ± 0.1
2_FN _i (0.1)	78 ± 10	4.5 ± 0.5	78 ± 8	2.9 ± 0.3	72 ± 13	4.1 ± 0.5	72 ± 3	4.3 ± 0.1	87 ± 15	7.7 ± 0.1

pd (pima diabetes); io (ionosphere)

(a) Means and deviations are presented in accuracy and % of the remaining training set size. In boldface the best results are highlighted in terms of the trade-off between accuracy and training set reduction.

	bcw	wdbc	glass	hc	hh	hepatitis	io	iris	pd	zoo
1_NE _i	0.3 ± 0.1	0.4 ± 0.1	1.0 ± 0.1	0.7 ± 0.1	0.7 ± 0.1	1.3 ± 0.3	0.6 ± 0.1	1.3 ± 0.3	0.3 ± 0	1.1 ± 0.4
2_NE _i	0.7 ± 0.2	0.8 ± 0.1	1.9 ± 0.2	1.4 ± 0.1	1.4 ± 0.2	2.5 ± 0.6	1.2 ± 0.3	2.8 ± 0.7	0.6 ± 0	2.0 ± 0.6
1_FN _i	0.3 ± 0.1	0.4 ± 0.1	1.0 ± 0.1	0.7 ± 0.0	0.8 ± 0.1	1.5 ± 0.6	0.6 ± 0.1	1.4 ± 0.3	0.3 ± 0	1.1 ± 0.4
2_FN _i	0.7 ± 0.3	0.8 ± 0.1	2.0 ± 0.2	1.4 ± 0.2	1.4 ± 0.2	2.9 ± 0.7	1.2 ± 0.2	2.8 ± 0.4	0.6 ± 0	2.0 ± 0.7

bcw (breast cancer wisconsin); hc (heart cleveland); hh (heart hungarian); pd (pima diabetes); io (ionosphere)

(b) Number of prototypes updated in each step using adaptive algorithm. Means and deviations are presented in % of updates.

	bcw(699)		wdbc(569)		glass(214)		hc(303)		hh(294)		hepatitis(155)		io(351)		iris(150)		pd(768)		zoo(101)	
	stat.	incr.	stat.	incr.	stat.	incr.	stat.	incr.	stat.	incr.	stat.	incr.	stat.	incr.	stat.	incr.	stat.	incr.	stat.	incr.
CNN	112.32		86.06		22.91		38.25		37.56		15.43		44.47		15.19		265.48		10.06	
FCNN	73.12		48.26		17.31		89.45		42.06		9.43		32.67		5.59		109.18		4.36	
MCNN	392.72		232.96		23.41		43.05		43.26		15.63		66.97		14.69		594.48		6.76	
1_NE	2733.22	0.12	1754.96	0.11	93.21	0.04	222.65	0.06	216.16	0.06	52.43	0.03	365.47	0.09	50.69	0.03	3991.58	0.13	37.46	0.02
2_NE	2481.42	0.09	1353.66	0.08	63.61	0.03	192.95	0.03	189.76	0.03	25.33	0.02	338.67	0.06	19.29	0.02	3600.08	0.09	6.76	0.02
1_FN	2432.02	0.13	1649.36	0.11	84.21	0.06	198.55	0.07	185.26	0.07	54.83	0.04	344.17	0.08	51.59	0.04	3044.58	0.12	42.06	0.03
2_FN	2401.82	0.08	1304.36	0.08	58.01	0.03	172.85	0.03	165.66	0.03	22.93	0.02	316.97	0.05	18.79	0.02	2934.38	0.08	6.56	0.02

bcw (breast cancer wisconsin); hc (heart cleveland); hh (heart hungarian); pd (pima diabetes); io (ionosphere)

(c) Means of time of experiments in seconds (static vs. incremental methods). The number close to the name are the examples size of the dataset.