New rank methods for reducing the size of the training set using the nearest neighbor rule

Juan Ramón Rico-Juan[a], José Manuel Iñesta[a]

[a]*Dpto. Lenguajes y Sistemas Informáticos, Universidad de Alicante, E-03071 Alicante, Spain*

Abstract

Some new rank methods to select the best prototypes from a training set are proposed in this paper in order to establish its size according to an external parameter, while maintaining the classification accuracy. The traditional methods that filter the training set in a classification task like editing or condensing have some rules that apply to the set in order to remove outliers or keep some prototypes that help in the classification. In our approach, new voting methods are proposed to compute the prototype probability and help to classify correctly a new sample. This probability is the key to sorting the training set out, so a relevance factor from 0 to 1 is used to select the best candidates for each class whose accumulated probabilities are less than that parameter. This approach makes it possible to select the number of prototypes necessary to maintain or even increase the classification accuracy. The results obtained in different high dimensional databases show that these methods maintain the final error rate while reducing the size of the training set.

Key words: editing, condensing, rank methods, sorted prototypes selection

1. Introduction

In classification problems, a statistical knowledge of the conditional density functions of each class is rarely available, so application of the optimal Bayes classification methods is not possible. The nearest neighbor (NN) rule and its extension (k-nearest neighbors) have been the most widely used non-parametric classifiers in practice.

The advantage of the NN rule lies in the fact that it combines its conceptual simplicity with an asymptotic error rate that is conveniently bounded in terms of the optimal

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

Bayes error (Cover and Hart, 1967). However, the NN rule also presents some problems when the number of prototypes is large, since it needs to store all the examples in a memory and is sensitive to noisy instances. Many researchers have studied how to reduce the training set and obtain the same classification ability as when the whole training set is used (Wilson and Martinez, 2000; Wilson, 1972; Hart, 1968).

There are two different ways to deal with the instance reduction problem (Li et al., 2005):

- New prototype generation that creates a new prototype set (Chang, 1974).
- Prototype selection, consisting in selecting a particular subset of prototypes from the original training set:
  - The condensing or reducing algorithms give the minimal subset of prototypes that lead to the same performance as when the whole training set is used.
  - The editing algorithm eliminates atypical prototypes from the original set and removes overlapping among classes.

For condensing algorithms, the key problem is to decide which examples should be retained. The difference between the different condensing algorithms is how the typicality of training examples is evaluated. Condensing algorithms give more emphasis to minimizing the size of the training set and its consistence, but noisy examples may often be selected for the prototype set and harm the classification accuracy. For editing algorithms, identifying these 'bad' training examples that harm the accuracy is the most important challenge.

In this paper, some new rank methods are proposed in order to reduce the size of the training set. This rank is based on estimating the probability that an instance participates in a correct classification using the nearest neighbor rule. So, using this methodology the user could control the size of the resulting training set.

In the second section, some classical methodologies to reduce the training set are explained. In the third section, our new methodologies are introduced with their detailed algorithms. In the fourth section, the results obtained when applying different

algorithms to reduce the size of the training set and their associated error rates on some widely used collection data are shown. Finally, some conclusions and future lines of work are presented.

2. Classical Methodologies

2.1. Condensed Nearest Neighbor Rule

The *Condensed Nearest Neighbor Rule* (CNN) (Hart, 1968) was one of the first techniques to reduce the size of the training set. This algorithm gives a subset S of the 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 algorithm selects a sample randomly from T. If this sample is incorrectly classified using S then it is added to S. The process is repeated until all samples belonging to T are selected and correctly classified using S.

This algorithm is especially sensitive to noise, because noisy instances are retained and the generalization accuracy is damaged because they are usually exceptions and thus do not represent the underlying function well.

There are some other extensions of the CNN technique such as *Selective Nearest Neighbor Rule* (SNN) (Ritter et al., 1975), which changes slightly the basic CNN rules; *Reduced Nearest Neighbor Rule* (Gates, 1972), which finds the reduction subset by decreasing the training set, while CNN uses an incremental approach. All these methods are more complex than CNN; hence, they are beyond our goals and will not be studied. However, Multiedit Condensing can be considered to be a technique derived from CNN due to its good performance.

2.2. Multiedit Condensing

Multiedit Condensing (MCNN) (Dasarathy et al., 2000) solves the problem CNN has with noisy examples. The goal of MCNN is to remove the outliers using an editing algorithm (Wilson, 1972) and then apply CNN. The editing algorithm starts with a subset S equal to T, and then each instance in S is removed if it does not agree with the majority of its k-NN. This edits out noisy instances as well as close border cases.

The multiedit algorithm applies the algorithm repeatedly until all remaining instances have a majority of their neighbors in the same class.

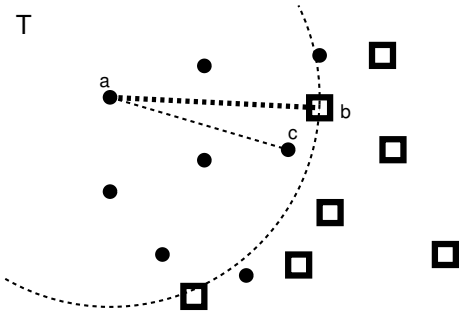
3. Our Methodology

In order to illustrate the main idea of our proposed methodology, a binary classification problem is considered and a distribution of training samples, T. In the figures shown in this section, the points of class 1 are indicated by circles and the points of class 2 by rectangles. The main idea is that the prototypes in the training set vote for the rest of the prototypes that help them to classify correctly and the method estimates a probability for each prototype that shows its importance in a classification task. The next step is to sort the training set according to that probability and select the best candidates before the accumulated probability exceeds the external parameter (from 0 to 1). This parameter allows the performance of this method to be controlled. Low values will reduce the size of the training set, and the performance will be similar to that of condensing algorithms. On the other hand, high values of this parameter will reduce the noise or outlier prototypes as editing algorithms do.

In order to compute the votes for the training set, different approaches are presented and detailed in the following subsections.

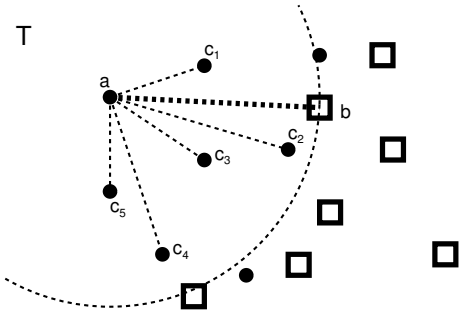
3.1. One vote per prototype farther

In the *one vote per prototype farther* (1-FN), only one vote per prototype in the training set is considered. So the algorithm is focused on finding the best candidate to vote for. A graphical representation of this situation is shown in figure 1(a). The process is repeated for each element of the training set. Then, a is a selected prototype (in this case it belongs to class 1) and the nearest enemy (NE), labeled as b, is found among the rest of the classes (in this case, class 2). The farthest prototype whose class is the same as that of a and whose distance is less than that to the nearest enemy, in this case c, is found. So, if the prototype c belongs to the training set, the prototype a is classified correctly using the NN rule, given the fact that $d(a,c) < d(a,b)$. In other words, there is a nearest neighbor of a whose distance is less than the distance to



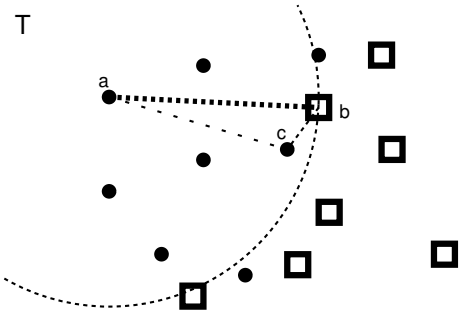
- T: Training set
- a: example selected
- b : The a nearest enemy
  $argmin_{x \in T \sim \{a\}}\{d(a,x) : class(a) \neq class(x)\}$
- c : The a farthest neighbor from the same class with a distance less than $d(a,b)$
  $argmax_{x \in T \sim \{a\}}\{d(a,x) : class(a) = class(x) \wedge d(a,x) < d(a,b)\}$

(a) The voting method for two class prototypes to one candidate far from the selected candidate.



- T: Training set
- a: example selected
- b : The a nearest enemy
  $argmin_{x \in T \sim \{a\}}\{d(a,x) : class(a) \neq class(x)\}$
- c : The a nearest neighbors from the same class with a distance less than $d(a,b)$
  $_{x \in T \sim \{a\}}\{d(a,x) : class(a) = class(x) \wedge d(a,x) < d(a,b)\}$

(b) The voting method for two class prototypes to k candidates.



- T: Training set
- a: example selected
- b : The a nearest enemy
  $argmin_{x \in T \sim \{a\}}\{d(a,x) : class(a) \neq class(x)\}$
- c : The b Nearest Neighbor with the same class as a and whose distance is less than $d(a,b)$
  $argmin_{x \in T \sim \{a\}}\{d(x,b) : class(x) = class(a) \wedge d(a,x) < d(a,b)\}$

(c) The voting method for two class prototypes to one candidate near to the enemy class.

Figure 1: Illustrations of the proposed voting methods.

the nearest enemy. In consequence, the vote from a is given to c. When the method is applied to all the training set prototypes, a score of votes for each prototype is obtained. An estimated probability is obtained normalizing this score using the vote summatory for a given class. This probability is an estimate of how many times a sample could be used as the NN to classify another sample correctly.

The algorithm to compute this method is detailed below:

```
function vote-1-FN(T)
  for a ∈ T do {Initializing votes}
    a.vote ← 1
  end for
  for a ∈ T do {Process all prototypes}
    b ← a.NearestEnemy(T)
    c ← a.FarthestNeighbour(T,b)
    if c = null then {if c does not exist, the prototype a is needed}
      a.vote++
    else
      c.vote++
    end if
  end for
  for a ∈ T do {Compute the probability}
    a.probability ← a.vote / a.sumClassVotes(T)
  end for
end function
```

In this algorithm T is the training set, a. NearestEnemy(T) is a method by which the prototype computes the nearest enemy using the T set, a. FarthestNeighbour(T,b) is a method by which the prototype computes the farthest neighbor from a with the same class and whose distance is less than that to b, a. and a.sumClassVotes(T) is a method by which the prototype sums all the votes from the prototypes that belong to the class of a. The votes are initialized to 1 (uniform distribution) in order to obtain probabilities greater than zero.

3.2. *k votes per prototype*

In this approach, *k votes per prototype* (k-NN) in the training set are allowed. This method is focused on finding the best candidates to vote for. In figure 1(b) a graphical representation of this situation is shown. The process described next is repeated for each element of the training set. First, a is a selected prototype (in this case it belongs to class 1) and the nearest enemy, labeled as b, is found among the rest of the classes (in this case, class 2). The nearest neighbor prototypes whose class is the same as a and whose distance is less than that to the nearest enemy, in this case c_i, are found. So, if the prototypes c_i belong to the training set, the prototype a is classified correctly using the NN rule. Given that $d(a, c_i) < d(a, b)$, in other words, there is a NN to a whose distance is less than that to the nearest enemy. In consequence, the vote from a is given to every c_i. When the method is applied to all training set prototypes, a score for each prototype is obtained with the votes. As in the previous subsection, the individual probability is computed by dividing the individual votes between the class accumulated votes.

The algorithm to compute this method is detailed below:

```
function vote-k(T)
  for a ∈ T do {Initializing votes}
    a.vote ← 1
  end for
  for a ∈ T do {Process all prototypes}
    b ← a.NearestEnemy(T)
    C ← a.NearestNeighbours(T,b)
    if |C| = 0 then {if C is empty the prototype a is needed}
      a.vote++
    else
      for c ∈ C do
        c.vote++
      end for
    end if
  end for
```

for $a \in T$ **do** {Compute the probability}
  a.probability $\leftarrow$ a.vote / a.sumClassVotes(T)
end for
end function

Where a. NearestNeighbours(T,b) is a method by which the prototype computes the nearest neighbors to a with the same class and whose distance is less than that to b. It returns a set of these prototypes. The votes are initialized to 1 (uniform distribution) as in the previous method.

3.3. One vote per prototype near the enemy

In this case (1-NE), one vote per prototype is considered and the process is repeated for all training set examples. A graphical example of this approximation is shown in figure 1(c). In the example, a is a selected prototype and the nearest enemy is labeled as b. The nearest prototype to the nearest enemy whose class is the same as that for a and whose distance is less than $d(a,b)$, in this case c, is selected to receive the vote from a. So, if c belongs to the new training set, the example a is classified correctly because $d(a,c) < d(a,b)$. In other words, the NN from a is near the NE.

The algorithm to compute this method is detailed below:

function vote-1-NE(T)
  for $a \in T$ **do** {Initializing votes}
    a.vote $\leftarrow$ 1
  end for
  for $a \in T$ **do** {Process all prototypes}
    $b \leftarrow a$.NearestEnemy(T)
    $c \leftarrow b$.NearestNeighbour(T,a)
    if c = *null* **then** {if c doesn't exist, the prototype a is needed}
      a.vote++
    else
      c.vote++
    end if
  end for

for $a \in T$ **do** {Compute the probability}

a.probability $\leftarrow$ a.vote / a.sumClassVotes(T)

end for

end function

Where b. NearestNeighbour(T,a) is a method by which the prototype computes the NN to b with the same class as a and whose distance to a is less than to $d(a, b)$.

In the following section, a number of variations on the previous algorithms are presented with a brief description of the difference.

3.4. Variations of the previous algorithms

One vote per prototype farther discarding outliers (1-FN-D3) is similar to the previous *one vote per prototype farther* (1-FN). The difference is that the algorithm is applied several times to compute the individual votes of the prototypes and it avoids computing the NE for the examples whose votes in the previous iteration are less than 3. Thus, the outliers are initialized to 1 and they only receive their own vote. At the end of the iteration, they will obtain 2 votes. These prototypes are discarded in the next iteration. The final rank is computed with the votes of the final iteration.

The *One vote per second prototype farther* (2-FN) is similar to the previous *one vote per prototype farther*. In order to discard the outliers (isolated examples that are surrounded by different class examples) the method to compute the b:=a.NearestEnemy(T) is replaced by b:=a.SecondNearestEnemy(T). This way, the isolated examples are not considered.

Next, a combination of classical methods and our method is proposed. The idea is to combine CNN and MCNN with some of the algorithms presented above.

The CNN rule selects the prototypes in an unsorted way, so the resulting training sets may be different for each run. If the best prototypes of each class are selected to start the CNN algorithm and the order of the selection is fixed (more important prototypes are examined first), this may help CNN to find a good candidate for the condensed training set. So, if the 1-FN method is selected to sort the training set, the new algorithm is called 1-FN-CNN and if the k-NN method is selected, the new algorithm is called as k-NN-CNN.

In the same way as in the previous paragraph, the multiedit condensing algorithm (MCNN) selects the prototypes in an unsorted way, so the resulting training set may also be different each time. In the first step, the 1-FN algorithm is applied and in the second step a MCNN is used to obtain the reduced training set. This new algorithm is called 1-FN-MCNN. If the k-NN is used in the first step, the new algorithm is called k-NN-MCNN.

4. Results

The experiments were performed using two well known isolated handwritten character databases and the UCI Machine Learning Repository (Asuncion and Newman, 2007). 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) as the similarity measure. In the case of the UCI database, some of the main collection sets are used. The prototypes are vectors of numbers and some of their components may have missing values. In order to cope with this problem, a normalised heterogeneous distance such as HVDM (Wilson and Martinez, 1997) is used.

The new algorithms proposed were tested using low, medium, and high parameter values (0.10, 0.25, 0.50, 0.75, 0.90) in order to control their performance with different training sets.

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 were extracted with 1300 examples each (26 classes and 50 examples per class) and the 4-fold cross-validation technique was used. So, 4 experiments were evaluated with 1300 prototypes as the size of the test set and 3900 (the rest) prototypes for the training set. As shown in table 1 and graphically in Fig. 3, the best classification results were

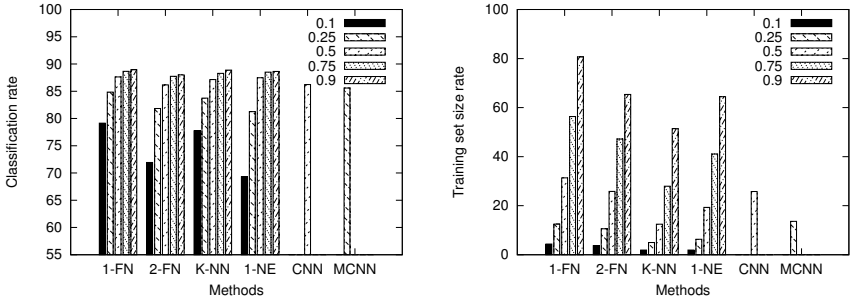


Figure 2: Comparison of the proposed methods grouped by method and parameter with the NIST database (uppercase characters): a) Classification rate b) Training set remaining size (%).

obtained for the original set, 1-FN, 2-FN, and 1-NE with 0.90 and 0.75 as the external parameter. Also, Fig. 2 shows the results grouped by method and parameter. The 1-FN method obtains the best classification rates for all parameters in our proposed methods, especially for the parameter value 0.1 that reduces dramatically the training set size to 3.69% with respect to the original set and achieves 79.15% accuracy. With regard to CNN and MCNN methods, better classification accuracies were obtained by our methods for similar training sizes. The 1-FN(0.90) and 2-FN(0.90) methods give a training set size of 80.67% and 65.37% , respectively. This performance is similar to that of the standard editing method. If about 25% of the training set is kept, 1-NE(0.50) and 2-FN(0.50) have the best average classification rates but they do not achieve any significant improvement with respect to the CNN and the MCNN methods.

Note that any reduction in the original training set size decreases the average accuracy. This may be due to the high intrinsic dimensionality of the feature vectors which causes all examples to be near the other class examples. This is supported by studies like Ferri et al. (1999) where the authors examined the sensitivity of the relation between edit methods based on the nearest neighbor rules and the accuracy. Nevertheless, Fig. 3 shows that the 1-NE(0.75) method has a classification rate similar to the original rate with only 41.1% of the training set prototypes, and significantly better than that of CNN and MCNN.

	accuracy (%)	training size	training size (%)
original	89.2 ± 0.9	3900 ± 0	100 ± 0
1-FN(0.90)	89.0 ± 1.1	3146 ± 1	80.7 ± 0.1
2-FN(0.90)	88.9 ± 1.3	2549 ± 6	65.4 ± 0.2
1-FN(0.75)	88.7 ± 1.3	2198 ± 8	56.4 ± 0.2
2-FN(0.75)	88.3 ± 1.1	1840 ± 3	47.2 ± 0.1
1-NE(0.90)	88.6 ± 1.3	2514 ± 3	64.5 ± 0.1
1-NE(0.75)	88.5 ± 1.6	1602 ± 4	41.1 ± 0.1
1-FN-D3(0.90)	88.0 ± 1.2	2172 ± 9	55.7 ± 0.2
1-FN-D3(0.75)	87.8 ± 1.2	1758 ± 20	45.1 ± 0.3
1-FN(0.50)	87.6 ± 0.9	1223 ± 10	31.3 ± 0.3
1-NE(0.50)	87.5 ± 0.8	753 ± 6	19.3 ± 0.2
2-FN(0.50)	87.2 ± 1.2	1009 ± 5	25.8 ± 0.1
K-NN-CNN	86.5 ± 1.4	1000 ± 20	25.7 ± 0.5
1-FN-MCNN	86.4 ± 0.8	538 ± 6	13.8 ± 0.2
K-NN(0.90)	86.3 ± 1.4	2000 ± 40	51 ± 1
1-FN-CNN	86.3 ± 1.1	1020 ± 20	26.1 ± 0.3
CNN	86.2 ± 1.4	1000 ± 20	25.8 ± 0.6
1-FN-D3(0.50)	86.2 ± 1.6	1120 ± 10	28.7 ± 0.4
MCNN	85.6 ± 1.3	530 ± 10	13.6 ± 0.4
K-NN-MCNN	85.5 ± 1.4	540 ± 20	13.8 ± 0.4
1-FN(0.25)	84.9 ± 0.4	487 ± 6	12.5 ± 0.1
K-NN-(0.75)	84.3 ± 0.8	1100 ± 20	27.9 ± 0.6
2-FN-(0.25)	83.8 ± 1.4	412 ± 3	10.6 ± 0.1
1-FN-D3(0.25)	81.8 ± 0.5	436 ± 6	11.2 ± 0.2
1-NE(0.25)	81.3 ± 1.6	246 ± 5	6.3 ± 0.1
1-FN(0.10)	79.2 ± 0.8	170 ± 4	4.3 ± 0.1
K-NN(0.50)	78.0 ± 0.3	490 ± 10	12.4 ± 0.3
2-FN(0.10)	77.8 ± 1.7	143 ± 4	3.6 ± 0.1
1-FN-D3(0.10)	71.9 ± 0.1	160 ± 10	4.1 ± 0.3
1-NE(0.10)	69.3 ± 1.6	75 ± 3	1.9 ± 0.1
K-NN-(0.25)	68 ± 2	194 ± 1	4.9 ± 0.1
K-NN-(0.10)	59 ± 3	75 ± 1	1.9 ± 0.1

Table 1: Comparison results sorted by classification accuracy rates over the NIST database (uppercase characters) with the number of training size prototypes and their percentage of the original. Means and deviations for the 4-fold-cross-validation experiments are presented.

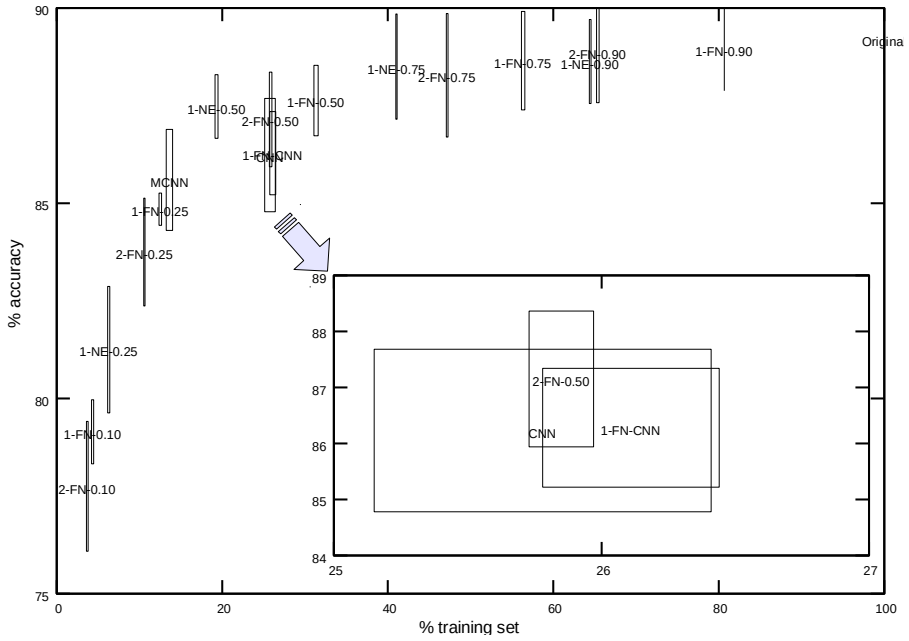


Figure 3: Graphical comparison of the proposed methods using accuracy vs. percentage of training set. The boxes around each method name represent graphically the *standard deviations*. The zoomed region represents CNN, 2-FN(0.5) and 1-FN-CNN more clearly.

The USPS database

In this classification task, the original digit database was divided in two sets: The training set with 7291 examples and test set with 2007 examples, so no cross-validation was performed.

The table 2 shows that the best classification rates are obtained by the original set, 1-NE(0.90) and 2-FN(0.90). In this case, the deviations are not available because there are only two sets (training and test). Again, due to the high dimensionality of the feature vectors, 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 remarkable that 1-NE(0.75) appears in the fourth position with only 37.77% of the training set. If about 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 less than that of CNN and MCNN.

UCI Machine Learning Repository

Some of the main data collections used in this study have already been used in other articles like Wilson and Martinez (2000). The 10-fold-cross-validation method was used. The size of the training and test set is different according to the total size of the collection.

The table 3 shows that in most cases our methods with 0.1 as external parameter obtained similar classification rates as those obtained using the whole set, but with a dramatic decrease in the size of the training set, as in the *bcw* case, with an average of 0.5% of the original size. These reductions are significantly better than those obtained with the CNN and MCNN methods. In the worst cases, such as *glass* and *io*, the results are comparable to those of MCNN.

5. Conclusions and future work

In this paper, new algorithms to reduce the size of the training set for use in a classification task have been presented. These algorithms give a different estimate of the probability that a new example may be classified correctly by a training set example.

	accuracy(%)	training size	*trainingsize*(%)
original	88.9	7291	100.0
1-NE(0.90)	88.4	4567	62.6
1-FN(0.90)	88.3	5837	80.0
1-NE(0.75)	88.2	2754	37.7
2-FN(0.90)	88.1	4602	63.1
1-FN-D3(0.90)	88.0	3830	52.5
1-FN(0.75)	87.9	3724	51.0
K-NN(0.90)	87.9	2991	41.0
1-FN(0.50)	87.7	1603	21.9
1-NE(0.50)	87.7	1007	13.8
2-FN(0.50)	87.4	1192	16.3
K-NN(0.75)	87.4	1859	25.5
k-MCNN	87.1	597	8.1
2-NN(0.75)	87.0	2855	39.1
1-FN-MCNN	87.0	592	8.1
MCNN	86.9	614	8.4
1-FN-D3(0.75)	86.5	2379	32.6
1-FN(0.25)	86.4	520	7.1
K-NN(0.50)	85.8	935	12.8
2-FN(0.25)	85.5	388	5.3
1-FN-CNN	85.5	1446	19.8
K-NN-CNN	85.1	1392	19.0
CNN	85.0	1418	19.5
1-NE(0.25)	85.0	279	3.8
K-NN(0.24)	83.8	362	4.9
1-FN-D3(0.50)	83.4	1030	14.1
2-FN(0.10)	81.2	109	1.4
K-NN(0.10)	80.3	120	1.6
1-FN(0.10)	79.8	153	2.1
1-NE(0.10)	77.1	71	1.0
1-FN-D3(0.25)	75.8	346	4.8
1-FN-D3(0.10)	63.3	105	1.4

Table 2: Comparison of results sorted by classification rates over the USPS digits database. Only one fold was made so no deviations are available.

	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
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(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
1-NE(0.25)	96.3 ± 1.7	1.4 ± 0.1	91 ± 4	2.8 ± 0.1	**80 ± 10**	**9.0 ± 0.6**	48.3 ± 7	13.4 ± 0.5	83 ± 9	8.1 ± 0.7
1-NE(0.5)	95 ± 1.4	7.3 ± 0.2	93 ± 4	12.6 ± 0.4	80 ± 10	23 ± 0	50.4 ± 8.6	35.6 ± 1.1	78 ± 8	26 ± 1
1-NE(0.75)	95.3 ± 1.2	50.2 ± 0.1	94 ± 2	50.1 ± 0.1	86 ± 6	53.3 ± 0.8	53.2 ± 8	60.7 ± 1.1	76 ± 6	52.6 ± 0.4
1-NE(0.9)	95.3 ± 1.5	80.2 ± 0.1	95.1 ± 1.6	80.2 ± 0.1	87 ± 9	81.7 ± 0.6	52.6 ± 5.2	82.3 ± 0.7	76 ± 8	80.3 ± 0.5
1-FN(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
1-FN(0.25)	96.6 ± 1.4	5.9 ± 0.3	94 ± 3	11.7 ± 0.4	78 ± 12	13.2 ± 0.4	50 ± 8	15.9 ± 0.7	79 ± 12	12.0 ± 0.5
1-FN(0.5)	96.7 ± 1.2	18.9 ± 0.6	95 ± 3	29.3 ± 0.7	85 ± 9	31.4 ± 1.2	54 ± 8	38 ± 1	77 ± 8	31.7 ± 0.8
1-FN(0.75)	95.1 ± 1.8	51.6 ± 0.2	95 ± 2	54.3 ± 0.8	89 ± 5	56.5 ± 1.1	54 ± 8	63.1 ± 1.2	75 ± 6	56.6 ± 0.8
1-FN(0.9)	95.6 ± 1.4	80.2 ± 0.1	95 ± 2	80.2 ± 0.1	87 ± 8	81.6 ± 0.5	54 ± 6	82.2 ± 0.8	76 ± 9	80.3 ± 0.5
2-FN(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**
2-FN(0.25)	96 ± 2	5.4 ± 0.3	93 ± 3	11.8 ± 0.5	72 ± 14	11.9 ± 0.6	50 ± 10	14.6 ± 0.6	77 ± 9	11.4 ± 0.5
2-FN(0.5)	95.9 ± 1.7	17.9 ± 0.6	93 ± 2.4	29.3 ± 1	79 ± 12	28.8 ± 1.4	52 ± 6	35.3 ± 0.7	75 ± 9	29.7 ± 0.6
2-FN(0.75)	95 ± 2	51.7 ± 0.4	95 ± 3	54.1 ± 1.1	85 ± 6	54.2 ± 1.1	54 ± 7	60.3 ± 0.8	72 ± 4	54.5 ± 0.7
2-FN(0.9)	95.4 ± 1.1	80.2 ± 0.1	95 ± 2	80.2 ± 0.1	87 ± 6	81.6 ± 0.6	53 ± 6	81.2 ± 0.8	78 ± 8	80.3 ± 0.5

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 ± 2	87 ± 5	23.3 ± 1.6	95 ± 4	18 ± 2	66 ± 5	48.2 ± 1.6	97 ± 6	17 ± 3
MCNN	84 ± 10	13.5 ± 1.1	85 ± 6	11.1 ± 1.1	95 ± 5	8.7 ± 1.7	73 ± 5	15 ± 1	94 ± 7	13 ± 2
1-NE(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
1-NE(0.25)	81 ± 6	5.4 ± 0.5	**80 ± 6**	**3.3 ± 0.3**	84 ± 11	5.3 ± 0.6	71 ± 7	10.6 ± 0.2	92 ± 9	8.6 ± 0.7
1-NE(0.5)	86 ± 6	20 ± 1	87 ± 5	14.1 ± 0.5	92 ± 7	13 ± 1	71 ± 6	29.8 ± 0.5	95 ± 6	20.5 ± 1.3
1-NE(0.75)	82 ± 12	52.5 ± 0.6	90 ± 4	50.3 ± 0.4	94 ± 5	51.4 ± 0.4	69 ± 5	54.8 ± 0.5	96 ± 6	54.0 ± 1.2
1-NE(0.9)	79 ± 16	80.6 ± 0.5	88 ± 4	80.3 ± 0.3	95 ± 6	80.8 ± 0.2	70 ± 6	80.2 ± 0.1	95 ± 6	83.5 ± 1.8
1-FN(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**
1-FN(0.25)	86 ± 9	12.0 ± 0.6	85 ± 7	10.4 ± 0.3	90 ± 6	10.9 ± 0.4	73 ± 5	13.6 ± 0.3	91 ± 13	9.1 ± 0.5
1-FN(0.5)	84 ± 9	29.3 ± 1.2	86 ± 6	28.1 ± 0.6	93 ± 5	26.4 ± 0.9	69 ± 7	33.9 ± 0.5	94 ± 5	20.7 ± 0.9
1-FN(0.75)	85 ± 10	54.1 ± 1.4	88 ± 4	53.2 ± 0.6	95 ± 6	54.8 ± 0.8	68 ± 6	58.9 ± 0.5	97 ± 6	54.3 ± 1.1
1-FN(0.9)	79 ± 14	80.6 ± 0.5	88 ± 4	80.3 ± 0.3	95 ± 7	80.8 ± 0.2	70 ± 6	80.2 ± 0	96 ± 6	83.5 ± 1.8
2-FN(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
2-FN(0.25)	84 ± 10	11.9 ± 0.6	82 ± 4	9.2 ± 0.3	93 ± 5	10.8 ± 1.2	68 ± 4	12.9 ± 0.4	86 ± 12	9.1 ± 0.5
2-FN(0.5)	86 ± 9	29.4 ± 1	88 ± 6	26.6 ± 0.7	93 ± 6	26.1 ± 1.6	67 ± 8	31.4 ± 0.4	93 ± 7	22.2 ± 1.9
2-FN(0.75)	82 ± 12	54.3 ± 1.1	90 ± 6	52.1 ± 0.5	93 ± 5	55.3 ± 1.4	68 ± 7	56.4 ± 0.4	95 ± 6	54.5 ± 1.9
2-FN(0.9)	81 ± 15	80.6 ± 0.5	88 ± 4	80.3 ± 0.3	94 ± 5	80.8 ± 0.2	67 ± 6	80.2 ± 0.1	96 ± 6	83.5 ± 1.8

pd (pima diabetes); io (ionosphere)

Table 3: Comparison results on some of the UCI Machine Repository databases. Means and deviations are presented in accuracy and % of remaining training set size.

The results obtained for accuracy are in most cases better than those obtained using the classical algorithms compared. Note the good performance shown by 1-FN, 2-FN and 1-NE algorithms with respect to the classification accuracy and the reduction in the number of prototypes in the training set. Moreover, these algorithms have a low time complexity, only $O\left(|T|^2\right)$ which is run offline, before the classification operation.

As future work, the new algorithms proposed could be compared with evolved instance-based learning algorithms such as DROP (Wilson and Martinez, 2000), although their method has higher complexity than the one presented here. In addition, how to estimate the control parameter of these algorithms to obtain optimal results in classification tasks remains to be studied. Finally, boosting techniques adapted to the proposed algorithms could explore to observe the behavior.

References

Asuncion, A., Newman, D., 2007. UCI machine learning repository. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`

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.

Dasarathy, B. V., Sánchez, J. S., Townsend, S., 2000. Nearest neighbour editing and condensing tools-synergy exploitation. Pattern Anal. Appl. 3 (1), 19–30.

Ferri, F. J., Albert, J. V., Vidal, E., 1999. Considerations about sample-size sensitivity of a family of edited nearest-neighbor rules. IEEE Transactions on Systems, Man, and Cybernetics, Part B 29 (5), 667–672.

Freeman, H., Jun. 1961. On the encoding of arbitrary geometric configurations. IRE Transactions on Electronic Computer 10, 260–268.

Gates, G., 1972. The Reduced Nearest Neighbor Rule. IEEE Transactions on Information Theory IT-18(3), 431–433.

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

Li, Y., Huang, J., Zhang, W., Zhang, X., Dec. 2005. New prototype selection rule integrated condensing with editing process for the nearest neighbor rules. In: IEEE International Conference on Industrial Technology. pp. 950–954.

Ritter, G. L., Woodruff, H. B., Lowry, S. R., Isenhour, T., 1975. An algorithm for a selective nearest neighbor rule. IEEE Transactions on Information Theory 21 (6), 665–669.

Wagner, R. A., Fischer, M. J., 1974. The string-to-string correction problem. J. ACM 21, 168–173.

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

Wilson, D., Martinez, T., 2000. Reduction techniques for instance-based learning algorithms. Machine Learning 38 (3), 257–286.

Wilson, D. R., Martinez, T. R., 1997. Improved heterogeneous distance functions. Journal of Artificial Intelligence Research, 1–34.