FHC: an adaptive fast hybrid method for k-NN classification

STEFANOS OUGIAROGLOU[1], GEORGIOS EVANGELIDIS,
Dept. of Applied Informatics, University of Macedonia, Greece.
E-mail: {stoug,gevan}@uom.gr

DIMITRIS A. DERVOS,
Dept. of Informatics, Alexander TEI of Thessaloniki, Greece.
E-mail: dad@it.teithe.gr

Abstract

A popular and easy to implement classifier is k-Nearest Neighbor (k-NN) algorithm. However, sequentially searching for nearest neighbors in large datasets leads to inefficient classification because of the high computational cost involved. This paper presents an adaptive hybrid and cluster-based method for speeding up k-NN classifier. The proposed method reduces the computational cost as much as possible while maintaining classification accuracy at high levels. The method is based on the well-known k-means clustering algorithm and consists of two main parts: (i) a preprocessing algorithm that builds a two-level, cluster-based data structure, and, (ii) a hybrid classifier that classifies new items by accessing either the first or the second level of the data structure. The proposed approach was tested on seven real life datasets and the experiential measurements were statistically validated by the Wilcoxon signed ranks test. The results show that the proposed classification method can be used either to achieve high accuracy with slightly higher cost or to reduce the cost at a minimum level with slightly lower accuracy.

Keywords: Nearest Neighbours, Classification, Clustering, Prototypes

1 Introduction

k-Nearest Neighbor (k-NN) classifier is one of the most widely-used classification methods [7]. k-NN classifier is a lazy classifier, i.e., it does not build any classification model. When a new unclassified item x must be classified, the classifier searches the available training data and retrieves the k nearest data items (neighbors) to x according to a distance metric. Then, x is classified to the most common class indicated via a majority vote of the nearest neighbors. If more than one classes are the most common (ties in voting), the major class is determined either randomly or by choosing the class of the nearest neighbor.

k-NN classifier is effective, easy to implement and has many applications. The classification cost depends on the size of the training set (TS). Particularly, for each incoming new item x, the classifier has to compute all distances between x and all training items. This cost may be very high and even prohibitive for large datasets. This drawback is an active research field that has attracted the interest of many researchers over the last decades. Therefore, many algorithms and techniques have been proposed to speed-up k-NN searching. A possible categorization of the speed-up

[1]S. Ougiaroglou is supported by a scholarship from the State Scholarships Foundation (I.K.Y.) of Greece

methods is: (i) multi-dimensional indexing methods, (ii) data reduction techniques (DRTs), and, (iii) cluster-based methods (CBMs). Contrary to indexes and CBMs, DRTs have the extra advantage of the reduction of storage requirements.

Indexes [24] are used to reduce the cost of the nearest neighbor searches. Their performance depends on data dimensionality. Practically, most of them are effective when datasets have moderate dimensionality (e.g., 2-10). In higher dimensions the "dimensionality curse" may render the use of indexes irrelevant since their performance degrades rapidly and can become even worse than that of a sequential search.

Data Reduction Techniques (DRTs) [27, 12, 30, 4, 19] build a small representative set of the initial TS that is called Condensing Set (CS). Thus, DRTs reduce both storage requirements and computational cost while attempting to maintain classification accuracy at high levels. DRTs can be divided into two main algorithm categories: prototype selection (PS) and prototype abstraction (PA) algorithms. PS algorithms select representative items (prototypes) from the initial TS, whereas, PA algorithms generate prototypes by summarizing similar training items.

Cluster-based Methods (CBMs) [15, 31, 28, 16] apply a clustering procedure during pre-processing in order to assign data items into clusters. In the classification step, for each incoming new item, CBMs dynamically form an appropriate training subset of the initial TS, and use it to classify the new item. This subset is called reference set and consists of the data of one or more clusters.

In [22], we proposed a hybrid schema for fast k-NN classification that combined the idea of DRTs with that of CBMs. In this paper we extend our work by proposing an efficient classification approach through the hybrid schema that takes into consideration the distribution of items in the classes. Moreover, we present additional experiments and results. Our contribution is the development of an adaptive, hybrid and cluster-based method for speeding-up k-NN classifier. More specifically, we develop a fast cluster-based preprocessing algorithm that builds a two level data structure and efficient classifiers that access either the first or the second level of the data structure. The first level stores the cluster centroids (representatives) for each class. The second level stores the set of items belonging to each cluster. Therefore, a DRT and a CBM are combined in a hybrid schema to achieve the desirable performance.

The rest of this paper is organized as follows. Section 2 briefly presents the related work. Section 3 considers in detail the proposed classification method. Section 4 presents the experimental study based on seven real life datasets. The results are validated by the Wilcoxon signed ranks test. The paper concludes in Section 5.

2 Related Work

Condensing Nearest Neighbor (CNN) rule [14] is the earliest and one of the best known PS algorithms. The idea behind the algorithm is that the non close-border (or "internal") training items can be removed without loss of accuracy, since they do not define the decision boundaries between classes. Thus, CNN-rule reduces the cost of k-NN classifier as well as the storage requirements by removing the internal training items. CNN-rule determines the amount of the selected prototypes automatically based on the level of noise in the data and the number of classes (the more the classes, the more boundaries exist and, as a consequence, the more the prototypes selected). CNN-rule performs as follows: Initially, a training item is moved from TS to CS. Then, the algorithm classifies the content of TS using the 1-NN classier on the content of CS. When an item is wrongly classified it is moved from TS to CS. The idea is that whenever an item is misclassified, it may be close to the decision boundaries, and thus, it must be placed in CS. This procedure continues until there are no moves from TS to CS during a complete pass of TS. This means that all the contents of TS can be correctly classified by the content of CS. Then, the content of TS is discarded to release memory. A drawback is that CNN-rule is order-dependent, i.e., it builds a different CS by examining the same data in a different order.

Although, there are many other PS algorithms that either extend CNN-rule or are based on the same idea, CNN-rule continues to be the reference PS algorithm and it is used in many papers for comparison purposes. Some of the these algorithms are the Reduced NN rule [13], the Selective NN rule [23], the Modified CNN rule [11], the Generalized CNN rule [6], the Fast CNN [3], and the family of IBL algorithms [1].

IB2 belongs to the popular family of IBL algorithms [1] and it is based on the idea of CNN-rule. Actually, it is a dynamic one-pass version of it. We characterize IB2 as dynamic because it is able to use new training items in order to update an already constructed CS and without requiring the previously discarded TS items. Therefore, IB2 can be used in dynamic environments where new TS items are gradually available and it can manage large datasets that cannot fit into the main memory. It works as follows: Initially an item is moved from TS to CS. Then, each TS item is classified by applying the 1-NN classifier on the content of CS. If it is correctly classified, it is moved from TS to CS. Otherwise, it is discarded. Here, there are no multiple passes over the TS data. Therefore, the algorithm can not ensure that the discarded items can correctly be classified by the content of the final CS. Moreover, like CNN-rule, IB2 automatically determines the size of CS and is order-dependent.

A recently proposed PS algorithm is Prototype Selection by Clustering (PSC) [20]. It is based on the concept of cluster homogeneity. PSC considers that homogeneous clusters (with items that belong to the same class) contain items that lie in the "internal" area of a class, whereas, non-homogeneous clusters contain close-border items. The algorithm adopts k-means clustering [18] in order to divide the TS into clusters. For the homogeneous clusters, it places the nearest item to the cluster centroid into CS. For the non-homogeneous clusters, PSC places the items that define the decision boundaries into CS. Although the resulting CS is not data order-dependent, it depends on the selection of the initial means for the k-means clustering.

Reduction through homogeneous clusters (RHC) [21] is a recently proposed PA algorithm. Like PSC and RSP3, it is based on the idea of homogeneity. Like IB2 and PSC, it has as main goal the fast data reduction. RHC continues building clusters by executing k-means clustering until all of them are homogeneous. More specifically, it begins by computing a mean item for each class in TS. These means are called class-centroids. Then, it executes k-means clustering by using the class-centroids as initial means for the k-means clustering. This clustering procedure is executed on the data of each non-homogeneous cluster. Whenever, a homogeneous cluster is identified, the cluster centroid is placed in CS and the items of the particular cluster are discarded. RHC is non-parametric and independent on the ordering of data in TS.

Chen and Jozwik proposed a well-known PA algorithm [5]. Chen and Jozwik algorithm (CJA) repetitively divides TS into subsets. Initially, it finds the pair of items that define the dataset diameter (the most distant items in TS). Suppose, A and B.

Then, CJA divides the TS into two subsets. One subset stores the items nearest to A, whereas, the other subset stores the items closer to B. Then, it keeps on dividing the subset with the larger diameter until a user-predefined number of subsets is built. In the end, CJA computes a mean item for each subset. The mean item is labelled by the most common class label in the corresponding subset and is placed in CS.

The family of Reduction by Space Partitioning (RSP) algorithms [25] includes three variations of CJA. Contrary to CJA, RSP1 does not ignore items in the subsets. It computes as many prototypes as the different classes in the subset. Consequently, RSP1 achieves lower reduction rates but higher classification accuracy. CJA and RSP1 select the subset that will be divided by adopting the criterion of the largest diameter. They assume that subsets with larger diameters contain more items and by splitting them first, a higher reduction rate can be achieved. RSP2 adopts the criterion of overlapping degree [25]. RSP3 continues splitting until all subsets are homogeneous, i.e., is non-parametric and determines the number of prototypes automatically. CS generation for CJA and RSP algorithms does not depend on the order of data in TS.

Some PS algorithms aim to increase accuracy rather than reduce cost and storage requirements. These are called editing algorithms and constitute a subcategory of PS algorithms. The goal of editing is achieved by removing noisy and close-border data and leaving smoother decision boundaries. Noisy items mislead many DRTs and affect their reduction rates. Therefore, when high levels of noise exist in the data, a successful classification task implies the execution of editing before the execution of the main data reduction procedure. There exist some hybrid PS approaches, such as the DROP algorithms [30] and IB3 [1], which integrate an editing mechanism in their main reduction procedure (see [12] for details).

Edited Nearest Neighbor (ENN) rule [29] is a popular editing algorithm. It works as follows: for each training item x, if the class of x does not agree with the majority class of its k nearest neighbors, x is discarded. ENN-rule constitutes an additional costly preprocessing step. It needs to compute $\frac{N*(N-1)}{2}$ distances, i.e., all the distances among the items in TS. Other well-known and also time-consuming editing algorithms are multi-edit [9] and all-kNN [26] that are variations of ENN-rule.

A great number of PS and PA algorithms have been proposed. Here, we presented in detail only the algorithms that we used for comparison purposes in Section 4.3. PA and PS algorithms are reviewed, categorized and compared to each other in [27] and [12], respectively. Other relevant reviews can be found in [30, 4, 19]

An effective CBM has been proposed by Hwang and Cho [15]. Hwang and Cho's method (HCM) also uses k-means clustering in order to find clusters in TS. Each cluster is divided into two sets. Items located in a certain distance from the cluster centroid are placed into the "core set". The rest are placed into the "peripheral set". When a new item must be classified, it is examined whether it lies within the "core area" of the nearest cluster. In that case, it is classified by retrieving the k-nearest neighbors from this cluster. Otherwise, its nearest neighbors are retrieved from the set dynamically formed by the items of the nearest cluster and the "peripheral" items of adjacent clusters. HCM has been used in our experimental study (see Section 4.3).

The Cluster-based Tree [31] is a CBM that is based on searching in a cluster hierarchy. Karamitopoulos and Evangelidis [16] proposed a CBM for fast time series classification. Finally, Wang introduced a CBM for fast k-NN classification that prunes the search space by using the properties of the triangle inequality.

3 The Proposed Adaptive Hybrid Method

The proposed method consists of two main parts. The first part is a two level data structure (TLDS) built by a clustering preprocessing procedure. We call this procedure TLDS construction algorithm (TLDSCA). The second part is a fast hybrid classifier (FHC) that accesses TLDS to perform classification. Subsection 3.1 presents TLDSCA. Subsections 3.2 and 3.3 present two versions of the proposed classifier. In Subsection 3.4 some general comments about the proposed method are presented.

3.1 Two-level data structure construction algorithm

TLDS is built by a repeated execution of k-means clustering on the TS data of each class. In particular, for each class k-means builds a number of clusters. TLDS consists of a list of cluster centroids (i.e., the mean vectors of all clusters for all classes) together with their corresponding class that we call the first level of TLDS. Each element of this list points to a list containing the items assigned to the specific cluster centroid. We refer to the collection of these lists of items as the second level of TLDS. We will be using the term prototypes for the cluster centroids of the first level of TLDS.

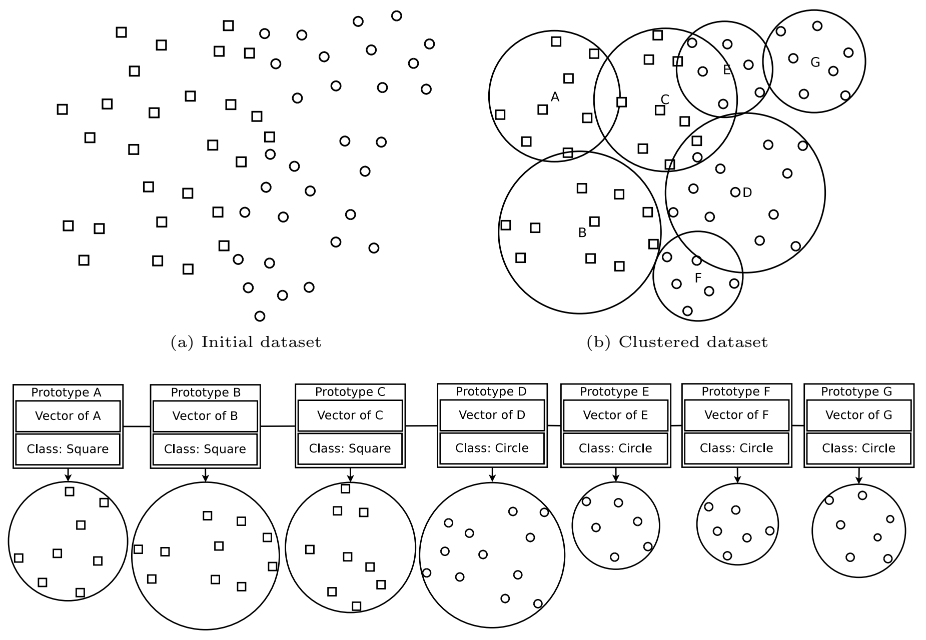
The number of clusters built is determined by the data reduction factor (DRF), that is a user-predefined parameter. For each class C, the number of clusters NC is estimated by $NC = \lceil \frac{|C|}{DRF} \rceil$, where $|C|$ is the number of items that belong to C. Therefore, DRF determines the length of TLDS. Figure 1 illustrates a two-dimensional example: There are two classes, square and circle. The initial TS contains 27 squares and 31 circles (Figure 1(a)). Thus, if $DRF = 10$, the classes Square and Circle will be represented by 3 and 4 prototypes respectively (Figure 1(b)). The result of TLDSCA will be the TLDS depicted in Figure 1(c). Class square is represented by the prototypes A–C, whereas class circle is represented by the prototypes D–G.

TLDSCA is easy to implement. Algorithm 1 accepts a TS and a DRF value and returns a TLDS. Initially, for each class, it estimates the number of clusters that will be built (lines 3–5). The algorithm continues by calling k-means. Then, the resulting clusters are added in TLDS (lines 7–11).

TLDSCA is fast because it is based on k-means clustering. Of course, the computational cost depends on how fast the clusters are consolidated. In addition, DRF also influences the cost. Typically, the higher the DRF value, the fewer and larger the clusters created, and consequently, the lower the cost involved.

Algorithm 1 TLDSCA

```
Input: $TS, DRF$, Output: $TLDS$
1: $TLDS \leftarrow$ empty list of records of the form $[class, prototype, list\_of\_items]$
2: for each class $C$ in $TS$ do
3:     $SC \leftarrow$ set of items $\in C$
4:     $NC \leftarrow \lceil \frac{|SC|}{DRF} \rceil$
5:     $NewClusters \leftarrow K$-MEANS($SC$, $NC$)
6:     for each $CL \in NewClusters$ do
7:         add in $TLDS$ element $[C, CL_{centroid}, CL_{items}]$
8:     end for
9: end for
10: return $TLDS$
```



(a) Initial dataset

(b) Clustered dataset

(c) Speed-up data structure

FIG. 1. k-means clustering on items of each class (*DRF*=10)

We should mention that the idea of creating multiple class representatives via clustering has also been proposed by Datta and Kibler in [8]. Their goal was the representation of distant and disjoint groups formed by items of the same class and the construction of a classifier capable of managing symbolic (nominal) attributes.

3.2 Fast Hybrid Classifier I

The first version of the proposed classifier (FHC-I) works by accessing either the first or the second level of TLDS. It uses three parameters that let the user define the desirable trade-off between accuracy and classification cost. The idea behind the algorithm is quite simple (see Algorithm 2). When a new item x arrives and has to be classified (line 1), FHC-I initially scans the first level of TLDS and retrieves the pk nearest prototypes to x (line 2). We call this procedure first level search. If the acceptance criterion introduced by the *npratio* parameter is met, these prototypes, through a majority vote, determine the class where x belongs to (line 4–6). Upon failure, x is classified by searching for the k "real" nearest neighbors within the clusters dynamically formed by the union of clusters indexed by the pk nearest prototypes (lines 7–10). This procedure is called second level search. Obviously, the more the items classified without the need of the second level search, the lower is the computational cost involved. Possible ties during the majority class voting of either the first or the second level search are resolved using 1-NN.

FHC-I uses parameter *npratio* to decide when to switch to a second level search.

npratio is a ratio that defines how many nearest prototypes should determine the majority class (the most common class among the *pk* nearest prototypes) in order to classify a new item (see lines 3–5). For example, suppose that the input parameters are set to be $k = 3$, $pk = 10$, and $npratio = 0.7$. Furthermore, suppose that a new item x has to be classified and a TLDS with 100 clusters is available. FHC-I, initially, examines the 10 nearest prototypes from the first level of TLDS. If seven or more of them belong to the majority class, then x is classified to this class. Otherwise, FHC-I proceeds to a second level search. The three "real" nearest neighbors are retrieved from the data subset formed by the union of clusters indexed by the ten found nearest prototypes, and they determine the class of x. Even in the case of the second level search, FHC-I avoids searching the rest 90 clusters. We note that if we define $npratio = 0$, all incoming items are classified by a first level search.

Algorithm 2 The Fast Hybrd Classifier I

```
Input: $TLDS$, $pk$, $npratio$, $k$
1: for each unclassified item $x$ do
2:   $pkprototypes \leftarrow$ Find the $pk$ nearest to $x$ prototypes $\in$ {prototypes of $TLDS$}
3:   $SMC_1 \leftarrow$ set of items $\in$ majority class $MC_1$ among $pkprototypes$
4:   if $\frac{|SMC_1|}{pk} \geq npratio$ then
5:     Classify $x$ to $MC_1$
6:   else
7:     $NNs \leftarrow$ Find the $k$ nearest neighbors to $x$ in the set formed by the union of the clusters
       indexed by the $pkprototypes$
8:     $MC_2 \leftarrow$ Find the majority class among $NNs$
9:     Classify $x$ to $MC_2$
10:  end if
11: end for
```

3.3 Fast Hybrid Classifier II

The performance of FHC-I depends on the distribution of TS items in the classes. If they are uniformly distributed, each class is represented by a similar number of prototypes in TLDS. Therefore, all unclassified items have the same probability to be classified by a second level search. In contrast, in cases of non-uniform distributions and since the value of *npratio* is the same for all classes, the probability of performing a second level search depends on which is the majority class of the first level search. Items belonging to rare classes always lose during the voting of the first level search and are classified by a second level search.

Fast hybrid classifier II (FHC-II) attempts to better manage unbalanced datasets. It considers the sizes of the classes and tries to reduce "costly" second level searches. FHC-II estimates *npratio* instead of using a pre-specified value for it. This is accomplished by using a range of *npratio* values defined by parameters $npratio_{low}$ and $npratio_{high}$. The value of *npratio* is dynamically adjusted to be between the particular range and depends on the majority class determined by the first level search.

Algorithmically, FHC-II is similar to FHC-I. However, for each class c, FHC-II counts how many TS items $|c|$ belong to c (or how many prototypes are created for c) and notes the corresponding *min* and *max* values. For each class c, FHC-II computes $npratio \in [npratio_{low}, npratio_{high}]$ as follows: $npratio = norm \times (npratio_{high} -$

$npratio_{low}) + npratio_{low}$ where $norm = \frac{|c|-min}{max-min}$. For each new item, the ratio of majority class MC_1 votes during first level search should be greater than the estimated *npratio* in order to avoid a second level search (see lines 2–5 in Algorithm 2).

For instance, suppose that a TS contains three classes, A, B and C with 3000, 2000 and 1000 items respectively. Also, suppose that $npratio_{low} = 0.5$ and $npratio_{high} = 1$. If class A is voted to be the majority class during the first level search, then $npratio = 1$ (because $norm = 1$). Namely, all np nearest prototypes must vote the majority class in order to classify the new item without the need of a second level search. Similarly, if class B is the majority class of the first level search, $npratio = 0.75$ (because $norm = 0.5$. Finally, if class C is the majority class then $npratio = 0.5$ (because $norm = 0$. That is, the value of *npratio* is adjusted for each class in the range $[npratio_{low}, npratio_{high}]$ depending on the size of the class, i.e., the smaller the class the lower the *npratio* and vice versa.

When correct prediction of "weak" (or rare) classes is critical, FHC-I should be used instead of FHC-II. FHC-II should be adopted when correct predictions for all classes have the same significance.

3.4 Discussion

Considering the proposed classifier, it is obvious that a new item that lies in a close-border area, is classified by a second level search. On the other hand, an item that lies in the "internal" area of a class, is classified by first level search. Thus, FHC is neither a CBM nor PA, since it dynamically decides on how to classify a new item. Classification via a first level search is a PA method. On the other hand, a second level search is a CBM that uses a dynamically-formed subset of the initial TS. Therefore, the method is hybrid. Of course, contrary to PS/PA algorithms and like CBMs and indexing methods, the proposed method does not reduce storage requirements.

Concerning the parameters, *pk* and *npratio* (or $npratio_{low}$ and $npratio_{high}$) should be determined by taking into account the *DRF* value that was used for TLDS construction. If accuracy is more critical than cost and a TLDS with few and large clusters is available, *pk* and *npratio* should have high values. If cost is more critical and a TLDS with many and small clusters is available, low *pk* and *npratio* values are recommended. Considering *DRF*: low *DRF* values are recommended for building accurate classifiers with high cost savings and high *DRF* values for building fast classifiers without significant accuracy loss. If our needs are not specified at the time that TLDSCA is executed, an intermediate *DRF* value is the most appropriate. In this case, the trade-off can be afterwards determined by adjusting *pk* and *pkratio*.

When FHC performs a second level search, it accesses a subset of the initial TS formed by the union of the *pk* clusters. Since each cluster contains items of a specific class, this subset does not contain items of irrelevant classes (it does not contain outliers of classes which are not represented by the *pk* prototypes) and, thus, we claim that accuracy is not affected as much by these outliers. Taking into account this property, FHC may be more accurate than the conventional-k-NN classifier (the one that uses the original TS) without the need of editing. Of course, noise removal can increase the cluster quality and the overall performance.

4 Performance Evaluation

4.1 Experimental Setup

The proposed method was coded in C and was evaluated using seven datasets distributed by the KEEL repository[2] [2] and summarized in Table 1. All datasets were used without normalization. Euclidean distance was adopted as the distance metric. We compared the performance of our method to six methods. Three PS and two PA algorithms, and one CBM. We used the methods presented in detail in Section 2. We selected the particular methods because: (i) CNN-rule [14], IB2 [1] and RSP3 [25] are popular and usually used in many papers for comparison purposes, (ii) we consider TLDSCA to be a fast algorithm, and, hence, we wanted to compare it to IB2 and RHC [21] that have been proven to be fast algorithms, and, (iii) our method, PSC [20], HCM [15] and RHC are based on k-means clustering. A comparison between them was desirable. MGT, LS and TXR datasets are distributed sorted on the class label. This affects the data order-dependent methods. We randomized these datasets.

Table 1. Dataset description

Dataset	**Size**	**Attributes**	**Classes**
Letter Recognition (LR)	20000	16	26
Magic G. Telescope (MGT)	19020	10	2
Pen-Digits (PD)	10992	16	10
Landsat Satellite (LS)	6435	36	6
Shuttle (SH)	58000	9	7
Texture (TXR)	5500	40	11
Phoneme (PH)	5404	5	2

We compare the methods by reporting three average measurements obtained via five-cross-fold validation for each one: (i) Accuracy (Acc), (ii) Classification Cost (CC), and, (iii) Preprocessing Cost (PC). Costs were estimated by counting distance computations. We used the five already constructed five pairs of training/testing sets distributed by the KEEL repository. Moreover, we wanted to evaluate all methods on noise-free data. Therefore, we ran all experiments twice, one on the original and one on the edited TS. For editing purposes, we used ENN-rule [29] by setting k=3 [30].

All methods involve a k parameter during the classification step: The DRTs execute k-NN classifier on CS. Similarly, when FHC performs a second level search, it retrieves and examines the k nearest neighbors. HCM applies k-NN classifier on the reference set. For all methods and datasets, we defined $k = 1$.

In addition, our method provides three extra parameters: DRF, pk, and $npratio$. Several experiments were conducted for the these parameters. The values tested for each one were: (i) $DRF = 2^i, i = 1, 2, \ldots 8$ (for dataset SH, $i = 3, 4, \ldots 8$), (ii) $pk = 2, 5, 7, 10, 12, \ldots, 27, 30$, and, (iii) for FHC-I, $npratio = 0.5, 0.7, 1$ (for FHC-II, see Subsection 3.3). Therefore, we built $8 \times 16 \times 3 = 384$ FHC-I classifiers. Then, we kept the most accurate FHCs for each reported classification cost (CC). In real life applications, there is no need to do such extensive tests to determine the appropriate values of the parameters. Here, our purpose was to fully understand how each parameter influences classifier construction and performance. In real life

[2]http://sci2s.ugr.es/keel/datasets.php

applications, the parameters should be determined by taking into consideration the significance of Acc and CC as well as the dataset used.

HCM also uses three parameters: C is the number of clusters, L is the number of adjacent clusters, D is the distance threshold used to define the core and peripheral items (see Section 2 or [15]). We set $C = \lfloor\sqrt{\frac{n}{2^i}}\rfloor$, i=1,...,7, where n is the number of items. Thus, for each dataset, we built 8 classifiers. The first classifier (for i=1) is based on the rule of thumb that defines $C = \lfloor\sqrt{\frac{n}{2}}\rfloor$ [17]. We decided to build classifiers that use small C values based on the observation that Hwang and Cho defined C=10 for a TS with 60919 items. Also, we set $L = \lfloor\sqrt{C}\rfloor$ as Hwang and Cho did in their experiments. Moreover, following the approach of Hwang and Cho, we considered as peripheral items, those whose distance from the cluster centroid was greater than the double average distance among the items of each cluster (i.e., D=2).

Another issue that needs attention is the number of clusters that PSC uses. In [20], since the main goal of the authors was the fast data reduction, they run experiments by constructing $r \times j$, $j = 2, 4, \cdots, 10$, clusters, where r is the number of classes. Although we are also interested in low PC, our main goal is to achieve high Acc at a low CC. So, we decided to test PSC with higher j values. We conducted several experiments with varying $j = 2^i, i = 1, 2, \ldots 9$ (for the noisy MG dataset, $i = 1, 2, \ldots, 11$).

Since only a first level search can be used to classify new items, we included its performance in the comparisons. We call this method first level search classifier (FLSC). It carries out the whole task when $npratio = 0$. Similarly to FHC-I, we kept only the most accurate FLSC, PSC and HCM classifiers for each reported cost.

4.2 Pre-processing performance

Table 2 presents PC measurements in millions of distance computations. Each cell has two PC values. The first value (or.) corresponds to PC estimated on the original (non-edited) data whereas the second value on the edited data. The first table row shows the PC needed for the execution of the editing procedure of ENN-rule. Of course, PC measurements on edited data do not include the cost of editing.

RSP3 is the most time-consuming method, since it must retrieve the pair of the farthest points in each subset. CNN-rule is faster than RSP3. IB2 and RHC seem to be the fastest approaches. PC measurements of all other methods depend on the parameter used and the size and distribution of the data in the multidimensional space. Considering the measurements of TLDSCA, we can conclude that its PC performance is satisfactory. We note that we have adopted the full cluster-consolidation as well as the random initialization of the means. Thus, TLDSCA could become even faster had we used more efficient consolidation and initialization criteria.

In most data mining tasks, preprocessing is executed only once. Hence, these measurements may not be so significant in real-life applications. However, PC is a comparison criterion and its measurements should be evaluated taking into account the performance that the corresponding classifiers achieve in terms of Acc and CC.

4.3 Classification performance

Each classifier was executed twice, once on original and once on edited data. Thus, the diagrams presented in figure 2, include the performance of each method twice.

TABLE 2. PC in millions of distance computations

Method		**LR**	**MGT**	**PD**	**LS**	**SH**	**TXR**	**PH**
ENN	or.	127.99	115.76	38.65	13.25	1076.46	9.68	9.35
CNN	or.	163.03	281.49	11.75	17.99	45.30	5.65	13.45
	ed.	112.20	68.61	9.25	6.49	26.02	3.90	5.57
IB2	or.	23.37	34.61	1.78	2.22	8.26	0.84	1.96
	ed.	18.35	8.48	1.51	0.99	6.35	0.72	0.86
RSP3	or.	326.52	511.67	86.66	37.70	17410.12	27.63	20.31
	ed.	300.51	318.82	85.16	30.64	15652.75	27.04	15.67
RHC	or.	41.84	4.08	2.88	1.69	16.83	3.63	0.66
	ed.	31.05	2.83	2.83	1.73	22.41	3.00	0.47
PSC i=1	or.	66.32	23.95	6.52	2.96	127.20	3.15	1.08
	ed.	55.13	11.44	6.73	2.86	107.47	3.35	0.68
PSC i=2	or.	110.06	17.21	15.93	5.85	54.07	7.90	0.94
	ed.	94.76	10.15	17.57	4.83	52.46	10.33	1.04
PSC i=3	or.	165.32	27.09	35.23	10.11	222.77	14.49	2.79
	ed.	138.41	12.42	32.33	9.97	189.71	11.10	2.18
PSC i=4	or.	221.07	77.15	63.33	18.42	296.88	21.07	4.80
	ed.	203.85	35.95	51.16	15.57	273.98	20.35	4.28
PSC i=5	or.	348.81	102.54	88.93	30.85	780.87	23.85	10.67
	ed.	303.11	75.18	88.32	24.46	689.34	25.39	9.26
PSC i=6	or.	431.32	157.47	123.82	42.31	1259.12	35.93	17.51
	ed.	422.79	115.30	114.03	35.96	1699.70	34.88	10.62
PSC i=7	or.	553.78	223.81	155.33	52.19	2548.15	43.37	30.78
	ed.	529.74	197.41	154.28	52.49	2516.00	42.83	20.23
PSC i=8	or.	830.67	349.12	225.12	63.26	3855.90	52.04	32.76
	ed.	713.12	234.09	205.71	54.65	3171.32	51.39	31.44
PSC i=9	or.	851.97	395.81	324.17	85.09	4479.18	38.72	54.01
	ed.	814.97	266.48	321.97	70.72	5673.10	37.77	47.16
HCM i=1	or.	88.88	111.22	28.80	12.52	744.82	8.10	9.87
	ed.	74.60	58.85	26.88	9.13	867.40	7.92	5.58
HCM i=2	or.	88.12	131.45	17.57	9.84	490.57	5.58	5.15
	ed.	80.87	49.53	17.53	7.81	557.07	5.40	4.15
HCM i=3	or.	63.66	73.69	11.27	6.34	399.23	4.80	3.70
	ed.	55.31	33.39	12.06	6.32	366.82	3.46	2.65
HCM i=4	or.	47.53	52.88	7.61	3.75	334.99	4.27	1.55
	ed.	30.23	27.84	7.12	3.32	254.83	4.58	1.78
HCM i=5	or.	26.35	27.69	5.97	3.39	105.13	3.41	1.33
	ed.	31.45	18.16	5.21	3.06	146.12	2.68	1.03
HCM i=6	or.	18.98	19.51	3.32	2.12	100.66	1.67	0.70
	ed.	25.96	12.65	2.99	1.37	106.02	1.82	0.77
HCM i=7	or.	10.89	7.50	1.70	0.94	34.78	0.52	0.74
	ed.	9.93	5.16	1.81	0.67	37.18	0.54	0.48
TLDSCA i=1	or.	19.62	404.23	18.38	10.56	-	3.50	34.69
	ed.	18.48	258.18	17.78	8.26		3.34	26.27
TLDSCA i=2	or.	15.79	291.78	14.29	9.18	-	2.74	24.65
	ed.	14.63	221.87	13.78	7.67		2.60	23.51
TLDSCA i=3	or.	11.58	252.06	10.60	6.87	3898.83	1.99	17.50
	ed.	10.13	177.57	10.78	5.92	4267.59	1.98	18.00
TLDSCA i=4	or.	7.74	192.26	7.28	4.45	2879.39	1.32	12.36
	ed.	7.01	145.16	7.21	4.51	3027.05	1.38	9.75
TLDSCA i=5	or.	4.80	159.82	4.54	3.75	1983.80	0.94	7.92
	ed.	4.25	92.64	5.06	3.07	2115.25	0.99	6.47
TLDSCA i=6	or.	2.69	105.35	2.95	1.82	1537.64	0.58	7.00
	ed.	2.37	64.63	3.29	1.79	1585.51	0.58	4.29
TLDSCA i=7	or.	1.23	54.04	1.68	1.15	855.87	0.28	3.27
	ed.	1.15	46.55	1.60	0.95	847.11	0.29	2.95
TLDSCA i=8	or.	0.60	34.55	0.69	0.47	551.25	0.08	1.21
	ed.	0.58	21.79	0.61	0.42	535.73	0.08	1.55

For each classifier, the diagrams report CC measurements on the x-axis (in terms of millions or thousands distance computations) and the corresponding Acc values on the y-axis. The closer to the "upper-left" corner of the diagram a classifier's point lies, the higher is the performance achieved. Since we want to clearly indicate classifiers of high performance, the diagrams present only a subset of performance points (points of some classifiers are omitted because they are out of the diagram range). In addition, we do not show the parameter values used to built the corresponding classifiers[3].

Table 3 shows Acc and CC results for the conventional 1-NN classifiers, i.e., classifiers that use original (1-NN (or)) or edited data (1-NN (ed)). Although editing is used to improve accuracy, ENN achieves that only in the case of MGT. This happens

[3]Tables with complete parameter values and performance measurements are available on url: `http://users.uom.gr/~stoug/IGPL_experiments.zip`

because MGT contains high levels of noise. Although, LS and PH also contain some noisy items, ENN does not improve accuracy. The rest datasets are almost noise-free.

TABLE 3. Conventional 1-NN: Acc and CC measurements

Method		**LR**	**MGT**	**PD**	**LS**	**SH**	**TXR**	**PH**
1-NN (or)	Acc.	95.83	78.14	99.35	90.60	99.82	99.02	90.10
	CC.	64.00	57.88	19.34	6.63	538.24	4.84	4.67
1-NN (ed)	Acc.	94.98	80.44	99.30	90.29	99.79	98.64	88.14
	CC.	61.23	46.26	19.21	6.02	537.24	4.78	4.14

Almost in all cases, FHC-I achieves high performance (see figure 2). In the cases of LR, MGT, PD, LS and PH, it is more accurate than 1-NN. This happens because when FHC-I performs a second level search, it accesses a TS subset that does not contain items of irrelevant classes. With the exception of SH, FHC-I achieves better performance than all DRTs. For SH, FHC-I can achieve higher Acc than all DRTs, but at a higher CC. Although FHC-I is more accurate than HCM in all datasets, in the cases of LR and SH the latter may be preferable when very fast classifiers are required. Of course, FHC-I performs better than FLSC. However, the latter achieves noteworthy performance that is comparable to the other speed-up methods.

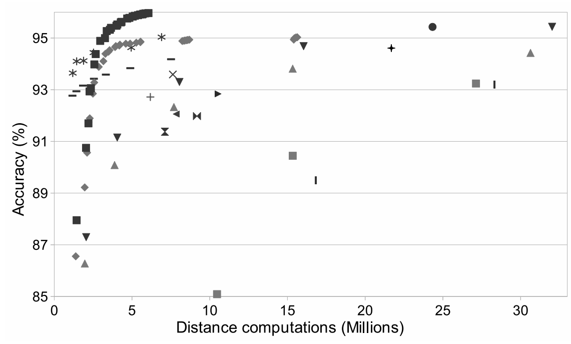
All diagrams of figure 2 show that when the speed-up methods are executed over edited data, they are faster than when they are executed over original data. Nevertheless, in some cases, either the CC gains are not very high or Acc is significantly reduced. In the case of the noisy MGT dataset, editing is necessary for all methods.

A final comment about the preprocessing and classification results is that the proposed method can perform comparable to or better than the other methods. The user can adapt the method to the application requirements by appropriately adjusting its parameters. We conclude that the proposed method can be adjusted to achieve high Acc with gains in CC or to reduce CC at a minimum level with slightly lower Acc.

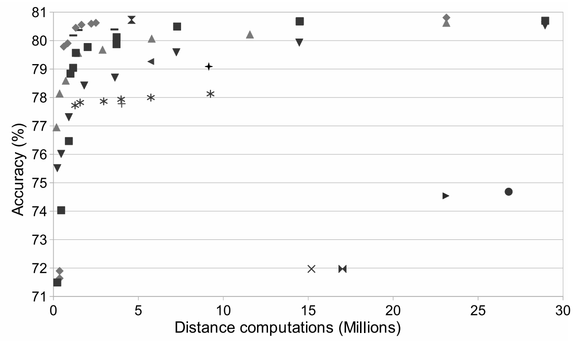
4.4 Statistical test of significance

Our experimental study is complemented with the results of a non-parametric statistical test. We ran the Wilcoxon signed ranks test [10] in order to validate the results presented in the previous subsections. The Wilcoxon test compares the speed-up methods in pairs taking into account their performance on each dataset. The test was run considering that the three comparison criteria (Acc,CC,PC) have the same significance. We ran the test on the results estimated on original and edited data . Consequently, it was run on 42 measurements (7 datasets $\times$ 3 criteria $\times$ 2 forms of each dataset). Of course, we could not include tests for all variations of parametric classifiers (FHC-I, FLSC, PSC, HCM). For each method, we selected a good representative variation for each dataset. Our criterion for choosing these representatives was relatively high Acc and low CC. PC measurements were also taken into account.

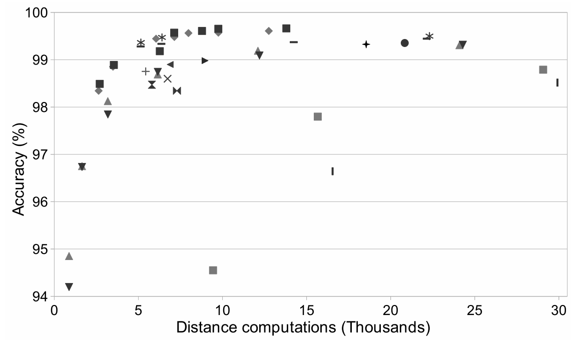
The performance of the “good” FHC-I and FLSC representatives were compared to each competitor. Note that FHC-I and FLSC imply the execution of TLDSCA during preprocessing. Table 4 presents the results of the test. Columns “w/l/t” list the number of wins/losses/ties for each pair. Column “Wilc.” lists the significance level. When that measure is lower than 0.05 (values in bold in table 4), we can claim that



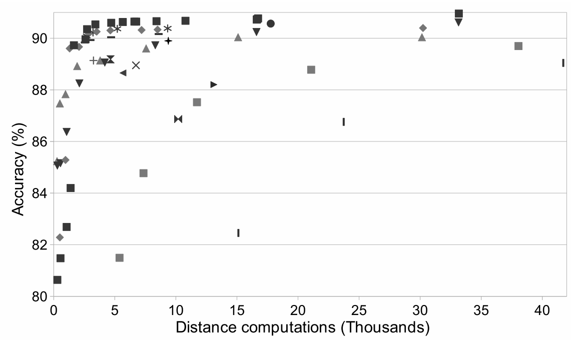
(a) LR



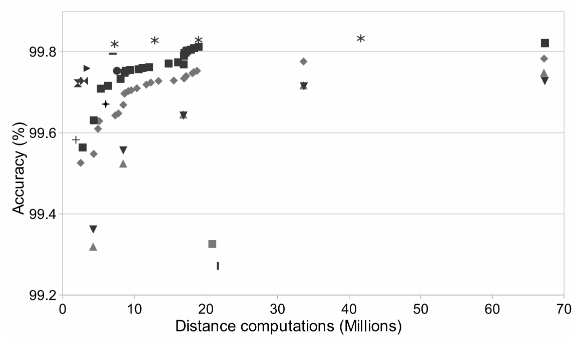
(b) MGT



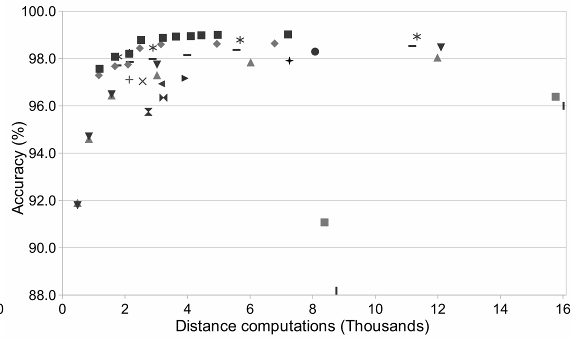
(c) PD



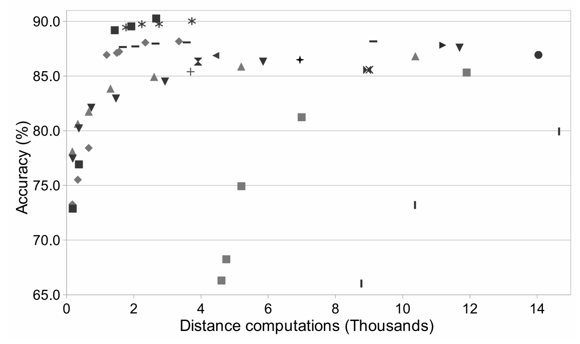
(d) LS



(e) SH



(f) TXR



(g) PH

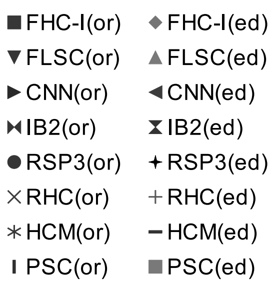


Fig. 2. Classification performance (Accuracy and Classification cost)

the difference between the two methods is statistically significant. As we expected, the test confirms that FHC-I performs better than its competitors. In all cases the significance value is lower than 0.05. The performance of FLSC is statistically better than that of RSP3 and PSC. In contrast, it is statistically worse than HCM.

TABLE 4. Wilcoxon signed ranks test

Methods	**w/l/t**	**Wilc.**	**Methods**	**w/l/t**	**Wilc.**
FHC-I vs CNN	35/7/0	**0.000**	FLSC vs CNN	28/14/0	0.103
FHC-I vs IB2	25/17/0	**0.013**	FLSC vs IB2	19/23/0	0.657
FHC-I vs PSC	42/0/0	**0.000**	FLSC vs PSC	39/3/0	**0.000**
FHC-I vs RSP3	40/2/0	**0.000**	FLSC vs RSP3	28/14/0	**0.006**
FHCI vs RHC	28/14/0	**0.011**	FLSC vs RHC	21/21/0	0.750
FHC-I vs HCM	30/12/0	**0.029**	FLSC vs HCM	12/29/1	**0.003**

4.5 FHC-II performance

Four of the eight datasets are unbalanced. LS includes six items with 626, 703, 707, 1358, 1508, 1533 items respectively. MGT has two classes with 12332 and 6688 items. Similarly, SH has seven classes with 45589, 8903, 3267, 171, 49, 13, 10 items and PH has two classes with 3818 and 1586 items. FHC-I does not manage fairly the items of all classes. FHC-II efficiently manage these datasets by reducing the probability of second level searches for the "weak" classes. This leads to even faster classifiers.

We ran FHC-II twelve times for each dataset using the following settings: (i) $pk = 15, 30$, (ii) $DRF = 16, 32$ and (iii) $(npratio_{low}, npratio_{high})$ values: $(0.7 - 1), (0.5 - 1), (0.3 - 0.7)$. These methods were compared to four FHC-I methods built by the same pk and DRF values and $npratio = 1$. Figure 3 presents the results. Each FHC-II method is noted with the following sequence: $DRF, pk, npratio_{low}, npratio_{high}$ in the figure's legend. Similarly, FHC-I is noted with $DRF, pk, npratio$.

In the case of the noisy MGT, FHC-II improve both Acc and CC measurements. In the case of the LS dataset, all FHC-II classifiers built using the $(0.3 - 0.7)$ range of $npratio$ values as well as the one that uses settings $DRF = 16$, $pk = 15$, $npratio_{low} = 0.5$, $npratio_{high} = 1$ are ineffective. They reduce costs, but they also reduce accuracy. All other FHC-II classifiers execute faster than the FHC-I classifiers without loss on accuracy. In the cases of SH and PH, FHC-I may be preferable to FHC-II. The later executes slightly faster than FHC-I. However this speed-up affects the accuracy.

5 Conclusions

We proposed an adaptive hybrid method for fast k-NN classification. The method involves the construction of a two level data structure and classifiers that make predictions using either the first or the second level of this structure. Actually, the method combines the idea of DRTs with that of CBMs in a hybrid schema. The method lets the user determine the trade-off between accuracy and cost. Thus, it can be used either to improve accuracy at a lower cost, or to reduce cost at a minimum level without sacrificing accuracy. Experiments showed that cost improvements may be achieved, with the accuracy remaining high and comparable to that of the conventional k-NN.

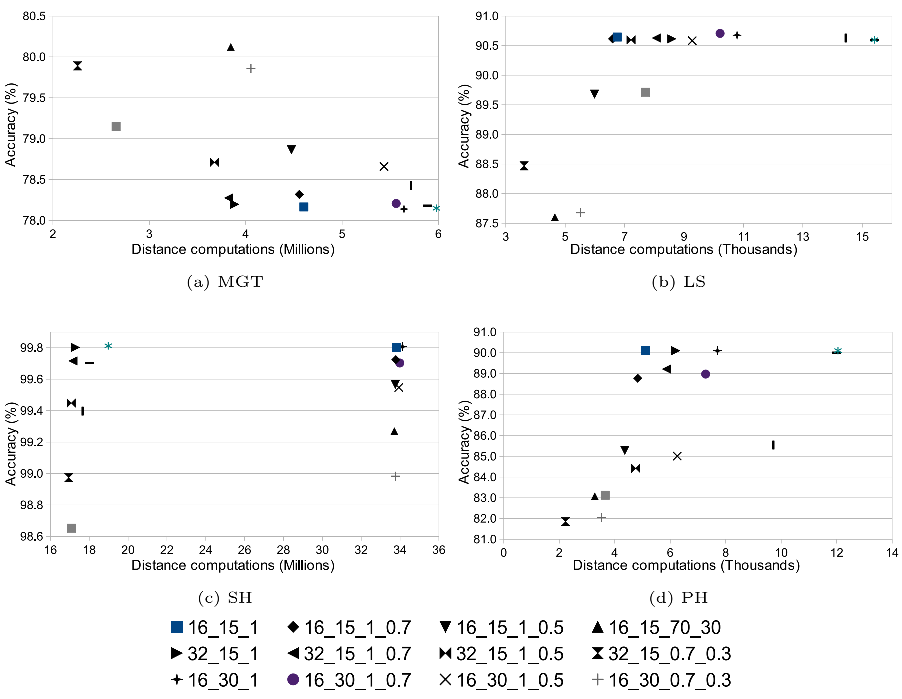


FIG. 3. FHC-I vs FHC-II

References

[1] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Mach. Learn.*, 6(1):37–66, Jan. 1991.

[2] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, and S. García. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2011.

[3] F. Angiulli. Fast condensed nearest neighbor rule. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 25–32, New York, NY, USA, 2005. ACM.

[4] H. Brighton and C. Mellish. Advances in instance selection for instance-based learning algorithms. *Data Min. Knowl. Discov.*, 6(2):153–172, Apr. 2002.

[5] C. H. Chen and A. Jóźwik. A sample set condensation algorithm for the class sensitive artificial neural network. *Pattern Recogn. Lett.*, 17:819–823, July 1996.

[6] C.-H. Chou, B.-H. Kuo, and F. Chang. The generalized condensed nearest neighbor rule as a data reduction method. In *Proceedings of the 18th International Conference on Pattern Recognition - Volume 02*, ICPR '06, pages 556–559, Washington, DC, USA, 2006. IEEE Computer Society.

[7] B. V. Dasarathy. *Nearest neighbor (NN) norms : NN pattern classification techniques.* IEEE Computer Society Press, 1991.

[8] P. Datta and D. F. Kibler. Learning symbolic prototypes. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 75–82, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[9] P. A. deVijVer and J. Kittler. On the edited nearest neighbor rule. In *Proceedings of the Fifth International Conference on Pattern Recognition.* The Institute of Electrical and Electronics Engineers, 1980.

[10] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, Dec. 2006.

[11] V. S. Devi and M. N. Murty. An incremental prototype set building technique. *Pattern Recognition*, 35(2):505–513, 2002.

[12] S. Garcia, J. Derrac, J. Cano, and F. Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3):417–435, Mar. 2012.

[13] G. W. Gates. The reduced nearest neighbor rule. ieee transactions on information theory. *IEEE Transactions on Information Theory*, 18(3):431–433, 1972.

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

[15] S. Hwang and S. Cho. Clustering-based reference set reduction for k-nearest neighbor. In *4th international symposium on Neural Networks: Part II–Advances in Neural Networks*, ISNN '07, pages 880–888. Springer, 2007.

[16] L. Karamitopoulos and G. Evangelidis. Cluster-based similarity search in time series. In *Proceedings of the Fourth Balkan Conference in Informatics*, BCI '09, pages 113–118, Washington, DC, USA, 2009. IEEE Computer Society.

[17] K. Mardia, J. Kent, and J. Bibby. *Multivariate Analysis.* Academic Press, 1979.

[18] J. McQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of 5th Berkeley Symp. on Math. Statistics and Probability*, pages 281– 298, Berkeley, CA : University of California Press, 1967.

[19] J. A. Olvera-López, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, and J. Kittler. A review of instance selection methods. *Artif. Intell. Rev.*, 34(2):133–143, Aug. 2010.

[20] J. A. Olvera-Lopez, J. A. Carrasco-Ochoa, and J. F. M. Trinidad. A new fast prototype selection method based on clustering. *Pattern Anal. Appl.*, 13(2):131–141, 2010.

[21] S. Ougiaroglou and G. Evangelidis. Efficient dataset size reduction by finding homogeneous clusters. In *Proceedings of the Fifth Balkan Conference in Informatics*, BCI '12, pages 168–173, New York, NY, USA, 2012. ACM.

[22] S. Ougiaroglou, G. Evangelidis, and D. A. Dervos. An adaptive hybrid and cluster-based model for speeding up the k-nn classifier. In *Proceedings of the 7th international conference on Hybrid Artificial Intelligent Systems - Volume Part II*, HAIS'12, pages 163–175, Berlin, Heidelberg, 2012. Springer-Verlag.

[23] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Trans. on Inf. Theory*, 21(6):665–669, 1975.

[24] H. Samet. *Foundations of multidimensional and metric data structures.* The Morgan Kaufmann series in computer graphics. Elsevier/Morgan Kaufmann, 2006.

[25] J. S. Sánchez. High training set size reduction by space partitioning and prototype abstraction. *Pattern Recognition*, 37(7):1561–1564, 2004.

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

[27] I. Triguero, J. Derrac, and S. G. andFrancisco Herrera. A taxonomy and experimental study on prototype generation for nearest neighbor classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 42(1):86–100, 2012.

[28] X. Wang. A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality. In *The 2011 International Joint Conference on Neural Networks (IJCNN)*, pages 1293 –1299, August 2011.

[29] D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE trans. on systems, man, and cybernetics*, 2(3):408–421, July 1972.

[30] D. R. Wilson and T. R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286, 2000.

[31] B. Zhang and S. N. Srihari. Fast k-nearest neighbor classification using cluster-based trees. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(4):525–528, 2004.