Pattern Recognition 43 (2010) 873–886

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Fast *k* most similar neighbor classifier for mixed data (tree *k-MSN*)

Selene Hernández-Rodríguez*, J. Fco Martínez-Trinidad, J. Ariel Carrasco-Ochoa

Computer Science Department, National Institute of Astrophysics, Optics and Electronics, Luis Enrique Erro No. 1, Sta. María Tonantzintla, Puebla, CP 72840, Mexico

ARTICLE INFO

Article history:
Received 14 July 2008
Received in revised form 9 July 2009
Accepted 6 August 2009

Keywords:
Nearest neighbor rule
Fast *k* nearest neighbor search
Mixed data
Non-metric comparison functions

ABSTRACT

The *k* nearest neighbor (*k-NN*) classifier has been a widely used nonparametric technique in Pattern Recognition, because of its simplicity and good performance. In order to decide the class of a new prototype, the *k-NN* classifier performs an exhaustive comparison between the prototype to classify and the prototypes in the training set *T*. However, when *T* is large, the exhaustive comparison is expensive. For this reason, many fast *k-NN* classifiers have been developed, some of them are based on a tree structure, which is created during a preprocessing phase using the prototypes in *T*. Then, in a search phase, the tree is traversed to find the nearest neighbor. The speed up is obtained, while the exploration of some parts of the tree is avoided using pruning rules which are usually based on the triangle inequality. However, in soft sciences as Medicine, Geology, Sociology, etc., the prototypes are usually described by numerical and categorical attributes (mixed data), and sometimes the comparison function for computing the similarity between prototypes does not satisfy metric properties. Therefore, in this work an approximate fast *k* most similar neighbor classifier, for mixed data and similarity functions that do not satisfy metric properties, based on a tree structure (Tree *k-MSN*) is proposed. Some experiments with synthetic and real data are presented.

1. Introduction

The *k-NN* [1] rule has been a widely used nonparametric technique in Pattern Recognition, because of its simplicity and good performance. In order to decide the class of a new prototype, the *k-NN* classifier performs an exhaustive comparison between the prototype to classify and the prototypes in the training set *T*. However, in some applications, the exhaustive comparison between the new prototype to classify and the elements in the training set becomes impractical. Many fast *k-NN* classifiers have been designed to avoid this problem. In some papers, different reviews of these fast *k-NN* classifiers are provided, for example, in [2–4].

The objective of the fast *k-NN* classifiers is to reduce the number of comparisons trying to keep the original classification accuracy obtained by *k-NN*. Speeding up the *k-NN* classifiers is required because some applications demand a rapid response on large datasets, for example online stock analysis, air traffic control, network traffic management, intrusion detection, etc. Also, fast *k-NN* classifiers are useful for those problems with high dimensionality where the comparison function could be very expensive [5,6], under this context, reducing the number of comparisons could be very important. For these reasons, although nowadays the computers are very fast, the development of fast *k-NN* classifiers is currently an active research area [7,8]. Nevertheless, most of the fast *k-NN* classifiers proposed in the literature have been designed for numerical prototype descriptions compared through a metric function. Thus, most of these methods use metric properties to reduce the number of comparisons.

However, in many real-world applications, prototypes are described by both numerical and categorical variables (mixed data). In these cases, sometimes the comparison function does not satisfy metric properties. For this reason, we cannot use most of the fast *k-NN* classifiers proposed for numerical prototype descriptions. There are two possible directions to work with non-metric similarity functions, which are:

1. Working on the prototype representation level, before the use of a classifier. In this approach, the main idea is to work with the dissimilarity matrix, which stores the relation of similarity between prototypes in *T*. Some directions to work with this prototype representation are:
   - *Correcting the dissimilarity matrix to make it Euclidean* [9]: A non-Euclidean dissimilarity matrix can be corrected to become Euclidean, if the comparison function is symmetric.
   - *Embeddings for dissimilarities* [9–12]: The objective of an embedding is to transform the dissimilarity matrix into a Euclidean space, such that the discrepancy between the original dissimilarities and the estimated distances is preserved,

* Corresponding author.
E-mail addresses: selehdez@inaoep.mx (S. Hernández-Rodríguez), fmartine@inaoep.mx, ariel@inaoep.mx (J.A. Carrasco-Ochoa).

0031-3203/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2009.08.014

in order to use favorable properties of the Euclidean space (which is topologic, inner product, normed and metric).
- *Using the dissimilarity representation*: This approach uses the dissimilarities to represent prototypes [9,13]. Thus, for a certain prototype, the dissimilarities against prototypes in R (where R is a small subset from T) are used as feature vector, instead of the features of the prototype in its original representation.

2. The second direction is to work with the original prototype representation space and developing algorithms suitable to work with mixed data and non-metric similarity functions [45].

Our work addresses the second direction, where if a metric is not available but a comparison function that evaluates the similarity between a pair of prototypes can be defined, given a new prototype to classify, the objective is to find the k most similar neighbors (*k-MSN*) and use them for classifying the new prototype. In this way, a *k-MSN* classifier uses a training set (T) of N prototypes, where each prototype is described by d attributes, which can be numerical or non numerical. Given a new prototype Q to classify, the classifier finds its k most similar neighbors in T (k *MSN*'s) according to a comparison function and assigns to Q a class, based on the k most similar neighbors. However, the exhaustive search of the *k-MSN*, as occurs with the *k-NN*, could be very expensive if T is large. Therefore, the aim of this paper is to present an approximate fast *k-MSN* classifier for mixed data based on a tree structure, which does not assume the comparison function satisfies metric properties. Our classifier, for avoiding the exhaustive search of the *k-MSN* will not use special properties of the comparison function. It only will assume that the comparison function reaches its maximum when a prototype is compared against itself, in this way, our classifier can be applied on different applications with different comparison functions.

This paper is organized as follows: Section 2 provides a brief review of fast *k-NN* classifiers. In Section 3, our fast *k-MSN* classifier is introduced. In Section 4, experimental results obtained using our classifier and a comparison against other fast classifiers is reported. Finally, in Section 5 we present our conclusions and future work.

2. Related work

Nowadays, with the current computer technology it is possible to store large amounts of information. In order to apply the *k-NN* classifier on applications where the training set is large, different approaches have been developed. Some approaches, like condensation or edition [14], are used to reduce the size of the training set, before applying the *k-NN* classifier. Another approach, which is followed in this paper, is developing fast *k-NN* classifiers. In the literature, different divisions of the fast *k-NN* classifiers have been proposed. Following [4] fast *k-NN* classifiers can be divided as follows:

1. *Tree-based*: In this case, the training data space is partitioned into "regions" and usually a tree structure is used for recording and indexing these regions. Given a new prototype, its *NN* is found using the tree. Some of these classifiers are: kd-tree [15], R-tree [16], SS-tree [17], SR-tree [18], pyramid [19] and Voronoi cell [20].
2. *Elimination-based*: These kinds of fast classifiers are based on pruning rules derived from the triangle inequality to avoid comparisons among prototypes. In some cases, also a tree structure is used [4,21–23] and in some other cases, projection based methods [15,24,25], are used.
3. *Approximate NN search*: Another research direction is to find an approximate *NN*, instead of the exact one [26–28].
4. *Application-dependant*: For example, in [29] an approach, applicable when the triangle inequality may not hold but it is possible to impose a geometric structure and dimensionality on the space, is proposed. Also, in [2] an algorithm applicable on high dimensional data, such as images, is presented. In [30] an algorithm based on the transformation of correlated data is introduced.

One of the first and most studied fast *k-NN* classifier was proposed by Fukunaga and Narendra [21]. In the Fukunaga and Narendra's (FN) classifier, a tree is created by decomposing the training set into C subsets, using the *C-Means* algorithm. In this case, each subset represents a node of the tree, which is divided again to construct the tree. In FN classifier a fixed size tree, with three levels and $C=3$, was used. Each node p of the tree contains four features, which are: the set of prototypes in the node $p(S_p)$, the number of prototypes in $p(N_p)$, the center of the node (M_p) and finally the maximum distance (R_p) between M_p and the prototypes in S_p. Given a new prototype Q to classify, FN fast classifier searches the *NN* based on a branch and bound method to traverse the tree. Two pruning rules are used to decide whether or not a node or a prototype of the tree is evaluated. These rules are based on the triangle inequality. The first pruning rule for nodes of the tree is

$$B+R_p<D(Q,M_p) \quad (1)$$

where B is the distance between Q and the current *NN* and D is the distance function. In this case, the nodes satisfying condition (1) cannot contain a prototype closer to Q than the current *NN* and therefore, those nodes are eliminated. The second pruning rule is applied to the prototypes that belong to a leaf node of the tree, in order to decide whether or not to compute the distance from Q to the prototypes of the node. The pruning rule for each prototype $O_i \in S_p$ is

$$B+D(O_i,M_p)<D(Q,M_p) \quad (2)$$

The prototypes that satisfy condition (2) cannot be closer to Q than the current *NN* and therefore, the distance to Q is not computed. The search process finishes when all nodes in the tree have been evaluated or eliminated by the first pruning rule. Finally, the class of the *NN* is assigned to Q. An extension to *k-NN* is also proposed in [21], where, in the search process, B is the distance between Q and the current *k-NN* instead of the current 1-*NN*. In this case, the majority class of its k nearest neighbors is assigned to Q.

In the last years, some improvements on FN classifier have been developed in two ways, the first one is the evaluation of different clustering algorithms in the tree construction and the second one, is the improvement of the pruning rules for making a faster FN classifier.

In [31] an improvement on the tree construction algorithm is proposed, building a binary tree where the leaves represent only one prototype. In the search phase, FN pruning rules are used. Also, in [32,33] other clustering algorithms are evaluated to construct the tree, but using the same FN search algorithm.

In [33] two improvements on the FN pruning rule, based on the information of sibling nodes, are also proposed. In this case, if the distance between the center of a node and the nearest prototype of the sibling node is too big, the sibling node can be pruned without comparing its center against Q (the new prototype to classify); this is called Sibling-Based pruning Rule (SBR). The second improvement is a rule that combines iteratively FN pruning rule and SBR, which is called Generalized pruning Rule (GR). According to their experiments, Gómez-Ballester's (GB) classifier obtained best results using GR rule.

In [34] an improvement on the FN pruning rule is presented (ONC classifier), where the pruning rule is applied without comparing Q against the center of p, instead of it, the distance between the current nearest neighbor of Q and the node p (which is previously computed and stored, in a preprocessing phase), is used to decide if p can be safely pruned.

The improvements mentioned before, are exact methods because they always find the same *k-NN* that would be found using the exhaustive search. However, finding the *k NN*'s (even using a fast method) is a slow process for some tasks, therefore in some other works [26,27], fast approximate *k-NN* classifiers, based on tree structures, have been proposed. An approximate classifier does not guarantee to find the *k-NN*, instead an approximation is obtained. In [27] the first FN pruning rule is modified in order to finish the search when the current *NN* is not too far from the exact *NN*, as follows:

$$(1+e)(D(Q,M_p)-R_p)>B \quad (3)$$

where e is an error margin that allows decreasing the number of comparisons. However, Moreno-Seco's (MS) classifier also relies on metric properties to avoid comparisons.

All the improvements of FN classifier were designed to work with numerical data when the prototype comparison function satisfies metric properties, in particular the triangle inequality. However, in [45] a fast *k-NN* classifier (Cluster tree) proposed to work with dissimilarity functions was introduced. Cluster tree creates a tree structure, similar to the structure used by Fukunaga and Narendra, but following a bottom-up approach instead of top-down. The tree construction algorithm uses the similarities to build the bottom level. To build the next levels of the tree (until the root is reached) the mean and the standard deviation of the dissimilarities between the nodes at the current level are used. In the classification stage of Cluster tree, one or more paths (determined by a parameter ς) are followed, from the root to the leaf level of the tree. The k nearest neighbors found in the reached leaves, are reported as the result.

From the related work about fast *k-NN* classifiers, we can notice that most of the work is focused on numerical data applications, where metric properties (of the prototype comparison function) are used to reduce the number of comparisons. However, some comparison functions for mixed data do not satisfy metric properties. Therefore, in this work, a fast approximate *k-MSN* classifier for mixed data based on a tree structure (Tree *k*-MSN), which can use any function for comparing prototypes, is proposed.

3. Proposed classifier

The proposed algorithm, Tree *k*-MSN, consists of two phases. The first one, or preprocessing phase, is the construction of a tree structure from the training set *T*, using strategies suitable for mixed data. In the second phase, two search algorithms, which are independent of metric properties of the comparison function, are proposed.

3.1. Preprocessing phase

In this phase, the training set is hierarchically decomposed to create a tree structure. Thus, the root of the tree contains the whole training set *T*. In order to create the following levels of the tree, each node n of the tree is divided in C clusters, in such a way that each cluster represents a descendant node of n. Each descendant node is divided again and this process is repeated until a stop criterion is satisfied.

Due to, in our algorithm, mixed data is allowed, instead of using the *CMeans* algorithm for building the tree structure, as in FN classifier, the *C-Means with Similarity Functions* algorithm (*CMSF*) [35,36], is used. Among the clustering algorithms for mixed data analysis, *CMSF* is the unique algorithm that allows creating C clusters, computing as representative of each cluster one prototype belonging to the cluster (i.e., a prototype contained in *T*); besides the *CMSF* allows using any similarity function.

Each node p of the tree contains three features which are: S_p the set of prototypes that belong to p; N_p the number of prototypes in p and unlike FN and MS classifiers, Rep_p a representative prototype

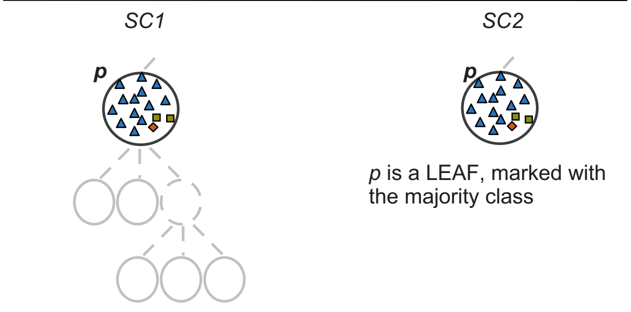

Fig. 1. Stop criterion 2(SC2).

of the node, which is the most similar on average to the rest of prototypes in the node. The algorithm to construct the tree structure is as follows:

```
1. $CurrentNode =$ root of the tree
2. $NodesToDivide = \{CurrentNode\}$
3. $p = 1$
4. while $|NodesToDivide| \neq 0$
5.   $CurrentNode = NodesToDivide[1]$
6.   $[PrototypesInCluster, RepresentativeOfCluster,$
       $SizeCluster] = CMSF(CurrentNode, C)$
7.     for cluster $i = 1$ to $C$      **C is the number of
       clusters
8.       $node_p =$ new descendant node of $CurrentNode$
9.       $node_p.S_p = PrototypesInCluster(i)$
10.      $node_p.Rep_p = RepresentativeOfCluster(i)$
11.      $node_p.N_p = SizeCluster(i)$
12.      if $Stop\ Criterion =$ true then      **SC1,
           SC2or SC3**
13.          $node_p$ is a leaf
14.        else $NodesToDivide = NodesToDivide \cup \{node_p\}$
15.      $p = p+1$
16.    end for
17.    $NodesToDivide = NodesToDivide$
18.end while
```

A node is marked as a leaf when a stop criterion is satisfied. In this work we used a stop criterion based on the node size (*SC*1), which is used in [21,23,31–33,37,38] and introduce two new stop criteria (*SC*2 and *SC*3), which take into account not only the number of prototypes of the node, but also the class distribution of these prototypes. The three stop criteria are the following:

1. *Stop criterion* 1 (*SC*1): This criterion is based on the node size. In this case, if the number of prototypes contained in a node is less than a predefined threshold ($N_p \le NoP$), then the node is considered as a leaf.
   Following this criterion, the set of prototypes contained in a node is divided, in order to obtain nodes with few prototypes, which implies that the prototypes are more similar among themselves. However, when most of the prototypes in a node of the tree belong to the same class, then it probably implies that the prototypes of the node are very similar among themselves even if the set is not small enough to be a leaf. For this reason, we propose a second stop criterion.
2. *Stop criterion* 2 (*SC*2): In this case, if most of the prototypes in a node belong to the same class, then the node is considered as a leaf and it is marked with the majority class, even if the set is not small enough according to the first stop criterion (see Fig. 1). In order to decide how many prototypes in the node must belong to

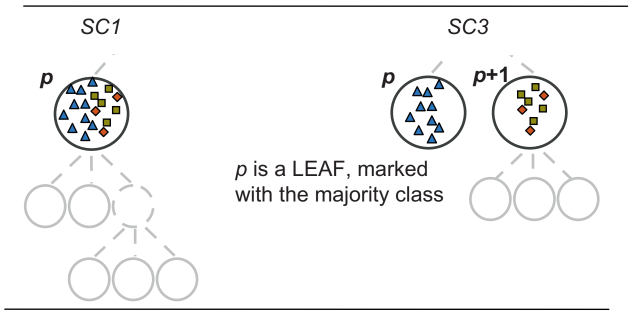

Fig. 2. Stop criterion 3(SC3).

the same class, for generalizing the class of a node, a percentage threshold (*PercThres*) is used.
In the nodes where this criterion is not satisfied, only the size of the node is considered to create leaf nodes (*SC*1).
When the node is generalized by the majority class, through *SC*2, if *PercThres* = 100%, it means that all prototypes in the node belong to the same class (the generalized class of the node). However, when *PercThres* < 100%, an error is introduced, because some prototypes in the node do not belong to the majority class. Therefore, we introduce a third criterion.

3. *Stop criterion* 3 (*SC*3): In this case, if certain percentage (*PercThres*) of the prototypes in a node belongs to the same class, two nodes are created (see Fig. 2). Using the prototypes that belong to the majority class, a leaf node is created and it is marked with the majority class. The rest of the prototypes are assigned to a second node. In the second node, the size is considered to decide if the node is a leaf (if $N_p \leq NoP$) or if the node will be divided again.
In the nodes where *SC*3 criterion is not satisfied, only the size of the node is considered to create leaf nodes (*SC*1).

3.2. Classification phase

In this phase, in order to avoid the exhaustive tree traversal, the existing fast *k-NN* classifiers rely on pruning rules (based on metric properties). As we are looking for a method applicable when the comparison function does not satisfy metric properties, pruning rules based on the triangular inequality cannot be used; therefore, we propose to stop the search when a leaf of the tree is reached. In the first search algorithm (local search), we propose to use a depth-first search strategy and in the second search algorithm (global search), we propose to use a best-first search strategy. The proposed algorithms for searching the *k-MSN* are described below:

Local search: It begins at the root of the tree, following the path of the most similar node and finishes when a leaf is reached. As each node of the tree is represented by a prototype of the training set, with known class, a list of the *k-MSN* is stored and updated during the tree traversal. When the first leaf node *l* is reached, if *l* is marked with the majority class, then only the representative prototype Rep_l is considered to update the *k-MSN* (because most of the prototypes in the node belong to the same class). If the node is not marked with the majority class, then a local exhaustive search in the node is done and the list of *k-MSN* is updated. After a leaf is processed, if the list of *k-MSN* does not have *k* elements, then the tree traversal makes backtracking to explore nodes closer to *Q*, in order to find *k MSN*'s. When the list is completed, the search stops. The algorithm of the *local search*, to classify prototype *Q*, is as follows:

```
1. CurrentNode = root of the tree
2. while Current Node ≠ Leaf
3.    DescNodes = {node_p: node_p is a descendant node of
        CurrentNode}
4.    MostSimilarNode = argmin(Sim(Q,Rep_p), ∀ node_p
        ∈ DescNodes
5.    CurrentNode = MostSimilarNode
6.    kMSN_List = UpdateList_kMSN(kMSN_List, Rep_p)
7. end while
8. If CurrentNode.MarkedMajorityClass = true
9.    kMSN_List = UpdateList_kMSN(kMSN_List,
        Rep_CurrentNode)
10.else
11.   Sim(Q,prototype_q), ∀ prototype_p ∈ CurrentNode
12.   kMSN_List = UpdateList_kMSN(kMSN_List,
        prototype_q)
13.   while |kMSN| < k
14.      [CL] = Backtracking(find the closer leaf
         to CurrentNode)
15.      CurrentNode = CL
16.   end while
17.end if-else
18.Class(Q) = majority class of k-MSN_List
```

Global search: It begins at the root of the tree, comparing *Q* against the descendant nodes of the root, which are added to a list (*List_tree_traversal*). After that, *List_tree_traversal* is sorted in such a way that the most similar node to *Q* is in the first place. The most similar node (first element) is eliminated from *List_tree_traversal* and its descendant nodes are compared against *Q*, and added to *List_tree_traversal*, which is sorted again. The search finishes when the first element of *List_tree_traversal* is a leaf. In this search, it is possible to reconsider nodes in levels of the tree already traversed if the first node of *List_tree_traversal* belongs to a previous level in the tree. The algorithm of the *global search*, to classify prototype *Q*, is as follows:

```
1. CurrentNode = root of the tree
2. List_tree_traversal = Empty
3. while Current Node ≠ Leaf
4.    Current Node = List_tree_traversal [1]
5.    List_tree_traversal = List_tree_traversal
6.    DescNodes = {node_p: node_p is a descendant node
        of CurrentNode}
7.    List_tree_traversal = List_tree_traversal ∪
        DescNodes
8.    Compute the similarity between Q and
        the nodes in DesNodes
9.    Order List_tree_traversal in such a way that
        the most similar prototype to Q is the first
        element of List_tree_traversal
10.   CurrentNode = List_tree_traversal [1]
11.   kMSN_List = UpdateList_kMSN(kMSN_List, Rep_p)
12.end while
13.if CurrentNode.MarkedMajorityClass = true
14.   kMSN_List = UpdateList_kMSN(kMSN_List,
        Rep_CurrentNode)
15.else
16.   Sim(Q,prototype_q), ∀ prototype_q ∈ CurrentNode
17.   kMSN_List = UpdateList_kMSN(kMSN_List,
        prototype_q)
18.   while |kMSN| < k
19.      [CL] = Backtracking(find the closer leaf to
         CurrentNode)
20.      CurrentNode = CL
21.   end while
22.end if-else
23.Class(Q) = majority class of k-MSN_List
```

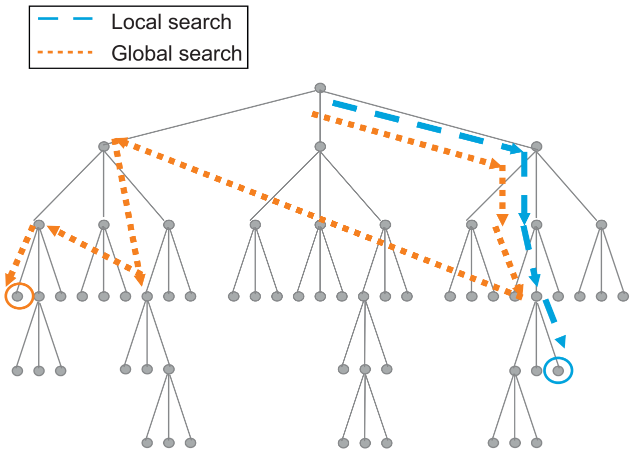

Fig. 3. Example of the search algorithms.

During the tree traversal, another list (*List_k-MSN*) containing the *k* current *MSN* is stored and updated. After a leaf is processed (in a similar way than in the local search), if *List_k-MSN* does not contain *k* elements (*MSN*), then the first element in *List_tree_traversal* is considered to follow a new route. The process stops when *List_k-MSN* contains *k* elements (*MSN*). However, using both search strategies (global and local), in practical problems where the training set is large, it is quite difficult that *List_k-MSN* does not have *k* elements (*MSN*) when the first leaf is processed.

After finding *k-MSN*, the majority class is assigned to the new sample *Q*.

In Fig. 3 the difference between both search algorithms is shown. As we can see, global search allows evaluating nodes in already traversed levels.

4. Experimental results

In this section, we report the evaluation of the fast approximate *k-MSN* classifier proposed in this work (Tree *k-MSN*). In order to compare the proposed classifier, the following tree based classifiers were considered: FN classifier [21], GB classifier using GR pruning rule [33], ONC classifier [34], MS classifier [27] and Cluster tree [45].

To compare FN, GB, ONC and MS classifiers with our proposed classifier (Tree *k-MSN*), we adapted these classifiers. The adaptation consisted in the use of the same tree structure proposed in Section 3.1 and the same function, suitable to work with mixed data, instead of a distance function. In this way, only the search algorithms are compared. Besides, since GB tree traversal search algorithm was proposed for a binary tree, in our tree GR pruning rule is applied to all of the *C*-1 sibling nodes. When a leaf node is reached, as it could contain more than one prototype, a local exhaustive comparison is performed to find the *k-MSN*.

Cluster tree was implemented and tested using the same comparison functions (*D, HVDM* and *HOEM*) to work with mixed data (during the preprocessing and classification stages), and the following parameters: $\alpha=2$ and $\varsigma=2$, as suggested by the authors.

There are other classifiers based in tree structures (for example: [4,23]). However, it is not possible to adapt these classifiers to work with mixed data and similarity functions, because some techniques, such as PCA, which are only applicable to numeric data, are involved.

Thus, in the experimentation, the next *k-MSN* classifiers were compared:

1. The exhaustive *k-MSN* classifier (using a dissimilarity function).
2. Adapted FN classifier.
3. Adapted GB classifier.
4. Adapted ONC classifier.
5. Adapted MS classifier.
6. Cluster tree.
7. Tree *k-MSN*, using local search.
8. Tree *k-MSN*, using global search.

In this work, the comparison functions, used for the experiments, were *Heterogeneous Value Difference Metric*, *HVDM* [39], *D*[1] [35] and *Heterogeneous Overlap-Euclidean Metric, HOEM* [40] functions. These three functions were selected because they allow comparing mixed data. In particular, *D* and *HVDM* comparison functions are not metric functions because they do not satisfy the triangle inequality property. Tough *HOEM* function satisfies all metric properties, using this function the lower average classification accuracy was obtained in the experiments. For the functions *F*, *HVDM* and *HOEM* the most similar neighbor is the one that minimizes the comparison function.

In order to compare the different classifiers, the accuracy (*Acc*) and the percentage of comparisons between prototypes (*Comp*), were considered. The accuracy was computed as follows:

$$Acc=\frac{No\ Correct\ prototypes*100}{No\ Test\ Prototypes} \tag{4}$$

where *NoCorrectPrototypes* is the number of correctly classified prototypes in the test set and *NoTestPrototypes* is the size of the test set. The percentage of comparisons between prototypes was computed as follows:

$$Comp=\frac{No\ Comp\ Fast\ Classifier*100}{No\ Training\ prototypes} \tag{5}$$

where *NoCompFastClassifier* is the number of comparisons done by the fast *k-NN* classifier, and *NoTrainingPrototypes* is the size of the training set. According to (5), for the exhaustive classifier, the 100% of the comparisons is done.

In the experiments, 17 datasets from the *UCI* repository [41] and 4 synthetic databases were considered. In Table 1, the description of the used datasets is shown.

In all the experiments ten-fold-cross-validation was used. According to this technique, the dataset is divided in ten partitions; nine of them are used for training and the last partition is used as testing set. This process is repeated ten times, in such a way that each partition is used once as testing set.

Before comparing the different classifiers, some experiments to determine the value of the following parameters, were carried out:

1. The parameter *e* used in MS classifier.
2. The parameter *C* of the *CMSF* algorithm in the tree construction.
3. The stop criteria in the tree construction.

The datasets used to make the experiments of Sections 4.1–4.3, were: Hepatitis, Flag, Echocardiogram, Hayes, Soybean large, Bridges, Glass, Iris and Wine. To evaluate the similarity between prototypes the *HVDM* function was used.

After setting the values of the required parameters, the classifiers are compared over synthetic and real datasets, using different values of *k* (for *k-MSN*) and three different comparison functions for mixed data; *D*, *HVDM* and *HOEM*.

[1] The name of this function (*D*) is not an acronym. *D* function is based on similar attributes.

Table 1
Datasets used in this section.

Dataset	No. of prototypes	No. of numerical features	No. of non numerical features	Classes	% Missing data
Hepatitis	155	6	13	2	5.38
Credit	690	6	9	2	5
Zoo	101	1	16	7	0
Flag	194	3	25	8	0
Echocardiogram	132	9	2	2	7.6
All-hyper	2800	6	22	4	0
Ann-thyroid	7200	6	15	3	0
Thyroid0387	9172	7	22	8	0
KDD	30 000	32	4	2	0
Tic tac	958	0	9	2	0
Hayes	132	0	4	3	0
Soybean-large	307	0	35	19	6.4
Bridges	108	0	11	7	10.3
Mushroom	8124	0	22	2	1.38
Glass	214	9	0	7	0
Iris	150	4	0	3	0
Wine	178	13	0	3	0
Phoneme	5401	5	0	2	0
Synthetic database	10 000	2	0	3	0
Synthetic database	20 000	2	0	3	0
Synthetic database	30 000	2	0	3	0
Synthetic database	40 000	2	0	3	0

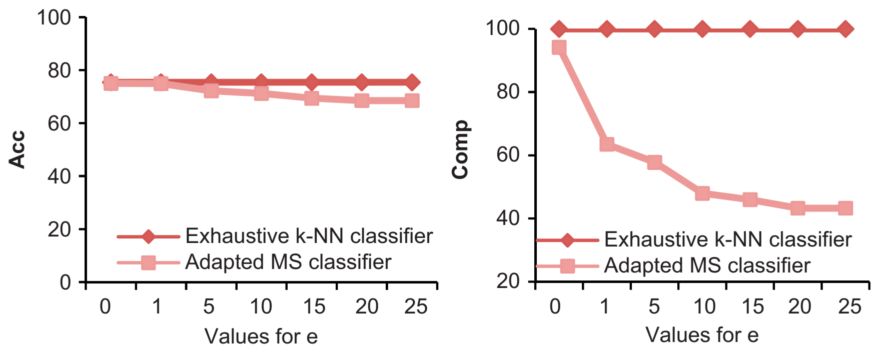

Fig. 4. Accuracy (Acc) and percentage of comparisons (Comp) obtained, using MS classifier with different values of *e*.

4.1. The parameter e used in MS classifier

Before using adapted MS classifier, some tests with different values of the error margin e ($e=0$, 1, 5, 10, 15 and 20) were done. The average classification accuracy (Acc) according to different values of e and the percentage number of comparisons (Comp) between prototypes are depicted in Fig. 4. In all of these experiments, while the error margin e grows, the number of comparisons between prototypes (Comp) decreases. In the next experiments, $e=20$ was used, because using this value the performance (Acc and Comp) of MS classifier is stabilized.

4.2. The parameter C of the CMSF algorithm used during the tree construction

In order to evaluate the performance of the compared classifiers taking into account the size of the tree, some experiments for selecting the value of C, were performed. The parameter C of the *CMSF* algorithm corresponds to the number of children of the nodes in the tree. In Figs. 5 and 6, the classification accuracy and the percentage of comparisons between prototypes for each value of C are respectively depicted. According to these experiments, the accuracy does not vary too much with the different values of C. However, the number of comparisons between prototypes increases for the adapted FN classifier when C grows. In the next experiments $C=3$ was used, because there is not a big variation of the accuracy and the number of prototypes comparisons is reduced for adapted FN and MS classifiers.

4.3. Stop criteria during the tree construction

Another important parameter in the tree construction algorithm is the stop criterion. Using the different stop criteria, described in Section 3.1, the performance of the classifiers was evaluated. In Figs. 7 and 8, the obtained classification accuracy and the percentage of comparisons among prototypes, done by the different classifiers, are shown.

Using the stop criterion based on the node size (*SC*1), the best accuracy, for all the classifiers, was obtained with $NoP=20$, in addition with this value the percentage of comparisons was reduced for all the classifiers.

Using the stop criterion 2 (*SC*2), the accuracy grows when the percentage threshold (*PercThres*) of prototypes belonging to the same class, is increased and the percentage of comparisons does not vary so much (see Figs. 7 and 8).

Using the stop criterion 3 (*SC*3), the accuracy was better than using *SC*2. However, a few more comparisons were done using *SC*3 than using *SC*2.

When the percentage threshold (*PercThres*) is 100, using *SC2* and *SC3* the obtained accuracy and the comparisons percentage are the

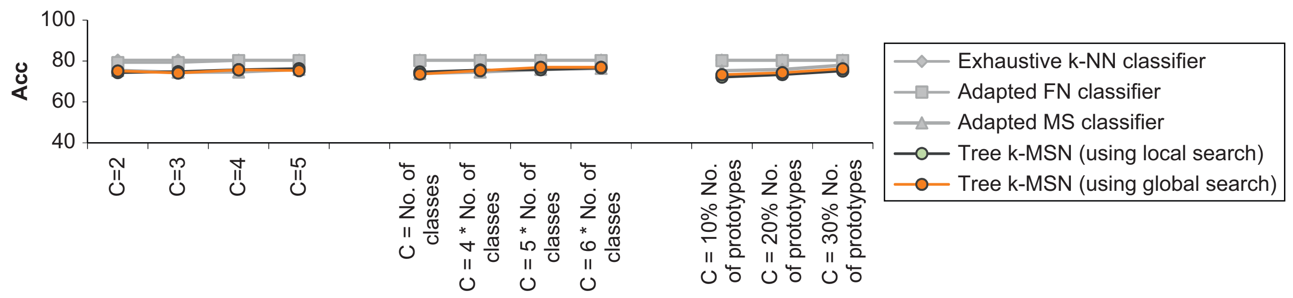

Fig. 5. Accuracy for different values of C when $C \in [2, 5]$, when C is related to the number of classes in each database and when C is related to the number of prototypes in each database.

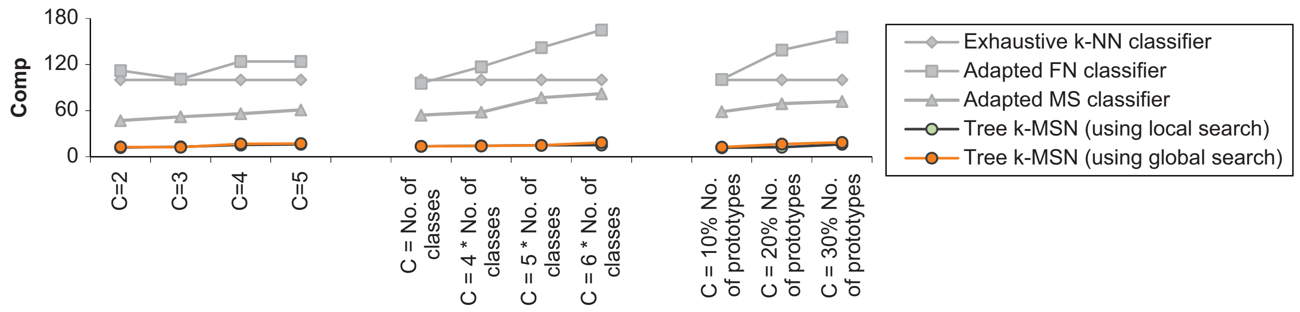

Fig. 6. Percentage of comparisons between prototypes for different values of C when $C \in [2, 5]$, when C is related to the number of classes in each database and when C is related to the number of prototypes in each database.

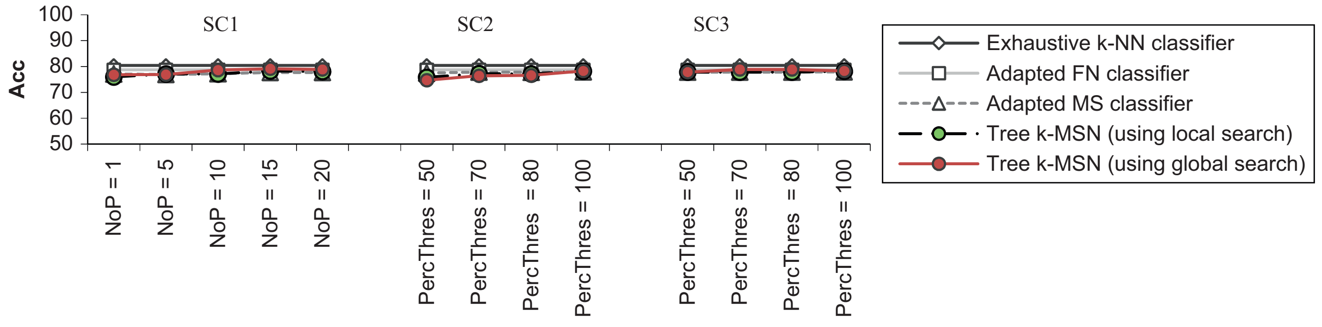

Fig. 7. Accuracy obtained using different stop criteria.

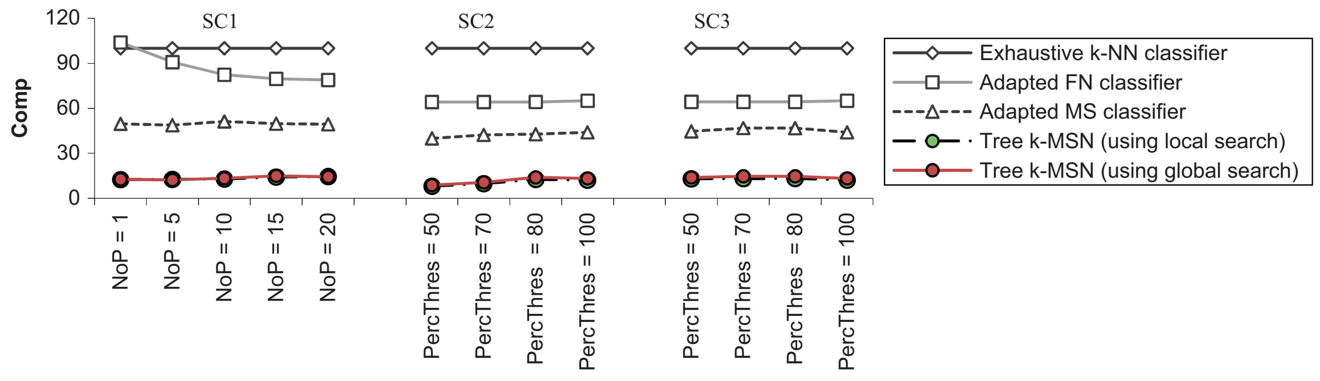

Fig. 8. Percentage of comparisons among prototypes obtained using different stop criteria.

Table 2
Condensed/edited method (GCNN) against the fast *k-MSN* classifier proposed in this work (Tree *k-MSN*).

No. of prototypes	Exhaustive *k-NN*		GCNN		Tree *k-MSN*			
	Acc	Comp	Acc	Comp	Using local search		Using global search	
					Acc	Comp	Acc	Comp
Flag	53.21	100	43.78	44.41	52.21	13.20	52.71	17.07
Tic tac	90.60	100	58.33	5.05	85.92	3.06	84.87	3.19
Soybean	90.54	100	72.53	46.18	85.26	9.72	84.28	13.10
Credit	80.90	100	77.10	5.70	79.40	4.41	77.13	4.24
Zoo	97.00	100	95.55	26.17	96.00	19.68	96.00	16.65
Echo	82.69	100	93.39	22.67	79.62	16.50	80.33	17.65
Bridge	63.36	100	68.20	88.20	60.36	15.80	57.45	17.65
Glasses	68.18	100	69.61	61.62	67.73	12.91	67.25	16.54
Iris	94.67	100	96.00	38.00	92.67	15.66	92.67	17.65
Wine	95.46	100	94.44	78.89	91.57	13.80	92.12	17.00
Average	81.66	100	76.89	41.69	79.07	12.47	78.48	14.07

Table 3
Classifiers evaluation using synthetic databases.

No. of prototypes	Exhaustive *k-NN* classifier		Adapted *k-NN* classifiers								Proposed classifier: Tree *k-MSN*			
	Acc	Comp	FN		GB		ONC		MS		Using local search		Using global search	
			Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp
10 000	95.60	100	95.60	0.96	95.60	0.97	95.60	0.96	95.64	0.92	95.58	0.27	95.58	0.45
20 000	95.60	100	95.60	0.96	95.60	0.91	95.60	0.90	95.64	0.92	95.53	0.37	95.58	0.55
30 000	99.63	100	99.63	0.47	99.63	0.36	99.63	0.34	97.06	0.46	99.64	0.198	97.12	0.30
40 000	100	100	100	0.21	100	0.16	100	0.17	100	0.21	99.92	0.09	99.87	0.14
Average	97.7	100	97.7	0.65	97.71	0.6	97.71	0.59	97.09	0.63	97.66	0.23	97.04	0.36

same, because in both cases, a leaf is marked with the majority class only when all of the prototypes in the node belong to the same class.

Therefore, in the next experiments, *SC*3 with *PercThres* = 100 (which is the same as using *SC2*, with *PercThres* = 100) and *NoP* = 20 (if the leaf is not marked by the majority class, then there are 20 prototypes, at most, in this leaf), were used.

The values of the parameters used in the following experiments are:

- The error margin $e = 20$ (MS classifier).
- The parameter of the *CMSF* algorithm C100 = 3 (Tree construction).
- The stop criterion *SC*3, with *NoP* = 20 and *PercThres* = 100% (Tree construction).

4.4. Experiments with condensed/edited methods

Since the family of condensed/edited methods are used to reduce the computational burden (by means of pruning the training set) this approach was considered to make comparisons against the fast *k-MSN* classifier proposed in this work. To make these experiments one of the most recent methods was selected: *Generalized Condensed Nearest Neighbor Rule* GCNN [14]. In Table 2 the results of these experiments are reported.

From Table 2, it is possible to observe that the classifier proposed (Tree *k-MSN*, using local and global search) did a smaller number of comparison between prototypes than GCNN. In this case, Tree *k-MSN* did the 12.47% of comparisons on average (with a higher classification accuracy) while GCNN required the 41.69% of comparisons. Besides, these experiments were carried out using some small datasets, since the computational cost of condensed/edited methods is high for medium and large datasets. For these reasons, condensed/edited methods are not included in the next experiments.

4.5. Experiments with tree-based fast k-NN classifiers using synthetic databases

In order to make some experiments with large synthetic datasets, datasets with 10 000, 20 000, 30 000 and 40 000 random samples (with two features) were created from a Gaussian distribution, with three different mean vectors and variances (three classes). In the first class, the mean vector was $M = (4, 5)$ and the variance for both attributes was 8, in the second class, the mean vector was $M = (10, 15)$ and the variance for both attributes was 6, and finally, in the third class, the mean vector was $M = (15, 10)$ and the variance for both attributes was 8.

In Table 3, the accuracy obtained (Acc) and the percentage of comparisons between prototypes (Comp) are shown. The number of comparisons performed by exhaustive search is considered as the 100% of comparisons. From these experiments, we can notice that the proposed classifier (using local and global search) is competitive in accuracy against the adapted FN, GB, ONC and GB classifiers (all the different classifiers achieve similar classification accuracy). However, the proposed classifier (Tree *k-MSN*, using local and global search) reduced the percentage of comparisons, for example, with 10 000 prototypes instead of 96 comparisons (done by FN classifier), only 27 were done (using local search).

Table 4
Comparison of the different classifiers using *HVDM* function, with $k = 1$, 3 and 5 *MSN*.

Datasets	Exhaustive *k-NN* classifier		Adapted fast *k-NN* classifier								Cluster tree [45]		Proposed classifier: Tree *k-MSN*			
	Acc	Comp	FN		GB		ONC		MS		Acc	Comp	Using local search		Using global search	
			Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp			Acc	Comp	Acc	Comp
$k = 1$																
Hepatitis	81.75	100	81.13	118.99	81.13	87.50	81.13	71.89	81.08	95.86	77.88	42.01	83.71	9.54*	83.71	15.65
Credit	80.90	100	80.90	56.32	80.10	52.78	80.35	51.65	80.10	54.95	71.10	16.70	79.40	4.41*	77.13	4.24
Zoo	97.00	100	97.00	24.68	97.00	23.46	97.00	22.16	97.00	21.86	95.00	41.63	96.00	19.68*	96.00	16.65
Flag	53.21	100	53.71	56.97	53.71	53.10	53.18	46.52	54.26	43.43	48.47	44.09	52.21	13.20*	52.71	17.07
Echocard.	82.69	100	82.69	121.01	82.69	84.01	82.69	72.82	84.18	93.43	83.24	41.75	79.62	16.50*	80.33	17.65
All-hyper	97.90	100	97.90	40.92	97.21	38.65	97.90	37.54	97.38	38.65	90.00	46.67	97.50	4.41*	96.80	4.24
Ann-thyroid	95.00	100	95.00	21.59	94.90	20.65	95.00	20.12	95.00	21.03	90.25	6.09	92.00	1.22*	91.25	1.36
Thyroid0387	87.8*	100	86.47*	18.95	86.74*	14.54	86.31*	13.95	86.26*	17.70	80.04	12.40	79.60	0.52*	84.70	0.59
KDD	99.93	100	99.93	13.92	99.86	12.95	99.63	12.08	99.61	13.08	99.42	12.45	99.56	0.61*	99.63	0.61
Tic tac toe	90.60	100	90.60	9.91	90.41	8.10	88.63	7.56	89.78	9.63	85.30	11.91	85.92	3.06*	84.87	3.19
Hayes	84.29	100	84.29	28.31	84.29	21.14	84.29	16.59	83.57	27.45	67.31	27.45	83.52	18.19*	81.26	19.65
Soyb. large	90.54	100	91.18	26.56	91.18	19.90	91.18	16.85	89.88	25.02	83.33	20.22	85.26	9.72*	84.28	13.10
Bridges	63.36	100	64.27	93.65	64.27	53.34	65.18	50.65	63.27	54.25	40.27	36.82	60.36	15.80*	57.45	17.65
Mushrooms	100	100	100.00	18.16	99.95	16.85	100.00	15.54	100.00	16.11	91.90	48.09	98.74	3.06*	98.10	3.19
Glass	68.18	100	68.18	35.96	68.18	27.04	68.18	20.16	67.71	34.36	60.30	34.64	67.73	12.91*	67.25	16.54
Iris	94.67	100	94.67	21.12	94.67	19.87	94.67	18.21	95.33	19.37	86.67	37.41	92.67	15.66*	92.67	17.65
Wine	95.46	100	95.46	43.78	95.46	32.66	95.46	27.88	94.35	34.31	93.24	45.85	91.57	13.80*	92.12	17.00
Phoneme	90.27	100	90.27	4.89	90.27	4.10	90.27	3.43	90.27	4.79	90.10	15.63	87.82	1.42*	87.18	1.84
Average	86.31	100	86.31	41.98	86.22	32.81	86.17	29.20	86.06	34.74	79.66	30.10	84.07	9.10	83.75	10.44
$k = 3$																
Hepatitis	80.58	100	80.50	118.99	80.50	87.50	81.13	71.89	81.75	95.86	77.83	42.01	79.83	9.54*	79.17	15.65
Credit	80.03	100	80.03	56.32	80.03	52.78	80.03	51.65	80.1	54.95	75.40	16.70	76.96	4.41*	77.98	4.24
Zoo	94.00	100	89.00	24.68	89.00	23.46	89.00	22.16	89.00	21.86	95.00	41.63	91.00	19.68*	90.00	16.65
Flag	54.13	100	52.13	56.97	52.63	53.10	53.68	46.52	52.13	43.43	47.92	44.09	51.11	13.20*	52.66	17.07
Echocard.	84.95	100	87.20	121.01	87.20	84.01	87.97	72.82	85.71	93.43	81.76	41.75	87.91	16.50*	87.86	17.65
All-hyper	97.9	100	97.3	40.92	97.3	38.65	97.9	37.54	97.70	38.65	92.50	46.67	97.3	4.41*	97.3	4.24
Ann-thyroid	95	100	94.5	21.59	93.61	20.65	94.85	20.12	93.54	21.03	91.28	6.09	92.9	1.22*	92.84	1.36
Thyroid0387	87.8	100	86.62	18.95	86.84	14.54	87.8	13.95	86.75	17.70	80.00	12.40	86.91	0.52*	84.7	0.59
KDD	99.93	100	99.33	13.92	99.55	12.95	99.93	12.08	99.55	13.08	99.31	12.45	99	0.61*	99.38	0.61
Tic tac toe	90.6	100	90.6	9.91	90.6	8.10	90.6	7.56	89.75	9.63	81.10	11.91	87.52	3.06*	87.12	3.19
Hayes	82.75*	100	82.58*	28.31	82.58*	21.14	81.81*	16.59	81.87*	27.45	61.92	27.45	76.65	18.19*	74.40	19.65
Soyb. large	88.23*	100	85.28	26.56	85.28	19.90	84.63	16.85	84.62	25.02	80.38	20.22	80.68	9.72*	79.71	13.10
Bridges	62.45	100	66.27	93.65	66.27	53.34	64.45	50.65	65.36	54.25	36.64	36.82	59.55	15.80*	58.64	17.65
Mushrooms	100	100	100	18.16	100	16.85	100	15.54	100	16.11	91.90	48.09	100	3.06*	100	3.19
Glass	71.06	100	71.06	35.96	71.06	27.04	71.06	20.16	66.82	34.36	56.10	34.64	65.84	12.91*	66.80	16.54
Iris	93.33	100	93.33	21.12	93.33	19.87	93.33	18.21	92.67	19.37	86.67	37.41	90.00	15.66*	90.00	17.65
Wine	94.35	100	94.35	43.78	94.35	32.66	94.35	27.88	95.52	34.31	93.79	45.85	91.60	13.80*	92.71	17.00
Phoneme	89.6	100	89.6	4.89	89.6	4.10	89.6	3.43	89.6	4.79	88.12	15.63	85.89	1.42*	85.4	1.84
Average	85.93	100	85.54	41.98	85.54	32.81	85.67	29.20	85.14	34.74	78.76	30.10	83.37	9.10	83.15	10.44
$k = 5$																
Hepatitis	82.58	100	83.71	118.99	83.71	87.50	83.04	71.89	82.38	95.86	79.08	42.01	79.17	9.54*	79.83	15.65
Credit	80.03	100	80.03	56.32	80.03	52.78	80.03	51.65	80.03	54.95	78.30	16.70	76.96	4.41*	77.98	4.24
Zoo	93.00	100	89.00	24.68	89.00	23.46	89.00	22.16	89.00	21.86	93.00	41.63	89.00	19.68*	88.00	16.65
Flag	59.32	100	56.21	56.97	56.21	53.10	55.74	46.52	56.76	43.43	48.92	44.09	50.53	13.20*	51.58	17.07
Echocard.	85.00	100	87.14	121.01	87.14	84.01	87.14	72.82	87.20	93.43	81.76	41.75	86.43	16.50*	86.37	17.65
All-hyper	80.03	100	80.03	40.92	80.03	38.65	80.03	37.54	79.64	38.65	92.50	46.67	76.96	4.41*	77.98	4.24
Ann-thyroid	97.9	100	97.3	21.59	97.3	20.65	97.9	20.12	97.9	21.03	91.19	6.09	97.3	1.22*	97.3	4.17
Thyroid0387	95	100	94.5	18.95	93.61	14.54	95	13.95	95	17.70	79.40	12.40	92.9	0.52*	92.84	0.59
KDD	87.8	100	86.62	13.92	86.84	12.95	86.94	12.08	85.641	13.08	84.45	12.45	86.91	0.61*	84.7	0.61
Tic tac toe	99.93	100	99.33	9.91	99.55	8.10	99.93	7.56	99.93	9.63	81.10	11.91	99	3.06*	99.38	3.19
Hayes	83.52*	100	74.34	28.31	74.34	21.14	72.86	16.59	73.63	27.45	65.71	27.45	73.63	18.19*	72.86	19.65
Soyb. large	85.95	100	83.33	26.56	83.33	19.90	83.33	16.85	82.02	25.02	80.04	20.22	79.73	9.72*	78.76	13.10
Bridges	62.55	100	64.55	93.65	64.55	53.34	64.55	50.65	62.64	54.25	36.64	36.82	60.55	15.80*	58.64	17.65
Mushrooms	80.03	100	80.03	18.16	80.03	16.85	80.03	15.54	100	16.11	91.90	48.09	76.96	3.06*	77.98	3.19
Glass	64.03	100	64.03	35.96	64.03	27.04	64.03	20.16	63.92	34.36	53.79	34.64	64.94	12.91*	65.89	16.54
Iris	94.00	100	94.00	21.12	94.00	19.87	94.00	18.21	92.00	19.37	85.33	37.41	88.00	15.66*	88.00	17.65
Wine	96.63*	100	96.63*	43.78	96.63*	32.66	96.63*	27.88	94.41	34.31	92.68	45.85	89.87	13.80*	90.98	17.00
Phoneme	89.18	100	89.18	4.89	89.18	4.10	89.18	3.43	89.18	4.79	87.45	15.63	85.00	1.42*	84.42	1.84
Average	84.25	100	83.33	41.98	83.31	32.81	83.30	29.20	83.96	34.74	77.96	30.10	80.77	9.10	80.75	10.59

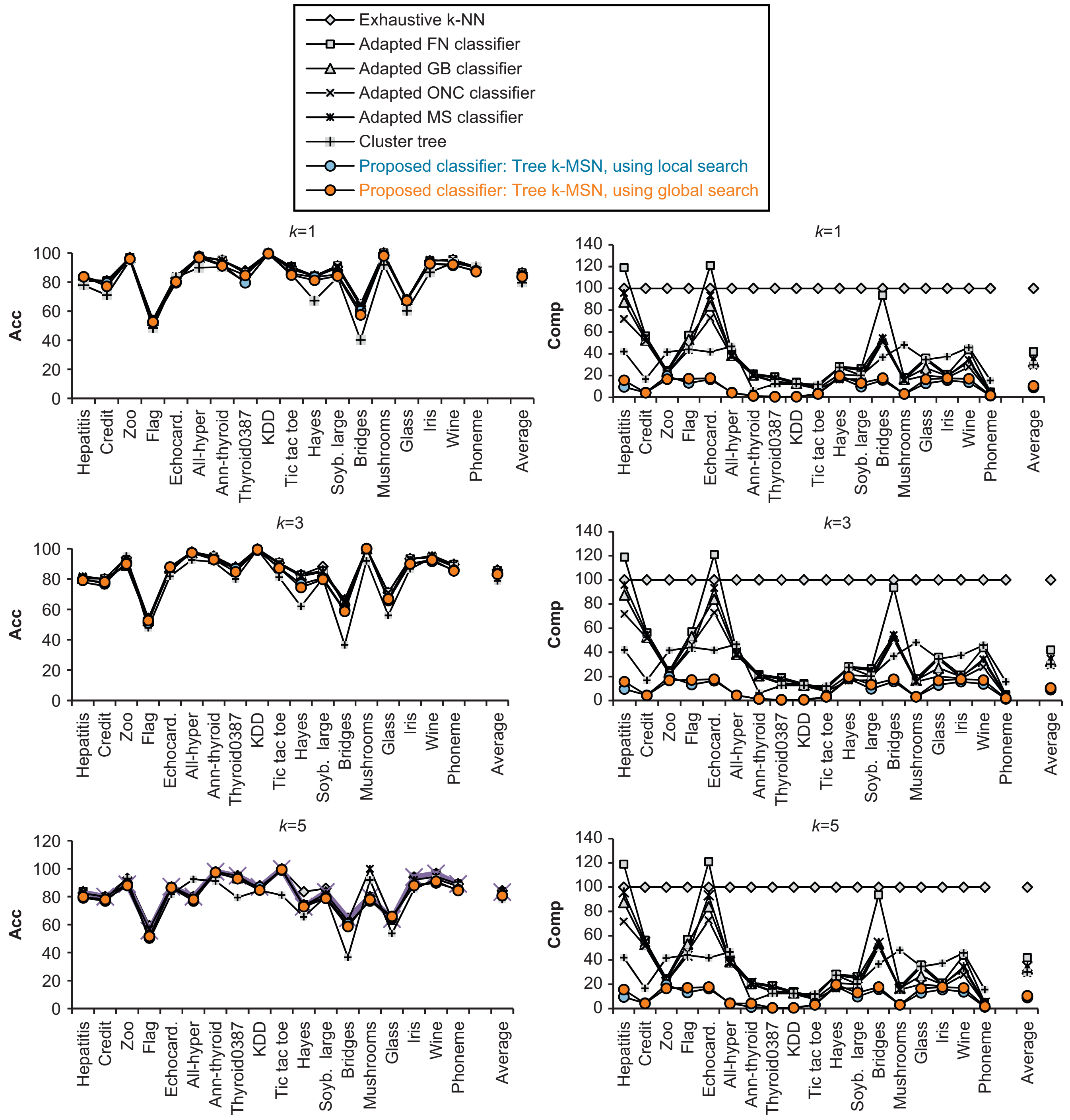

Fig. 9. Evaluation of the different classifiers using *HVDM* function, with $k = 1$, 3 and 5 *MSN*.

4.6. Experiments with tree-based fast k-NN classifiers using real datasets

In order to make experiments with real datasets, the classifiers were evaluated using 18 datasets form the UCI repository. In Table 4, the obtained accuracy (**Acc**) and the percentage of comparisons between prototypes (**Comp**), using different number of *k NN*'s (*MSN*'s), are shown.

The original FN, GB and ONC classifiers are exact approaches because they use the triangle inequality property of the distance function. However, as the *HVDM* function does not necessarily satisfies this property, adapted classifiers (using the *HVDM* function) become inexact. Using the adapted FN classifier, the average accuracy does not decrease on average, with $k = 1$. However, the average number of comparisons between prototypes (see Table 4, in the obtained results with $k = 1$) is only reduced from 100% to 41.98%. The advantage of our proposal is that the average percentage of comparisons is reduced (9.10% with local search and 10.44% with global search) more than using Cluster tree (30.95%), MS (34.74%), ONC (29.20%) and GB (32.81%) classifiers and moreover the classification accuracy obtained by Tree *k-MSN* (84.07% using local search) is higher than the obtained by Cluster tree (79.66%). It is important to notice that,

Table 5
General averages using *HVDM*, *HOEM* and *D* functions, with $k = 1$, 3 and 5 *MSN*.

No. of *k MSN*	Exhaustive *k-MSN* classifier		Adapted *k-NN* classifiers								Cluster tree		Proposed classifier: Tree *k-MSN*			
	Acc	Comp	FN		GB		ONC		MS		Acc	Comp	Using local search		Using global search	
			Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp			Acc	Comp		
HVDM																
$k = 1$	86.31	100	86.31	41.98	86.22	32.81	86.17	29.20	86.06	34.74	79.66	30.10	84.07	9.10	83.75	10.44
$k = 3$	85.93	100	85.54	41.98	85.54	32.81	85.67	29.20	85.14	34.74	78.76	30.10	83.37	9.10	83.15	10.44
$k = 5$	84.25	100	83.33	41.98	83.31	32.81	83.30	29.20	83.96	34.74	77.96	30.10	80.77	9.10	80.75	10.59
HOEM																
$k = 1$	83.98	100	83.98	43.65	83.98	31.65	83.98	30.85	82.54	33.85	76.65	28.75	80.92	9.06	80.59	10.75
$k = 3$	83.01	100	83.01	43.65	83.01	31.65	83.01	30.85	82.27	33.85	76.52	28.75	80.05	9.06	78.99	10.75
$k = 5$	82.89	100	82.89	43.65	82.89	31.65	82.89	30.85	81.27	33.85	75.95	28.75	79.46	9.06	78.46	10.75
D																
$k = 1$	85.97	100	85.39	45.37	85.36	34.09	85.37	30.25	84.38	34.57	78.64	27.86	81.99	8.67	79.94	11.37
$k = 3$	84.42	100	84.36	45.37	84.36	34.09	84.36	30.25	83.72	34.57	79.52	27.86	81.53	8.67	79.72	11.37
$k = 5$	84.02	100	83.87	45.37	83.81	34.09	83.82	30.25	83.68	34.57	78.62	27.86	80.96	8.78	79.68	11.39

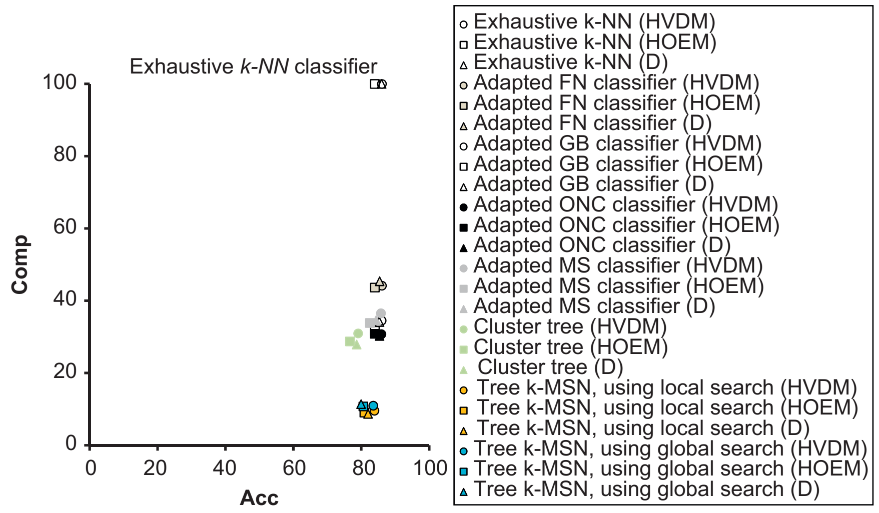

Fig. 10. Accuracy against the comparisons percentage using the different classifiers and the three different comparison functions, with $k = 1$.

although FN, ONC and GB obtained a slightly higher classification accuracy than Tree *k-MSN*, these classifiers needed to be adapted (using our tree structure) in order to allow them to work on mixed data, since the original algorithms cannot be applied under these circumstances.

Besides, in Tables 4 the symbol "*" next to the classification accuracy (Acc) and the number of prototype comparisons (Comp) indicates that there is a statistically significant difference with respect to the proposed classifier (Tree *k-MSN*, using local search), according to the *k-fold cross-validated paired t test* [42], with 9 degrees of freedom and a confidence of 95%. To make the 10-*fold cross-validated paired t test*, the classification accuracy and the percentage of prototype comparisons, obtained by each trial of the 10-*fold-cross-validation*, are considered. From these experiments, we can observe that the difference in classification accuracy between the proposed classifier and FN, ONC, GB, MS and Cluster tree, is not statistically significant for most of the datasets, while the prototype comparison reduction reached by Tree *k-MSN* (using local search) is statistically significant for all datasets.

The fast *k-NN* classifiers are proposed for problems where the set *T* is large; such is the case of the datasets: All-hyper, Ann-thyroid and Mushrooms. For these databases, our proposed classifier (Tree *k-MSN*, using local and global search) obtained a big reduction on the average number of comparisons between prototypes (for the different values of *k*). When large datasets were considered, the average classification accuracy was higher (see Table 4, which includes large datasets, and Fig. 5, where only small datasets were considered).

In Fig. 9, the accuracy obtained by the different fast classifiers listed before over each database (for $k = 1$, 3, 5 in *k-MSN*) and a general average (last point) are shown; here we can see that the performance of the five classifiers is very similar. The number of comparisons between prototypes per database is also depicted. From this figure, it is possible to see that the number of comparisons using adapted FN classifier sometimes is even higher than the exhaustive search. With adapted MS, ONC and GB classifiers, the comparisons are always less than using adapted FN classifier, but the proposed classifier (Tree *k-MSN*, using local and global search) always did fewer comparisons than adapted FN, ONC, MS and GB classifiers, for the different values of *k*.

The experiments with different values of *k* (*k-MSN*) were repeated using *D* and *HOEM* functions. As we can see from Table 5, the results of the different classifiers were similar for the three prototype comparisons functions. Using *HOEM* function, FN and GB classifiers are exact approaches, it happens because *HOEM* function is a metric.

Table 6
Average runtimes for each step of 10 fold cross validation for different size datasets, using our adapted fast *k-MSN* classifiers, with $k = 1$.

No. of prototypes	Adapted *k-NN* classifiers											
	FN			GB			ONC			MS		
	Acc	Comp	Runtime	Acc	Comp	Runtime	Acc	Comp	Runtime	Acc	Comp	Runtime
2000	93.70	9.93	9.071	93.60	5.87	18.25	93.60	3.54	8.16	93.55	9.52	8.86
3000	94.20	7.78	17.43	94.13	4.30	34.20	94.23	2.47	13.88	93.80	7.48	17.28
4000	95.03	6.59	27.22	95.05	3.52	54.63	95.07	1.99	14.95	94.82	6.35	25.67
7200	94.97	5.76	83.04	94.90	5.36	98.45	95.00	1.99	42.52	94.88	4.63	80.53
Average	94.48	7.52	34.19	94.42	4.76	51.38	94.48	3.00	19.88	94.26	7.00	33.09

Table 7
Average runtimes for each step of 10 fold cross validation for different size datasets, using the exhaustive, cluster tree and the proposed fast *k-MSN* classifiers, with $k = 1$.

No. of prototypes	Exhaustive *k-NN* classifier			Cluster tree			Proposed classifier: Tree *k-MSN*					
	Acc	Comp	Runtime	Acc	Comp	Runtime	Using local search			Using global search		
							Acc	Comp	Runtime	Acc	Comp	Runtime
2000	93.70	100	1279	91.20	12.15	11.62	93.60	1.66	0.485	93.40	2.41	1.03
3000	94.90	100	3714	90.56	10.64	18.45	94.00	1.16	0.812	94.03	1.71	1.60
4000	95.03	100	7958	90.16	9.15	22.85	95.00	0.89	1.139	94.85	1.34	2.35
7200	95.00	100	52 606	90.25	6.09	32.62	94.26	0.01	7.63	94.02	0.82	8.95
Average	94.66	100	16 389	90.54	10.64	21.38	94.22	0.93	2.52	94.08	1.57	3.48

However, in our experiments for all classifiers the lower accuracy was obtained using *HOEM* function.

In Fig. 10 a scatter graphic of the accuracy against the comparisons percentage using the different fast *k-MSN* classifiers, with $k = 1$ and the three prototype comparisons functions (*HVDM*, *D* and *HOEM*) is shown. From Fig. 10, we can note that all the classifiers obtained similar accuracy but the classifiers proposed in this work (using local and global search) did the smallest number of comparisons.

4.7. Runtime experiments

Since the objective of the approximate fast *k-NN* classifiers is to reduce the number of comparisons, between pairs of prototypes, trying to keep the classification accuracy obtained by the exhaustive *k-NN*, for most of the fast *k-NN* classifiers, the authors report and compare the number of comparisons. However, in order to show the performance of the proposed methods, in Tables 6 and 7, a comparison among the different classifiers, regarding runtimes, is presented. In these tables the average classification runtime (Runtime column) needed for each *k-NN* classifier (using $k = 1$ and *HVDM* function), in each step of the *ten fold cross validation*, is shown. For this experiment, the mixed dataset *Ann-thyroid,* taking 2000 (each fold has 1800 prototypes for training and 200 for testing), 3000 (each fold has 2700 prototypes for training and 300 for testing), 4000 (each fold has 3600 prototypes for training and 400 for testing), and 7200 (each fold has 6480 prototypes for training and 720 for testing) prototypes, was used. All the experiments were done using a PC computer with a 3.0 GHz Pentium 4 processor and 1 GB RAM.

From Tables 6 and 7, we can observe that comparing our method against the exhaustive search, the number of comparisons and the classification runtime, were significantly reduced. For example for classifying 720 prototypes using a training set with 6480 prototypes, which is a medium size dataset, our classifier required 7.63 s, while the exhaustive search required 52 606 s (14.61 h). More over, if we wanted to classify 720 000 prototypes instead of 720, using the same training set, our classifier would require approximately 7630 s (2.1 h), and the exhaustive search would require approximately 52 606 000 s (608.75 days = 1 year and 8 months). From these results, we can observe that the exhaustive search becomes inapplicable for some applications that require classifying a large amount of prototypes.

4.8. Comparison against other fast k-NN classifiers

In this section, the performance of the proposed classifier (Tree *k-MSN*) is compared against tree-based and Approximating–Eliminating classifiers.

The Approximating–Eliminating classifiers compared in this work were: AESA [43], LAESA, using $|BP| = 20\%$ and 50% of the objects in the dataset [5], TLAESA [37] and Modified TLAESA [44].

HVDM function was used. For this experiment, 10 datasets from the UCI repository were used. The largest datasets from the previous experiment were not included because the Approximating–Eliminating algorithms cannot be applied due to their high memory requirements.

From Table 8, we can observe that when the comparison function does not satisfy the triangle inequality, AESA, LAESA, TLAESA and modified TLAESA algorithms become inexact (i.e., the obtained results are not the same as using the exhaustive search). However, the percentage of comparisons is, on average, reduced from 100%, done by the exhaustive search, to 24.71%, 28.41%, 27.38%, 53.04% and 30.88%, respectively.

Comparing the biggest reduction among the Approximating–Eliminating classifiers (achieved by AESA, with 24.71%) and the tree-based classifiers (achieved by ONC classifier, with 36.37%), we can observed that better results are obtained using AESA classifier. However, a drawback of AESA classifier is the spatial cost to store a matrix of distances among all pair of prototypes in the training set.

From these experiments, we can also observe that the classifier proposed in this work (Tree *k-MSN*) achieves the biggest reduction on the percentage of prototypes comparisons (14.5% using local search, and 16.53% using global search).

Table 8
Comparison of tree-based fast *k-NN* classifiers (a), Approximating-Eliminating fast *k-NN* classifiers (b), the exhaustive *k-MSN* classifier, cluster-tree and tree *k-MSN* (c), with $k=1$, using *HVDM* comparison function.

(a)

Datasets	Exhaustive *k-NN* classifier		Adapted *k-NN* classifiers							
	Acc	Comp	FN		GB		ONC		MS	
			Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp
Hepatitis	81.75	100	81.13	118.99	81.13	87.50	81.13	71.89	81.08	95.86
Zoo	97.00	100	97.00	24.68	97.00	23.46	97.00	22.16	97.00	21.86
Flag	53.21	100	53.71	56.97	53.71	53.10	53.18	46.52	54.26	43.43
Echoc..	82.69	100	82.69	121.01	82.69	84.01	82.69	72.82	84.18	93.43
Hayes	84.29	100	84.29	28.31	84.29	21.14	84.29	16.59	83.57	27.45
Soyb.	90.54	100	91.18	26.56	91.18	19.90	91.18	16.85	89.88	25.02
Bridges	63.36	100	64.27	93.65	64.27	53.34	65.18	50.65	63.27	54.25
Glass	68.18	100	68.18	35.96	68.18	27.04	68.18	20.16	67.71	34.36
Iris	94.67	100	94.67	21.12	94.67	19.87	94.67	18.21	95.33	19.37
Wine	95.46	100	95.46	43.78	95.46	32.66	95.46	27.88	94.35	34.31
Avg.	81.12	100	81.26	57.10	81.26	42.20	81.30	36.37	81.06	44.93

(b)

Datasets	AESA		LAESA |PB| = 20% of the prototypes in *T*		LAESA |PB| = 50% of the prototypes in *T*		TLAESA |PB| = 20% of the prototypes in *T*		Modified TLAESA |PB| = 20% of the prototypes in *T*	
	Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp
Hepatitis	81.68	51.03	80.61	60.73	80.68	59.68	81.64	84.64	81.03	68.26
Zoo	97.00	21.34	96.00	25.23	96.00	23.43	95.75	55.77	95.78	27.34
Flag	53.60	27.23	52.82	27.57	52.01	25.34	52.01	49.25	50.19	42.94
Echoc.	82.54	64.34	82.08	67.39	82.23	62.68	82.25	75.84	82.12	38.30
Hayes	83.71	21.23	80.71	21.84	80.73	19.23	80.73	49.44	80.48	25.49
Soyb.	89.87	2.07	89.87	5.12	89.87	4.23	89.87	38.44	87.23	18.35
Bridges	63.21	24.23	60.37	26.23	59.37	27.68	59.37	48.45	59.49	38.92
Glass	68.18	13.20	68.18	24.53	68.18	26.34	68.18	49.54	68.18	22.39
Iris	94.67	8.23	94.67	10.68	94.67	9.89	94.67	42.54	94.67	13.29
Wine	95.46	14.23	95.46	14.75	95.46	15.34	95.46	36.45	95.46	13.52
Avg.	80.99	24.71	80.08	28.41	79.92	27.38	79.99	53.04	79.46	30.88

(c)

Datasets	Cluster tree		Proposed classifier: Tree *k-MSN*			
	Using local search		Using global search			
Acc	Comp	Acc	Comp	Acc	Comp	
Hepatitis	77.88	42.01	83.71	9.54	83.71	15.65
Zoo	95.00	41.63	96.00	19.68	96	16.65
Flag	48.47	44.09	52.21	13.20	52.71	17.07
Echoc..	83.24	41.75	79.62	16.50	80.33	17.65
Hayes	67.31	27.45	83.52	18.19	81.26	19.65
Soyb.	83.33	20.22	85.26	9.72	84.28	13.10
Bridges	40.27	36.82	60.36	15.80	57.45	17.65
Glass	60.30	34.64	67.73	12.91	67.25	16.54
Iris	86.67	37.41	92.67	15.66	92.67	17.65
Wine	93.24	45.85	91.57	13.80	92.12	17
Avg.	73.57	37.18	79.27	14.50	78.8	16.53

5. Conclusions

In this work, an approximated fast *k-MSN* classifier for mixed data, which allows using any prototype comparison function, was proposed. In order to compare our classifier, FN, MS, ONC and GB classifiers were adapted, using our proposed tree structure, to allow them working on mixed data and to use any prototype comparison function, because of under these circumstances the original algorithms cannot be applied.

Based on our experimental results, in comparison with the exhaustive classifier, and the FN, MS, ONC and GB adapted classifiers, the proposed classifier (using *k-MSN* local and global search algorithms), obtained a big reduction on the number of comparisons between prototypes, which is of particular importance in applications where a fast response is required. As we showed in the runtime experiment, when a large amount of mixed data prototypes needs to be classified, the exhaustive search become impractical, and therefore a fast *k-NN* classifier is required. In this situation, our proposal is the best choice.

Additionally, a comparison against Cluster tree [45], which does not assume metric properties of the comparison function, was done. From these experiments we can also notice that Cluster tree obtains lower classification accuracy than the proposed classifier (Tree *k-MSN*) and requires more prototype comparisons.

Finally, we can notice that for large mixed datasets and non-metric prototype comparison functions, if a better accuracy is required, the best choice is our adapted ONC classifier, but if reducing the runtime is more important, the best option is our proposed Tree *k-MSN* classifier using local search.

As future work, we plan to look for pruning rules, not based on metric properties, which would allow us to reduce even more the number of prototype comparisons.

References

[1] T.M. Cover, P.E. Hart, Nearest neighbor pattern classification, Transactions on Information Theory 13 (1967) 21–27.
[2] S.A. Nene, S.K. Nayar, A simple algorithm for nearest neighbour search in high dimensions, IEEE Transactions in Pattern Analysis and Machine Intelligence 19 (9) (1997) 989–1003.
[3] V. Ramasubramanian, K. Paliwal, Fast nearest-neighbor search based on approximation–elimination search, Pattern Recognition 33 (2000) 1497–1510.
[4] C. Yong-Sheng, H. Yi-Ping, F. Chiou-Shann, Fast and versatile algorithm for nearest neighbor search based on lower bound tree, Pattern Recognition Letters 40 (2) (2007) 360–375.
[5] L. Micó, J. Oncina, E. Vidal, A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing-time and memory requirements, Pattern Recognition Letters 15 (1994) 9–17.
[6] M. Denny, M.J. Franklin, Operators for expensive functions in continuous queries, in: Proceedings of the 22nd International Conference on Data Engineering, ICDE'06, vol. 03 (issue 07) 2006, p. 147.
[7] M. Adler, B. Heeringa, Search Space Reductions for Nearest-Neighbor Queries, in: Lecture Notes in Computer Science, 2008, pp. 554–567.
[8] R. Panigrahi, An Improved Algorithm Finding Nearest Neighbor Using Kd-trees, in: Lecture Notes in Computer Science, vol. 4957, 2008, pp. 387–398.
[9] E. Pekalska, R. Duin, The Dissimilarity Representation for Pattern Recognition, in: Foundations and Applications, World Scientific, Singapore, December 2005.
[10] H. Chang, D. Yeung, W. Cheung, Relaxational metric adaptation and its application to semi-supervised clustering and content-based image retrieval, Pattern Recognition 39 (2006) 1905–1917.
[11] D. Kushnir, M. Galun, A. Brandt, Fast multiscale clustering and manifold identification, Pattern Recognition 39 (2006) 1876–1891.
[12] J. Laub, V. Roth, J. Buhmann, K. Muller, On the information and representation of non-Euclidean pairwise data, Pattern Recognition 39 (2006) 1815–1826.
[13] M. Lozano, J. Martínez Sotoca, J. Sánchez, F. Pla, E. Pekalska, R. Duin, Experimental study on prototype optimisation algorithms for prototype-based classification in vector spaces, Pattern Recognition 39 (2006) 1827–1838.
[14] C. Chien-Hsing, K. Bo-Han, C. Fu, The generalized condensed nearest neighbor rule as a data reduction method, in: Proceedings of the 18th International Conference on Pattern Recognition, vol. 02, 2006, pp. 556–559.
[15] J.H. Friedman, F. Baskett, L. Shustek, An algorithm for finding nearest neighbors, IEEE Transactions on Computers (1975) 1000–1006.
[16] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Boston, MA, pp. 47–57.
[17] D.A. White, R. Jain, Similarity indexing with the SS-tree, in: Proceedings of the International Conference on Data Engineering, New Orleans, LA, 1996, pp. 516–523.
[18] N. Katayama, S. Satoh, The SR-tree: an index structure for high dimensional nearest neighbor queries, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Tucson, AZ, USA, 1997, pp. 369–380.
[19] S. Berchtold, C. Böhm, H. Kriegel, The pyramid-technique: towards breaking the curse of dimensionality, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Seattle, WA, 1988, pp. 142–153.
[20] S. Berchtold, D.A. Keim, H. Kriegel, T. Seidl, Indexing the solution space: a new technique for nearest neighbor search in high dimensional space, IEEE Transactions on Knowledge Data Engineering 12 (1) (2000) 45–57.
[21] K. Fukunaga, P. Narendra, A branch and bound algorithm for computing *k*-nearest neighbors, IEEE Transactions on Computers 24 (1975) 743–750.
[22] S. Brin, Near neighbor search in large metric spaces, in: Proceedings of the International Conference on very Large Data Bases, Zurich, Switzerland, 1995, pp. 574–584.
[23] J. McNames, A fast nearest neighbor algorithm based on a principal axis search tree, IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (9) (2001) 964–976.
[24] M.R. Soleymani, S.D. Morgera, An efficient nearest neighbor search method, IEEE Transactions on Communications COM 35 (6) (1987) 677–679.
[25] A. Djouadi, E. Bouktache, A fast algorithm for the nearest-neighbor classifier, IEEE Transactions in Pattern Analysis and Machine Intelligence 19 (3) (1997) 277–282.
[26] S. Arya, D. Mount, N. Netanyahu, R. Silverman, A. Wu, An optimal algorithm for approximate nearest neighbor searching in high dimensions, Journal of the ACM 45 (6) (1998) 891–923.
[27] F. Moreno-Seco, L. Mico, J. Oncina, Approximate nearest neighbor search with the Fukunaga and Narendra algorithm and its application to chromosome classification, CIARP, in: Lecture Notes in Computer Science, vol. 2905, 2003, pp. 322–328.
[28] K. Figueroa, E. Chávez, G. Navarro, R. Paredes, On the last cost for proximity searching in metric spaces, in: WEA 2006, Lecture Notes in Computer Science, vol. 4007, 2006, pp. 279–290.
[29] A. Faragó, T. Linder, G. Lugosi, Fast nearest-neighbor search in dissimilarity spaces, IEEE Transactions in Pattern Analysis and Machine Intelligence 15 (9) (1993) 957–962.
[30] C. Hsieh, Y. Liu, Fast search algorithms for vector quantization of images using multiple triangle inequalities and wavelet transform, IEEE Transactions on Image Processing 9 (3) (2000) 321–328.
[31] I. Kalantari, G. McDonald, A data structure and an algorithm for the nearest point problem, IEEE Transactions on Software Engineering 9 (1983) 631–634.
[32] S. Omachi, H. Aso, A fast algorithm for a k-nn classifier based on branch and bound method and computational quantity estimation, Systems and Computers in Japan 31 (6) (2000) 1–9.
[33] E. Gómez-Ballester, L. Mico, J. Oncina, Some approaches to improve tree-based nearest neighbor search algorithms, Pattern Recognition Letters 39 (2006) 171–179.
[34] J. Oncina, F. Thollard, E. Gómez-Ballester, L., Micó, F. Moreno-Seco, A tabular pruning rule in tree-based fast nearest neighbor search algorithms, IbPRIA, in: Lecture Notes in Computer Science, vol. 4478, 2007, pp. 306–313.
[35] J.R. García-Serrano, J.F. Martínez-Trinidad, Extension to C-means algorithm for the use of similarity functions, in: Lectures Notes in Artificial Intelligence, vol. 1704, 1999, pp. 354–359.
[36] J.F. Martínez-Trinidad, J.R. García-Serrano, I.O. Ayaquica-Martínez, C-means algorithm with similarity functions, Computación y Sistemas 5 (4) (2002) 241–246.
[37] L. Mico, J. Oncina, R. Carrasco, A fast branch and bound nearest neighbor classifier in metric spaces, Pattern Recognition Letters 17 (1996) 731–739.
[38] W. D'haes, D. Dyck, X. Rodel, PCA-based branch and bound search algorithms for computing K nearest neighbors, Pattern Recognition Letters 24 (2002) 1437–1451.
[39] D. Wilson, T. Martínez, Reduction techniques for instance based learning algorithms, Machine Learning 38 (2000) 257–286.
[40] D. Wilson, T. Martínez, Improve heterogeneous distance functions, Journal of Artificial Intelligence Research 6 (1997) 1–34.
[41] C. Blake, C. Merz, UCI repository of machine learning databases, 1998. ⟨http://www.uci.edu/mlearn/databases/⟩, Department of Information and Computer Science, University of California, Irvine, CA.
[42] T. Dietterich, Statistical tests for comparing supervised classification learning algorithms, Neural Computation 10 (7) (1998) 1895–1923.
[43] E. Vidal, An algorithm for finding nearest neighbours in (approximately) constant average time complexity, Pattern Recognition Letters 4 (1986) 145–157.
[44] K. Tokoro, K. Yamaguchi, S. Masuda, Improvements of TLAESA nearest neighbor search and extension to approximation search, in: ACSC'06: Proceedings of the 29th Australian Computer Science Conference, 2006, pp. 77–83.
[45] B. Zhang, S. Srihari, Fast k nearest neighbour classification using cluster-based tree, IEEE Transactions on Pattern Analysis and Machine Intelligence 26 (4) (2004) 525–528.

About the Author—JOSÉ FRANCISCO MARTÍNEZ TRINIDAD Titular Professor of Computer Science and Pattern Recognition Category B at National Institute of Astrophysics, Optics and Electronics, Mexico, since 2002. Previous appointments include Professor of Computer Science and Pattern Recognition at Computer Science Research Center of the National Polytechnic Institute, Mexico from 1996 to 2001. Professor of Computer Science at Technological Institute of Toluca, Mexico from 1995 to 1996. Professor of Computer Science at Autonomous University of Puebla, Puebla, Mexico during 1994. His research interests are logical combinatorial pattern recognition, mixed data analysis, clustering, conceptual clustering, feature selection, prototype selection, fast most similar neighbor classifiers, text analysis, text categorization, sequential patterns, and pattern recognition methods for data mining.

About the Author—JESÚS ARIEL CARRASCO OCHOA Titular Professor of Computer Science and Pattern Recognition at National Institute of Astrophysics, Optics and Electronics, Mexico. His research interests are logical combinatorial pattern recognition, mixed data analysis, clustering, conceptual clustering, feature selection, prototype selection, fast most similar neighbor classifiers, text analysis, text categorization, sequential patterns, and pattern recognition methods for data mining.

About the Author—SELENE HERNÁNDEZ-RODRÍGUEZ Student of Ph.D. at the National Institute of Astrophysics, Optics and Electronics, Mexico. She is currently working on the development of new classifiers for mixed data as part of my Ph.D. research.