Fast tree-based classification via homogeneous clustering

Georgios Pardis[1], Konstantinos I. Diamantaras[1][0000−0003−1373−4022], Stefanos Ougiaroglou[1,2][0000−0003−1094−2520], and Georgios Evangelidis[2][0000−0003−1639−2152]

[1] Department of Information and Electronic Engineering, International Hellenic University, 57400 Sindos, Greece
george.pardis@gmail.com, kdiamant@it.teithe.gr
[2] Department of Applied Informatics, School of Information Sciences, University of Macedonia, 54636 Thessaloniki, Greece
{stoug, gevan}@uom.edu.gr

Abstract. Data reduction aims to reduce the number of training data in order to speed-up the classifier training. They do that by collecting a small set of representative prototypes from the original patterns. The Reduction by finding Homogeneous Clusters algorithm is a simple data reduction technique that recursively utilizes k-means clustering to build a set of homogeneous clusters. The means of the clusters constitute the set of the prototypes. Based on the same idea, we propose two new classifiers, which recursively produce homogeneous clusters and achieve higher performance than current homogeneous clustering methods with significant speed up. The key idea is the development of a tree data structure that holds the constructed clusters. Tree nodes consist of clustering models. Leaves correspond to homogeneous clusters where the corresponding class label is stored. Classification is performed by simply traversing the tree. The two algorithms differ on the clustering method used to build tree nodes: the first uses k-means while the second applies EM clustering. The proposed algorithms are evaluated in different datasets and compared with well-known methods. The results demonstrate very good classification performance combined with large computational savings.

Keywords: Classification, k-means, EM, Prototype Generation

1 Introduction

Handling large volumes of training data in classification systems is an open topic that has attracted the interest of the research community. The motivation behind all the efforts is the fast training process as well as the fast and accurate classification. In this context, many Data Reduction Techniques (DRTs) [5, 9] have been proposed in an attempt to reduce the classification cost of the k-Nearest Neighbour classifier (k-NN) [2]. However, DRTs can be also applied to eager classifiers [7]. The goal of DRTs is to build a small representative set of the training set. This set is called condensing set and has the advantage of low

computational cost without sacrificing accuracy. A DRT can be either Prototype Selection (PS) [5] or Prototype Generation (PG) [9]. The DRTs that belong to first category select representative prototypes form the original training set. On the other hand, the DRTs that adopt the PG approach generate prototypes by summarizing similar items. Most DRTs are based on a simple idea: The instances that lie far from the class decision boundaries are redundant and increase the computational cost. Hence, DRTs try to select or generate many prototypes for the close borders areas. It is worth mentioning that the editing algorithms is a special subcategory of PS that aim to remove noise.

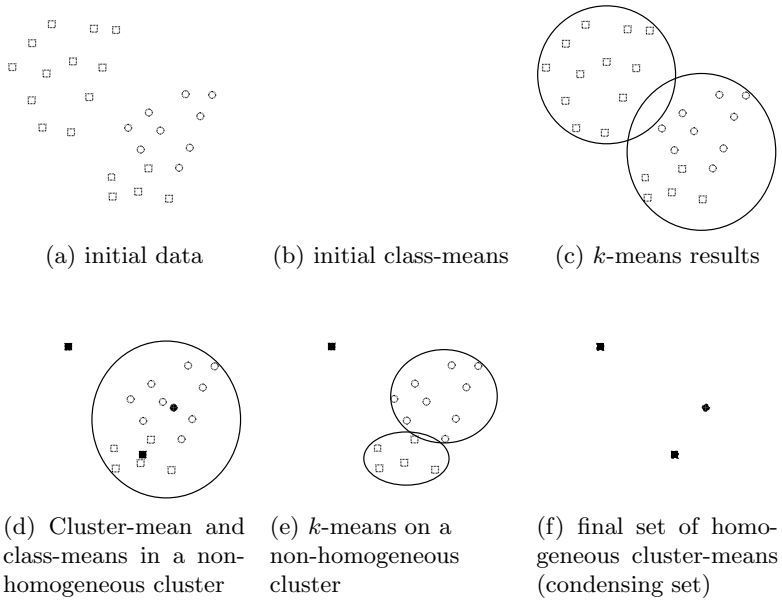
Reduction by finding Homogeneous Clusters (RHC) [8] is a simple and fast DRT that is based on a recursive procedure that builds homogeneous clusters, i.e. clusters that contains items of a specific class. For each non-homogeneous cluster, RHC applies k-means clustering using as initial seeds the means of the classes that are present in the cluster. This procedure is repeated on all non-homogeneous clusters and terminates when all become homogeneous. The means of the homogeneous clusters constitute the condensing set.

RHC discards the means of the non-homogeneous clusters. However, they can be used to build a tree data structure, where the non-leaf nodes store the means of non-homegeneous clusters while the leaf-nodes store the means of homogeneous clusters. Thus, instead of applying k-NN over the condensing set (i.e., the set of leaf nodes), the classification could be performed by traversing the tree. This simple idea constitutes the motive behind the present work. The contribution of the paper is the development of two classifiers that utilize a tree-based model. When an unclassified instance is to be classified, the prediction process of the proposed algorithms traverses the tree and the instance gets assigned to the unique class stored in the leaf node. The two algorithms differ to each other on the clustering algorithm that they use, i.e., k-means and EM, respectively.

The rest of the paper is organized as follows: Section 2 reviews RHC and ERHC. Section 3 presents in detail the proposed tree-based classifiers. Section 4 presents the results of the experimental study. Section 5 concludes the paper.

2 The RHC and ERHC Algorithms

The RHC algorithm is a non-parametric PG algorithm that recursively applies k-means and keeps on constructing clusters until all of them are homogeneous. Initially, RHC considers the whole training set as a non-homogeneous cluster. The algorithm begins by computing the mean for each class by averaging the attribute values of the corresponding instances. Therefore, for a dataset with n classes, RHC computes n class means. Then, RHC executes k-means using the n aforementioned class-means as initial seeds and builds n clusters. For each homogeneous cluster, its mean is placed in the condensing set as prototype. For each non-homogeneous cluster, the above procedure is applied recursively. RHC stops when all clusters are homogeneous. In the end, the condensing set contains all the mean items of the homogeneous clusters. Note that using the class means as initial means for k-means clustering, the number of clusters is determined



(a) initial data (b) initial class-means (c) k-means results

(d) Cluster-mean and class-means in a non-homogeneous cluster (e) k-means on a non-homogeneous cluster (f) final set of homogeneous cluster-means (condensing set)

Fig. 1: Data generation through RHC

automatically. Figure 1 demonstrates the execution of RHC on a dataset with two classes. The algorithm recursively identifies three homogeneous clusters, and, in the end, the initial dataset is replaced by only three prototypes.

RHC generates more prototypes for the close-class-border data areas and less for the central class data areas. Therefore, datasets with many classes and noisy items create more prototypes and thus lower reduction rate is achieved.

Editing and Reduction through homogeneous clusters (ERHC) [6] is a simple variation of RHC. It is based on a following idea: a cluster that contains only one item is probably noise. ERHC removes that type of clusters. Therefore, ERHC is more noise tolerant than RHC and achieves higher accuracy and reduction rates as well as lower computational cost especially on datasets with noise.

3 The Proposed Algorithms

In this paper, we propose two classifiers. Both recursively apply clustering methods to create a tree data structure whose leaves represent homogeneous clusters. The first algorithm applies k-means while the second one employs EM clustering [3]. The nodes store cluster models trained on items belonging to more than

Algorithm 1 Proposed algorithms

Input: TS
Output: $HomogeneousTree$

```
1: HomogeneousTree ← ∅
2: queue ← enqueue(clusters obtained by applying EM or k − means on TS)
3: repeat
4:    C ← dequeue(queue)
5:    m ← trained model of C
6:    if C is homogeneous then
7:       Insert leaf m in HomogeneousTree
8:    else
9:       Insert internal node m in HomogeneousTree
10:      queue ← enqueue(clusters obtained by applying EM or k − means on C
11:   end if
12: until IsEmpty(queue)
13: return HomogeneousTree
```

one classes, i.e., they are non-homogeneous. The idea behind this is that, during classification stage, instead of comparing the distance between each unclassified item with all prototypes, the algorithm will only compare the distances to the closest ones. Also, the homogeneous clustering technique comes up with a weakness. Since more prototypes are generated to close-class-borders, datasets with many classes and noisy items create more prototypes and thus lower reduction rate is achieved. To overcome this weakness, the proposed algorithms remove the noise with a smart technique that leads to a higher rate of reduction without negatively affecting the processing cost, as the editing algorithms usually do.

The two algorithms operate in the same way and the only difference is the applied clustering technique. Initially, they consider the whole training set to be a non-homogeneous cluster. If the cluster contains items from c classes, they compute c clusters and then apply k-means or EM on each cluster recursively. The model is saved in the tree as a root node and the children of this node are the c different clusters. For each child node, homogeneity is checked. If the cluster is homogeneous or is "almost" homogeneous (i.e., a large percentage of its items share the same class label) this node becomes an appropriately labeled leaf. In case of a large percentage of homogeneity, the small remaining percentage of items probably lies in a region of a different class (noise) or lies close to a decision-boundaries region, and thus, it is removed and the node is considered to be homogeneous. If the cluster node is not homogeneous this procedure is applied recursively. Each unclassified item is classified only by descending the tree. The pseudo-code in Algorithm 1 describes the process.

3.1 Training HC_EM

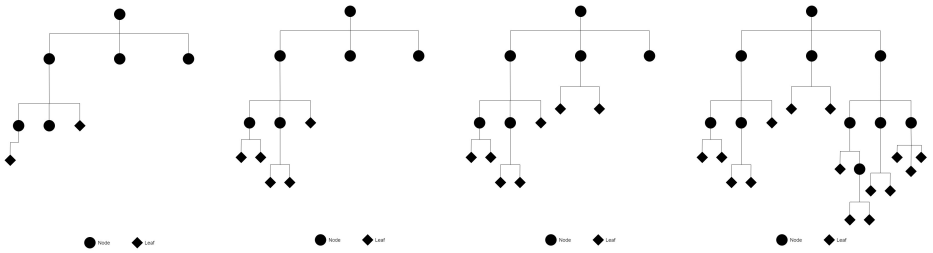
The first algorithm (HC_EM) uses EM as a training method. In the beginning, the whole dataset is given to the method. The method checks if the dataset

consists of items of only one class and if it does, this set is homogeneous so a leaf is created that carries the label of this class. In case it is not homogeneous, the method finds the most popular label and compares its items with all the rest. If the percentage of the most popular label is more than ninety-five, the set is considered as homogeneous and a leaf is created carrying this label. In case there are less than five items in total, a leaf that carries the label of the most popular class is created. The numbers five and ninety-five were chosen after a trial and error procedure to find the combination that gives the best results on average. Almost always, the initial dataset is not homogeneous so the algorithm will try to find the best parameters to create an effective model. The first parameter that the training method uses, is the weight of each class. The other parameters are the number of classes, the means of each class and the initial precisions [4]. Now that the model is ready, we train it using the dataset and it produces some predictions about the class each item belongs to. In this point, we create different clusters consisting of the items of each cluster label that the model predicted. Then, we produce a node that carries the trained model and has as many children as the different clusters created before. For each different child cluster we repeat the whole training procedure from the beginning given as input the set of items with their real class label and not the predicted one.

3.2 Training HC_KMEANS

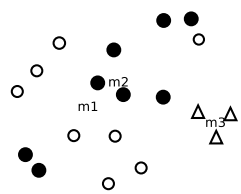
HC_KMEANS is the second of the proposed algorithms. It uses the k-means as a training method. The reason we decided to construct HC_KMEANS is because although the EM leads to better results, it is slow. By using the k-means, we achieve faster execution, without a significant cost in accuracy. The first steps of the training procedure are the same as we described for the HC_EM above. The first difference is that the number of the minimum items capable to consist a cluster is three and the percentage of superiority of a class that can make it a homogeneous cluster is ninety-eight. Again, these numbers were chosen after a trial and error procedure. The other difference is that the model created takes as a parameter only the number of the different classes and the mean (representative) of each one. The rest of the procedure is the same. A node is created and contains the trained model and the different child clusters.

Figure 2 visualizes the training procedure. Once the training set is loaded in the root of the tree, Expectation Maximization or k-means algorithm trains a model to predict as many clusters as the different classes in the set. This model is stored in the node. For every cluster we test if there are any misclassified items and the procedure repeats until all clusters are homogeneous. In Figure 3, one can visualize a complete example of training when using k-means. The initial dataset contains three class labels. Eventually, a tree model is constructed using recursive application of k-means. The nodes of the tree contain the cluster models, in our case cluster centroids and a class label. The leaves of the tree correspond to the homogeneous clusters whose items form the initial dataset.

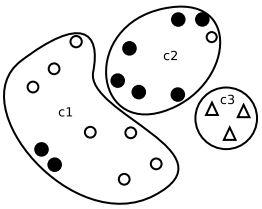


(a) The creation of the first leaf (b) First node completed (c) First two nodes completed (d) The completed tree structure

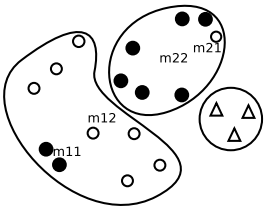
Fig. 2: General Training procedure



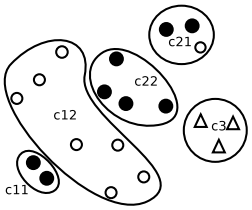
(a) Initial dataset with m_i indicating the class representatives



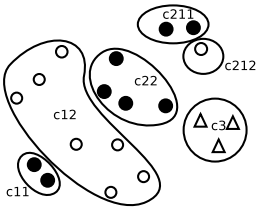
(b) Resulting clusters after applying k-means using m_i's as initial seeds. c3 is homogeneous.



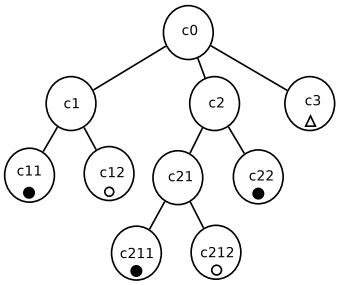
(c) Computing class representatives m_{1i} and m_{2i} for clusters c1 and c2, respectively.



(d) Resulting clusters after applying k-means on c1 and c2. c11, c12 and c22 are homogeneous.



(e) Resulting clusters after applying k-means on c21. c211 and c212 are homogeneous.



(f) The completed tree structure that replaces the initial dataset. Notice the labeled leaves.

Fig. 3: HC_KMEANS Training procedure. In the final model, leaves correspond to the centroids of the homogeneous clusters and internal nodes to the centroids of the non-homogenous clusters.

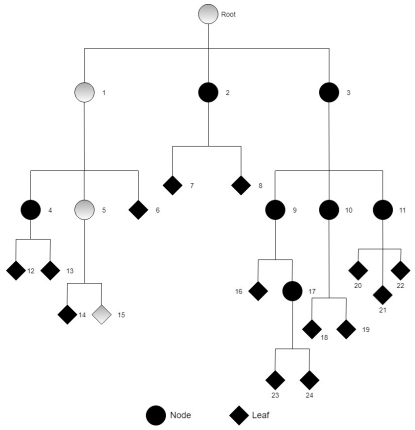


Fig. 4: Classification procedure

3.3 Prediction procedure

Each unclassified item descends the tree structure and it ends up in a leaf whose cluster label adopts as the predicted one. If the adopted cluster label matches the class label of the item then, the item has been classified correctly. At the beginning, the item will be at the root of the tree and the trained model of the node, which had been stored in the training procedure, will be used to predict the cluster label. Then, it will descend to the child cluster whose cluster label matches the predicted one. If the child is a leaf, the item is assigned to the cluster label of the leaf, otherwise the procedure is repeated until a leaf is reached.

Figure 4 illustrates the prediction procedure. The predicted label in this example matched the cluster labels of internal nodes 1 and 5 and finally of the leaf node 15. It is possible that during the descend the item may get assigned to different classes until it adopts the final predicted cluster label of a leaf. As already mentioned, the trained model that is stored in each node predicts a cluster label for each item. The way each model makes that prediction is related to the algorithm we use. In case of k-means, the only criterion is the Euclidean distance which determines how similar an item is to a group of items. The model predicts the closest cluster mean each testing sample belongs to. On the other hand, EM uses the weight of each cluster, the means of each mean and the precisions, to create a likelihood logarithm value of the most probable mixture component for the given sample.

Each one of the clusters of a parent node, will be dominated by data items belonging to a certain class label. This means that the model created a cluster which consists of items with similar properties. It is normal for some items to be mislabelled at first, because their properties are more alike those of items belonging to a different label than theirs. These items may constitute noise and in most cases they lay in the border of two or more different classes. In some cases, these items may be a very small cluster within the boundaries of a bigger

one. In our tree structure, during the prediction procedure for an item each new model will predict a new label that may or may not be the same with the previously predicted one. However, statistically, a new label is more likely to be the same as the previous one. The schema of the algorithm we created, leads to greater accuracy and exceptionally low classification costs altogether. Each item needs at most as many predictions as the depth of the tree to be classified. In most of the datasets we tested our algorithm on, the depth was not over three.

4 Performance Evaluation

4.1 Experimental Setup

The proposed algorithms were evaluated using fourteen datasets distributed by the KEEL repository [1] and summarized in Table 1. For comparison purposes, we used the Conventional k-NN classification (with $k = 1$), RHC, ERHC and the Decision Tree C4.5 algorithms. We selected these methods because: like RHC and ERHC, our algorithms are based on the concept of homogeneity. Also, the Decision Tree (DT) C4.5 constructs a tree structure similar to ours. It is also, one of the most well-known algorithms and one of the best in data mining [10].

The thirteen datasets (except the KDD dataset) were used without data normalization. Also, we wanted to evaluate our algorithms on datasets with noise. We built two additional datasets by adding 10% random noise in the LS and PD datasets. We refer to these datasets as LS10 and PD10 respectively. The noise was added by setting the class label of the 10% of the training items to a randomly chosen different class label.

For each dataset and algorithm, we report three average measurements obtained via five-fold cross-validation: (i) Accuracy, (ii) Training time in seconds and (iii) Testing cost in total number of distance computations. For all datasets (except the KDD dataset), we used the five already constructed pairs of training/testing sets hosted by KEEL. Algorithm implementations were written in Python and all the experiments were done on an 8GB laptop with an AMD A8-6410 CPU and Windows 10 operating system.

The original form of KDD has 41 attributes. However, for simplifying the experimentation process, we removed the three nominal and the two fixed value attributes that are present in the dataset. In addition, we removed all the duplicates. Furthermore, the attribute value ranges of the KDD dataset vary extremely. Therefore, we normalized them in the range [0, 1]. We then randomized the transformed KDD dataset and divided it into the appropriate folds.

In order not to favor our algorithms against DT C4.5, after a trial and error procedure, we determined the maximum depth of the DT C4.5 that gives the best average results for all datasets. Maximum depth can be used as a parameter to set a limit on how deep a tree can be. The depth that gave the best results is nineteen (19), although the results for different values do not have significant difference in classification accuracy, especially for the values that are close to the chosen one. Our measurements show that both k-means and EM implementations of the proposed algorithms are superior to the best tuned classification decision trees.

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
KddCup (KDD)	494020/141481	36	23
Balance (BL)	625	4	3
Banana (BN)	5300	2	2
Ecoli (ECL)	336	7	8
Yeast (YS)	1484	8	10
Twonorm (TN)	7400	20	2
MONK 2 (MN2)	432	6	2

4.2 Results and Discussion

Table 2 reports the accuracy results including the conventional 1-NN on the initial datasets. Table 3 reports the time needed to train the classification models. 1-NN is not included there, since as a lazy classifier it does not build any model. Table 4 reports the total number of distances computed during testing. Notice that for HC_EM, we report the number of likelihood functions computed. It may by the case that these computations are slightly more expensive than plain euclidean distance computations. Also, we do not report the testing cost for C4.5 since C4.5 outperforms all the other algorithms during testing by traversing its trees using plain comparisons and not distance computations. The tests confirm that HC_EM algorithm achieves the highest accuracy, with 1-NN and ERHC being close, and clearly outperforms all the other algorithms that perform distance computations during testing, with HC_KMEANS being a close second.

Among the two proposed algorithms, we observe that HC_EM has better results in accuracy than HC_KMEANS. However, the total training process time is about twice as much for HC_EM compared to HC_KMEANS, which we attribute to the higher computational cost of EM. The lower testing times are due to the higher rates of reduction HC_EM achieves. That briefly means that the depth of the produced tree is smaller and so the computations made are less. Regarding the performance of the algorithms on the noisy datasets, ERHC achieves the best overall accuracy, with HC_EM achieving the highest accuracy in PD10. We attribute this to the pruning mechanism used by HC_EM.

It must be noted that our tree-based approach resembles the well-known decision tree (DT) models. However, there is a major difference on the discrimination function applied in each node of the tree. In a typical classification tree, applied on numerical data, each node corresponds to an area in the input space which is divided by a linear separating surface. This surface must be perpendicular to one

of the data dimensions. Clearly, imposing such a restriction limits the separating capabilities of DTs. Nonlinear discriminating surfaces as well as linear surfaces forming some random angle with respect to the axes are not allowed. In our case each tree node is divided using the much more versatile Gaussian split, in the case of the HC_EM, or the Voronoi tessellation in the case of HC_KMEANS. As Table 2 shows, the results of the DT models are inferior to our HC_EM method in most of the cases we have studied. In some cases the difference is significant, such as, for example, the case of the TN (HC_EM accuracy = 97.66%, vs. DT accuracy = 85.42%, a difference of 12.24%) and BL datasets (HC_EM accuracy = 91.61%, vs. DT accuracy = 76.94%, a difference of 14.67%). The highest accuracy difference in favor of the DT models is observed for the LS dataset where HC_EM accuracy = 81.26%, and DT accuracy = 85.04%, a difference of 3.78%).

We can claim that our approach is a decision tree method with nonlinear discriminating surfaces per node.

Table 2: Experimental results: Accuracy (%)

Dataset	**1-NN**	**DT C4.5**	**RHC**	**ERHC**	**HC_ KMEANS**	**HC_ EM**
LR	**95.83**	87.09	93.61	92.89	90.33	91.27
MGT	78.14	83.06	72.45	76.94	76.70	**85.38**
PD	**99.35**	96.21	98.30	98.63	97.78	97.71
LS	**90.60**	85.04	89.08	89.16	88.02	81.26
SH	99.82	**99.98**	99.05	98.69	98.48	99.73
TXR	99.02	92.68	96.96	97.22	95.98	**99.84**
PH	**90.10**	87.02	85.70	86.72	85.29	85.98
KDD	99.71	**99.77**	99.39	99.39	99.44	99.39
BL	78.40	76.94	69.52	75.97	78.06	**91.61**
BN	86.91	87.08	84.32	84.32	**88.93**	86.99
ECL	**79.78**	74.62	69.19	79.19	74.64	73.43
YS	52.02	52.74	48.48	**53.00**	50.77	52.00
TN	94.88	85.42	89.41	91.55	**97.82**	97.66
MN2	90.51	**100**	94.62	95.08	98.35	**100**
PD10	89.50	87.29	77.15	97.25	85.57	**97.45**
LS10	82.02	80.25	79.81	**87.08**	81.98	80.92
Avg	87.91	85.95	84.19	87.69	86.76	**88.79**

5 Conclusions

The two proposed algorithms are simple PG algorithms that achieve high performance. They improve the quality of the training data and as a consequence the

Table 3: Experimental results: Training time (seconds)

Dataset	**DT C4.5**	**RHC**	**ERHC**	**HC_ KMEANS**	**HC_ EM**
LR	<1	32.85	30.94	20.46	261.10
MGT	1.11	71.25	82.94	42.35	55.88
PD	<1	5.22	5.20	3.06	30.79
LS	<1	7.80	8.64	5.39	21.41
SH	<1	8.94	9.23	4.60	44.86
TXR	<1	3.77	3.97	3.00	5.22
PH	<1	12.80	13.34	7.76	20.97
KDD	4.38	26.41	25.71	7.79	20.45
BL	<1	1.19	1.16	<1	1.49
BN	<1	11.11	11.40	6.05	4.08
ECL	<1	1.50	<1	<1	1.83
YS	<1	8.65	5.72	2.73	13.81
TN	<1	7.86	6.51	<1	<1
MN2	<1	<1	<1	<1	<1
PD10	<1	30.32	23.67	26.57	39.83
LS10	<1	16.22	12.00	14.10	42.05

Table 4: Experimental results: Distance computations in thousands. For HC_EM, likelihood functions computed in thousands.

Dataset	**1-NN**	**RHC**	**ERHC**	**HC_ KMEANS**	**HC_ EM**
LR	64000	7573	5057	405	254
MGT	57881	15219	8984	440	252
PD	19335	673	492	96	44
LS	6625	669	466	72	33
SH	538228	2465	1679	501	135
TXR	4840	257	196	40	30
PH	4673	899	557	78	57
KDD	3202767	6581	5234	78	56
BL	62	13	8	5	0.7
BN	4494	217	196	70	40
ECL	18	5	2	2	2
YS	352	176	72	20	23
TN	8761	286	212	33	11
MN2	30	1	1	1.6	1.4
PD10	19335	5205	2567	228	139
LS10	6625	1605	792	122	70
Avg	246127	2615	1657	137	72

accuracy through fast preprocessing that removes noisy and mislabeled training items. The main advantage is that the two algorithms use a tree structure of

trained models that lead to improved accuracy and significantly less computations, since a misclassified item in a parent node can correct its predicted label in the child node computation.

The algorithms were experimentally evaluated on known datasets and compared to RHC and ERHC, two very fast and effective algorithms, the conventional 1-NN classifier and the C4.5 decision tree which is an popular algorithm in data mining. We measured the accuracy, the training time and the testing cost. Through experimental studies, we have demonstrated that they have met the goals for which they were developed and led to improved performance.

References

1. Alcalá-Fdez, J., Fernández, A., Luengo, J., Derrac, J., García, S.: 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)
2. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theor. **13**(1), 21–27 (Sep 2006). https://doi.org/10.1109/TIT.1967.1053964, `http://dx.doi.org/10.1109/TIT.1967.1053964`
3. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society: Series B (Methodological) **39**(1), 1–22 (1977)
4. scikit-learn developers: Scikit-learn user guide. Scikit-learn.org (March 2019)
5. Garcia, S., Derrac, J., Cano, J., Herrera, F.: Prototype selection for nearest neighbor classification: Taxonomy and empirical study. IEEE Trans. Pattern Anal. Mach. Intell. **34**(3), 417–435 (Mar 2012). https://doi.org/10.1109/TPAMI.2011.142, `http://dx.doi.org/10.1109/TPAMI.2011.142`
6. Ougiaroglou, Stefanosand Evangelidis, G.: Efficient editing and data abstraction by finding homogeneous clusters. Annals of Mathematics and Artificial Intelligence **76**(3), 327–349 (2015). https://doi.org/10.1007/s10472-015-9472-8, `http://dx.doi.org/10.1007/s10472-015-9472-8`
7. Ougiaroglou, S., Diamantaras, K.I., Evangelidis, G.: Exploring the effect of data reduction on neural network and support vector machine classification. Neurocomputing **280**, 101 – 110 (2018). https://doi.org/https://doi.org/10.1016/j.neucom.2017.08.076, `http://www.sciencedirect.com/science/article/pii/S0925231217317757`, applications of Neural Modeling in the new era for data and IT
8. Ougiaroglou, S., Evangelidis, G.: RHC: Non-parametric cluster-based data reduction for efficient k-nn classification. Pattern Analysis and Applications **19**(1), 93–109 (2016)
9. Triguero, I., Derrac, J., Garcia, S., Herrera, F.: A taxonomy and experimental study on prototype generation for nearest neighbor classification. Trans. Sys. Man Cyber Part C **42**(1), 86–100 (Jan 2012). https://doi.org/10.1109/TSMCC.2010.2103939, `http://dx.doi.org/10.1109/TSMCC.2010.2103939`
10. Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S., Zhou, Z.H., Steinbach, M., Hand, D.J., Steinberg, D.: Top 10 algorithms in data mining. Knowledge and Information Systems **14**(1),

1–37 (Jan 2008). https://doi.org/10.1007/s10115-007-0114-2, https://doi.org/10.1007/s10115-007-0114-2