Improving Data Reduction by Merging Prototypes

Pavlos Ponos[1], Stefanos Ougiaroglou[1,2], and Georgios Evangelidis[1]

[1] Dept. of Applied Informatics, School of Information Sciences,
University of Macedonia, 54636 Thessaloniki, Greece
pponos@uom.edu.gr, gevan@uom.gr
[2] Dept. of Information Technology,
Alexander TEI of Thessaloniki, 57400 Sindos, Greece
stoug@uom.edu.gr

Abstract. A well-known and adaptable classifier is the k-Nearest Neighbor (kNN) that requires a training set of relatively small size in order to perform adequately. Training sets can be reduced in size by using conventional data reduction techniques. Unfortunately, these techniques are inappropriate in streaming environments or when executed in devices with limited resources. dRHC is a prototype generation algorithm that works in streaming environments by maintaining a condensed training set that can be updated by continuously arriving training data segments. Prototypes in dRHC carry an appropriate weight to indicate the number of instances of the same class that they represent. dRHC2 is an improvement over dRHC since it can handle fixed size condensing sets by removing the least important prototypes whenever the condensing set exceeds a predefined size. In this paper, we exploit the idea[1] that dRHC or dRHC2 prototypes could be merged whenever they are close enough and represent the same class. Hence, we propose two new prototype merging algorithms. The first algorithm performs a single pass over a newly updated condensing set and merges all prototype pairs of the same class under the condition that each prototype is the nearest neighbor of the other. The second algorithm performs repetitive merging passes until there are no prototypes to be merged. The proposed algorithms are tested against several datasets and the experimental results reveal that the single pass variation performs better both for dRHC and dRHC2 taking into account the trade-off between pre-processing cost, reduction rate and accuracy. In addition, the merging appears to be more appropriate for the static version of the algorithm (dRHC) since it offers higher data reduction without sacrificing accuracy.

Keywords: k-NN Classification, Data Reduction, Prototype Merging, Data Streams, Clustering

[1] We thank Prof. Yannis Manolopoulos for his excellent remarks during ADBIS 2017 that led to that idea.

1 Introduction

The attention of the Data Mining and Machine Learning communities has been attracted by the problem of dealing with fast data streams [1] and large data sets that cannot fit in main memory. What is more, researchers focus on how to perform data mining tasks on devices with limited memory, instead of transferring data to powerful processing servers. Classification, being a typical data mining task, has many applications on all above-mentioned environments.

Classification algorithms (or classifiers) can be categorized to eager (model based) or lazy (instance based). Both eager and lazy classifiers assign unclassified items to a predefined set of class values, with their difference being on how they work. Eager classifiers use a training set to build a model that is used to classify new items. On the other hand, lazy classifiers do not build any models and classify a new item by examining the whole training set. What is of utmost importance for both types of classifiers is the size and the quality of the training set, as both dictate the classifier's effectiveness and efficiency.

A well known and widely used lazy classifier is the k-Nearest Neighbors (k-NN) [2]. Once a new unclassified item arrives, its k nearest neighbors are retrieved from the training data. This is achieved by using a distance metric, i.e. Euclidean distance. Then, the unclassified item is assigned to the most common class among the classes of the k nearest neighbors.

k-NN is an effective classifier, especially when it is used on small training sets. In the event of large training sets, all distances between a new item and the training data have to be computed, and as a result its performance degrades. Opposite to an eager classifier that discards the training data after the construction of its classification model, k-NN classifier has higher storage requirements as it must have the training set always available. Another drawback of k-NN is that noise can negatively affect its accuracy. A pre-processing step that builds a small condensing set through a Data Reduction Technique (DRT) can cope with these weaknesses.

The Dynamic RHC (dRHC) algorithm proposed in [3] is a DRT that is based on RHC [8] (Reduction through Homogeneous Clusters) and can gradually build its condensing set. Whenever a new training data segment becomes available, the existing condensing set gets updated incrementally without needing to keep the complete training set and regenerate the prototypes. dRHC is applicable to dynamic environments where training data progressively becomes available and for the cases where data cannot fit in main memory. Despite the fact that dRHC is a fast DRT that achieves high reduction rates with no significant loss in accuracy when applying k-NN, the condensing set that it builds may outpace the available physical memory. dRHC2 [4] is an improvement over dRHC in that it keeps the size of the condensing set fixed. The experimental study in [4] shows that dRHC2 is faster than dRHC while keeping accuracy at high levels.

The motivation of the current work is the scenario shown in Figure 1. The figure depicts a condensing set produced by dRHC2. One can notice that there exist prototypes that could be merged, for example the two prototypes that belong to class "circle". The paper examines the conditions under which proto-

types could be merged without inhibiting the performance of the k-NN classifier. Two variations of a prototype merging algorithm are introduced for both dRHC and dRHC2. The main idea is to merge pairs of prototypes that belong to the same class, only when certain criteria are met. The difference between the two variations lies on whether after the arrival of a data segment that updates the condensing set, the merging phase is executed only once or repetitively until there are no more prototype to be merged.

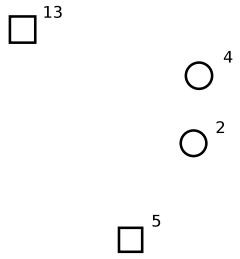


Fig. 1. Items of the same class that could be merged. The two prototypes of class "circle" could be merged. What about the two prototypes of class "square"?

The rest of this paper is structured as follows: Section 2 discusses the background knowledge on DRTs and their limitations. Section 3 reviews the dRHC and dRHC2 algorithms. Section 4 considers in detail the proposed algorithms. In Section 5, the four new algorithms are experimentally compared to dRHC and dRHC2 on fourteen data sets. Section 6 concludes the paper and proposes directions for future work.

2 Background Knowledge

In the literature, Data Reduction Techniques (or DRTs in short) can be classified into two main categories: (i) Prototype Generation (PG) algorithms that generate prototypes to summarize similar items [5] and (ii) Prototype Selection (PS) algorithms that collect prototypes from the initial training set [6]. Prototype selection algorithms can be further categorized into condensing or editing algorithms. PS-condensing and PG algorithms are used for data condensation, i.e., construction of a condensing set from the initial training data. On the other hand, PS-editing algorithms are used for noise and outlier removal from the training data.

The basic idea behind both PG and PS algorithms is that without loss in accuracy we can remove items that do not delineate decision boundaries between classes. Therefore, PG algorithms generate a few prototypes for the internal areas and many more for the close-class borders, whereas PS algorithms try to collect

items the are close to decision boundaries. A point worth mentioning is that both are sensitive to noise, hence an editing algorithm must be applied beforehand.

PS and PG algorithms have been reviewed, and compared to each other in [5], [6] and [7]. A prevalent feature of both is that the whole training set must reside in main memory, which in general, makes DRTs improper for very large datasets, especially for the cases where algorithms are executed in devices with limited resources or when the training set cannot fit in memory.

In addition, as soon as the condensing sets are constructed these DRTs cannot contemplate new items. In other words, they cannot update their condensing set in a dynamic manner. What makes DRTs inappropriate for streaming environments is that training items must always be available once the condensing set is built. For each new training item (D) that becomes available, the algorithm must run from scratch in order to calculate the new condensing set. In order to tackle this issue the Dynamic RHC and Dynamic RHC2 algorithms [3, 4] can be used in dynamic and/or streaming environments.

3 The dRHC and dRHC2 Algorithms

The dRHC algorithm maintains all properties of RHC (Reduction through Homogeneous Clusters) algorithm [3, 8], and in addition it can also manage large or streaming datasets.

The idea behind RHC is to apply k-Means clustering on the training set in order to form as many clusters as the distinct values of the class variable using as initial seeds the corresponding class representatives. Homogeneous clusters, i.e., clusters with all items belonging to the same class, are replaced by their centroid, whereas, the clustering procedure is applied recursively to all non-homogeneous clusters. RHC is shown to be a fast and effective DRT that outperforms other well-known DTRs in terms of data reduction and accuracy [3, 8].

In an analogous fashion, dRHC engages two stages: (i) *initial condensing set construction* and (ii) *condensing set update*. As soon as the first data segment arrives, the *condensing set construction* phase is executed. The only difference with RHC's condensing set is that a weight attribute that denotes the number of training items that are represented is stored for each prototype. All the subsequent data segments that arrive, are processed by the *condensing set update* phase. In this phase, the prototypes of the current condensing set and the items of the incoming data segment are used, so that a new set of initial clusters is built. Then, dRHC algorithm proceeds alike to RHC.

An example of the execution of the *condensing set update* phase is depicted in Figure 2. More specifically, in Figure 2a we can see a condensing set with three prototypes and their corresponding weights. When a new data segment with seven items arrives (Figure 2b), each item is assigned to its nearest prototype (Figure 2c). Cluster A is homogeneous, therefore the prototype's attributes are updated so that it slightly "moves" towards the new items (Figure 2d), and its weight is updated to be the sum of the weights of all items it now represents (all items in the arriving data segment have weight equal to 1). For cluster B,

there is no new item assigned to it, hence the corresponding prototype remains unchanged. On the other hand, cluster C becomes non-homogeneous and RHC is applied on it. k-means creates two homogeneous clusters (Figure 2e). Finally, a new cluster centroid is computed for each cluster and the final updated condensing set is depicted in Figure 2f.

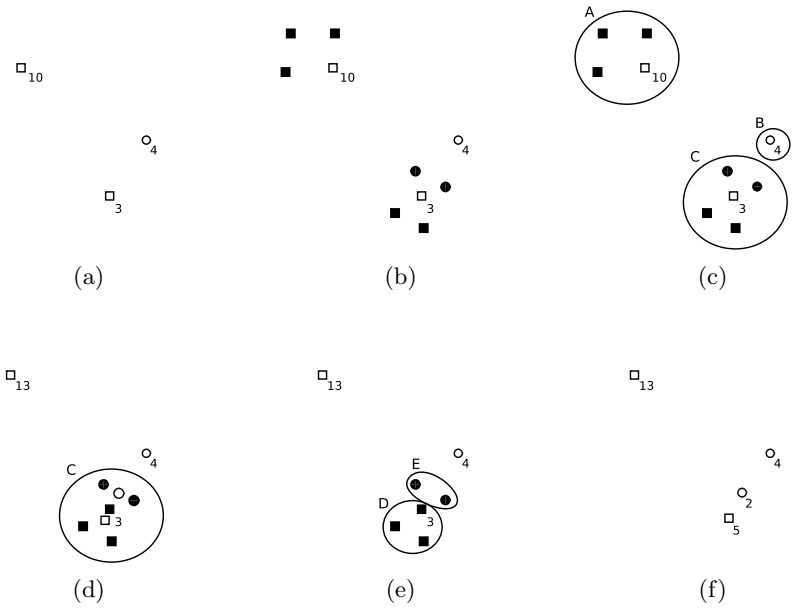


Fig. 2. Example of execution of the *condensing set update* phase of dRHC

Notwithstanding the fact that dRHC seems to be a good fit for data streams or large training datasets, after repetitive condensing set update phases, the condensing set may become too large. An algorithm that confronts with this drawback is dRHC2 [4], in which the size of the condensing set is maintained to a fixed pre-specified threshold. Practically, the only difference between dRHC and dRHC2 is the post-processing step that is presented in [4]. In dRHC2, the condensing set never exceeds a pre-specified size that is determined as a trade-off between computational cost, accuracy, system limitations as well as the level of noise in data.

dRHC2 encapsulates a mechanism where prototypes are ranked according to their importance. The highest the importance of a prototype, the more likely to survive a condensing set update phase. In order to judiciously rank prototypes, dRHC2 takes into account not only the prototype's weight, but also its age.

Lastly, in case of data with noise, dRHC2 performs better than dRHC due to the fact that the noisy prototypes have lower weight and AnA values and are eventually removed.

4 The Proposed Algorithms

As discussed in Section 3, both dRHC and dRHC2 algorithms perform well in streaming environments, with the latter algorithm having a clear advantage after repetitive condensing set update phases. As we demonstrated in Figure 1, we may encounter cases where prototypes that belong to the same class are close enough to be considered for merging. In effect, the size of the condensing set can be further reduced.

We propose two new prototype merging algorithms that:

- in the case of dRHC are applied after each condensing set updating due to the arrival of a new training data segment, and
- in the case of dRHC2 are applied between the condensing set updating phase and the prototype removal via ranking phase.

After examining various merging strategies, we propose a strategy that is not very aggressive and manages to improve data reduction while maintaining accuracy at acceptable levels. The first prototype merging algorithm performs a single pass over the condensing set and *merges prototype pairs belonging to the same class where each prototype is the nearest neighbor of the other*. The notation "sm" that stands for single pass merging is used to denote the new variations of dRHC and dRHC2, namely, dRHCsm and dRHC2sm.

The second prototype merging algorithm performs multiple passes over the condensing set until no more prototype pairs can be merged. The notation "mm" that stands for multiple pass merging is used to denote the new variations of dRHC and dRHC2, namely, dRHCmm and dRHC2mm.

dRHCsm and dRHC2sm being descendants of dRHC and dRHC2 respectively, retain all their properties, with the difference being that once a new data segment arrives and updates the condensing set, the merging phase described in Algorithm 1 is performed.

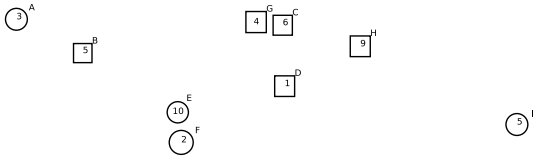
The merging algorithm accepts as input a condensing set CS. Initially, *newCS* is empty. Then, the algorithm for each prototype x checks whether the nearest neighbor of the nearest neighbor y of x is x itself and whether both x and y belong to the same class (line 4). If this is the case, x and y are merged to prototype m, m is added to *newCS* and x, y are removed from *CS* (lines 5–7). Otherwise, x moves to *newCS* (line 9).

A visual representation of the execution of the single pass merging phase is shown in Figure 3. More particularly, the initial condensing set is depicted in Figure 3a. Prototype pairs (E, F) and (C, G) satisfy the merging requirements (Figure 3b), and are merged to form prototypes EF and CG respectively (Figure 3c).

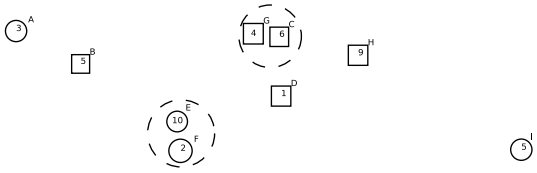
Algorithm 1 Single Pass Merging Phase

Input: CS
Output: $newCS$

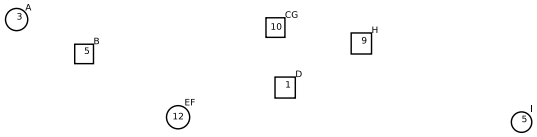
```
newCS ← ∅
for each x in CS do
    y = NN(x)
    if NN(y) == x and class(x) == class(y) then
        m = merge (x,y)
        add m to newCS
        remove x,y from CS
    else
        move x to newCS
    end if
return newCS
end for
```



(a) Initial condensing set



(b) Pairs of prototypes to be merged



(c) CS after the single pass merging phase

Fig. 3. Single Pass Merging Algorithm

In the case of dRHCmm and dRHC2mm, the merging phase is again performed after the condensing set update phase. Algorithm 2 is essentially a variation of Algorithm 1, where merging is applied repetitively until no more pairs of prototypes can be merged. Thus, starting with the condensing set depicted in Figure 3c, with two additional merging passes, first CG and D are merged and then CGD and H are merged for the final condensing set (see Figure 4).

Algorithm 2 Multiple Pass Merging Phase

Input: CS
Output: $newCS$

```
1: newCS ← ∅
2: mergeflag ← True
3: while mergeflag == True do
4:     mergeflag ← False
5:     for each x in CS do
6:         y = NN(x)
7:         if NN(y) == x and class(x) == class(y) then
8:             m = merge (x,y)
9:             mergeflag ← True
10:            add m to newCS
11:            remove x,y from CS
12:        else
13:            move x to newCS
14:        end if
15:    end for
16: end while
17: return newCS
```

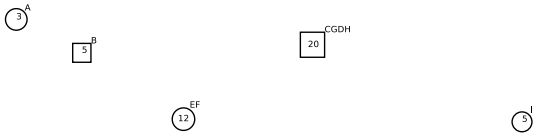


Fig. 4. Multiple Pass Merging Algorithm: condensing set after the multiple pass merging phase

5 Performance Evaluation

5.1 Experimental Setup

The performance of dRHCsm, dRHCmm, dRHC2sm and dRHC2mm was tested against dRHC and dRHC2 using fourteen datasets distributed by the KEEL dataset repository[3] [9]. Table 1 summarizes the datasets used.

Table 1. Dataset description

Dataset	Size	Attributes	Classes	Data segment
Letter Image Recognition (LIR)	20000	16	26	2000
Magic G. Telescope (MGT)	19020	10	2	1902
Pen-Digits (PD)	10992	16	10	1000
Landsat Satellite (LS)	6435	36	6	572
Shuttle (SH)	58000	9	7	1856
Texture (TXR)	5500	40	11	440
Phoneme (PH)	5404	5	2	500
Balance (BL)	625	4	3	100
Pima (PM)	768	8	2	100
Ecoli (ECL)	336	7	8	200
Yeast (YS)	1484	8	10	396
Twonorm (TN)	7400	20	2	592
MONK 2 (MN2)	432	6	2	115
KddCup (KDD)	141481	36	23	4000

The chosen distance metric was the Euclidean distance. All algorithms were implemented in C. All datasets except KddCup were not normalized. We randomized the datasets that were distributed sorted on the class label (last value in each row of the datasets). For each algorithm and dataset, we measured three average values via five-fold cross-validation. These values are *Accuracy (Acc)*, *Reduction Rate (RR)* and *Pre-processing cost (PC)*.

Acc was estimated by running k-NN classification with $k = 1$ and PC in terms of distance computations. A point worth mentioning is that PC measurements do not include the small cost overhead introduced by the ranking of the prototypes.

All four algorithms presented in Section 4 accept data segments as input. The data segment sizes that were adopted for each dataset are listed in the last column in Table 1, therefore the initial training sets were split into specific segments. Data segment size can relate to the size of the available memory, in the scenario of limited main memory, or to the buffer size accepting data from a streamer. Experiments with data segments of different size were not conducted in this study, due to the fact that in [3] dRHC's performance was not found to be influenced at all by the chosen segment size.

In dRHC2, dRHC2sm and dRHC2mm the maximum allowed condensing set size is provided as an input in form of the T parameter. In order to be comparable with [4], the T parameter was adjusted to the 85%, 70%, 55% and 40% of the size of the condensing sets constructed by dRHC.

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

5.2 Results and Discussion

In Table 2, the performance of the dRHC was compared against dRHCsm and dRHCmm. The prevalent values are highlighted in bold. The pre-processing cost measurements are in million distance computations while values of accuracy and reduction rate are reported as percentages.

Due to the extra cost that is introduced by Algorithm 1, pre-processing cost in dRHC was lower compared to dRHCsm and dRHCmm. In addition to that, since dRHCmm may perform many passes over the condensing set in order to merge all eligible prototype pairs, the pre-processing cost of this algorithm was the highest in all datasets. On the other hand, reduction rate with dRHCsm and, especially, dRHCmm was improved at the cost of a small loss in accuracy in some datasets. Interestingly, in some other datasets (BL, MN2, TN, ECL, YS) accuracy increased, signifying an improvement in the quality of the condensing set after the merging phase.

Table 2. Comparison of dRHC, dRHCsm and dRHCmm in terms of Accuracy (ACC(%)), Reduction Rate (RR(%)) and Preprocessing Cost (PC (millions of distance computations))

	ACC (%)			RR (%)			PC (M)		
Dataset	dRHC	dRHCsm	dRHCmm	dRHC	dRHCsm	dRHCmm	dRHC	dRHCsm	dRHCmm
BL	70.56	**70.88**	**70.88**	78.12	80.88	**81.24**	**0.029**	0.051	0.084
KDD	**99.42**	99.30	99.28	99.22	99.32	**99.33**	**54.70**	56.54	67.94
LS	**88.50**	88.17	88.14	88.35	89.29	**89.42**	**1.53**	2,63	4.74
LIR	**93.92**	92.59	92.53	88.18	90.68	**90.78**	**19.57**	28.14	53.56
MGT	**72.97**	72.23	72.26	74.62	76.46	**76.48**	**26.03**	67.23	162.04
MN2	97.68	**97.91**	**97.91**	96.88	**97.17**	**97.17**	**0.004**	**0.004**	**0.004**
PD	**98.49**	97.19	96.63	97.23	98.07	**98.28**	**1.44**	1.57	1.68
PH	**85.38**	84.49	84.55	82.34	83.61	**83.70**	**1.64**	3.68	7.05
SH	**99.70**	99.36	99.32	99.50	99.60	**99.62**	**7.98**	8.09	8.92
TXR	**97.60**	95.91	95.96	94.95	96.40	**96.63**	**0.68**	0.74	1.01
TN	93.08	**93.34**	91.22	95.37	95.68	**97.29**	**0.695**	0.893	1.691
ECL	71.46	71.73	**71.74**	68.92	69.67	**69.74**	**0.015**	0.029	0.035
PM	**63.93**	63.40	63.41	65.11	66.33	**66.63**	**0.064**	0.210	0.325
YS	48.38	**48.51**	**48.51**	51.23	52.58	**52.63**	**0.306**	0.779	1.394

Similarly, in Table 3 one can compare the performance of dRHC2 against dRHC2sm and dRHC2mm for the different values of T that is provided as an input to the algorithms. We omit the measurements of the reduction rate (RR) since the values of T fix the size of the condensing set as a percentage of the size of the condensing set generated by dRHC. To understand the concept behind the T value, take for example the LIR dataset. In Table 2, one can observe that for the LIR dataset, dRHC achieves Acc=93.92 with RR=88.18 (it practically generates only 2364 prototypes out of the 20000 instances of the dataset). In Table 2, we observe that for T=40 (or by fixing the max size of the condensing set to be 40% of the condensing set produced by dRHC, i.e., the top ranked 946 prototypes) dRHC2, dRHCsm and dRHCmm achieve accuracies 90.08, 90.41 and 90.00 respectively.

As depicted in Table 3, for both dRHC2sm and dRHC2mm higher pre-processing cost values were measured, which is justified by the extra costs introduced with the merging step. Other than this, despite the much smaller con-

densing sets used, accuracy was in most cases slightly affected (negatively or positively) as in the case of the results presented in Table 2 for dRHC.

Table 3. Comparison of dRHC2, dRHC2sm and dRHC2mm in terms of Accuracy (ACC(%)), Reduction Rate (RR(%)) and Preprocessing Cost (PC (millions of distance computations)) taking into account four different values of T

		ACC (%)			PC (M)		
Data	T %	dRHC2	dRHC2sm	dRHC2mm	dRHC2	dRHC2sm	dRHC2mm
LIR	85	**93.40**	92.59	92.53	**19.18**	28.14	53.56
	70	**92.84**	92.44	92.33	**17.72**	27.65	53.47
	55	**91.85**	91.82	91.79	**15.36**	24.06	45.31
	40	90.08	**90.41**	90.00	**12.29**	18.58	32.64
MGT	85	**74.19**	72.89	72.91	**25.85**	67.23	162.04
	70	**74.64**	74.05	74.00	**24.41**	63.21	156.08
	55	**75.11**	74.55	74.13	**21.71**	53.07	124.89
	40	**75.97**	75.48	75.43	**17.73**	39.75	89.54
PD	85	**98.60**	97.19	96.63	**1.41**	1.56	1.68
	70	**98.63**	97.22	96.63	**1.32**	1.37	1.68
	55	**98.34**	97.13	96.53	**1.16**	1.27	1.64
	40	**97.73**	96.95	96.76	**0.93**	1.04	1.33
LS	85	**88.61**	88.07	88.13	**1.51**	2.62	4.72
	70	**88.53**	88.33	88.35	**1.42**	2.46	4.49
	55	**88.58**	87.94	87.96	**1.27**	2.11	3.74
	40	**87.89**	87.62	87.69	**1.03**	1.63	2.72
SH	85	**99.69**	99.38	99.31	**7.61**	7.65	7.71
	70	**99.61**	99.37	99.36	**6.91**	6.92	7.77
	55	**99.56**	99.26	99.36	**5.95**	6.14	6.75
	40	**99.37**	99.28	99.23	**4.73**	4.93	5.23
TXR	85	**97.38**	95.91	95.96	**0.67**	0.74	1.01
	70	**97.00**	96.00	95.98	**0.62**	0.74	1.01
	55	**96.46**	95.87	95.66	**0.53**	0.67	0.94
	40	**95.76**	95.44	95.33	**0.43**	0.53	0.72
PH	85	**86.14**	84.96	85.03	**1.62**	3.65	7.00
	70	85.62	85.59	**85.70**	**1.52**	3.34	6.73
	55	**85.21**	84.97	85.18	**1.33**	2.77	6.04
	40	84.88	**85.36**	84.60	**1.08**	2.06	3.92
BL	85	**71.84**	70.40	70.88	**0.029**	0.050	0.083
	70	73.12	**73.92**	72.32	**0.027**	0.050	0.081
	55	**77.28**	74.56	73.76	**0.025**	0.043	0.070
	40	**81.60**	78.56	78.56	**0.021**	0.035	0.052
PM	85	65.23	**65.62**	65.49	**0.063**	0.200	0.324
	70	**67.96**	65.88	65.88	**0.060**	0.180	0.320
	55	**68.09**	67.44	67.31	**0.055**	0.150	0.260
	40	**68.23**	67.58	**68.23**	**0.046**	0.110	0.194
ECL	85	74.73	**72.95**	**72.95**	**0.015**	0.029	0.035
	70	**76.22**	75.62	75.03	**0.015**	0.027	0.040
	55	**78.28**	76.81	76.51	**0.014**	0.024	0.034
	40	**79.75**	77.70	78.30	**0.013**	0.022	0.026
YS	85	48.31	48.72	**48.78**	**0.306**	0.779	1.394
	70	48.65	**48.92**	**48.92**	**0.306**	0.779	1.394
	55	48.99	**49.46**	49.26	**0.278**	0.682	1.146
	40	**52.83**	51.62	51.62	**0.244**	0.567	1.004
TN	85	**94.03**	94.00	91.22	**0.688**	0.892	1.691
	70	94.54	**94.69**	91.22	**0.654**	0.850	1.691
	55	**95.45**	95.24	91.54	**0.590**	0.746	1.752
	40	**95.93**	95.34	91.54	**0.495**	0.604	1.471
MN2	85	**96.28**	**96.28**	**96.28**	**0.0039**	0.0041	0.0041
	70	96.29	**96.99**	**96.99**	**0.0038**	0.0040	0.0040
	55	**94.45**	90.50	91.66	**0.0038**	0.0040	0.0040
	40	**93.52**	**93.52**	**93.52**	**0.0040**	0.0042	0.0042
KDD	85	**99.47**	99.31	99.27	**53.56**	56.45	68.00
	70	**99.51**	99.38	99.36	**49.81**	53.80	66.53
	55	**99.50**	99.38	99.39	**43.48**	47.32	57.97
	40	**99.48**	99.38	99.40	**34.60**	37.34	43.85

6 Conclusions and Future Work

This paper introduces four new algorithms (dRHCsm, dRHCmm, dRHC2sm and dRHC2mm) that are variations of dRHC and dRHC2. The newly proposed algorithms inherit the characteristics of the dRHC and dRHC2 algorithms while

trying to further condense the training set by merging pairs of prototypes. In theory, we would expect higher reduction rates while maintaining the accuracy at the same level and a slight increase in pre-processing costs, especially for the "mm" versions.

The experimental study, demonstrates that pre-processing costs are significantly higher in the case of the "mm" versions of the merging algorithm compared to the "sm" versions, an increase that is not justified by the small improvement in reduction rate, in the case of dRHC. Overall, the merging of prototypes appears to make more sense in the case of dRHC where it offers increased data reduction without affecting accuracy. In the case of dRHC2, where condensing sets are eventually truncated via ranking, merging of prototypes could be skipped.

In the near future, we plan to further investigate alternative prototype merging algorithms, for example based not only on the proximity of the prototypes but on their weights or age as well. We will examine the effect of the application of prototype merging on additional data reduction algorithms.

7 Acknowledgments

This research is funded by the University of Macedonia Research Committee as part of the "Principal Research 2019" funding program.

References

1. Aggarwal, C.: Data Streams: Models and Algorithms. Advances in Database Systems Series, Springer Science+Business Media, LLC (2007)
2. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theor. 13(1), 21–27 (Sep 2006), http://dx.doi.org/10.1109/TIT.1967.1053964
3. Ougiaroglou, S., Evangelidis, G.: RHC: a non-parametric cluster-based data reduction for efficient k-NN classification. Pattern Analysis and Applications 19(1), 93–109 (2014), http://dx.doi.org/10.1007/s10044-014-0393-7
4. Ougiaroglou S., Arampatzis G., Dervos D.A., Evangelidis G. (2017) Generating Fixed-Size Training Sets for Large and Streaming Datasets. In: Kirikova M., Nrvg K., Papadopoulos G. (eds) Advances in Databases and Information Systems. ADBIS 2017. Lecture Notes in Computer Science, vol 10509. Springer, Cham
5. 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), http://dx.doi.org/10.1109/TSMCC.2010.2103939
6. 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), http://dx.doi.org/10.1109/TPAMI.2011.142
7. M. Lozano. Data Reduction Techniques in Classification processes (Phd Thesis). Universitat Jaume I, 2007

8. Ougiaroglou, S., Evangelidis, G.: Efficient dataset size reduction by finding homogeneous clusters. In: Proceedings of the Fifth Balkan Conference in Informatics. pp. 168–173. BCI '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2371316.2371349
9. 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)