Generating fixed-size training sets for large and streaming datasets

Stefanos Ougiaroglou[1,2], Georgios Arampatzis[1], Dimitris A. Dervos[1], and Georgios Evangelidis[2]

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

Abstract. The k Nearest Neighbor is a popular and versatile classifier but requires a relatively small training set in order to perform adequately, a prerequisite not satisfiable with the large volumes of training data that are nowadays available from streaming environments. Conventional Data Reduction Techniques that select or generate training prototypes are also inappropriate in such environments. Dynamic RHC (dRHC) is a prototype generation algorithm that can update its condensing set when new training data arrives. However, after repetitive updates, the size of the condensing set may become unpredictably large. This paper proposes dRHC2, a new variation of dRHC, which remedies the aforementioned drawback. dRHC2 keeps the size of the condensing set in a convenient, manageable by the classifier, level by ranking the prototypes and removing the least important ones. dRHC2 is tested on several datasets and the experimental results reveal that it is more efficient and noise tolerant than dRHC and is comparable to dRHC in terms of accuracy.

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

1 Introduction

The problem of handling fast data streams [1] or large datasets that cannot reside in main memory has attracted the attention of the Data Mining and Machine Learning research communities. Moreover, researchers focus on how to run data mining algorithms on devices with limited memory (e.g., sensor devices) avoiding data transferring costs to powerful processing servers. Classification is a typical data mining task that has many applications on all aforementioned environments.

Classification algorithms (or classifiers) try to assign unclassified items to a set of predefined classes, on the basis of the available training dataset, i.e., a set of already classified items. Classifiers can be either model-based (eager) or instance-based (lazy). Both aim at accurate class prediction but they differ on how they

work. An eager classifier pre-processes the training set and builds a model that is then used to classify unclassified items. In contrast, lazy classifiers do not build any model. They classify a new item by scanning the whole training set. The size and the quality of the training set is vital for both types of classifiers. These factors determine the effectiveness and the efficiency of the classifier. However, if the size of the training set is very large, the usage of any classifier is prohibitive due to the high computational cost involved.

k-Nearest Neighbors (k-NN) is a well-known and extensively used lazy classifier [5]. When an unclassified item arrives, the algorithm scans the available training data and retrieves the k nearest items or neighbors to it according to a distance metric (e.g., Euclidean distance). Then, the unclassified item is assigned to the most common class among the classes of the k nearest neighbors.

The k-NN classifier is an effective classifier especially when it is used on small training sets. For larger training sets, its performance degrades because all distances between the new item and the training data items must be computed. In addition, contrary to the eager classifiers that can discard the training data after the construction of the classification model, the k-NN classifier has higher storage requirements since it must have the training set always available. In addition, k-NN classifier is not noise tolerant. Noise misleads the classifier and downgrades classification accuracy. The application of a Data Reduction Technique (DRT) that builds a small representative (condensing) set of the initial training data as a preprocessing step can deal with these weaknesses.

Since the k-NN classifier does not build any classification model, it can be easily adapted in streaming environments [1]. However, it can be used only on a portion of a training stream (e.g., a data window thresholded by the user). Another approach could be the maintenance of a condensing set built from the data stream.

Dynamic RHC (dRHC) [12] is a DRT that incrementally builds its condensing set. A new training data segment can be used to update an existing condensing set without applying the prototype generation procedure from scratch over the complete training data (new and old training data). Therefore, dRHC is appropriate for dynamic environments where new training data becomes gradually available and for very large datasets that can not fit in main memory.

The experimental results presented in [12] demonstrate that dRHC is a fast DRT, achieves high reduction rates and does not degrade the classification accuracy achieved by the k-NN classifier on the original training set. However, after repetitive updates of the condensing set, its size may exceed the size of the available memory. This observation is behind the motivation of the present work. The contribution is the development of a new version of dRHC that remedies this drawback by keeping the size of the condensing set fixed. In particular, the paper proposes dRHC2 that ranks the prototypes and removes the weakest ones when the size of the condensing set exceeds a predefined threshold value. The experimental study shows that dRHC2 is faster than dRHC while keeping accuracy at high levels.

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 algorithm. Section 4 considers in detail the proposed dRHC2 algorithm. In Section 5, both algorithms are experimentally compared to each other on fourteen datasets. The experimental results are validated by the Wilcoxon signed rank test. Section 6 concludes the paper and proposes directions for future work.

2 Background knowledge

Data Reduction Techniques can be grouped into two main categories: (i) Prototype Selection (PS) algorithms that collect representative items (or prototypes) from the initial training set [8] , and, (ii) Prototype Generation (PG) algorithms that generate prototypes by summarizing on similar items [15]. PS algorithms can be either condensing or editing. The latter aim for improved classification accuracy by removing noise from the training data and by "cleaning" the decision boundaries between the discrete classes. On the other hand, PS-condensing and PG algorithms aim for data condensation, i.e., the construction of a condensing set of the initial training data.

Most PS-condensing and PG algorithms are based on a simple observation: the items that do not define the decision boundaries between classes can be removed without loss of accuracy. Consequently, PS-condensing algorithms try to collect only the items that are close to the decision boundaries (border items). On the other hand, PG algorithms generate many prototypes for the close-border areas and few for the "internal" data areas. It is worth mentioning that most of the PG and PS-condensing algorithms are sensitive to noise. Hence, for training sets with noise, an editing algorithm must be applied beforehand.

A great number of DRTs have been proposed in the literature. PS and PG algorithms are reviewed, categorized and compared to each other in [15] and [8], respectively. Although there are some exceptions (e.g., IBL algorithms [2, 4]), DRTs are usually memory-based. This implies that the whole training set must reside in main memory. This property renders DRTs inappropriate for very large training sets that cannot fit into the device's memory or for devices with limited memory (e.g., sensor devices) without transferring data to a server over the network for processing.

Furthermore, these DRTs cannot consider new training items after the construction of the condensing sets, i.e., they cannot update their condensing set in a dynamic manner. Suppose that a DRT is applied over a training set TS and builds a condensing set. Moreover, suppose that new training items D become available. For the construction of an updated condensing set, the DRT must run from scratch over the complete training set $TS \cup D$. Therefore, all training items must always be available. Hence, DRTs are inappropriate for dynamic/streaming environments where new training items become gradually available. The Dynamic RHC is a PG algorithm that can be used in dynamic/streaming environments [12].

3 The dRHC algorithm

Dynamic RHC (dRHC) is a descendant of the Reduction through Homogeneous Clusters (RHC) algorithm [12, 11]. The latter is based on the concept of cluster homogeneity. RHC utilizes k-means clustering. Initially, it considers the training set with D classes as a non-homogeneous cluster C. The algorithm computes a class-mean for each class in C by averaging on the corresponding items. Then, k-means clustering is applied on C by using these class-means as initial seeds and D clusters are formed. Each item in C is assigned to one of the D clusters. Then, RHC examines the D clusters. If a cluster is homogeneous (i.e., has items of only one class), its cluster-mean constitutes a prototype and is placed in the condensing set. If a cluster is non-homogeneous, the aforementioned procedure is applied recursively on it. RHC terminates when there are no non-homogeneous clusters left. In other words, each homogeneous cluster contributes a prototype.

RHC builds many prototypes for close-border areas and fewer for the "internal" areas. By using the class-means as initial seeds for k-means clustering, quick discovery of large homogeneous clusters is feasible (the larger clusters discovered, the higher reduction rates achieved). The main disadvantage of RHC is that it is memory-based. Moreover, it cannot cope with training sets that cannot fit in memory or streaming data.

The dRHC algorithm retains all the properties of RHC. In addition, it can also manage large or streaming datasets by considering the available data in the form of data segments. The application of dRHC involves two phases (see Figure 1): (i) *initial condensing set construction* and (ii) *condensing set update*. The *initial condensing set construction* phase is executed only once, when the first data segment arrives. The following data segments are processed by the *condensing set update* phase. The *initial condensing set construction* phase is almost identical to RHC. The only difference is that each prototype of the Condensing Set (CS) stores a weight attribute that counts how many items are represented by the specific prototype. The *condensing set update* phase uses the prototypes of the current condensing set and the items of a new data segment in order to build a set of initial clusters and then it proceeds similarly to RHC.

The *condensing set update* phase can be easily understood by considering Algorithm 1. The algorithm has two input parameters: an already constructed (old) condensing set (OCS) and a data segment (Seg) with new training items. The output is an updated (new) condensing set (NCS). The *condensing set update* phase utilizes a queue (Q) data structure to hold the unprocessed clusters (lines 2–3). Initially, the *condensing set update* phase initializes as many clusters, as the number of prototypes in OCS (lines 3–6). Each cluster contains only the prototype. Then, for each item x in Seg, the algorithm identifies the nearest prototype and assigns x to the corresponding cluster (lines 7–11). All items in Seg have $weight = 1$. Hence, each cluster contains an old prototype with $weight >= 1$ and items (from Seg) with $weight = 1$. The clusters are enqueued to Q for further processing (lines 12–14).

The *condensing set update* phase generates the updated condensing set NCS considering the weight values. For each homogeneous cluster, it computes the

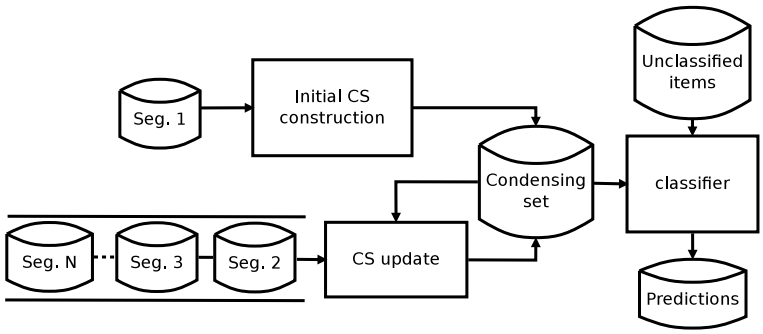


Fig. 1. Classification using dRHC

weighted mean of the cluster (lines 19–22). For each non-homogeneous cluster C, a weighted mean is estimated for each class in C (lines 24–29). Like RHC, dRHC utilizes the weighted class means as initial seeds for k-Means clustering (line 30). Of course, dRHC uses the version of k-means that computes the cluster means by considering the corresponding weights. Thus, a vector attribute a_j, $j = 1, 2, \ldots, n$ of a class or cluster mean m_C (lines 20, 26) is computed by the following formula:

$$m_C.a_j = \frac{\sum_{x_i \in C} x_i.a_j \times x_i.weight}{\sum_{x_i \in C} x_i.weight}$$

k-means clustering builds as many clusters as the number of different classes in C. They are enqueued to Q for further processing. The repeat-until loop (lines 17,35) ends when all clusters become homogeneous, i.e., when Q is empty. Note that old prototypes usually have weights greater than one and have higher influence in the computation of a new weighted class or cluster mean than any item of a new data segment, whose weight is one.

An example of the execution of the *condensing set update* phase is depicted in Figure 2. More specifically, Figure 2(a) presents an existing condensing set created by either the *initial CS construction* phase or by a previously executed *condensing set update* phase. The condensing set has three prototypes with the corresponding weights. Suppose that a segment with seven new training items is available and is about to be processed (Figure 2(b)). Each new item has a weight equal to one. The first step is the assignment of each new item to the cluster of the nearest prototype (Figure 2(c)). The new items assigned to cluster A have the same class as the class of the prototype in A. Therefore, the prototype "moves" towards the new items (Figure 2(d)). This is achieved by computing the weighted mean in A. The latter constitutes the new prototype and is placed in the condensing set along with its new weight. On the other hand, there is no item that has been assigned to cluster B. Hence, the corresponding prototype remains unchanged. Cluster C becomes non-homogeneous. For each class in C,

Algorithm 1 dRHC: condensing set update phase

Input: OCS, Seg
Output: NCS

```
{Queue Initialization}
Q ← ∅
CLi ← ∅ {empty list of clusters}
for each prototype m ∈ OCS do
    add new cluster C = {m} in CLi
end for
for each item x ∈ Seg do
    x.weight = 1
    find the Cluster C_x ∈ CLi with the nearest to x prototype
    C_x ← C_x ∪ {x} {The mean of C_x is not recomputed}
end for
for each cluster C in CLi do
    Enqueue(Q, C)
end for
{Construction of NCS}
NCS ← ∅
repeat
    C ← Dequeue(Q)
    if C is homogeneous then
        m ← weighted mean of C
        m.weight ← Σ_{x_i ∈ C} x_i.weight
        NCS ← NCS ∪ {m}
    else
        M ← ∅ {M is the set of weighted class means}
        for each class L in C do
            m_L ← weighted mean of L
            m_L.weight ← Σ_{x_i ∈ L} x_i.weight
            M ← M ∪ {m_L}
        end for
        NewClusters ← k-means(C, M)
        for each cluster NC ∈ NewClusters do
            Enqueue(Q, NC)
        end for
    end if
until IsEmpty(Q)
return NCS
```

a weighted class mean is computed, k-means is executed and two homogeneous clusters are built (see Figure 2(d,e)). Finally, a weighted cluster mean for each cluster is computed and its weight is estimated. They constitute new prototypes and they are placed in the condensing set (Figure 2(f)).

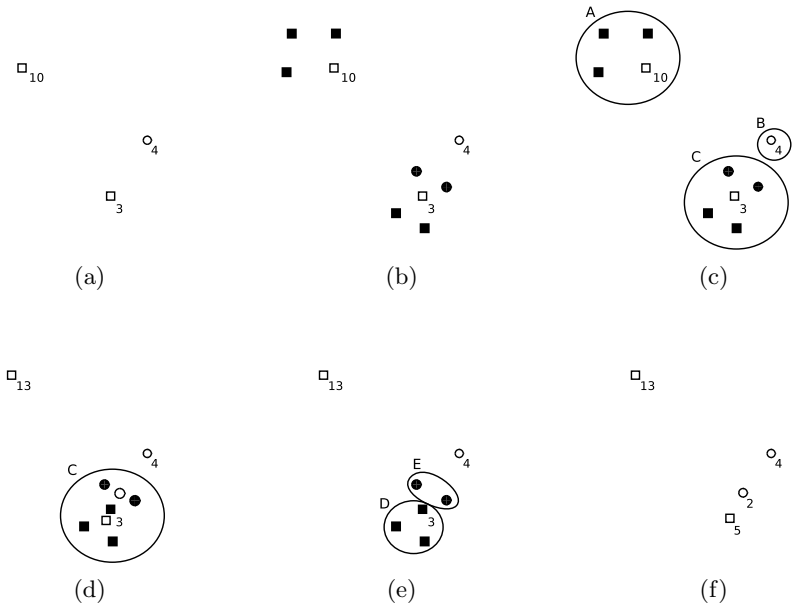


Fig. 2. Example of execution of the *condensing set update* phase of dRHC

In the experimental study presented in [12], RHC and dRHC were compared to each other and against state-of-the-art PS [9, 2, 10, 17] and PG [14] algorithms. It turns out that dRHC is the fastest algorithm (with the lowest preprocessing cost) and builds the smallest condensing set without loss of classification accuracy.

4 The proposed dRHC2 algorithm

Although dRHC seems to be appropriate for large training datasets or streaming environments, it has a major drawback: after repetitive executions of the *condensing set update* phase, the condensing set may become very large. Therefore, there is a need to keep the size of the condensing set fixed. The dRHC2 algorithm is a new version of dRHC that copes with this drawback.

dRHC2 accepts an input parameter that is the desirable maximum size of the condensing set. In effect, dRHC2 executes similarly to dRHC, but when

the size of the condensing set exceeds the maximum size (T), the algorithm removes prototypes to save space. In other words, dRHC maintains the size of the condensing set equal to T by removing the least important prototypes from the updated condensing set. Thus, dRHC2 introduces a post-processing step in the *condensing set update phase* presented in Section 3. T is adjusted by the user taking into account the trade-off between accuracy and computational cost, the level of noise in the data and of course the system limitations.

The dRHC2 algorithm includes a mechanism that ranks the prototypes according to their importance. Next, it retains only the T first prototypes and removes the rest. An essential role to the determination of the prototypes' "importance" plays the concept of prototype weight. One may claim that a prototype with high weight value is more important than a prototype with lower weight. This is not absolutely true. If a prototype was generated in a recent *condensing set update* phase, it probably has a lower weight value than an older prototype. Therefore, it is doomed to be discarded. On the other hand, an old prototype that has survived many executions of the *condensing set update* phase, probably has a high weight value. Thus, the latter tends to be favored against the new prototypes and will survive in the condensing set. Consequently, if dRHC2 ranks the prototypes according to their weight, it may discard most of the prototypes arriving in later stages.

In order to more appropriately rank the prototypes, dRHC2 keeps a counter of the data segments that have arrived and adds an extra attribute to each prototype that denotes the number (r) of the data segment on which the prototype was generated. After each execution of the *condensing set update* phase, dRHC2 ranks the prototypes according to a measure called AnA, which stands for Average number of Arrivals and takes into account the weight and the age of each prototype. In effect, AnA estimates the prototype weight per data segment. AnA is computed as follows:

$$AnA = \frac{w}{ds - r + 1}$$

where

- w is the prototype weight that is the number of items summarized together and represented by the specific prototype
- r is the number of the data segment on which the prototype was generated
- ds is the number of the current data segment

By using AnA, no prototypes are favored in the ranking. The AnA of the most recent prototypes, i.e., the ones created by the last *condensing set update* phase, is equal to their w. On the other hand, the AnA of older prototypes is practically their w divided by their age.

Algorithm 4 summarizes the post-processing step introduced by dRHC2. This step is the only difference between dRHC and dRHC2. The algorithm accepts the set maximum condensing set size T and the condensing set produced by either the *initial condensing set construction* phase or a *condensing set update* phase (NCS) as input parameters. When $|NCS| > T$, the algorithm returns a new condensing set NCS with $|NCS| = T$.

Algorithm 2 dRHC2: Post-processing step

Input: NCS, T
Output: NCS
1: **if** $|NCS| > T$ **then**
2: $NCS \leftarrow$ keep the T prototypes with the highest AnA
3: **end if**
4: **return** NCS

In case of data with noise, dRHC inevitably generates prototypes that represent noisy items. However, these prototypes usually have low weight and AnA values. If we adopt dRHC2 instead of dRHC, these prototypes are eventually removed. Therefore, we expect dRHC2 to handle noisy training data and, in such cases, achieve higher accuracy than dRHC.

One can claim that dRHC2 maintains a condensing set that adapts according to the concept drift [16] that may exist in the data stream. This assumption may not be absolutely correct. The newly generated prototypes or the prototypes that are updated after a *condensing set update* are not favored against the older ones.

5 Performance Evaluation

5.1 Experimental setup

The performance of dRHC2 was tested against dRHC using fourteen datasets distributed by the KEEL dataset repository[3] [3]. Table 1 summarizes on the datasets used. In the experimental study presented in [12], RHC and dRHC were evaluated against five state-of-the-art data reduction techniques [9, 2, 10, 17, 14] on the same fourteen datasets. Therefore, we "indirectly" compare dRHC2 to these techniques. The reader can execute dRHC and dRHC2 over the particular datasets using WebDR[4] [13]

The Euclidean distance was adopted as the distance metric. All algorithms were implemented in C. Except KDD, all the other datasets were used without normalization. We randomized the datasets that were distributed sorted on the class label. For each dataset and algorithm, three average measurements obtained via five-fold cross-validation were estimated: (i) Accuracy (ACC), (ii) Reduction Rate (RR), and, (iii) Preprocessing Cost (PC) in terms of distance computations. To be fair, we should notice that PC measurements do not include the small cost overhead introduced by the ranking of the prototypes. Accuracy was estimated by running k-NN classification with $k = 1$. Note that we did not use special evaluation methods for data streams (e.g., test-then-train) [7], since the used datasets do not exhibit concept drift.

For KDD, we removed the three nominal and the two fixed-value attributes that exist in the dataset. Moreover, KDD contains many duplicates that were

[3] `http://sci2s.ugr.es/keel/datasets.php`
[4] `https://atropos.uom.gr/webdr`

Table 1. Dataset description

Dataset	Size	Attributes	Classes	Memory/ Buffer size
Letter Image Recognition (LIR)	20,000	16	26	2,000
Magic G. Telescope (MGT)	19,020	10	2	1,902
Pen-Digits (PD)	10,992	16	10	1,000
Landsat Satellite (LS)	6,435	36	6	572
Shuttle (SH)	58,000	9	7	1,856
Texture (TXR)	5,500	40	11	440
Phoneme (PH)	5,404	5	2	500
Balance (BL)	625	4	3	100
Pima (PM)	768	8	2	100
Ecoli (ECL)	336	7	8	200
Yeast (YS)	1,484	8	10	396
Twonorm (TN)	7,400	20	2	592
MONK 2 (MN2)	432	6	2	115
KddCup (KDD)	141,481	36	23	4,000

also removed. The attribute ranges of KDD vary extremely. We normalized them to the $[0, 1]$ range.

The dRHC and dRHC2 algorithms consume data in the form of data segments. Thus, the training sets of the datasets were split into segments of a specific size. The data segment that we adopted for each dataset is presented by the last column in Table 1. The segment size corresponds to either the size of the buffer that accepts data from a stream or the size of the available memory (scenario of limited main memory and/or large datasets). The experimental study presented in [12] shows empirically that the size of the data segment does not affect the performance of dRHC. Therefore, we did not conduct experiments with different segment sizes. The performance measurements were estimated following the arrival of all the data segments.

The dRHC2 algorithm accepts as an input parameter the set maximum condensing set size (T). For each dataset, T was adjusted to a certain percentage of the size of the condensing set generated by dRHC. The T values chosen were the 85%, 70%, 55% and 40% of the size of the condensing sets constructed by dRHC. In other words, when dRHC builds a condensing set with 1000 prototypes, four experiments were conducted to evaluate the performance of dRHC2 with the following T values: $T = 850$, $T = 700$, $T = 550$, $T = 400$.

5.2 Comparisons

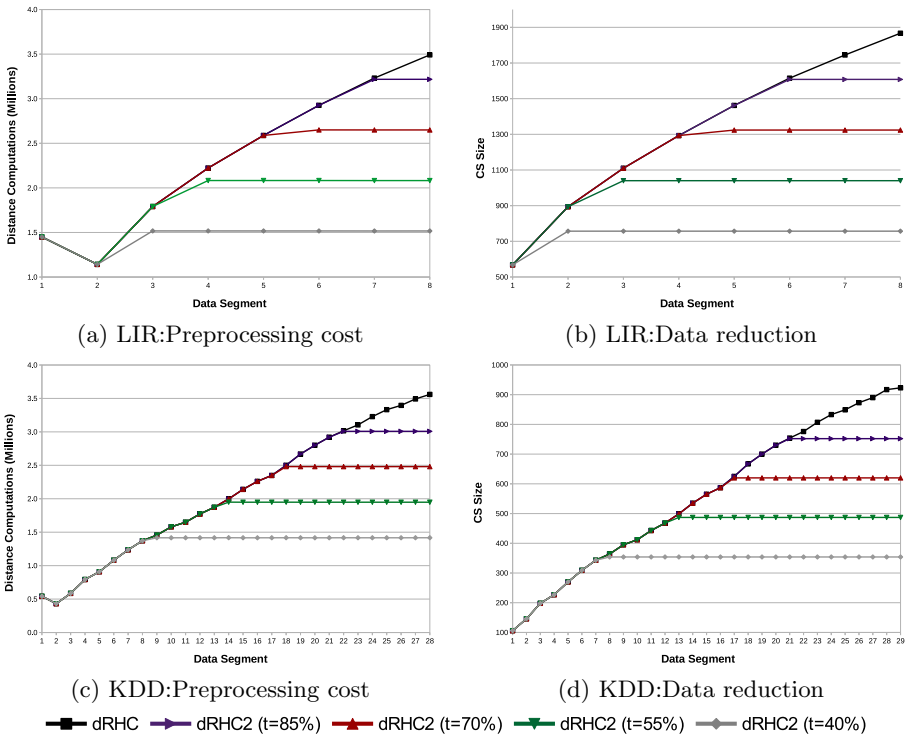
Table 2 presents the performance measurements of dRHC and dRHC2. The best measurements are highlighted in boldface. The preprocessing cost measurements are in million distance computations. Although, both dRHC and dRHC2 are adopted when the conventional k-NN classifier cannot be applied due to its limitations, for reference, we report the accuracy measurements achieved by applying k-NN on the original complete training set.

Obviously, the lower the T value used, the higher reduction rates achieved, the lower preprocessing cost needed and, of course, the faster the classifier is. Therefore, the results that concern RR and PC measurements are to be expected

Table 2. Comparison of dRHC2 and dRHC in terms of Accuracy (ACC(%)), Reduction Rate (RR(%)) and Preprocessing Cost (PC (millions of distance computations))

		ACC (%)			**RR (%)**		**PC (M)**	
Data	***T* %**	**1NN**	**dRHC**	**dRHC2**	**dRHC**	**dRHC2**	**dRHC**	**dRHC2**
LIR	85:	95.83	**93.92**	93.40	88.18	89.95	19.57	19.18
	70:			92.84		91.73		17.72
	55:			91.85		93.50		15.36
	40:			90.08		95.27		12.29
MGT	85:	78.14	72.97	74.19	74.62	78.42	26.03	25.85
	70:			74.64		82.24		24.41
	55:			75.11		86.04		21.71
	40:			**75.97**		89.95		17.73
PD	85:	99.35	98.49	98.60	97.23	97.65	1.44	1.41
	70:			**98.63**		98.06		1.32
	55:			98.34		98.48		1.16
	40:			97.73		98.90		0.93
LS	85:	90.60	88.50	**88.61**	88.35	90.09	1.53	1.51
	70:			88.53		91.84		1.42
	55:			88.58		93.59		1.27
	40:			87.89		95.34		1.03
SH	85:	99.82	**99.70**	99.69	99.50	99.58	7.98	7.61
	70:			99.61		99.65		6.91
	55:			99.56		99.72		5.95
	40:			99.37		99.80		4.73
TXR	85:	99.02	**97.60**	97.38	94.95	95.71	0.68	0.67
	70:			97.00		96.46		0.62
	55:			96.46		97.23		0.53
	40:			95.76		97.98		0.43
PH	85:	90.10	85.38	**86.14**	82.34	84.99	1.64	1.62
	70:			85.62		87.65		1.52
	55:			85.21		90.29		1.33
	40:			84.88		92.95		1.08
BL	85:	78.40	70.56	71.84	78.12	81.40	0.029	0.029
	70:			73.12		84.60		0.027
	55:			77.28		88.00		0.025
	40:			**81.60**		91.20		0.021
PM	85:	68.36	63.93	65.23	65.11	70.41	0.064	0.063
	70:			67.96		75.61		0.060
	55:			68.09		80.81		0.055
	40:			**68.23**		86.02		0.046
ECL	85:	79.78	71.46	74.73	68.92	73.61	0.015	0.015
	70:			76.22		78.44		0.015
	55:			78.28		82.90		0.014
	40:			**79.75**		87.73		0.013
YS	85:	52.02	48.38	48.31	51.23	58.59	0.306	0.306
	70:			48.65		65.91		0.306
	55:			48.99		73.23		0.278
	40:			**52.83**		80.47		0.244
TN	85:	94.88	93.08	94.03	95.37	96.06	0.695	0.688
	70:			94.54		96.76		0.654
	55:			95.45		97.45		0.590
	40:			**95.93**		98.14		0.495
MN2	85:	90.51	**97.68**	96.28	96.88	97.63	0.0040	0.0039
	70:			96.29		97.80		0.0038
	55:			94.45		98.27		0.0038
	40:			93.52		98.84		0.0040
KDD	85:	99.71	99.42	99.47	99.22	99.34	54.70	53.56
	70:			**99.51**		99.46		49.81
	55:			99.50		99.58		43.48
	40:			99.48		99.69		34.60
AVG	85:	86.89	84.36	84.84	84.29	86.67	8.19	8.04
	70:			85.22		89.01		7.49
	55:			85.51		91.36		6.55
	40:			**85.93**		93.73		5.26

since dRHC2 adopts a ceiling value for the maximum size of its condensing set and maintains it throughout the whole execution. Figure 3 depicts this property of dRHC2 by presenting diagrams for two indicative datasets. The diagrams illustrate how the preprocessing cost and the size of the condensing set initially increases and remains constant when T is reached.



(a) LIR:Preprocessing cost (b) LIR:Data reduction

(c) KDD:Preprocessing cost (d) KDD:Data reduction

Fig. 3. Processing per data segment

The ACC performance measurements for dRHC2 are most promising (see in Table 2). In cases of datasets with low level of noise, dRHC2 achieves accuracy comparable to that of dRHC. On the other hand, in cases of datasets with a high level of noise, dRHC2 achieves higher accuracy than dRHC. This happens because some less important dRHC prototypes originate from noisy data. These prototypes probably have low AnA values, they are ranked low, and they get discarded, eventually. Hence, in cases of datasets with high level of noise, we observe that dRHC2 with the lowest T value achieves even higher accuracy than the conventional k-NN classifier.

One final comment concerning the average measurements depicted in the last row of Table 2: dRHC2 can achieve even better accuracy than dRHC by

avoiding the arbitrary growth in size of the condensing set, and by reducing the preprocessing cost.

5.3 Wilcoxon signed rank test

The experimental study is complemented by providing the results of Wilcoxon signed rank test [6] to statistically validate the ACC measurements presented in Table 2. The test compares dRHC and dRHC2 in pairs considering the performance on each dataset. All four versions of dRHC2 (with different T values) were included in the test. Since dRHC2 is the dominant algorithm in terms of RR and PC, there is no need to include the corresponding measures in the test.

Table 3 presents the results of the Wilcoxon test. Columns labeled with "w/l" show the number of wins and losses, respectively. The Wilcoxon value ("Wilc." column) shows how significant the difference between the algorithms is. If it is lower than 0.05, one can claim that the difference is statistically significant. The results of the test reveal that dRHC and dRHC2 do not statistically differ in terms of accuracy. Thus, dRHC2 can be used instead of dRHC when there is need for a condensing set with a fixed size.

Table 3. Results of Wilcoxon signed rank test

Methods	**Accuracy**	
	w/l	**Wilc.**
dRHC vs dRHC2 (T=85%)	5/9	0.158
dRHC vs dRHC2 (T=70%)	4/10	0.103
dRHC vs dRHC2 (T=55%)	6/8	0.363
dRHC vs dRHC2 (T=40%)	7/7	0.397

6 Conclusions and future work

The paper reports on dRHC2, a noise-tolerant PG algorithm that maintains a fixed size condensing set by monitoring a training data stream or by managing a large dataset which cannot reside in memory. The experimental study yields promising results. Even when the condensing set generated by dRHC2 is less than half the size of that generated by dRHC, there is no loss in accuracy and in many cases the accuracy achieved by dRHC2 is even higher. In addition, because the size of the condensing set is not arbitrary increased and remains constant, the preprocessing cost remains low and constant till the end of the execution.

Our plans for future work include the development of variations of dRHC2 that will be able to fully handle data streams with concept drift. This could be achieved by increasing the importance of newly generated prototypes.

References

1. Aggarwal, C.: Data Streams: Models and Algorithms. Advances in Database Systems Series, Springer Science+Business Media, LLC (2007)

2. Aha, D.W., Kibler, D., Albert, M.K.: Instance-based learning algorithms. Mach. Learn. 6(1), 37–66 (Jan 1991), http://dx.doi.org/10.1023/A:1022689900470
3. 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)
4. Beringer, J., Hüllermeier, E.: Efficient instance-based learning on data streams. Intell. Data Anal. 11(6), 627–650 (Dec 2007), http://dl.acm.org/citation.cfm?id=1368018.1368022
5. 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
6. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. 7, 1–30 (Dec 2006), http://dl.acm.org/citation.cfm?id=1248547.1248548
7. Gama, J.a., Sebastião, R., Rodrigues, P.P.: Issues in evaluation of stream learning algorithms. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 329–338. KDD '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1557019.1557060
8. 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
9. Hart, P.E.: The condensed nearest neighbor rule. IEEE Transactions on Information Theory 14(3), 515–516 (1968)
10. Olvera-Lopez, J.A., Carrasco-Ochoa, J.A., Trinidad, J.F.M.: A new fast prototype selection method based on clustering. Pattern Anal. Appl. 13(2), 131–141 (2010)
11. 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
12. 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
13. Ougiaroglou, S., Evangelidis, G.: WebDR: A web workbench for data reduction. In: Calders, T., Esposito, F., Hüllermeier, E., Meo, R. (eds.) Machine Learning and Knowledge Discovery in Databases, Lecture Notes in Computer Science, vol. 8726, pp. 464–467. Springer Berlin Heidelberg (2014), http://dx.doi.org/10.1007/978-3-662-44845-8_36
14. Sánchez, J.S.: High training set size reduction by space partitioning and prototype abstraction. Pattern Recognition 37(7), 1561–1564 (2004)
15. 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
16. Tsymbal, A.: The problem of concept drift: definitions and related work. Tech. Rep. TCD-CS-2004-15, The University of Dublin, Trinity College, Department of Computer Science, Dublin, Ireland (2004)
17. Wilson, D.R., Martinez, T.R.: Reduction techniques for instance-based learning algorithms. Mach. Learn. 38(3), 257–286 (Mar 2000), http://dx.doi.org/10.1023/A:1007626913721