Fast Data Reduction by Space Partitioning via Convex Hull and MBR computation

Thomas Giorginis[a], Stefanos Ougiaroglou[b,c,*], Georgios Evangelidis[c], Dimitris A. Dervos[a]

[a] *Dept. of Information and Electronic Engineering, School of Engineering, International Hellenic University, 57400 Sindos, Greece*
[b] *Dept. of Digital Systems, School of Economics and Technology, University of the Peloponnese, 23100 Sparta, Greece*
[c] *Dept. of Applied Informatics, School of Information Sciences, University of Macedonia, 54636 Thessaloniki, Greece*

Abstract

Large volumes of training data introduce high computational cost in instance-based classification. Data reduction algorithms select or generate a small (condensing) set of representative training prototypes from the available training data. The Reduction by Space Partitioning algorithm is one of the most well-known prototype generation algorithms that repetitively divides the original training data into subsets. This partitioning process needs to identify the diameter of each subset, i.e., its two farthest instances. This is a costly process since it requires the calculation of all distances between the instances in each subset. The paper introduces two new very fast variations that, instead of computing the actual diameter of a subset, choose a pair of distant-enough instances. The first variation uses instances belonging to an exact 3d convex hull of the subset, while the second one uses instances belonging to the minimum bounding rectangle of the subset. Our experimental study shows that the new variations vastly outperform the original algorithm without a penalty in classification accuracy and reduction rate.

*Corresponding author
Email addresses: giotoman123@gmail.com (Thomas Giorginis), stoug@uop.gr (Stefanos Ougiaroglou), gevan@uom.edu.gr (Georgios Evangelidis), dad@ihu.gr (Dimitris A. Dervos)
URL: https://www.iee.ihu.gr/~stoug (Stefanos Ougiaroglou), http://users.uom.gr/~gevan (Georgios Evangelidis), https://www.iee.ihu.gr/~dad (Dimitris A. Dervos)

Keywords: `Reduction by Space Partitioning, RSP3, Classification, Prototype generation, Big training data, Convex Hull, Minimum Bounding Rectangle (MBR)`
2010 MSC: 00-01, 99-00

1. Introduction

Large volumes of training data become available on a daily basis and their handling has attracted the interest of both academia and industry. The major motive behind many research efforts is the reduction of the high computational cost involved in the processing of large datasets. The problem is intense in instance-based classifiers because, for each new instance to be classified, all the available training instances must be examined. The k-Nearest Neighbors (k-NN) [1] classifier is a typical example of an instance-based classifier that can not deal with large volumes of training data. It classifies a new instance by retrieving its k nearest training instances (neighbors). The new instance is assigned to the most common class among the classes of its k nearest neighbors. Although the task seems to be simple, it is CPU intensive and has memory requirements for storing the training data.

There are various approaches for handing large data volumes in instance-based classification. Indexing can be used to avoid exhaustive searches in the training set, however, indexing becomes problematic with high dimensional data [2]. Data dimensionality could be lowered by applying a feature selection or feature extraction technique, but, such approaches in many cases harm classification accuracy. An approach that does not alter the original feature space is to adopt a data reduction technique that selects or generates a small but sufficient subset of representatives from the original training data. In this paper, we focus on the latter approach.

A simple data reduction technique can be a random stratified sampling to construct a much smaller training set. However, the latter may not be representative of the original dataset and, as a consequence, accuracy may be harmed.

Thus, data reduction aims to produce a small set that is as much as possible representative of the initial (large) training set. The small representative set is called the condensing set and it has the benefits of low computational cost and low storage requirements.

Data Reduction Techniques (DRTs) [3, 4] pre-process the original training set and produce a condensing set to be used by the k-NN classifier. DRTs can be either Prototype Selection (PS) [3] or Prototype Generation (PG) [4] algorithms. PS algorithms create their condensing set by retaining some representative instances, also called prototypes, from the original training set. On the other hand, PG algorithms generate prototypes by summarizing instances. The goal is to select or to generate as few as possible prototypes that represent the original data. Both approaches are based on the following idea: The instances that define the boundaries between the different classes in the data space are the only essential instances for the classification. Therefore, the instances that lie on data areas far from the boundaries can be discarded without loss of accuracy. This paper focuses on the PG category of DRTs.

The Reduction by Space Partitioning (RSP3) [5] is one of the most popular and effective PG algorithms. It is simple and parameter-free, properties that are desirable to researchers and practitioners. It is based on a repetitive partitioning process that divides the training set into homogeneous subsets (i.e., containing instances of a single class). In effect, RSP3 keeps dividing non-homogeneous subsets until all of them become homogeneous. For each homogeneous subset, a prototype is generated by averaging the instances of that subset. The prototype is stored in the condensing set by replacing all instances of that subset. To divide a non-homogeneous subset, RSP3 finds the pair of farthest instances (seeds) in the subset by computing all the possible distances between its instances.

In this paper, we illustrate that the choice of seeds for dividing a non-homogeneous subset in RSP3 is a computationally expensive task that renders the use of RSP3 inefficient for time-sensitive environments and prohibitive for large datasets. The motivation of our research was to propose fast methods for choosing the seeds during subset divisions and thus, transform RSP3 to a fast

PG algorithm.

The contribution of the paper is the development of mechanisms that exploit the notions of Convex Hull [6] and Minimum Bounding Rectangle (MBR) to speed-up via approximation the aforementioned computationally expensive task. In the recent literature, Convex Hulls [7, 8, 9] and Convex Cones [10] are employed in classification tasks in the pattern recognition domain.

The paper proposes two new RSP3 variations. The first variation computes an approximate convex hull for each subset by utilizing the Quick hull algorithm in the 3d space [11]. For datasets with up to three dimensions the algorithm computes the exact convex hull. For datasets with more dimensions, the convex hull is approximated by considering only the three most significant dimensions, i.e., the ones chosen by an attribute selection method. Notice that the generated prototypes are still in the original dimensionality. The second variation approximates the convex hull of a subset using instances that belong to its Minimum Bounding Rectangle (MBR). The experimental results show that the new variations reduce pre-processing computational cost to a minimum level without harming accuracy and reduction rate. The results obtained are validated by conducting non-parametric Friedman tests with Conover's Post Hoc Comparisons.

The rest of the paper is organized as follows: Section 2 briefly reviews the recent research conducted in the field of PG algorithms. This short review reveals that RSP3 is cited in the most recent research and this is an additional motive behind the present paper. Section 3 reviews the family of Reduction by Space Partitioning algorithms. Section 4 explains how convex hulls and minimum bounding rectangles can be exploited in order to speed-up the RSP3 algorithms and, then, considers in detail the proposed RSP3 variations. Section 5 presents and discusses the contacted experimental study, and finally, Section 6 concludes the paper.

2. Related work

Many Prototype Generation (PG) algorithms have been proposed in the past decades. The explosion of Big Data maintains the research field active and challenging. [4] reviews all PG algorithms that had been proposed until 2012 and includes a taxonomy and an experimental study where they are compared to each other. The results reveal that RSP3 is a PG algorithm that achieves high accuracy. This section reviews the most recent research efforts, i.e., the ones that became available since 2012.

Reduction through Homogeneous Clustering (RHC) [12] is a fast PG algorithm based on k-means clustering. Like RSP3, RHC is based on the concept of homogeneity. Initially, RHC considers the whole training set as a non-homogeneous cluster and it computes a mean instance for each class present in the cluster. Then, it uses those mean instances as initial seeds for k-means clustering. If a homogeneous cluster (i.e., a cluster with instances of a single class) is discovered, the mean of the cluster constitutes a prototype. For each non-homogeneous cluster, the aforementioned procedure is applied recursively. RHC terminates when all discovered clusters are homogeneous and the final condensing set is the set of the means of the homogeneous clusters. The experimental results presented in [12] show that RSP3 is more accurate than RHC, but RHC is much faster and achieves higher reduction rates than RSP3. Moreover, RHC is one of the fastest approaches in the experimental study. It is worth mentioning that RHC was recently modified and applied in string data spaces [13, 14].

A simple clustering-based method to speed-up the k-NN classification is presented in [15]. Initially, the method discovers clusters in the training set by using c-means clustering. Then, the k-NN classifier performs classification by searching for the nearest neighbours in the nearest cluster. The number of clusters is an input parameter defined by the user. In addition, the authors try to further improve their method by using Neural Codes that are feature-based representations extracted by Deep Neural Networks. Neural Codes gather instances of

the same class so that they are placed within the same cluster. Although the authors refer to the cluster means as prototypes, their method is not a PG algorithm since it does not reduce the training data. However, the proposed method was empirically compared against several Data Reduction Techniques including RHC, but not RSP3. The results show that RHC achieved the highest performance against all data reduction techniques. Hence, we decided to include RHC in the experimental study of the present paper.

In [16], S. Impedovo et al present a PG algorithm for handwriting digit recognition. Their technique involves two stages. The first stage uses the Adaptive Resonance Theory [17] to determine the number of prototypes and generates the initial set of prototypes. The second stage uses a naive evolution strategy to generate the final set of prototypes. The proposed algorithm is incremental and can be adapted to changes in the writing styles by adding prototypes or modifying the previous generated prototypes.

In [18], a swarm-based metaheuristic search technique is adapted to generate prototypes. The technique is called Gravitational search algorithm (GSA) and has been inspired by Newtonian laws of gravity and motion [19]. It is worth mentioning that RSP3 is included in the experimental study of this paper.

Particle Swarm Optimization for Prototype Generation is applied in [20] alongside methods for improving the classification performance. The first method is a fitness function called error rank that aims to enhance the generation ability of the nearest neighbour classifier by considering the misclassified instances. The second method is a multi-objective optimization strategy that pursues the performance on multiple subsets of data in order to avoid over-fitting. The proposed approach is not parameter free. The authors include RSP3 in their experiments.

In 2018, M. Elkano et al proposed CHI-PG [21], an one-pass Prototype Generation algorithm that exploits the Map-Reduce paradigm. The algorithm generates prototypes by using fuzzy rules. A key feature of the algorithm is that the generated prototypes are exactly the same regardless of the number of Mappers/Reducers used. However, the algorithm uses parameters that need to

be empirically determined.

Another interesting PG algorithm is proposed by H.J. Escalante et al in [22]. The algorithm is called Prototype Generation via Genetic Programming (PGGP). It is based on genetic programming in which many training instances are combined through arithmetic operators in order to generate prototypes. The genetic program has as goal to generate prototypes that maximize an estimate of the generalization performance of the nearest neighbour classifier. The algorithm automatically selects the number of prototypes for each class.

J.Calvo-Zaragoza et al proposed the Prototype Generation on structural data using dissimilarity space representation [23]. The work focuses on structural data such as strings, trees or graphs and does not propose a new algorithm. It introduces dissimilarity space methods as an intermediate stage for mapping the initial structural representation to a feature-based representation, thereby allowing the use of PG algorithms. RSP3 and other two PG algorithms were used in the experimental study. The results depict that RSP3 is the most accurate PG algorithm.

In [24], a Learning Vector quantization algorithm based on granular computing is proposed. It is equipped with incremental learning mechanisms appropriate for Big Data. The algorithm uses a simple one-pass clustering task to quickly group instances with similar features. Then, it generates prototypes to cover the class distribution. The algorithm includes two stages. The first stage controls the number of prototypes by a usage-frequency indicator. In effect, the algorithm keeps the best prototype through a life index attached to each prototype. The second one prunes the useless dimensions of the training database.

The multi-objective evolutionary algorithm for prototype generation is presented in [25]. The algorithm aims to simultaneously optimize accuracy and reduction rate as well as to achieve better trade-off between them. This is achieved by formulating prototype generation as a multi-objective optimization problem. The key factors are precisely the amount of prototypes and an estimate of generalization performance achieved by the selected prototypes. The

paper uses RSP3 in the experimental study.

3. Reduction by Space Partitioning algorithms

Reduction by Space Partitioning (RSP) algorithms [5] comprise a popular family of three Prototype Generation algorithms proposed by Santchez in 2004. The Chen and Jozwik algorithm (CJA) [26] constitutes the ancestor of the family. CJA initially retrieves the two farthest instances, x and y, in the training set. The distance between x and y is the maximum distance in the training set and constitutes the diameter of the dataset. The algorithm divides the training set into two subsets. The instances that lie closer to x are placed in S_x whereas the instances that lie closer to y are placed in S_y. Then, CJA continues by first dividing the non-homogeneous subset with the largest diameter. If all subsets become homogeneous, the algorithm proceeds by dividing the homogeneous subset with the largest diameter. This procedure continues until the number of subsets becomes equal to a pre-specified value. In the end, for each subset S, CJA summarizes the instances in S by averaging them and creates a prototype (a mean instance) that is assigned the label of the majority class in S. The set of mean instances constitutes the final condensing set.

Each prototype p is computed by averaging the t attribute values of instances x_i, $i = 1, 2, \ldots, |S|$ that belong to S. Therefore, the j^{th} attribute $p.d_j$ of p is computed as follows:

$$p.d_j = \frac{1}{|S|} \sum_{x_i \in S} x_i.d_j, j = 1, 2, \ldots, t$$

CJA always divides the subset with the largest diameter. The idea is that this subset will likely contain the largest number of instances, and, when this happens, a high reduction rate is achieved. The algorithm has two weak points: (a) it is a non-parameter free algorithm since the user has to specify the desired number of subsets (or size of the condensing set), and, (b) the instances that do not belong to the most common class of a final subset are not represented

in the condensing set. Hence, it may be the case that a class label disappears from the condensing set.

RSP1 deals with the second drawback of CJA, i.e., all the original instances are represented by a prototype and none is ignored. More specifically, in the prototype generation step, RSP1 computes as many mean instances as the number of distinct classes in a final subset. Therefore, it averages the instances that belong to each class in the subset. Obviously, RSP1 builds larger condensing sets than CJA. However, it attempts to improve classification accuracy since it takes into account all training instances and all classes.

RSP1 and RSP2 differ in the way they select the next subset to be divided. RSP2 uses the highest overlapping degree as the splitting criterion. The overlapping degree of a subset is the ratio of the average distance between instances belonging to different classes, to the average distance between instances belonging to the same class. Hence, RSP2 assumes that instances that belong to the same class lie as close to each other as possible, whereas instances that belong to different classes lie farther apart.

RSP3 differs in that it divides only non-homogeneous subsets and stops when all of them become homogeneous. Since all non-homogeneous subsets are eventually divided, the choice of splitting criterion is irrelevant. RSP3 is the only space partitioning algorithm that automatically determines the size of the condensing set, hence it does not involve any input parameter. Consequently, RSP3 addresses both of the CJA weaknesses. Like all the aforementioned algorithms, the condensing set constructed by RSP3 does not depend on the instance order in the training set.

RSP3 constructs many small subsets for close-class-border areas where instances from different classes are close to each other. Consequently, more prototypes are generated for representing those areas. On the other hand, fewer and larger subsets are constructed for the non close-class-border ("internal") areas. RSP3 builds smaller homogeneous subsets for the "noisy" data areas and, as a consequence, more prototypes are generated. Thus, the reduction rate is negatively affected by noise and an editing algorithm [3] for noise removal

should be applied beforehand. The experimental study in [5] and the findings in other studies have shown that RSP3 generates a small and accurate set of training prototypes. When the k-NN classifier utilizes the RSP3 output instead of the original training data, it achieves the same classification accuracy but at a much lower computational cost. Still, the computational cost for constructing the condensing set remains very high.

In Figure 1, we depict a flowchart of an iterative version of RSP3 and we emphasize our contribution, namely, the selection of the seed instances for dividing a non-homogeneous subset (see flowchart process box with bold-italics font).

Coincidentally, this is the most expensive process in the given flowchart with a complexity of $O(n^2)$, since, in order to find the farthest instances in a subset one has to compute all distances among the instances. The process that computes the representative of a homogeneous subset and the process that assigns each instance of a subset to its nearest seed, both have a complexity of $O(n)$. The variations of RSP3 that we present in Section 4.2, make the process of selecting the seed instances and dividing a subset a $O(nlogn)$ or $O(n)$ task.

4. The proposed algorithms

Based on the observation that the farthest instances of a dataset belong to its convex hull, in this section, we discuss issues related to convex hull computation. We also list some observations that led us to the proposed fast variations of RSP3.

4.1. Exploiting the concept of Convex Hull

The computation of the convex hull of a dataset in the Euclidean space comprises a fundamental problem in computational geometry. The convex hull of a dataset S can be illustrated as the shape that encloses S [27, 28] (see figure 2). As already mentioned, an important for our purposes property of the convex hull of a dataset is that it contains the instances that define the diameter

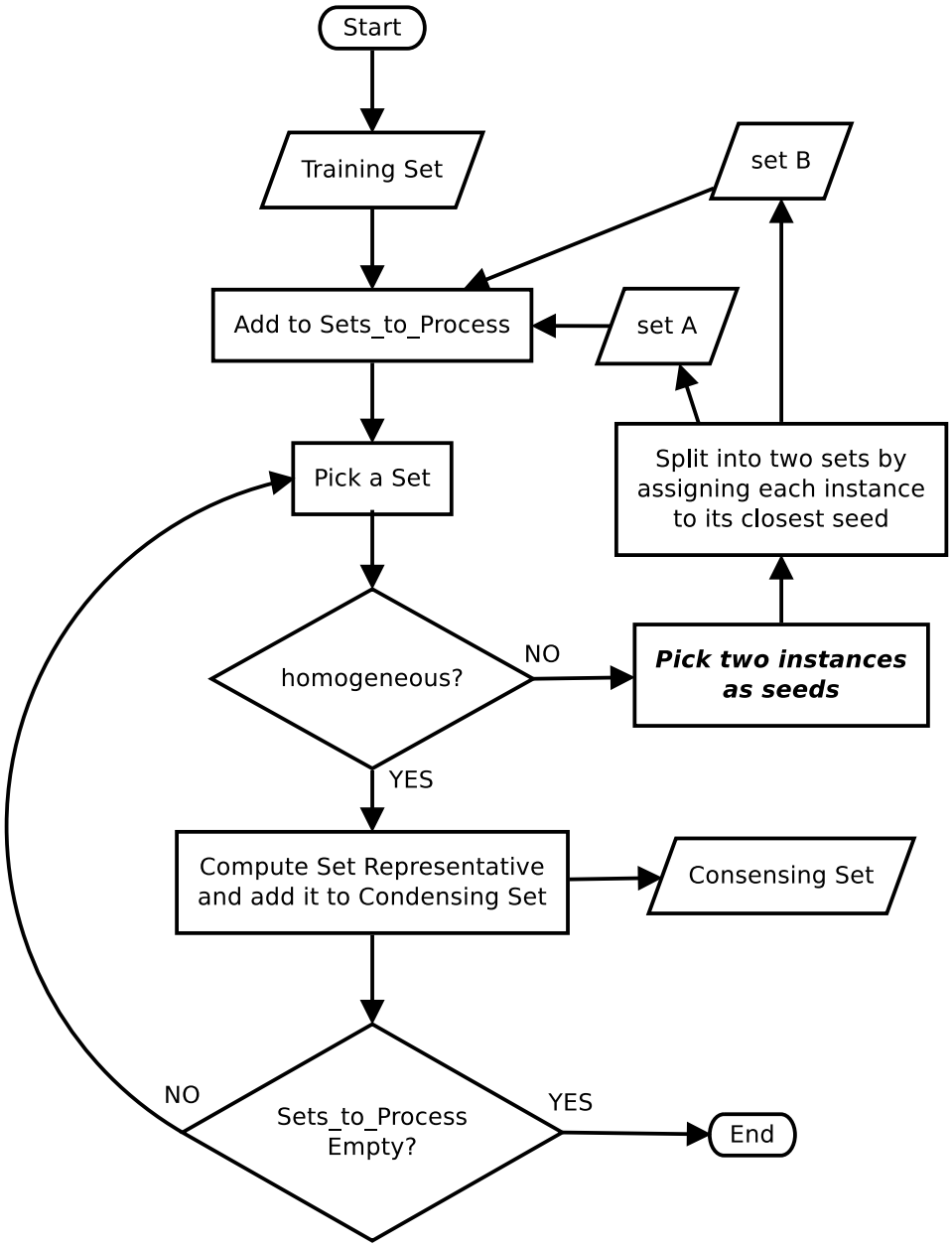


Figure 1: RSP3 algorithm flow chart (iterative version)

of the dataset. If $p1$ and $p2$ are the farthest instances within a set, $p1$ and $p2$ are strictly convex hull instances (see figure 2). Therefore, if the convex hull is known, one has to only compute all distances among convex hull instances in order to find the farthest instances of a subset.

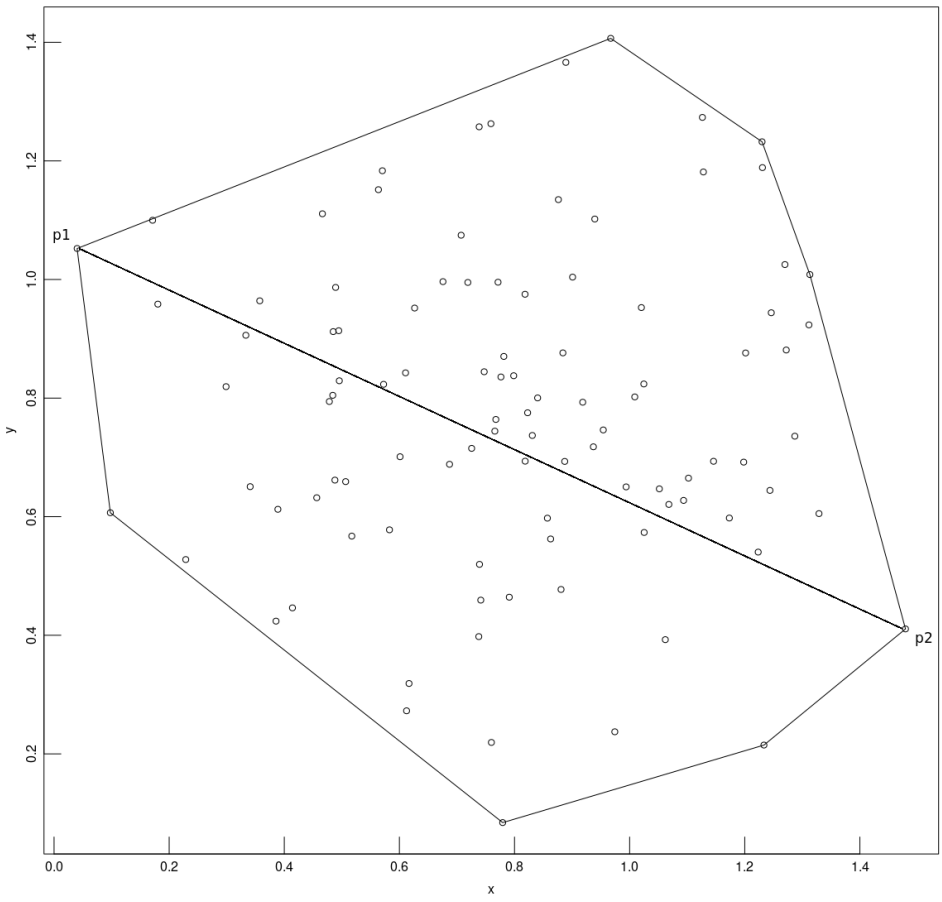


Figure 2: The convex hull of a dataset and the farthest instances $p1$ and $p2$

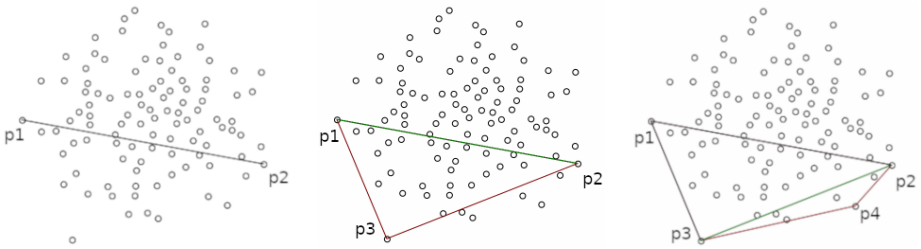
Graham's scan algorithm [29] is the oldest algorithm for convex hull computation on two-dimensional Euclidean planes. The algorithm finds the convex hull vertices by utilizing a stack data structure. The complexity of the algorithm

is $O(nlogn)$ where n is the number of instances in the dataset. The algorithm cannot be applied on higher dimensional euclidean spaces (see [29] for details). Timothy Chan proposed a faster approach where the computational cost depends on the size of the output [30]. The algorithm takes $O(nlogh)$ time, where h is the number of output vertices. Chan's algorithm combines Graham scan with Jarvis's march [31]. Chan's algorithm is simpler than Graham's scan and it is applicable to three-dimensional spaces.

Quick Hull [11] is the most widely used algorithm for convex hull computation. It was proposed by Barber [11] in 1996. Quick Hull uses a divide-and-conquer approach similar to that of Quick Sort, from which its name originates. Although the average algorithm complexity cannot be easily computed, it is assumed to be $O(nlogn)$, and in the worst case $O(n^2)$.

In two dimensional spaces, Quick Hull operates as follows: initially, the algorithm finds the instances (p_1 and p_2) that have the minimum and maximum x coordinates. In all cases, p_1 and p_2 are part of the convex hull. Quick Hull uses the line defined by the two instances to divide the set in two subsets and applies the procedure described below recursively (see Figure 3(a)). The algorithm determines the instance p_3, on one side of the line, with the maximum distance from the line. That instance is a convex hull instance. Now, p_1, p_2 and p_3 form a triangle (Figure 3(b)). The instances lying inside that triangle are ignored since they are not part of convex hull. Now, there are two new lines, i.e., the line from p_1 to p_3 and that from p_2 to p_3. The aforementioned procedure is repeated for these two lines, and up to two new instances are selected. In the example of Figure 3, p_4 is selected to be part of the convex hull. Quick Hull proceeds recursively and terminates when there are no new instances to be selected. All the selected instances constitute the final set of convex hull instances. Algorithm 1 outlines the recursive Quick Hull algorithm in pseudo-code.

In general, Quick Hull performs well. However, its execution may turn slow in cases of high symmetry or points lying on the circumference of a circle. It is worth mentioning that Quick Hull is available in Matlab and Octave. Moreover,



(a) (b) (c)

Figure 3: example of Quick Hull Execution

the algorithm is distributed by the Geometry Center as a C library through `http://qhull.org/` where explanatory material and detailed instructions are available[1].

Many algorithms implement convex hull computation, yet few of them scale to three-dimensional spaces and even fewer to multidimensional spaces [32]. Thus, it is not straightforward to incorporate such algorithms into a DRT. The Geometry Center that developed the C Quick Hull library have generalized the algorithm to operate in n-dimensional spaces. However, there are constraints concerning the datasets used [11]. Also, even if a dataset satisfies the constraints, it is not certain that its convex hull is successfully computed. The C Quick Hull library developers warn that as space dimensionality increases, the complexity and the memory requirements of the algorithm increase, too. In effect, the algorithm's overall complexity is unknown, since it is highly dependent on the dataset used.

Consequently, a variation of RSP3 that utilizes the generalized Quick Hull algorithm is inappropriate for multi-dimensional data. In addition to the potential problems during the computation phase, the overall computational cost and the memory requirements would probably overcome those of RSP3.

[1]The Geometry Center (`http://www.geom.uiuc.edu/`) has been a mathematics research and education center of the University of Minnesota. It had been established by the US National Science Foundation, and operated until 1998. However, the Quick Hull C library is still maintained by former members of the Center.

Algorithm 1 QuickHull

Input: S
Output: $ConvexHull$

$ConvexHull \leftarrow \varnothing$
$p_1 \leftarrow$ left most point
$p_2 \leftarrow$ right most point
$ConvexHull \leftarrow ConvexHull \cup \{p_1, p_2\}$
$S_1 \leftarrow$ instances from the right side of line p_1 to p_2
$S_2 \leftarrow$ instances from the right side of line p_2 to p_1
$ConvexHull \leftarrow$ FindConvexHull$(S_1, p_1, p_2, ConvexHull)$
$ConvexHull \leftarrow$ FindConvexHull$(S_2, p_2, p_1, ConvexHull)$
return $ConvexHull$

FindConvexHull$(S_k, p_a, p_b, ConvexHull)$

if $S_k = \varnothing$ **then**
  return
end if
$c \leftarrow$ the distant instance from line from p_a to p_b
$ConvexHull \leftarrow ConvexHull \cup \{c\}$
$S_0 \leftarrow$ instances inside triangle c, p_a and p_b $\{S_0$ is ignored$\}$
$S_1 \leftarrow$ instances from the right side of line p_a to c
$S_2 \leftarrow$ instances from the right side of line c to p_b
$ConvexHull \leftarrow$ FindConvexHull$(S_1, p_a, c, ConvexHull)$
$ConvexHull \leftarrow$ FindConvexHull$(S_2, c, p_b, ConvexHull)$
return $ConvexHull$

Hence, our proposal is to avoid the computation of the exact convex hull of high dimensional datasets and instead opt for convex hull approximations. The development of the two RSP3 variations we propose in the following section was based on the following observations:

- RSP3 uses a subset's diameter in order to divide it. However, the termination of the algorithm does not depend on whether the chosen pair of instances actually represents the diameter of the subset. In a hypothetical scenario where subsets are divided by using any two of their instances, the algorithm still returns a condensing set. However, the algorithm will likely require more time to execute, because of the additional number of subset divisions, and also a lower data reduction rate will be achieved. Based on this observation, we developed RSP3-RND, a dump baseline version

of RSP3, that chooses two random instances to perform the division of a subset.

- We have also empirically noticed that dividing a subset using the instances that define its diameter does not always lead to the optimal data reduction. Hence, it may turn out that it is not necessary to calculate the actual diameter of a subset. Instead, it may be appropriate to use a pair of distant enough instances in the subset in order to divide it.

- Convex hull algorithms require one or more instances that belong to the convex hull to be known / given in the beginning of their execution. In all cases, these instances are those with the minimum or maximum value in one of the dimensions. It turns out that finding the instances with the minimum and maximum values in each dimension of a subset, i.e., instances that belong to the subset's Minimum Bounding Rectangle, results into a set of instances that could approximate the convex hull of the subset. In the best case, the distance between the farthest instances in this set will be the subset's diameter, whereas, in the worst case, that distance will comprise a good approximation of the latter.

Since our goal is to speed-up RSP3 the proposed approximate convex hull computations in RSP3 should be as fast as possible. In effect, the total time for approximating the convex hull plus finding the farthest instances of the approximation should be significantly less than the time needed to find the farthest instances in the subset.

4.2. The new RSP3 variations

In this respect, we propose two new RSP3 variations that aim to reduce the distance computations required for the division of each subset. Both variations avoid the computation of the actual convex hull of a subset. Instead, they attempt to efficiently compute an approximation of the convex hull of a subset and use the farthest instances of this approximation in order to perform the division of the subset.

In addition, we implement a dump version of RSP3, where instead of using the farthest instances of a subset in order to divide it, we choose two random instances to perform the division. To the best of our knowledge, this version of RSP3 has not been proposed/used before. We consider it to be a baseline version of RSP3 that should justify the reason why distant instances are needed during subset division. We refer to this version as RSP3-RND.

4.2.1. RSP3-QH3d

We first propose a variation of RSP3, called "RSP3-QH3d" that uses Quick Hull for only three selected dimensions of the dataset, in the case of high dimensional data (> 3).

Thus, we propose to compute the exact 3d convex hull of a dataset in three selected dimensions and find the farthest instances of these 3d convex hull instances using all their attributes in order to get a good approximation of the farthest instances in the dataset.

For datasets with high dimensionality (> 3), RSP3-QH3d works as follows: Initially, an attribute selection method is applied in order to obtain the three most important attributes of the dataset. These attributes are then used for the 3d convex hull computation in each subset, i.e., all attributes, except those three, are ignored during the computation of the convex hull. Thus, the instances used for subset division are the farthest instances in the computed convex hull with their full set of attributes. The division of subsets and the prototype generation in homogeneous subsets also use the full set of attributes, like in the standard RSP3. Obviously, for datasets with up to three dimensions, attribute selection is not applied.

We also propose two versions of RSP3-QH3d that differ on the technique they employ to select the three most important attributes that Quick Hull uses for the computation of the exact 3d convex hull.

The first technique is based on the Interquartile Range (IQR), a measure of volatility. It involves dividing the values of an attribute into quarters. The quarters divide the sorted values of the attribute into four equal parts. The val-

ues that determine the borders of the quarters are called first, second and third quartiles and are denoted by Q_1, Q_2 and Q_3, respectively. IQR is computed as follows: $IQR = Q_3 - Q_1$. For each dataset, we selected the top three attributes in terms of IQR. The idea is that the higher the IQR value is for an attribute, the larger is the dispersion of the attribute's distribution of values. Therefore, in theory, the attribute in question contributes more to space partitioning, than the rest of the attributes do. From now on, we call RSP3-QH3d-IQR the RSP3-QH3d version that utilizes IQR.

The second technique selects the three most important attributes using the Gain Ratio [33] attribute selection algorithm. It involves the consideration of each one attribute and the evaluation of its importance for class label prediction. The RSP3-QH3d version that utilizes the Gain ratio technique is hereafter code-named RSP3-QH3d-GR. Obviously, any attribute selection algorithm [34, 35, 36, 37] could be used in the place of Gain Ratio.

We stress again, that in both versions of RSP3-QH3d, the three selected attributes are used by Quick Hull for identifying the 3d convex hull instances, only. The full set of attributes is considered during the formation of the subsets and the computation of the homogeneous subset mean instances (prototypes).

4.2.2. RSP3-MBR

We also propose a second variation of RSP3, that approximates the farthest instances of a subset by exploiting the Minimum Bounding Rectangle (MBR) of the subset.

The instances that contain the minimum or maximum value of a dimension (attribute) belong to the dataset's Minimum Bounding Rectangle (MBR). Thus, we call the RSP3 variation that uses the MBR approximation, "RSP3-MBR". Note that the MBR of a dataset coincides with the MBR of the dataset's convex hull. Figure 4 illustrates a two-dimensional example of the MBR and convex hull in the case of a hypothetical dataset.

Algorithm 2 depicts the MBR algorithm in pseudo-code. The input to the algorithm is a dataset S with d dimensions. The output consists of the set of

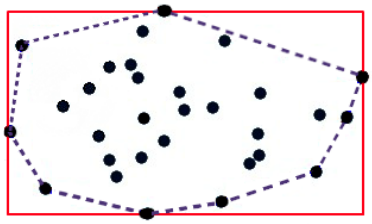


Figure 4: Convex Hull and Minimum Bounding Rectangle (MBR) of a set of instances

instances MI belonging to the dataset's MBR. The MBR algorithm maintains a $d \times 2$ array of indexes to the minimum and maximum instances in each dimension. Initially, the MBR algorithm initializes MMI by setting the first instance of S as the minimum and maximum for each dimension. Next, with one pass over S, MMI is updated with the indexes to the MBR instances. Evidently, an instance may be the minimum or the maximum for more than one dimension. Thus, the algorithm terminates by removing the duplicate MBR instances. Duplicate instance removal reduces the number of distances to be calculated when finding the farthest MBR instances.

Algorithm 2 MBR algorithm

Input: S
Output: MI

1: $MI \leftarrow \varnothing$, $low \leftarrow 1$, $high \leftarrow 2$
2: **for** $j \leftarrow 1$ to d **do**
3: $\quad MMI[j][low] \leftarrow 1, MMI[j][high] \leftarrow 1$ {index of the first instance in S}
4: **end for**
5: **for** $i \leftarrow 2$ to $|S|$ **do**
6: $\quad$ **for** $j \leftarrow 1$ to d **do**
7: $\quad\quad$ **if** $S[i][j] < S[MMI[j][low]][j]$ **then**
8: $\quad\quad\quad MMI[j][low] \leftarrow i$
9: $\quad\quad$ **else if** $S[i][j] > S[MMI[j][high]][j]$ **then**
10: $\quad\quad\quad MMI[j][high] \leftarrow i$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: **end for**
14: **for** $j \leftarrow 1$ to d **do**
15: $\quad MI \leftarrow MI \cup \{S[MMI[j][low]], S[MMI[j][high]]\}$
16: **end for**
17: Remove duplicates from MI
18: **return** MI

The complexity of MBR algorithm is $O(nd)$, where n is the size of the dataset and d its dimensionality. But, in most real life problems $n >> d$, thus, the complexity is closer to $O(n)$. Besides its simplicity, the algorithm is scalable to high dimensional datasets and it always returns a MBR that contains at most twice the number of input dimensions ($d \times 2$) instances, regardless the size of the dataset.

To summarize, RSP3-MBR first finds the MBR instances of a subset, then identifies the pair of farthest instances among them, and, finally, uses them to divide the subset.

5. Experimental study

5.1. The setup

The original RSP3 algorithm, the baseline variation RSP3-RND and the three proposed variations RSP3-QH3d-IQR, RSP3-QH3d-GR and RSP3-MBR were implemented in C++. The implementations are available on Github[2]. We used the Quick Hull C library distributed by Geometry Center. Moreover, RHC and NOP (no data reduction) were used for comparison purposes. Like RSP3, RHC is a fast parameter-free PG algorithm and is also based on the concept of homogeneity. These properties render RHC suitable for the present experimental study. The experiments were conducted on a Debian GNU/Linux server with two 64-bit Quad-Core CPUs (8 threads) and 8GB RAM.

1-NN classification using the five RSP3 algorithms, RHC and NOP was tested against sixteen (16) classification datasets distributed by the KEEL repository[3] [38]. Table 1 summarizes on the datasets used. We used various types of datasets regarding the number of instances, attributes and classes involved. All attributes were numerical and were normalized to the $[0, 1]$ range and we used the Euclidean distance. The datasets did not contain missing values. Regard-

[2]https://github.com/giotoman/RSP3

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

Table 1: datasets characteristics

Dataset	Instances	Attributes	Classes
Banana (BN)	5300	2	2
Eye State (EEG)	14980	14	2
KDD Cup (KDD)	494020	40	23
Letter Image Recognition (LIR)	20000	16	26
Landsat Satellite (LS)	6435	36	7
Magic Gamma Telescope (MGT)	19020	11	2
Pen Digits (PD)	10992	16	10
Phoneme (PH)	5404	5	2
Ring (RNG)	7400	20	2
Segment (SG)	2310	19	7
Shuttle (SH)	58000	9	7
Twonorm (TN)	7400	20	2
Textrue (TXR)	5500	40	11
Waveform (WF)	5000	21	3
Wine Quality White (WQW)	4898	11	11
Yeast (YST)	1484	8	10

ing the KDD dataset, we removed all nominal and fixed-value attributes and all duplicate instances, hence, the resulting dataset contained 141,481 instances.

The goal of data reduction is to build a small condensing set from the original dataset. The computational cost for producing the condensing set should be as low as possible and the use of the condensing set in the place of the original dataset as a training set (model) in kNN classification should achieve comparable accuracy.

Therefore, for each algorithm and dataset used, we calculated and report three measures by applying 5-fold cross validation: (i) Classification Accuracy (ACC), (ii) Reduction Rate (RR), and, (iii) condensing set construction CPU time in seconds (CPU). For the two RSP3-QH3d versions, we do not report the CPU time required for attribute selection. Accuracy is usually the most significant measure in classification tasks. The main goal of data reduction is to reduce the size of the training set as much as possible without loss of accuracy. Hence, reduction rate is considered more significant than the condensing set construction CPU time. The latter remains significant and should not be ignored especially when the training dataset is large and the data reduction is applied

in time-sensitive environments.

5.2. Experimental results

Table 2 presents the experimental results for each dataset. The best measurements are in bold while the second best measurements are in italic typeface. Figure 5 presents three histograms with the average values of each compared algorithm for each one measure.

The table and the histogram plots reveal no substantial difference between the original RSP3 and the proposed variations, namely RSP3-QH3d-IQR, RSP3-QH3d-GR and RSP3-MBR, regarding classification accuracy (ACC) and reduction rate (RR). All four have been found to perform similarly with respect to these two measures. In Figure 5, we plot these two results in an exponential scale to demonstrate how close the four algorithms perform. As expected, the baseline variation RSP3-RND is the worst RSP3 variation in terms of reduction rate. On the other hand, RSP3-RND is the fastest (lowest CPU cost) RSP3 approach and does not achieve lower accuracy than the original RSP3 and the proposed variations.

However, one notices that for all the datasets used, all RSP3 variations (RSP3-RND included) outperform the original RSP3 in terms of CPU Cost. This happens because RSP3 computes an extremely large number of distances. In contrast, the two RSP3-QH3d versions and RSP3-MBR retrieve 3d convex hull or MBR instances, respectively, and only compute distances among them, whereas, RSP-RND simply chooses two random instances. For the larger datasets in particular (i.e., KDD and SH), the original RSP3 leads to noticeably more intensive CPU usage, when compared to the RSP3 variations herewith proposed. Thus, one may safely conclude that the two RSP3-QH3d versions and RSP3-MBR each outperform the original RSP3 in execution time with no sacrifice in the classification accuracy and the reduction rate achieved. More specifically, each one of the proposed RSP3 variations executes at less than 0.15% of the RSP3 CPU cost on average. Therefore, the proposed variations can be applied on large datasets where the computational cost involved prohibits

the use of RSP3.

As expected, RHC is the fastest approach, with the exception of RSP3-RND, and achieves the highest reduction rate. This happens because the proposed variations need to compute the 3d convex hull or the MBR of each subset and then compute the distances between their instances. It is worth mentioning that, in several datasets, the proposed RSP3 variations require CPU time close to that of RHC. On the other hand, almost in all datasets used, RHC achieves the lowest accuracy.

Upon closely inspecting the results in Table 2, one notices that the reduction rate of RSP3-MBR is as high as that of RSP3-QH3d-GR, while RSP3-QH3d-IQR yields a slightly lower reduction rate when compared to the former and RSP3. Therefore, the approximation technique in RSP3-MBR and the convex hull computation in the three-dimensional space applied by the two RSP3-QH3d versions have a negligible impact on the reduction rate measure.

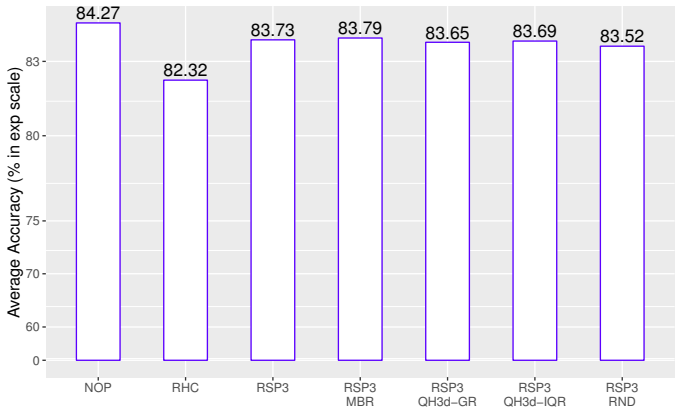
Furthermore, the accuracy achieved by RSP3-MBR and the two RSP3-QH3d versions is as high as that of RSP3. Although the difference is negligible, it is noteworthy that RSP3-MBR was measured to achieve the highest average accuracy measurement. The finding confirms that using the actual maximum distance in a subset does not necessarily produce optimal results.

RSP3's CPU Cost is much higher than that of the rest of the algorithms. There is no significant difference in CPU Cost between RSP3-MBR and the two RSP3-QH3d versions although they use completely different approaches for approximating the diameter of the subsets. RSP3-MBR generates an almost constant number of instances, whereas, the output of the two RSP3-QH3d versions depends on how the data instances are distributed within the dataset, not simply on the number of attributes involved.

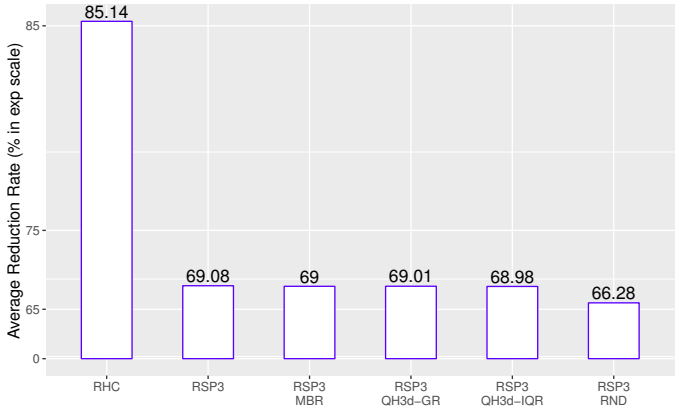
To summarize, RSP3-QH3d and RSP3-MBR seem to outperform RHC in terms of accuracy, RSP3-RND in terms of reduction rate, and, the original RSP3 in terms of CPU cost. This significant improvement in the required computational cost of the proposed RSP3 variations over the original RSP3, makes the proposed RSP3 variations valid contenders of RHC. Therefore, in case

Table 2: Comparison in terms of Accuracy (Acc(%)), Reduction Rate (RR(%)), CPU Time(secs)

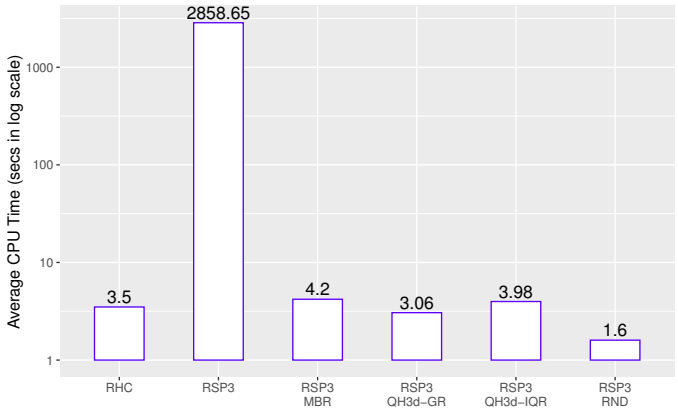
Dataset		NOP	RHC	RSP3	RSP3 QH3d-IQR	RSP3 QH3d-GR	RSP3 MBR	RSP3 RND
BN	ACC:	86.86	83.17	*84.42*	*84.42*	*84.42*	**84.53**	84.10
	RR:	-	**79.39**	75.13	*75.18*	*75.18*	75.32	74.13
	CPU:	-	**0.10**	5.72	0.53	0.55	0.41	*0.20*
EEG	ACC:	45.70	**47.54**	47.30	47.33	46.80	*47.53*	46.51
	RR:	-	**77.22**	53.71	53.71	53.37	*53.81*	49.54
	CPU:	-	**1.44**	333.15	6.53	6.36	6.15	*4.49*
KDD	ACC:	99.71	99.30	**99.60**	99.50	99.30	*99.58*	99.50
	RR:	-	**99.19**	98.54	98.19	*98.63*	98.56	97.95
	CPU:	-	38.54	41021.00	13.54	*6.68*	17.37	**5.08**
LIR	ACC:	95.87	93.49	**95.45**	95.37	*95.41*	95.38	95.25
	RR:	-	**88.01**	61.81	61.57	*62.13*	61.56	56.05
	CPU:	-	*6.55*	373.16	13.90	9.67	14.39	**5.90**
LS	ACC:	90.29	88.93	90.10	**90.35**	90.07	*90.21*	90.07
	RR:	-	**90.04**	73.20	73.47	*74.10*	73.11	70.82
	CPU:	-	*0.94*	118.22	1.59	1.48	2.18	**0.49**
MGT	ACC:	80.89	74.79	*77.82*	77.55	77.59	77.58	**78.01**
	RR:	-	**79.38**	58.44	58.85	58.77	*58.88*	57.44
	CPU:	-	**1.24**	297.58	13.51	10.19	12.71	*4.78*
PD	ACC:	99.40	98.50	**99.10**	99.06	98.94	99.04	*99.09*
	RR:	-	**96.48**	*89.22*	88.77	88.66	89.10	84.44
	CPU:	-	*0.80*	138.44	1.17	1.14	1.04	**0.37**
PH	ACC:	89.84	85.06	87.10	86.89	*87.30*	**87.47**	86.56
	RR:	-	**80.74**	69.09	*69.40*	69.35	69.30	67.01
	CPU:	-	**0.16**	12.49	0.99	0.96	0.68	*0.30*
RNG	ACC:	74.87	*82.20*	81.84	**82.22**	81.85	81.30	79.46
	RR:	-	**90.14**	57.15	56.80	56.17	55.75	*57.18*
	CPU:	-	*1.06*	94.29	3.11	3.18	3.44	**0.86**
SG	ACC:	97.09	94.24	95.97	96.23	*96.40*	**96.49**	95.75
	RR:	-	**92.21**	81.95	*82.45*	82.44	82.24	78.76
	CPU:	-	*0.09*	9.89	0.32	0.25	0.23	**0.07**
SH	ACC:	99.94	**99.79**	99.39	99.48	99.56	99.69	*99.70*
	RR:	-	**99.63**	*99.38*	99.29	99.32	99.37	98.82
	CPU:	-	*1.86*	3104.68	2.15	2.31	2.07	**0.59**
TN	ACC:	94.80	89.76	*93.00*	92.86	92.79	92.52	**93.37**
	RR:	-	**96.65**	*84.58*	83.87	83.75	84.07	79.75
	CPU:	-	*0.73*	73.15	0.96	0.97	1.01	**0.30**
TXR	ACC:	99.05	97.11	98.54	*98.67*	**98.80**	98.58	98.45
	RR:	-	**94.51**	82.60	82.72	82.70	*82.97*	78.84
	CPU:	-	1.47	97.99	*1.06*	1.12	1.38	**0.30**
WF	ACC:	77.25	75.18	**78.40**	77.17	77.48	77.78	*78.17*
	RR:	-	**88.78**	56.83	56.55	*56.96*	56.87	50.93
	CPU:	-	**0.42**	16.70	0.93	0.91	0.99	*0.59*
WQW	ACC:	64.26	60.63	61.68	61.86	*61.88*	**62.42**	61.56
	RR:	-	**59.45**	35.34	35.17	34.69	*35.65*	32.52
	CPU:	-	**0.40**	39.91	2.95	2.96	2.80	*1.12*
YST	ACC:	52.51	47.47	50.00	50.14	49.86	*50.54*	**50.81**
	RR:	-	**50.41**	*28.35*	27.72	27.94	27.45	26.23
	CPU:	-	**0.10**	2.07	0.43	0.30	0.38	*0.15*



(a) Accuracy



(b) Reduction Rate



(c) CPU Time

Figure 5: Average experimental measurements

of large datasets and time-sensitive environments, where the use of the original RSP3 is prohibitive because of its high CPU cost, either RSP3-QH3d or RSP3-MBR should be used instead of RHC when the need for higher accuracy can justify lower reduction rates. On the other hand, RHC outperforms all RSP3 variations in terms of reduction rate and comes close in terms of accuracy. Thus, RHC is preferable in scenarios where storage is limited. In the following subsection, we support the above claims with the help of extended statistical tests.

5.3. Statistical comparisons

The non-parametric Friedman test with Conover's Post Hoc Comparisons was used in order to check for differences among the algorithms. We used the Friedman test provided by the JASP statistical software [39]. The test was run three times, one for each criterion measured (ACC, CPU, RR). For brevity in the presentation, in all the tables that follow, we use the names IQR, GR, MBR, and RND for the proposed RSP3 variations.

Regarding ACC, there was a statistically significant difference in performance depending on which reduction algorithm was used, $\chi^2(6) = 28.834$, $p < .001$. Conover's post hoc comparisons (with the Holm correction applied) revealed that the algorithms did not differ among each other with the exception of RSP3-MBR that was rated better than RHC ($p = .031$) and NOP that was rated better than RSP3-RND ($p = .038$) and RHC ($p < .001$) (see Table 3). This is not a surprising result considering the very high reduction rates achieved by RHC. On the other hand, the quite good performance of RSP3-RND is attributed to the mediocre reduction rate of the algorithm (see statistical analysis of Reduction Rate below).

Regarding CPU, there was a statistically significant difference in performance depending on which reduction algorithm was used, $\chi^2(5) = 61.429$, $p < .001$. Conover's post hoc comparisons (with the Holm correction applied) revealed that RSP3 was rated worse than all other algorithms, RSP3-RND was rated better than all other RSP3 algorithms but equal to RHC, and RHC was

Table 3: Conover's Post Hoc Comparisons - Accuracy

		T-Stat	df	W_i	W_j	p	p_{bonf}	p_{holm}
RSP3	IRQ	0.322	90	68.000	64.000	0.748	1.000	1.000
	GR	0.645	90	68.000	60.000	0.521	1.000	1.000
	MBR	0.403	90	68.000	73.000	0.688	1.000	1.000
	RND	1.007	90	68.000	55.500	0.316	1.000	1.000
	RHC	2.861	90	68.000	32.500	0.005	0.110	0.094
	NOP	2.176	90	68.000	95.000	0.032	0.676	0.418
IRQ	GR	0.322	90	64.000	60.000	0.748	1.000	1.000
	MBR	0.725	90	64.000	73.000	0.470	1.000	1.000
	RND	0.685	90	64.000	55.500	0.495	1.000	1.000
	RHC	2.539	90	64.000	32.500	0.013	0.270	0.206
	NOP	2.498	90	64.000	95.000	0.014	0.300	0.214
GR	MBR	1.048	90	60.000	73.000	0.298	1.000	1.000
	RND	0.363	90	60.000	55.500	0.718	1.000	1.000
	RHC	2.216	90	60.000	32.500	0.029	0.613	0.409
	NOP	2.821	90	60.000	95.000	0.006	0.124	0.100
MBR	RND	1.410	90	73.000	55.500	0.162	1.000	1.000
	RHC	3.264	90	73.000	32.500	0.002	0.033	0.031
	NOP	1.773	90	73.000	95.000	0.080	1.000	0.876
RND	RHC	1.854	90	55.500	32.500	0.067	1.000	0.805
	NOP	3.183	90	55.500	95.000	0.002	0.042	0.038
RHC	NOP	5.037	90	32.500	95.000	$< .001$	$< .001$	$< .001$

rated better than RSP3-QH3d-IQR ($p = .002$) and RSP3-MBR ($p = .036$) and marginally better than and RSP3-QH3d-GR ($p = .056$) (see Table 4).

Finally, regarding RR, there was a statistically significant difference in performance depending on which reduction algorithm was used, $\chi^2(5) = 52.102$, $p < .001$. Conover's post hoc comparisons (with the Holm correction applied) revealed that RHC was rated better than all other algorithms, RSP3-RND was rated worse than all other algorithms, and all other RSP3 variations were rated equal among each other (see Table 5).

Thus, we conclude that the two versions of RSP3-QH3d and RSP3-MBR outperform RSP3 by involving only 0.13% - 0.15% of the computational cost of the latter, without sacrifices in classification accuracy and reduction rate. Also, the new variations of RSP3 come very close to RHC in terms of computational cost, but RHC remains the best algorithm in terms of data reduction rates.

Table 4: Conover's Post Hoc Comparisons - CPU Cost

		T-Stat	df	W_i	W_j	p	p_{bonf}	p_{holm}
RSP3	IRQ	2.882	75	96.000	65.000	0.005	0.077	0.036
	GR	3.439	75	96.000	59.000	$< .001$	0.014	0.011
	MBR	3.160	75	96.000	62.000	0.002	0.034	0.020
	RND	6.786	75	96.000	23.000	$< .001$	$< .001$	$< .001$
	RHC	6.042	75	96.000	31.000	$< .001$	$< .001$	$< .001$
IRQ	GR	0.558	75	65.000	59.000	0.579	1.000	1.000
	MBR	0.279	75	65.000	62.000	0.781	1.000	1.000
	RND	3.904	75	65.000	23.000	$< .001$	0.003	0.003
	RHC	3.160	75	65.000	31.000	0.002	0.034	0.020
GR	MBR	0.279	75	59.000	62.000	0.781	1.000	1.000
	RND	3.346	75	59.000	23.000	0.001	0.019	0.013
	RHC	2.603	75	59.000	31.000	0.011	0.167	0.056
MBR	RND	3.625	75	62.000	23.000	$< .001$	0.008	0.006
	RHC	2.882	75	62.000	31.000	0.005	0.077	0.036
RND	RHC	0.744	75	23.000	31.000	0.459	1.000	1.000

Finally, RSP3-RND was rated as the worst algorithm in terms of reduction rate, thus, it was shown that looking for distant instances when dividing a subset, effectively leads to higher data reduction.

6. Conclusion and future work

This paper proposes two fast RSP3 variations that can be used on large datasets. Contrary to the conventional RSP3 algorithm, which finds the farthest instances in non-homogeneous subsets by computing all the possible distances in each subset, both variations find instances belonging to approximations of the convex hull of each subset. Then, they compute the distances between those instances and retrieve the two farthest ones. The first variation is called RSP3-QH3d and utilizes the Quick Hull algorithm for 3d convex hull computation. Although, for datasets with more than three attributes, the three most important ones are used for the computation of the 3d convex hull of each subset, all other tasks are performed on the original dimensionality and the prototypes generated are in the original space. We conducted experiments by using the Interquartile Range as an attribute selection technique as well as the Gain ra-

Table 5: Conover's Post Hoc Comparisons - Reduction Rate

		T-Stat	df	W_i	W_j	p	p_{bonf}	p_{holm}
RSP3	IRQ	0.047	75	54.000	53.500	0.963	1.000	1.000
	GR	0.047	75	54.000	54.500	0.963	1.000	1.000
	MBR	0.372	75	54.000	58.000	0.711	1.000	1.000
	RND	3.163	75	54.000	20.000	0.002	0.034	0.018
	RHC	3.908	75	54.000	96.000	$< .001$	0.003	0.003
IRQ	GR	0.093	75	53.500	54.500	0.926	1.000	1.000
	MBR	0.419	75	53.500	58.000	0.677	1.000	1.000
	RND	3.117	75	53.500	20.000	0.003	0.039	0.018
	RHC	3.954	75	53.500	96.000	$< .001$	0.003	0.002
GR	MBR	0.326	75	54.500	58.000	0.746	1.000	1.000
	RND	3.210	75	54.500	20.000	0.002	0.029	0.018
	RHC	3.861	75	54.500	96.000	$< .001$	0.004	0.003
MBR	RND	3.536	75	58.000	20.000	$< .001$	0.011	0.008
	RHC	3.536	75	58.000	96.000	$< .001$	0.011	0.008
RND	RHC	7.071	75	20.000	96.000	$< .001$	$< .001$	$< .001$

tio attribute selection. The second RSP3 variation is called RSP3-MBR and approximates the convex hull using the Minimum Bounding Rectangle of each subset. RSP3-MBR can be used with datasets of any dimensionality without the need of attribute selection.

The goal of our research was to improve RSP3 in terms of CPU cost, since RSP3 was already one of the best PG algorithms in terms of accuracy and good enough in terms of reduction rate. The experimental results and the statistical tests we performed, demonstrated that the proposed variations of RSP3 have all the desired properties, i.e., they outperform (a) RHC in terms of accuracy (this is true for RSP3-MBR), (b) RSP3-RND in terms of reduction rate - a fact that demonstrates that choosing instances that are far apart as seeds for subset division is meaningful, and, (c) the original RSP3 in terms of CPU cost.

The final conclusion is that the proposed RSP3 variations are preferable over the original RSP3 (always and especially when having large datasets), RSP3-RND (always), and, RHC (when higher accuracy justifies lower reduction rate).

In the context of RSP3-QH3d, the computation of 3D convex hull implies the usage of attribute selection beforehand, so that the three most important

attributes are selected. Although the simple method we used in this study performed very well, we plan to conduct an extensive experimental study using a variety of attribute selection algorithms [34, 35, 36, 37]. Furthermore, we keep working on data reduction and RSP3 variations. More specifically, we plan to adapt RSP3 and other DRTs on more complex classification problems, such as multi-label data and data streams.

References

[1] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inf. Theor. 13 (1) (1967) 21–27 (Sep. 1967). doi:10.1109/TIT.1967.1053964.

[2] R. Weber, H.-J. Schek, S. Blott, A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces, in: Proceedings of the 24rd International Conference on Very Large Data Bases, VLDB '98, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998, pp. 194–205 (1998).
URL http://dl.acm.org/citation.cfm?id=645924.671192

[3] S. Garcia, J. Derrac, J. Cano, F. Herrera, Prototype selection for nearest neighbor classification: Taxonomy and empirical study, IEEE Trans. Pattern Anal. Mach. Intell. 34 (3) (2012) 417–435 (Mar. 2012). doi:10.1109/TPAMI.2011.142.

[4] I. Triguero, J. Derrac, S. Garcia, F. Herrera, A taxonomy and experimental study on prototype generation for nearest neighbor classification, Trans. Sys. Man Cyber Part C 42 (1) (2012) 86–100 (Jan. 2012). doi:10.1109/TSMCC.2010.2103939.

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

[6] R. Seidel, Handbook of discrete and computational geometry, CRC Press, Inc., Boca Raton, FL, USA, 1997, Ch. Convex Hull Computations, pp.

361–375 (1997).
URL http://dl.acm.org/citation.cfm?id=285869.285890

[7] R. Cupec, I. Vidovi, D. Filko, P. urovi, Object recognition based on convex hull alignment, Pattern Recognition 102 (2020) 107199 (2020). doi:https://doi.org/10.1016/j.patcog.2020.107199.
URL https://www.sciencedirect.com/science/article/pii/S0031320320300066

[8] H. Cevikalp, High-dimensional data clustering by using local affine/convex hulls, Pattern Recognition Letters 128 (2019) 427–432 (2019). doi:https://doi.org/10.1016/j.patrec.2019.10.007.
URL https://www.sciencedirect.com/science/article/pii/S0167865519302806

[9] A. Nemirko, J. H. Dul, Nearest convex hull classification based on linear programming, Pattern Recognition and Image Analysis 31 (2) (2021) 205211 (Apr 2021). doi:10.1134/s1054661821020139.
URL http://dx.doi.org/10.1134/S1054661821020139

[10] N. Sogi, R. Zhu, J.-H. Xue, K. Fukui, Constrained mutual convex cone method for image set based recognition, Pattern Recognition 121 (2022) 108190 (2022). doi:https://doi.org/10.1016/j.patcog.2021.108190.
URL https://www.sciencedirect.com/science/article/pii/S0031320321003502

[11] Barber, C. Bradford and Dobkin, David P. and Dobkin, David P. and Huhdanpaa, Hannu, The Quickhull Algorithm for Convex Hulls, ACM Trans. Math. Softw. 22 (4) (1996) 469–483 (12. 1996). doi:10.1145/235815.235821.
URL http://doi.acm.org/10.1145/235815.235821

[12] S. Ougiaroglou, G. Evangelidis, Rhc: Non-parametric cluster-based data reduction for efficient k-nn classification, Pattern Anal. Appl. 19 (1) (2016)

93109 (Feb. 2016). doi:10.1007/s10044-014-0393-7.
URL https://doi.org/10.1007/s10044-014-0393-7

[13] F. J. Castellanos, J. J. Valero-Mas, J. Calvo-Zaragoza, Prototype generation in the string space via approximate median for data reduction in nearest neighbor classification, Soft Computing (Sep 2021). doi:10.1007/s00500-021-06178-2.
URL http://dx.doi.org/10.1007/s00500-021-06178-2

[14] J. J. Valero-Mas, F. J. Castellanos, Data reduction in the string space for efficient knn classification through space partitioning, Applied Sciences 10 (10) (2020) 3356 (May 2020). doi:10.3390/app10103356.
URL http://dx.doi.org/10.3390/app10103356

[15] A.-J. Gallego, J. Calvo-Zaragoza, J. J. Valero-Mas, J. R. Rico-Juan, Clustering-based k-nearest neighbor classification for large-scale data with neural codes representation, Pattern Recogn. 74 (C) (2018) 531543 (Feb. 2018). doi:10.1016/j.patcog.2017.09.038.
URL https://doi.org/10.1016/j.patcog.2017.09.038

[16] S. Impedovo, F. Mangini, D. Barbuzzi, A novel prototype generation technique for handwriting digit recognition, Pattern Recogn. 47 (3) (2014) 10021010 (Mar. 2014). doi:10.1016/j.patcog.2013.04.016.
URL https://doi.org/10.1016/j.patcog.2013.04.016

[17] G. A. Carpenter, S. Grossberg, Adaptive Resonance Theory (ART), MIT Press, Cambridge, MA, USA, 1998, p. 7982 (1998).

[18] M. Rezaei, H. Nezamabadi-pour, Using gravitational search algorithm in prototype generation for nearest neighbor classification, Neurocomputing 157 (2015) 256 – 263 (2015). doi:https://doi.org/10.1016/j.neucom.2015.01.008.
URL http://www.sciencedirect.com/science/article/pii/S0925231215000296

[19] E. Rashedi, H. Nezamabadi-pour, S. Saryazdi, Gsa: A gravitational search algorithm, Information Sciences 179 (13) (2009) 2232 – 2248, special Section on High Order Fuzzy Sets (2009). doi:https://doi.org/10.1016/j.ins.2009.03.004.
URL http://www.sciencedirect.com/science/article/pii/S0020025509001200

[20] W. Hu, Y. Tan, Prototype generation using multiobjective particle swarm optimization for nearest neighbor classification, IEEE Transactions on Cybernetics 46 (12) (2016) 2719–2731 (2016). doi:10.1109/TCYB.2015.2487318.

[21] M. Elkano, M. Galar, J. Sanz, H. Bustince, Chi-pg: A fast prototype generation algorithm for big data classification problems, Neurocomputing 287 (2018) 22 – 33 (2018). doi:https://doi.org/10.1016/j.neucom.2018.01.056.
URL http://www.sciencedirect.com/science/article/pii/S0925231218300894

[22] H. J. Escalante, M. Graff, A. Morales-Reyes, Pggp: Prototype generation via genetic programming, Applied Soft Computing 40 (2016) 569 – 580 (2016). doi:https://doi.org/10.1016/j.asoc.2015.12.015.
URL http://www.sciencedirect.com/science/article/pii/S1568494615007942

[23] J. Calvo-Zaragoza, J. J. Valero-Mas, J. R. Rico-Juan, Prototype generation on structural data using dissimilarity space representation, Neural Comput. Appl. 28 (9) (2017) 24152424 (Sep. 2017). doi:10.1007/s00521-016-2278-8.
URL https://doi.org/10.1007/s00521-016-2278-8

[24] I. Cruz-Vega, H. J. Escalante, An online and incremental grlvq algorithm for prototype generation based on granular computing, Soft Comput.

21 (14) (2017) 39313944 (Jul. 2017). doi:10.1007/s00500-016-2042-0.
URL https://doi.org/10.1007/s00500-016-2042-0

[25] H. J. Escalante, M. Marin-Castro, A. Morales-Reyes, M. Graff, A. Rosales-Pérez, M. Montes-Y-Gómez, C. A. Reyes, J. A. Gonzalez, Mopg: A multi-objective evolutionary algorithm for prototype generation, Pattern Anal. Appl. 20 (1) (2017) 3347 (Feb. 2017). doi:10.1007/s10044-015-0454-6.
URL https://doi.org/10.1007/s10044-015-0454-6

[26] C. H. Chen, A. Jóźwik, A sample set condensation algorithm for the class sensitive artificial neural network, Pattern Recogn. Lett. 17 (8) (1996) 819–823 (Jul. 1996). doi:10.1016/0167-8655(96)00041-4.
URL http://dx.doi.org/10.1016/0167-8655(96)00041-4

[27] De Berg, M. and Cheong, O. and Van Kreveld, M., Computational geometry: algorithms and applications, Springer-Verlag New York Inc, 2008 (2008).

[28] Knuth, Donald E., Axioms and hulls, Lecture Notes in Computer Science, Springer-Verlag, 2008 (2008).

[29] Graham, Ronald L., An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set, Inf. Process. Lett. 1 (4) (1972) 132–133 (1972).

[30] Timothy M. Chan, Optimal output-sensitive convex hull algorithms in two and three dimensions (1996).

[31] L. Cinque and C. Di Maggio, A BSP realisation of Jarvis's algorithm, in: Proceedings 10th International Conference on Image Analysis and Processing, 1999, pp. 247–252 (9. 1999).

[32] Adam Jwik, A method for solving the n-dimensional convex hull problem, Pattern Recognition Letters 2 (1) (1983) 23 – 25 (1983). doi:https://doi.org/10.1016/0167-8655(83)90018-1.
URL http://www.sciencedirect.com/science/article/pii/0167865583900181

[33] J. R. Quinlan, Induction of decision trees, Mach. Learn. 1 (1) (1986) 81–106 (Mar. 1986). doi:10.1023/A:1022643204877.
URL http://dx.doi.org/10.1023/A:1022643204877

[34] S. S. Sankari, C. Jayakumar, Opportunities and challenges of feature selection methods for high dimensional data: A review, Ingénierie des Systèmes d Inf. 26 (1) (2021) 67–77 (2021). doi:10.18280/isi.260107.
URL https://doi.org/10.18280/isi.260107

[35] S. K. Biswas, M. Bordoloi, B. Purkayastha, Review on feature selection and classification using neuro-fuzzy approaches, Int. J. Appl. Evol. Comput. 7 (4) (2016) 28–44 (2016). doi:10.4018/IJAEC.2016100102.
URL https://doi.org/10.4018/IJAEC.2016100102

[36] L. M. Q. Abualigah, Feature Selection and Enhanced Krill Herd Algorithm for Text Document Clustering, 1st Edition, Springer Publishing Company, Incorporated, 2018 (2018).

[37] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, H. Liu, Feature selection: A data perspective, ACM Comput. Surv. 50 (6) (Dec. 2017). doi:10.1145/3136625.
URL https://doi.org/10.1145/3136625

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

[39] JASP Team, JASP (Version 0.14.1)[Computer software] (2020).
URL https://jasp-stats.org/