

A Convex Hull-based Data Selection Method for Data Driven Models

H.R. Khosravani², A.E. Ruano^{2,1}, P.M. Ferreira^{3,1}

1. IDMEC, Instituto Superior Técnico, Universidade de Lisboa, Portugal

2. University of Algarve, Portugal

3. LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

emails: hkhosravani@csi.fct.ualg.pt, aruano@ualg.pt, pmf@ciencias.ulisboa.pt

Abstract

The accuracy of classification and regression tasks based on data driven models, such as Neural Networks or Support Vector Machines, relies to a good extent on selecting proper data for designing these models, covering the whole input range in which they will be employed. The convex hull algorithm can be applied as a method for data selection; however the use of conventional implementations of this method in high dimensions, due to its high complexity, is not feasible. In this paper, we propose a randomized approximation convex hull algorithm which can be used for high dimensions in an acceptable execution time, and with low memory requirements. Simulation results show that data selection by the proposed algorithm (coined as ApproxHull) can improve the performance of classification and regression models, in comparison with random data selection.

Keywords: Convex Hull, Data Selection Problem, Classification, Regression, Neural Networks, Support Vector Machines.

1 Introduction

In many machine learning and data mining problems two basic tasks have to be considered: feature selection and instance selection. The former denotes choosing a subset of all available features so that the selected subset has the strongest relation to the model output and yields improved model performance. The latter refers to sample selection where we are interested in selecting a subset of useful and informative data samples (denoted S) among all existing data samples (denoted T). The goal is that the model obtained by using S can maintain or even exceed the performance level (for instance, accuracy) that would be attained by using T . The instance selection process not only helps decreasing the run time of the training process but also has the benefit of reducing the memory requirements of learning algorithms. This is important when classification or regression tasks rely on existing large-size training sets.

Generally speaking, instance selection methods can be classified from the search direction and selection criterion points of view. Regarding the search direction, the methods are categorized as incremental or decremental. In the former, the selection process starts with $S = \emptyset$ and progresses iteratively by inserting selected samples from T into S . In the latter, in contrast, the selection process starts with $S = T$ and superfluous samples are discarded from S in an iterative manner. Finally, for both methods, we have that $S \subset T$ at the end of the selection process [1, 2]. From the selection criterion point of view, instance selection methods are classified as wrapper or filter methods.

Wrapper methods use a model as a selection criterion, where the performance of the model is evaluated based on a subset of samples iteration by iteration to select those samples which have the most contribution on the model accuracy. Most works found in literature on wrapper methods relate to classification tasks. Collectively they may be further subcategorized into two groups. The first gathers methods which are based on $k - NN$ (K Nearest Neighbors) classifiers, whereas the second group involves a broad class of wrapper methods that can be based on any classifier. In the group of $k - NN$ based methods, the CNN (Condensed Nearest Neighbor rule) is one of the earliest incremental methods which was proposed by Hart [3]. It focuses on misclassified samples as critical samples that matter the most to the $k - NN$ classifier to ensure that unlabeled samples which are similar to the misclassified ones are correctly classified [4]. To improve the CNN method, the SNN (Selective Nearest Neighbor rule) and the GCNN (Generalized Condensed Nearest Neighbor rule) were proposed by Ritter *et al* [5] and Chou *et al* [6], respectively. Besides considering critical samples, some noteworthy $k - NN$ rule based methods are focused on the removal of noisy samples: the ENN (Edited Nearest Neighbor rule) [7], the *all* $k - NN$ [8], a family of five incremental methods coined DROP1 to DROP5 (Decremental Reduction Optimization Procedure, 1 to 5) [2], and the ICF (Iterative Case Filtering) [9]. Regarding general wrapper methods, arbitrary classifiers can be used as the criterion for instance selection. Search algorithms have had a significant contribution, from which evolutionary algorithms have received most attention in the literature of instance selection. Specifically, the application of Genetic Algorithms (GA) can be found in [10-13]. Besides evolutionary algorithms, other search algorithms such as tabu search, BSE (Backward Sequential Edition) and RFOS (Restricted Floating Object Selection) have also been considered [14-17].

Unlike wrapper methods, filter methods employ a model independent selection function to choose informative samples [1]. This means that the accuracy of the model does not have any contribution in the selection criterion; instead, a selection rule is applied. One group of filter methods applied in classification problems focuses on selecting class border samples [18, 19]. Alternatively some authors employed clustering methods in order to select instances that can be considered representatives for a cluster of instances [20-25].

Neural networks and Support Vector Machines (SVM) are two examples of well-established data driven machine learning approaches for classification and regression tasks. Since the models generated by these approaches are data driven, data must be selected in such way that it covers the whole input range in which the model is to be employed. To achieve this goal two different methods using Principal Components Analysis (PCA) and convex hull have been proposed [26, 27]. Since SVM training has high a learning time, having a large training dataset

for online classification is infeasible as the existing model should be retrained with newly arriving samples along with an already large training dataset. In this condition, a reasonable portion of the large training dataset should be selected for retraining the existing model. For this purpose, methods for data reduction in SVM classification models have been proposed [28, 29], including methods based on the convex hull [27, 30].

The identification of convex hull vertices is a time consuming task, as the complexity of real convex hull algorithms in high dimensions is $O(n^{\lfloor \frac{d}{2} \rfloor})$ [31] (n and d denoting the number of samples and sample dimension respectively). In this paper we propose a Randomized Approximation convex hull algorithm, coined ApproxHull, to overcome both the high execution time and memory requirements, resulting from the curse of dimensionality associated with the convex hull algorithm complexity. ApproxHull can be used not only for off-line training, but also for online model adaptation.

The rest of the paper is organized as follows: Section 2 gives a brief description on existing convex hull algorithms. Section 3 addresses the proposed algorithm for determining an approximation of the convex hull in high dimensions. Simulation results are presented in Section 4. The run time analysis of ApproxHull is addressed in Section 5. The comparison of the proposed algorithm with Quickhull [32] from memory requirements point of view is described in Section 6. Conclusions and future work are addressed in Section 7.

2 Definitions and Related Work

2.1 Convex Hull Definition

From a computational geometry's point of view, an object in Euclidean space is convex if for every pair of points within the object, every point on the straight line segment that joins them is also within the object. A set S is convex if, for every pair, $u, v \in S$, and all $t \in [0,1]$, the point $(1-t)u + tv$ is in S . Moreover, if S is a convex set, for any $u_1, u_2, \dots, u_r \in S$, and any nonnegative numbers $\{\lambda_1, \lambda_2, \dots, \lambda_r\}$: $\sum_{i=1}^r \lambda_i = 1$, the vector $\sum_{i=1}^r \lambda_i u_i$ is called a convex combination of $u_1, u_2, \dots, u_r$. According to the definitions above, the convex hull or convex envelope of set X of points in the Euclidean space can be defined in terms of convex sets or convex combinations:

- the minimal convex set containing X , or
- the intersection of all convex sets containing X , or
- the set of all convex combinations of points in X .

Based on the definition of convex hull, a k -simplex is a k -dimensional polytope which is the convex hull of $k + 1$ affinely independent points. Intuitively, 0-simplex, 1-simplex, 2-simplex and 3-simplex correspond to a point, a line segment, a triangle and a tetrahedron, respectively. Generally, a k -simplex consists of the elements called i -faces where $i \leq k - 1$. 0-faces, 1-faces and $(k - 1)$ -faces are called vertices, edges and facets of the k -simplex, respectively.

2.2 Convex Hull Algorithms

Convex hull algorithms can be categorized from three points of view. An algorithm can be deterministic or randomized depending on the order of vertices found. If the order is fixed from run to run, the algorithm is deterministic [33]; otherwise, it is randomized [34].

Furthermore, an algorithm can be considered as a real or approximation algorithm. If it is capable of identifying all vertices of the real convex hull, the algorithm is coined as real [32]; otherwise, it is considered an approximation algorithm [35, 36]. Finally, we can also classify convex hull algorithms into offline and online algorithms. The former uses all the data to compute the convex hull, while the latter employ newly arrived points to adapt an already existing convex hull [31].

Although many algorithms have been proposed for identifying the convex hull of datasets in low dimensions, still there is no efficient algorithm available to find the convex hull in higher dimensions. The complexity of the majority of proposed algorithms for two or three dimensions is $O(n \log n)$ while for high dimensions the complexity is $O(n^{\lfloor \frac{d}{2} \rfloor})$, where n is the number of samples in dataset and d is the sample dimension. According to the upper bound theory in computational geometry [37], the maximum number of facets for a convex hull with m vertices is $O(m^{\lfloor \frac{d}{2} \rfloor})$, which reflects the large memory requirements for those algorithms that construct the convex hull by enumerating facets, e.g., randomized incremental algorithms [34] and Quickhull [32].

Among all proposed algorithms, Quickhull is considered a quick deterministic real convex hull algorithm which is faster than other proposed algorithms in low dimensions. For dimensions $d \leq 3$ Quickhull runs in time $O(n \log r)$, where n and r are the number of points in the underlying dataset and the number of processed points, respectively. For $d \geq 4$, Quickhull runs in time $O(n f_r / r)$, where f_r is the maximum number of facets for r vertices. Since $f_r = O(r^{\lfloor \frac{d}{2} \rfloor} / \lfloor \frac{d}{2} \rfloor!)$, for high dimensions a massive number of facets would be generated for r vertices. Consequently, Quickhull is not feasible for high dimensions, both in terms of execution time and memory requirements (e.g., Quickhull algorithm suffers from insufficient memory in the case where it is applied on an artificial dataset including 1000 uniformly distributed random samples in 9 dimensions, in a computer with 4 Gigabytes RAM). The trend of memory consumption in Quickhull is described in Section 6.

Very recently, an on-line algorithm [27] has been proposed for application to SVMs. Its time complexity is at most $O(nd^4)$, which means that, for problems with $d > 8$, it has smaller complexity than the existing techniques. It incrementally forms an approximated convex hull of a dataset on the basis of two thresholds, L and M . The algorithm starts from a d -simplex and ends with an approximated convex hull with at most M vertices. Since a d -simplex has $d+1$ facets, it divides the space into $d+1$ partitions. In the first step, each partition whose number of samples is greater than L is divided into d new partitions, based on the furthest sample to the corresponding facet of each partition. This task is performed repeatedly for the new generated partitions and the furthest samples are marked as convex hull vertices. Afterwards, the sample whose distance to the current generated convex hull is maximum is marked as a vertex of convex hull. The procedure is executed until the number of vertices reaches the threshold M . Although the algorithm proposed in [27] is feasible to execute in high dimensions, it incorporates vertices which do not belong to the set of vertices of the real convex hull, as will be demonstrated in the results.

3 Proposed Algorithm

In order to overcome the shortcomings of Quickhull and the algorithm proposed in [27], we propose a randomized approximation algorithm that on the one hand, treats memory complexity efficiently and on the other hand, identifies vertices which belong to the real convex hull. Moreover, this algorithm is applicable in high dimensions efficiently.

In order to explain ApproxHull, first we need to explain two notions in computational geometry which are the hyperplane distance [38, 39] and the convex hull distance.

3.1 Hyperplane Distance

Suppose $V = [v_1, v_2, \dots, v_n]^T$ is a point, F is an n -vertex facet, and H is the corresponding hyper-plane of facet F in a n -dimensional Euclidean space. Also assume $a_1x_1 + a_2x_2 + \dots + a_nx_n + b = 0$, is the corresponding equation of H where $N = [a_1, a_2, \dots, a_n]^T$ and b are the normal vector and the offset, respectively.

The distance from point V to hyperplane H is computed by (1).

$$d_s(V, H) = \frac{a_1v_1 + a_2v_2 + \dots + a_nv_n + b}{\sqrt{a_1^2 + a_2^2 + \dots + a_n^2}} \quad (1)$$

3.2 Convex Hull Distance

Given a set $P = \{x_i\}_{i=1}^n \subset \mathbb{R}^d$ and a point $x \in \mathbb{R}^d$, the Euclidean distance between x and the convex hull of P , denoted by $\text{conv}(P)$, can be computed by solving the quadratic optimization problem stated in (2)

$$\begin{aligned} \min_a \quad & \frac{1}{2} a^T Q a - c^T a \\ \text{s. t.} \quad & e^T a = 1, a \geq 0 \end{aligned} \quad (2)$$

Where $e = [1, 1, \dots, 1]^T$, $Q = X^T X$ and $c = X^T x$, with $X = [x_1, x_2, \dots, x_n]$.

Suppose that the optimal solution of (2) is a^* ; then the distance of point x to $\text{conv}(P)$ is given by (3).

$$d_c(x, \text{conv}(P)) = \sqrt{x^T x - 2c^T a^* + a^{*T} Q a^*} \quad (3)$$

3.3 The proposed Algorithm

The idea behind ApproxHull is inspired from Quickhull, where the vertices of the real convex hull are identified based on the hyperplane distance of samples to the facets of the current convex hull. Like Quickhull, ApproxHull is an incremental algorithm; it starts with an initial convex hull and then the current convex hull grows iteratively by adding the new vertices into it. In order to overcome the challenges of time complexity and memory requirements in high dimensions, ApproxHull has two main properties. Firstly, it is an approximation algorithm which is translated into having a subset of the most informative vertices of the real convex hull through the use of a user-defined threshold. On the contrary, Quickhull finds the total vertices of the real convex hull which prevents it to be applicable in large, high dimensional datasets. Secondly, the convex hull obtained by the ApproxHull is only given by vertices, whereas Quickhull presents the convex hull in terms of both vertices and facets, which makes it

infeasible to be run for high dimensional datasets, due to the problem of high time complexity and memory requirements.

A pre-processing phase is performed on the original data set before applying ApproxHull. Duplicated rows (equal samples) and columns (same features), rows with missing values, and rows having non-numerical values are removed to decrease the risk of generating a singular matrix corresponding to a random invalid facet in ApproxHull.

ApproxHull consists of five main steps:

Step 1: Scaling each dimension to the range $[-1, 1]$.

Step 2: Identifying the maximum and minimum samples with respect to each dimension. These samples are considered as vertices of the initial convex hull.

Step 3: Generating a population of k facets based on current vertices of convex hull. In this step, the validity of all generated facets is checked in each iteration. A facet F of d points in d dimensions is valid if F is a full rank matrix. In ApproxHull, to guarantee that the population contains valid facets, two actions are considered. When d vertices of the current convex hull are selected to constitute a facet of the population, its validity is checked. Invalid facets are ignored, being substituted by another combination of d vertices of the current convex hull until a valid facet is found. In the case that there is a potential of generating invalid facets iteratively, in order to reduce the time being spent to ignore the invalid facets and generate the valid substitutions for them, the joggling method used in the Quickhull [40] can be employed as an optional action in the data preprocessing phase. Joggling the input is performed to solve precision error in computational geometry context. Mainly, in joggling (also called random perturbation) the input of each cell of the data set is modified by a small random quantity (positive or negative) to solve the problem of coplanar points that have the potential of generating invalid facets.

Step 4: Identifying the furthest points to each facet in the current facets population as new vertices of convex hull, if they have not been detected before. In order to detect the furthest points (i.e., those samples whose hyperplane distances are maximum with respect to a particular facet), firstly, the corresponding hyperplane equation of the facet is obtained by Algorithm 3 in terms of the normal vector and the offset. Secondly, the hyperplane distance of samples to the corresponding hyperplane is computed by (1). In fact, Algorithm 3 computes the hyperplane equation relying on the fact that the distance of the center point (a row vector whose elements are obtained by averaging each dimension of the dataset) to the hyperplane is a negative value. Algorithm 3 obtains the equation of the corresponding hyperplane equation of the facet in form $Ax = b$ where A and b are normal and offset of the hyperplane equation respectively.

Step 5: Updating the current convex hull by adding the newly found vertices to the current set of vertices.

Steps 3 to 5 are executed iteratively until one of the following two termination criteria is met:

- There are no newly found vertices in Step 4

- Let dc be the maximum of the approximated distances of the furthest points to the current convex hull in each iteration. If there are new vertices as a consequence of Step 4 and the difference between the maximum and the minimum of dc over the w last iterations is less than a user-defined threshold β (default value of 0.1), the algorithm ends.

The basic idea behind the second criterion is to avoid selecting new vertices that are very close to the current convex hull, not contributing this way with new information. As the convex hull generated by ApproxHull grows iteratively, the dc has a descending trend over iterations. Hence when the difference between the maximum and minimum of dc over the w last iterations is small, meaning the new found vertices are very close to the current convex hull, they can be ignored and the ApproxHull can be terminated.

In the ApproxHull algorithm the facets population size, the sliding window width, and the user-defined threshold β , can be tuned to manage the number of vertices of the approximated convex hull. As an example, in the experimental part of this work we are interested in having at most half of the training set points from the convex hull and the remaining points selected randomly from the complete data set. The value $\beta = 0.1$, which was obtained by trial and error, satisfies our expectation for a facet population size <1000 and a sliding window width <10 , in cases where the data was uniformly or normally sampled. In other applications where the data size (number of samples and dimensions) is high and the data distribution is unknown, the value of β can be tuned differently to meet the user-defined specification of the maximum percentage of approximated convex hull vertices in the training set.

Since computing the distance from a point to the current convex hull by solving the quadratic optimization problem defined in (2) is complex and time consuming in high dimensions, in ApproxHull the approximated distance of a newly found vertex to the current convex hull is computed on the basis of $2 * d$ vertices, where d denotes dimension, which are the nearest neighbors to the newly found vertex in the current convex hull.

In Step 3 of ApproxHull, in order to generate a population of facets based on the vertices of the current convex hull, two policies were tested: 1) a stochastic policy; 2) a GA [41, 42] based policy. In the first, k facets are generated in such a way that each vertex of a specific facet is generated by random selection among the vertices of the current convex hull. The stochastic policy algorithm is summarized in Algorithm 1. In the GA-based policy, k facets are generated by a GA so that $k = ps * ng$, where ps and ng denote the population size and the number of generations of the GA, respectively. In each generation of the GA execution, a new population of ps facets is created after employing crossover and mutation operators. In this policy the population of each generation is appended to the total population so that, in the end, there is a total population of facets with size $ps * ng$. The GA-based algorithm is summarized in Algorithm 2.

Since the idea behind ApproxHull is generating many different facets to help finding new vertices of the real convex hull, the diversity of generated facets is an important issue in the algorithm. In other words, more diversity in facets population provides a higher chance of detecting new vertices. Hence, the fitness value of a facet can be defined in terms of the inverse of occurrence ratio of its vertices over the current population. The fitness value of a

specific facet is high if, in average, the occurrence ratio of its vertices over the current population is low. In ApproxHull with GA-based Policy, the fitness value of a facet in population is measured by (4):

$$\frac{ps*d}{\sum_{i=1}^{d+1} N_i} \quad (4)$$

where ps , d and N_i are population size, dimension and number of facets in current population which share the i^{th} vertex of the facet, respectively.

In each iteration of the GA, parents are selected for mating using the Roulette Wheel method. Uniform crossover [42] is applied with a swapping probability of 0.5. The crossover probability is set to 0.7. For mutation, a vertex of a facet is selected randomly and replaced with another random vertex which has not been seen in the current population. The mutation probability is set to 0.05.

Figure 1 illustrates a simplified flowchart of the operations involved in ApproxHull.

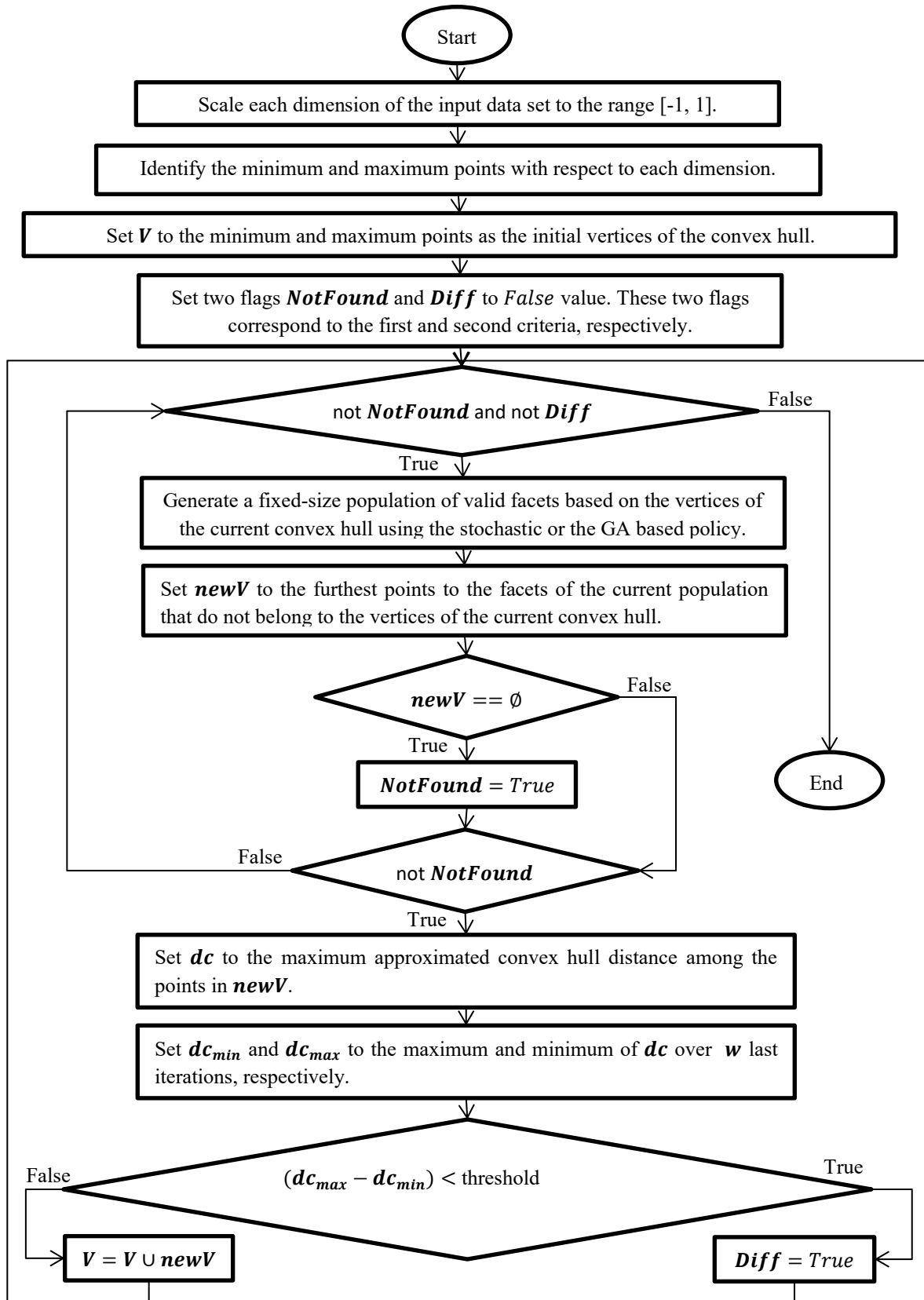


Fig. 1. Flow chart of ApproxHull.

Algorithm 1: ApproxHull with Stochastic Policy

Input: $DS = \{x_i\}_{i=1}^n \subseteq R^d$ as a set of samples obtained after the preprocessing of the original data, k denotes the population size of facets in d -dimensional space, w is an integer value as width of the sliding window and β as a user-defined threshold

1. Scaling each dimension of DS to the range $[-1, 1]$.
2. Let V denotes the maximum and minimum samples with respect to each dimension in DS .
3. $NotFound = False$
4. $Diff = False$
5. $iteration = 1$
6. $DC = \{\}$
7. **While** (not $NotFound$ and not $Diff$) **do**
8. Let P be an empty population.
9. **For** ($i = 1; i \leq k; i++$) **do**
10. $isValid = False$
11. **While** (not $isValid$) **do**
12. $j = 1$
13. Let F be an empty facet.
14. **While** ($j \leq d$) **do**
15. Select randomly a vertex v from V
16. **If** (v is not in F) **then**
17. $F = F \cup \{v\}$
18. $j = j + 1$
19. **End If**
20. **End While**
21. **If** ($\det(F) \neq 0$) **then**
22. $isValid = True$
23. **End If**
24. **End While**
25. $P = P \cup \{F\}$
26. **End For**
27. $newV = \{\}$
28. **For each** facet F in P **do**
29. Let H be the corresponding hyperplane equation of facet F which is obtained by Algorithm 3.
30. $mds = \max_{x_i \in DS} d_s(x_i, H)$
31. $FP = \{x | x \in DS \text{ and } d_s(x, H) = mds\}$
32. **For each** point fp in FP **do**
33. **If** (fp is not in V) **do**
34. $newV = newV \cup \{fp\}$
35. **End If**
36. **End For**
37. **End For**
38. **If** ($newV = \{\}$) **then**
39. $NotFound = True$
40. **End If**
41. **If** (not $NotFound$) **then**
42. $D = \emptyset$
43. **For** ($i = 1; i \leq |newV|; i++$) **do**
44. $N_i = \emptyset, T = V$
45. **For** ($k = 1; k \leq 2 * d; k++$) **do**
46. $nn = \arg \min \|z_i - v_j\|_2$
 $v_j \in T, j = 1, \dots, |T|$
 where $z_i \in newV$
47. $N_i = N_i \cup \{nn\}$
48. $T = T \setminus \{nn\}$
49. **End For**
50. $D = D \cup \{d_c(z_i, N_i)\}$
51. **End For**
52. $dc = \max D$
53. $DC = DC \cup \{dc\}$
54. **If** ($iteration \geq w$) **then**
55. Let dc_{min} be the minimum of dc in DC over w last iterations.
56. Let dc_{max} be the maximum of dc in DC over w last iterations.
57. **If** ($(dc_{max} - dc_{min}) < \beta$) **then**
58. $Diff = True$
59. **Else**
60. $V = V \cup \{newV\}$
61. **End If**
62. **End If**
63. **End If**
64. $iteration = iteration + 1$
65. **End While**

Output: V

Algorithm 2: ApproxHull with GA-based Policy

Input: $DS = \{x_i\}_{i=1}^n \subseteq R^d$ as a set of samples obtained after the preprocessing of the original data, ps as population size, pc as crossover probability, pm as mutation probability, ng as number of generation for GA, w is an integer value as width of the sliding window and β as a user-defined threshold.

1. Scaling each dimension of DS to the range $[-1, 1]$.
2. Let V denotes the maximum and minimum samples with respect to each dimension in DS .
3. $NotFound = False$
4. $Diff = False$
5. $iteration = 1$
6. $DC = \{\}$
7. **While** (not $NotFound$ and not $Diff$) **do**
8. Let P be an empty population with maximum size $ps * ng$.
9. Let G be an empty population with maximum size ps .
10. **For** ($i = 1; i \leq ps; i++$) **do**
11. $isValid = False$
12. **While** (not $isValid$) **do**
13. $j = 1$
14. Let F be an empty facet.
15. **While** ($j \leq d$) **do**
16. Select randomly a vertex v from V
17. **If** (v is not in F) **then**
18. $F = F \cup \{v\}$
19. $j = j + 1$
20. **End If**
21. **End While**
22. **If** ($\det(F) \neq 0$) **then**
23. $isValid = True$
24. **End If**
25. **End While**
26. $G = G \cup \{F\}$
27. **End For**
28. $ga_Iteration = 1$
29. **While** ($ga_Iteration \leq ng$) **do**
30. $P = P \cup G$
31. $Offsprings = \{\}$
32. **For each** $indv$ in G **do**
33. Compute the corresponding fitness value of $indv$.
34. **End For**
35. Let $MPool$ denotes an empty mating pool.
36. **For** ($i = 1; i \leq ps; i++$) **do**
37. Select one random parent $P1$ from G using Roulette Wheel method.
38. $MPool = MPool \cup \{P1\}$
39. **End For**
40. $j = 0$
41. **While** ($j < ps$) **do**
42. Select two random parents $P1$ and $P2$ from $MPool$.
43. Let rc is a random number from range $[0, 1]$.
44. **If** ($rc \leq pc$)
45. Do uniform crossover on $P1$ and $P2$ and consider $children$ for the result of crossover.
48. **If** ($rm \leq pm$)
49. Do mutation on $child$
50. **End If**
51. **End For**
52. **For each** $child$ in $children$ **do**
53. **If** ($\det(child) \neq 0$) **then**
54. $Offsprings = Offsprings \cup \{child\}$
55. **End If**
56. **End For**
57. $j = |Offsprings|$
58. **End If**
59. **End While**
60. Let G includes the first ps individuals of $Offsprings$.
61. $ga_Iteration = ga_Iteration + 1$
62. **End While**
63. $newV = \{\}$
64. **For each** facet F in P **do**
65. Let H be the corresponding hyperplane equation of facet F which is obtained by Algorithm 3.
66. $mds = \max_{x_i \in DS} d_s(x_i, H)$
67. $FP = \{x | x \in DS \text{ and } d_s(x, H) = mds\}$
68. **For each** point fp in FP **do**
69. **If** (fp is not in V) **do**
70. $newV = newV \cup \{fp\}$
71. **End If**
72. **End For**
73. **End For**
74. **If** ($newV = \{\}$) **then**
75. $NotFound = True$
76. **End If**
77. **If** (not $NotFound$) **then**
78. $D = \emptyset$
79. **For** ($i = 1; i \leq |newV|; i++$) **do**
80. $N_i = \emptyset, T = V$
81. **For** ($k = 1; k \leq 2 * d; k++$) **do**
82. $nn = \argmin \|z_i - v_j\|_2$
 $v_j \in T, j = 1, \dots, |T|$
where $z_i \in newV$
83. $N_i = N_i \cup \{nn\}$
84. $T = T \setminus \{nn\}$
85. **End For**
86. $D = D \cup \{d_c(z_i, N_i)\}$
87. **End For**
88. $dc = \max D$
89. $DC = DC \cup \{dc\}$
90. **If** ($iteration \geq w$) **then**
91. Let dc_{min} be the minimum of dc in DC over w last iterations.
92. Let dc_{max} be maximum of dc in DC over w last iterations.
93. **If** ($(dc_{max} - dc_{min}) < \beta$) **then**
94. $Diff = True$
95. **Else**
96. $V = V \cup \{newV\}$

46.	For each <i>child</i> in <i>children</i> do	97.	End If
47.	Let <i>rm</i> is a random number from	98.	End If
	range [0, 1]	99.	End If
		100.	<i>iteration</i> = <i>iteration</i> + 1
		101.	End While
			Output: V

Algorithm 3: Obtaining the corresponding hyperplane of a facet

Input: $DS = \{x_i\}_{i=1}^n \subseteq R^d$ as a set of samples and $F = \{v_i\}_{i=1}^d$ as a particular facet so that v_i is a row vector which denotes a specific sample in DS .

1. Let c is a row vector which denotes the center point of all samples in DS .

2. $U = \{u_i \mid u_i = v_i - c\}_{i=1}^d$

3. $A = \{\}$

4. $b = \{\}$

5. $A = U^{-1}e$ where $e = [1, 1, \dots, 1]^T$

6. $b = 1 + cA$

7. $t = \sqrt{\sum_{i=1}^d a_i^2}$ where $a_i \in A$
for $i = 1, 2, \dots, d$

8. $nA = \{\frac{a_i}{t} \mid a_i \in A\}_{i=1}^d$

9. $nb = \frac{b}{t}$ **Output:** nA and nb .

4 Simulation Results

Three experiments were executed to evaluate ApproxHull performance and its effect on the accuracy in classification and approximation tasks. The algorithm has been implemented in Python and C languages, and was executed in a computer with an Intel i5 CPU core and 4 Gigabytes of RAM.

4.1 Experiment 1

ApproxHull was applied on four artificial datasets named UDS1, UDS2, UDS3 and UDS4. All datasets are composed of uniformly distributed random samples which were generated by the built-in MATLAB function *rand*. The description of the datasets is given in Table I.

Dataset Name	dim	#S
UDS1	3	4000
UDS2	4	4000
UDS3	5	4000
UDS4	6	4000

TABLE I - DESCRIPTION OF ARTIFICIAL DATASETS CONSISTING OF UNIFORMLY DISTRIBUTED RANDOM SAMPLES. DIM AND #S DENOTE THE NUMBER OF DIMENSIONS AND SAMPLES, RESPECTIVELY.

Since Quickhull is a deterministic algorithm, in this experiment it is considered a baseline to which ApproxHull and Wang's algorithm [27], both being approximation convex hull algorithms, are compared. We use two criteria for comparison: P and R defined in (5) and (6).

$$P = \frac{\#(V_R \cap V_P)}{\#V_P} * 100 \quad (5)$$

$$R = \frac{\#(V_R \cap V_P)}{\#V_R} * 100 \quad (6)$$

V_R is the set of vertices obtained by employing the Quickhull algorithm and V_P is the set of vertices obtained by applying one of the other algorithms. Basically, criterion P shows the amount of precision of an algorithm in approximating the Quickhull results, while criterion R

denotes how much the results obtained by an algorithm are similar to those obtained by Quickhull.

In this experiment ApproxHull considering both facet generation policies and Wang's algorithm were executed for ten runs. For the latter L was set to $0.01n$ for all datasets, and M was set as $M \geq 0.02n$, $M \geq 0.07n$, $M \geq 0.1n$ and $M \geq 0.14n$, for UDS1, UDS2, UDS3 and UDS4, respectively. n is the number of samples.

For ApproxHull with Stochastic Policy the sliding window size, w , was set to 10 for all datasets and k (population size) was set to 4000, 5000, 6000, and 7000 for datasets UDS1, UDS2, UDS3 and UDS4, respectively. For ApproxHull with GA-based Policy, w was also set to 10 for all datasets and the number of generations, ng , was set to 50. ps (population size) was set to 80, 100, 120 and 140 for datasets UDS1, UDS2, UDS3 and UDS4, respectively.

Fig. 2 and 3 show the average values of P and R for the results obtained on datasets UDS1 to UDS4 by ApproxHull (with both policies) and by Wang's algorithm. By analyzing Fig. 2 it may be concluded that ApproxHull identifies only true vertices that belong to the real convex hull while Wang's algorithm selects vertices which are not in the real convex hull. Moreover, according to Fig. 3, ApproxHull, using either Stochastic Policy or GA-based Policy, detects more vertices of the real convex hull in comparison to the Wang's algorithm. Fig. 3 also shows that ApproxHull with Stochastic Policy could identify more vertices of the real convex hull in comparison to ApproxHull employing the GA-based policy.

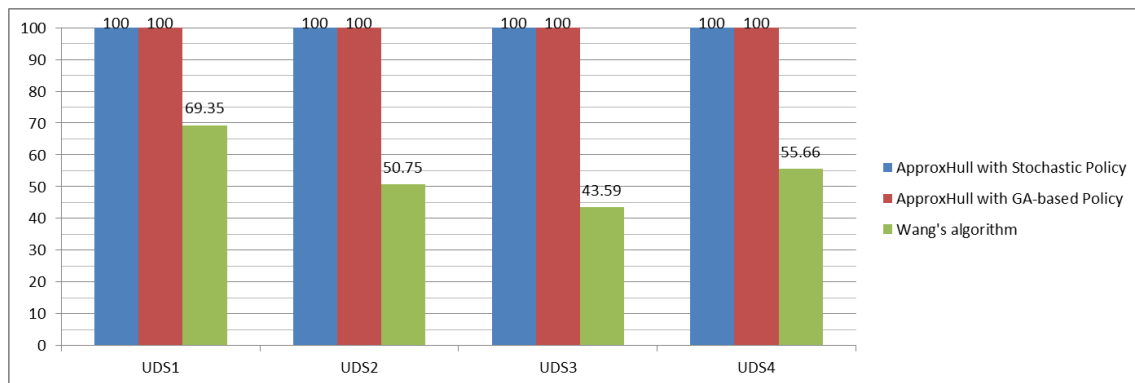


Fig. 2. Average value of criterion P for ApproxHull with both policies and Wang's algorithm on UDS1 to UDS4.

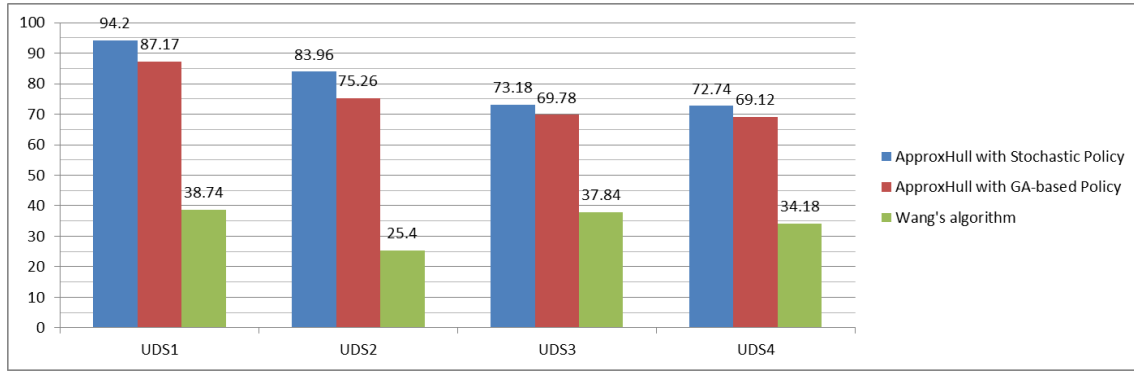


Fig. 3. Average value of criterion R for ApproxHull with both policies and the Wang's algorithm on UDS1 to UDS4.

Another experiment was conducted using normally distributed random samples which were generated by built-in MATLAB function *normrnd*, where the mean and the standard deviation were set to 0 and 1, respectively. In this experiment ApproxHull with both policies was applied on four normally distributed artificial datasets for ten runs. The experiments were executed in the same conditions as described before.

Fig. 4 shows the average value of R for the results obtained by ApproxHull using both policies on datasets NDS1 to NDS4. The results in terms of criterion P were 100%, as in the previous case. Table II and III also show the run time of ApproxHull using both policies on datasets in both cases.

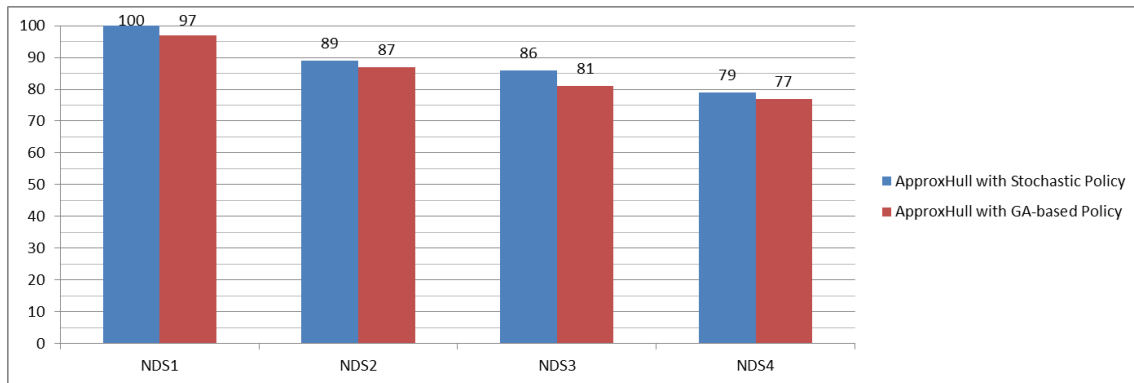


Fig. 4. Average value of criterion R for ApproxHull using both policies on NDS1 to NDS4.

	UDS1	UDS2	UDS3	UDS4
Stochastic Policy	11.50	25.86	55.44	115.59
GA-based Policy	17.11	58.66	208.03	323.68

TABLE II – RUN TIME (IN SECONDS) OF APPROXHULL WITH STOCHASTIC POLICY AND GA-BASED POLICY ON DATASETS UDS1-4.

	NDS1	NDS2	NDS3	NDS4
Stochastic Policy	6.30	11.86	30.33	50.20
GA-based Policy	10.06	19.24	90.06	156.78

TABLE III – RUN TIME (IN SECONDS) OF APPROXHULL WITH STOCHASTIC POLICY AND GA-BASED POLICY ON DATASETS NDS1-4.

Analyzing Fig. 3 and Fig. 4, ApproxHull with Stochastic Policy on both groups of artificial datasets has a better performance than the version employing the GA-based policy. In addition, according to Table II and III, the former is faster than the latter for all cases considered. For this reason, the Stochastic Policy will be used subsequently.

4.2 Experiment 2

In this experiment, ApproxHull was applied as a method for data selection in classification tasks. In order to evaluate the accuracy of the classification model, two cases were considered. In the first, ten training datasets were generated by random selection of samples from the whole dataset. In the second case, ten training datasets were generated, each one of them incorporating vertices of the approximated convex hull (which were obtained by ApproxHull) as well as random samples from the remaining dataset. The algorithm was applied separately for positive and negative classes. The datasets employed for classification were taken from [43]. The MATLAB SVM tool with Gaussian RBF (Radial Basis Function) kernel was used to design classifiers in both scenarios. The description of each dataset along with their corresponding hyper-parameters' values for the SVM classifiers is given in Table IV.

In this experiment, the following classification rate criterion was used:

$$\text{Classification Rate} = \frac{TN+TP}{TN+TP+FP+FN} , \quad (7)$$

where TP , TN , FP and FN denote the number of True Positive, True Negative, False Positive and False Negative results respectively. Table V shows the results obtained in the two cases for the datasets described in Table IV.

According to the fourth column of Table V, for all datasets the data selection mechanism employing ApproxHull has improved the accuracy of the corresponding classifiers in comparison with the random data selection method. For the Breast Cancer and the Letter datasets, the highest and lowest improvements were achieved, respectively. For the Cover Type, both algorithms achieved perfect classification. The average classification rate for datasets Satellite, Letter and Cover Type, in the second case is equal to 1 which means that perfect classification is obtained for these datasets.

Dataset	Class1 / Class2	#F	#DS	#TR	#TE	C	γ
Breast Cancer	"Malignant" / "Benign"	30	569	376	193	1	0.05
Parkinson	"Yes" / "No"	26	1040	686	354	200	0.1
Satellite	"Red Soil" / "Grey Soil"	36	2033	1342	691	500	0.1142
Letter	"A" / "B"	16	1555	1026	529	1	0.6576
Cover Type	"Douglasfir" / "Krummholz"	54	37877	24999	12878	1	0.5

TABLE IV - DESCRIPTION OF THE DATASETS USED IN CLASSIFICATION. #F, #DS, #TR, #TE ARE THE NUMBER OF FEATURES, TOTAL NUMBER OF SAMPLES, NUMBER OF TRAINING SAMPLES AND TEST SAMPLES, RESPECTIVELY. C AND γ ARE THE SVM HYPER-PARAMETERS.

Dataset	$CR_{Te}(1)$	$CR_{Te}(2)$	$CR_{Te}(2) - CR_{Te}(1)$
Breast Cancer	0.963	0.981	0.018
Parkinson	0.656	0.667	0.011
Satellite	0.990	1.000	0.010
Letter	0.993	1.000	0.007
Cover Type	1.000	1.000	0.000

TABLE V - AVERAGE CLASSIFICATION RATE FOR TEST DATASET IN TWO CASES FOR ALL DATASETS IN TABLE IV. $CR_{Te}(1)$ AND $CR_{Te}(2)$ DENOTE THE CLASSIFICATION RATES FOR THE TEST DATASET IN THE FIRST CASE (RANDOM SELECTION) AND SECOND CASE (DATA SELECTION USING APPROXHULL) RESPECTIVELY.

Additionally, as the previous datasets were balanced data sets, the same procedure, using the same SVM tool, was employed to a problem of automatic diagnosis of Cerebral Vascular Accidents (CVAs), from Computerized Tomography (CT) images. The application is described in, for instance, [44, 45] and, for the point of view of this paper, is a binary classification problem, using 51 features, with the aim of classifying each pixel in the intracranial area of each CT slice as normal, or abnormal (corresponding to a lesion). Using 150 CT slices corresponding to 7 exams, we had 1,867,602 pixels, from which 64,786 (around 3.5%) were abnormal. This is clearly a very bad-balanced problem. ApproxHull has been applied for this large data set, and training and test datasets with the sizes of 20,000 and 14,000 were constructed as explained previously. The average values of the classification rate obtained for the test sets, over 10 experiments, for random selection and using ApproxHull were 0.972 and 0.983, respectively. This example demonstrates that, for a completely unbalanced problem, the use of ApproxHull again achieved better results than random selection, and that it is applicable to large datasets.

In order to assess the statistical significance of the last results, as well as the ones presented in Table V, we performed, according to the suggestion of [46], a Wilcoxon signed-ranks test [47]. This is a non-parametric test, which ranks the differences in performances of, in this case, the application of two different training sets using the same classifier for each data set, ignoring the signs, and compares the ranks for the positive and the negative differences. Assume that d_i is the difference between the performance scores (in this case the classification rate of the test sets) of the two approaches on the i^{th} out of N data sets. The differences are ranked according to their absolute values; average ranks are assigned in case of ties. Let R^+ be the sum of ranks for the data sets on which the second approach outperformed the first, and R^- the sum of ranks for the opposite. Ranks of $d_i = 0$ are split evenly among the sums; if there is an odd number of them, one is ignored:

$$R^+ = \sum_{d_i > 0} Rank(d_i) + \frac{1}{2} \sum_{d_i = 0} Rank(d_i) \quad R^- = \sum_{d_i < 0} Rank(d_i) + \frac{1}{2} \sum_{d_i = 0} Rank(d_i)$$

Assume that T is the smaller of the sums, $T = \min(R^+, R^-)$. For a large number of data sets

$$\text{(in this case } N = 60\text{), the statistic } z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \text{ is distributed approximately}$$

normally. With $\alpha = 0.05$ the null hypothesis (both approaches perform equally well) can be

rejected if z is smaller than -1.96. For this case, $z = -5.35$, which clearly indicates that the improvements obtained with ApproxHull are statistically significant.

4.3 Experiment 3

Experiment 3 was conducted to find out how much improvement can be obtained for regression models by using data selection employing ApproxHull.

As in Experiment 2, two approaches were analyzed for comparison: 1) generating ten training datasets by random selection; 2) generating ten training datasets by applying ApproxHull together with random selection. The datasets which are used for regression were taken from [43, 48]. The description of each dataset is given in Table VI. The MLP (Multilayer Perceptron Neural Network) implemented in MATLAB was employed with two hidden layers and the output layer with one linear neuron. For all datasets except the Concrete dataset, both hidden layers had ten sigmoidal neurons. For the Concrete dataset, both hidden layers employed five sigmoidal neurons. The training algorithm described in [49] is employed, terminating if one of the following conditions is met: early-stopping, the number of training iterations exceeds 100 iterations, or the optimization criterion described in [49] is met, where $\tau = 10^{-3}$ is a measure of the desired number of correct digits in the training criterion.

Dataset	#F	#DS	#TR	#TE	#VAL
Puma	32	8192	4915	1638	1639
Bank	32	8192	4915	1638	1639
CompAct	21	8192	4915	1638	1639
Concrete	8	1030	618	206	206
Skillcraft	18	3338	2003	667	668

TABLE VI - DESCRIPTION OF THE DATASETS USED IN REGRESSION. #F, #DS, #TR, #TE AND #VAL ARE THE NUMBER OF FEATURES, TOTAL SAMPLES, TRAINING SAMPLES, TEST SAMPLES AND VALIDATION SAMPLES, RESPECTIVELY.

The RMSE (Root Mean Squared Error) criterion is employed to evaluate the accuracy of the models. Tables VII and VIII show the results obtained in the test and validation for the datasets described in Table VI.

Table VII shows the average RMSE for the test datasets (i.e., data used for early stopping) in the two mentioned cases. As it may be seen in the sixth column, the regression model which resulted from the data selected by ApproxHull has lower approximation error. Table VIII shows the average RMSE for the validation sets (i.e., data not used in the model design). Again, it may be concluded that the use of ApproxHull in the data selection phase, decreases the error for all datasets except for Skillcraft, which has an identical value.

Dataset	Initial $E_{Te}(1)$	$E_{Te}(1)$	Initial $E_{Te}(2)$	$E_{Te}(2)$	$E_{Te}(1) - E_{Te}(2)$
Puma	0.336	0.076	0.326	0.073	0.003
Bank	0.293	0.209	0.260	0.195	0.014
CompAct	0.193	0.082	0.166	0.049	0.033
Concrete	0.374	0.161	0.329	0.143	0.018
Skillcraft	0.428	0.404	0.382	0.337	0.067

TABLE VII - AVERAGE RMSE FOR TEST DATASET IN TWO CASES FOR ALL DATASETS IN TABLE VI. $E_{Te}(1)$ AND $E_{Te}(2)$ DENOTE RMSE FOR TEST DATASET IN FIRST CASE (RANDOM SELECTION) AND SECOND CASE (DATA SELECTION USING APPROXHULL) RESPECTIVELY.

Dataset	Initial $E_{val}(1)$	$E_{val}(1)$	Initial $E_{val}(2)$	$E_{val}(2)$	$E_{val}(1) - E_{val}(2)$
Puma	0.339	0.076	0.324	0.073	0.003
Bank	0.296	0.209	0.259	0.194	0.015
CompAct	0.194	0.061	0.169	0.048	0.013
Concrete	0.359	0.162	0.329	0.147	0.015
Skillcraft	0.406	0.334	0.382	0.334	0.000

TABLE VIII - AVERAGE RMSE FOR VALIDATION DATASET IN TWO CASES FOR ALL DATASETS IN TABLE VI. $E_{val}(1)$ AND $E_{val}(2)$ DENOTE RMSE FOR VALIDATION DATASET IN FIRST CASE (RANDOM SELECTION) AND SECOND CASE (DATA SELECTION USING APPROXHULL) RESPECTIVELY.

For completeness, the second and the fourth columns in the last two tables illustrate the average initial values (before training was performed) of the RMSEs, for both approaches, for the test and validation datasets.

Performing the Wilcoxon signed-ranks test on the RMSEs of the test and validation data sets, with $N=50$, the z values obtained were are -4.73 and -4.18, respectively. In the same way as in the classification problems, the improvements obtained with ApproxHull were considered to be statistically significant.

To summarize the results, among the 15 performance values presented in Tables V, VII and VIII, as well as for the CVA problem, the use of ApproxHull for data selection achieves better results than those obtained by using random data selection in 14 cases, and achieves equal performance in 2 cases.

5 Run Time Analysis

The ApproxHull run time with Stochastic Policy depends on five factors including the size of the involved dataset, i.e., the number of samples and features, population size (input parameter k), number of iterations, number of vertices of convex hull found, and on the distribution of samples in the dataset. In order to analyze the dependency of the run time with these factors, two experiments were conducted. First, the algorithm was applied to all the datasets described in Table IV and VI for ten times. For all datasets, k (population size) and w (width of sliding window) were set to 1000 and 5, respectively. Fig. 5 shows the average percentage of total samples identified as vertices of convex hull for each dataset described in Tables IV and VI.

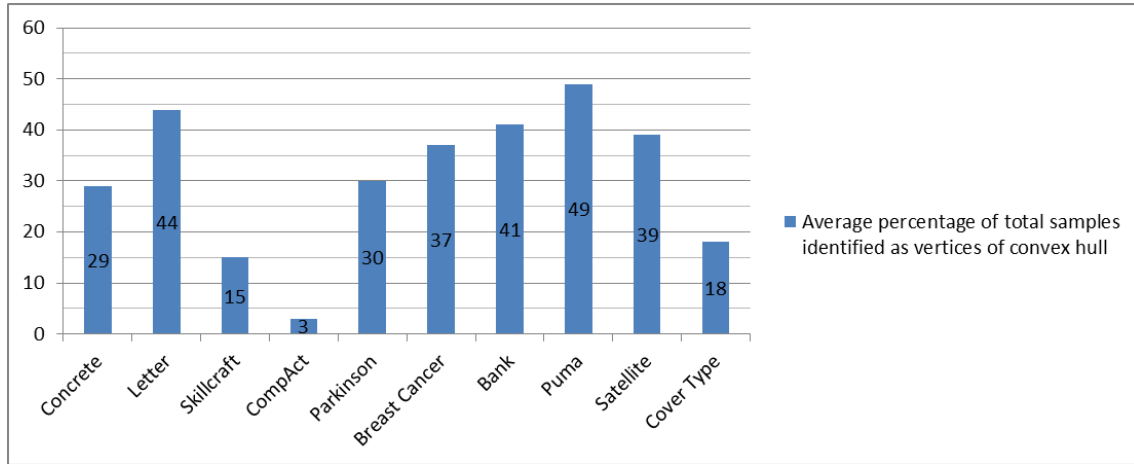


Fig. 5. Average percentage of total samples identified as vertices of convex hull for each dataset described in Tables IV and VI.

Fig. 6 illustrates the average number of iterations that were used to terminate the algorithm for each dataset. The corresponding average run time for each dataset is given in Table IX. As it can be seen in this table, the highest and lowest average run times are related to datasets Cover Type and Concrete, respectively. Cover Type is the largest dataset in terms of number of samples and features while Concrete has the least number of features and is the second smallest dataset with respect to number of samples. Although datasets Bank and Puma have the same size, the average run time for Bank is larger than that for Puma, because the average number of iterations for Bank is larger than that for Puma. This specific result related to datasets Puma and Bank reveals the fact that the distribution of samples can influence the run time.

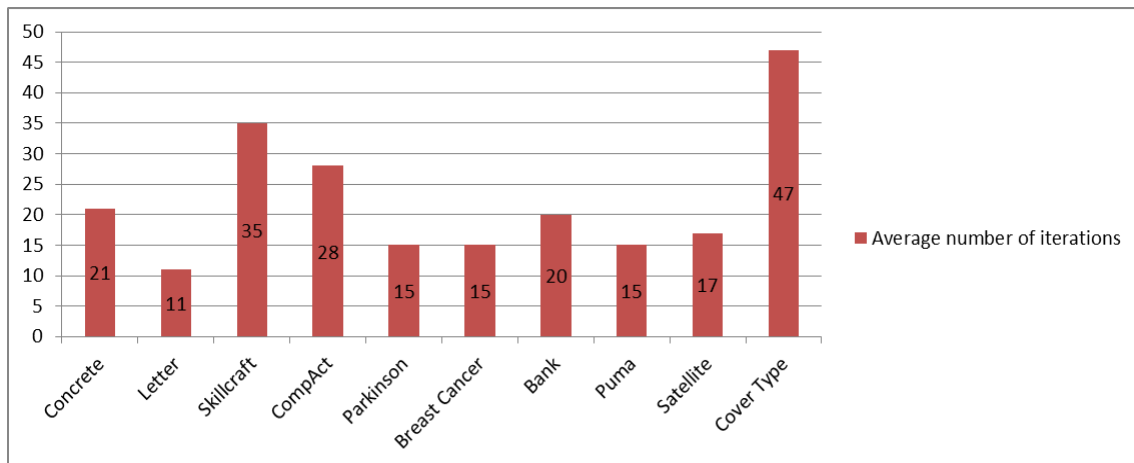


Fig. 6. Average number of iterations in ApproxHull for each dataset described in Tables IV and VI.

Dataset	Average Run Time (in seconds)
Concrete	11.78
Letter	19.13
Skillcraft	37.70
ComAct	37.39
Breast Cancer	8.10
Bank	257.34
Puma	174.24
Satellite	62.80

Cover Type	1280.16
------------	---------

TABLE IX. AVERAGE RUN TIME OF APPROXHULL ON DATASETS DESCRIBED IN TABLES IV AND VI.

Another experiment based on two groups of artificial datasets was conducted to clarify further the relationship between the run time of ApproxHull and the above mentioned factors.

The first group included thirty datasets which were composed of uniformly distributed random samples. The number of samples and the dimensions are in ranges [1000, 5000] and [4, 30] respectively. The second group employed the same number of datasets, in the same conditions, but using normally distributed random samples.

Fig. 7 and 8 show the effect of dataset size (i.e., number of samples and dimensions) on the run time of ApproxHull on five datasets with different number of samples and dimensions. For all datasets of both groups, population size (i.e., input parameter k) is set to 2000. It can be seen that in both groups of datasets, for a constant number of samples, by increasing the dimension, the run time of ApproxHull rises; for a constant dimension, by raising the number of samples, the run time also increases. It can also be observed that an increase in dimension is translated into a larger increase of the run time than an increase of the number of samples.

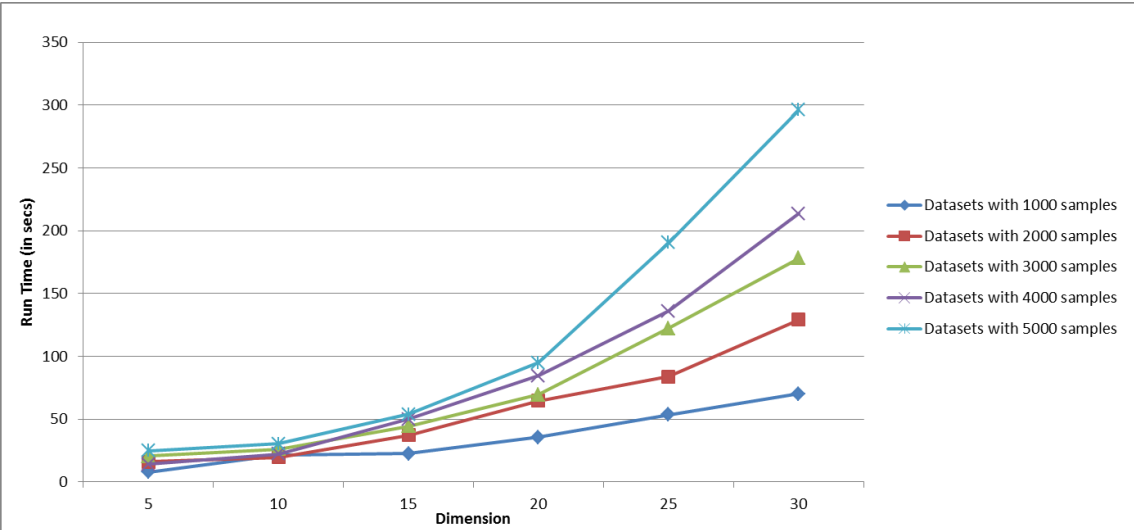


Fig. 7. Relationship between the size of five datasets containing uniformly distributed random samples and the run time of ApproxHull.

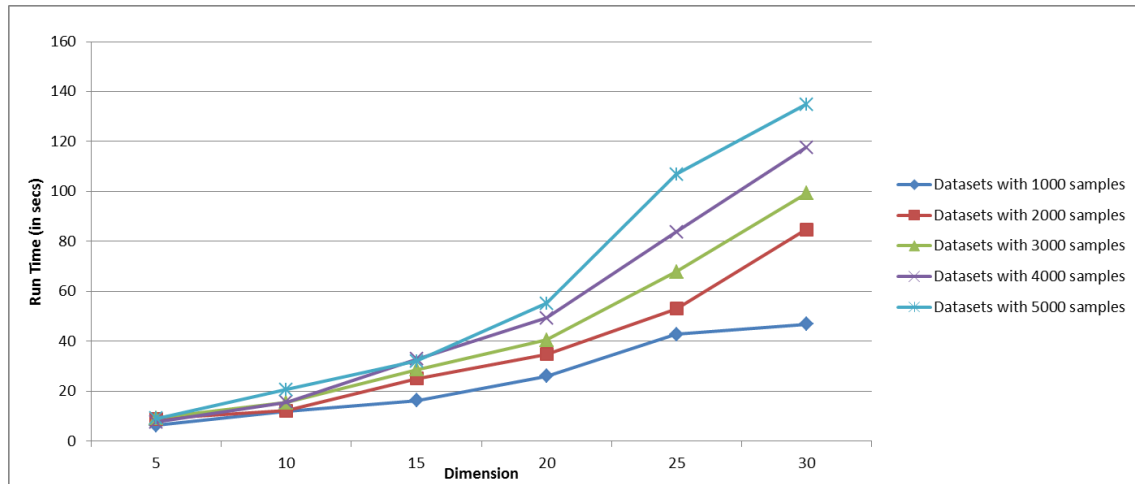


Fig. 8. Relationship between the size of five datasets containing normally distributed random samples and the run time of ApproxHull.

Figs. 9 and 10 illustrate the influence of population size (i.e., input parameter k) on the run time of ApproxHull for both groups of datasets. For each group, six datasets with 5000 samples are considered, with varying dimension. As it can be seen, on the one hand the population size has less influence on the run time in comparison with the effect of dataset size; on the other hand, if we enlarge the population size, this is not always translated into a run time increase (although this usually happens).

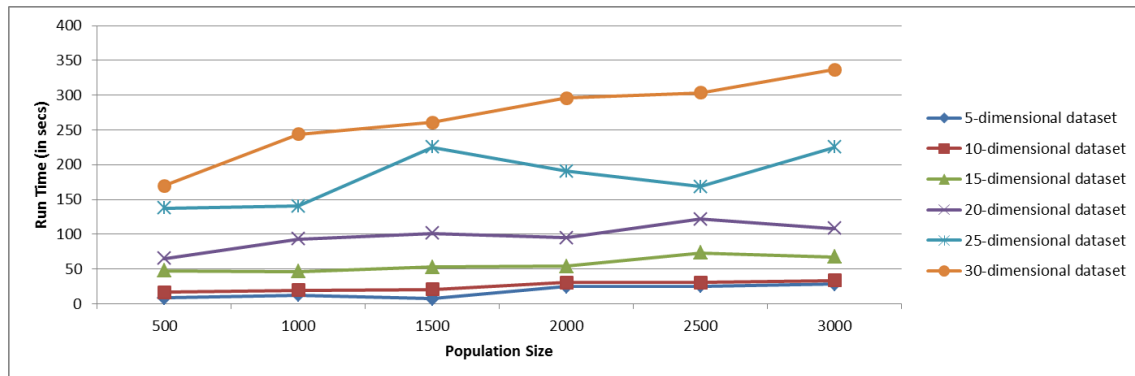


Fig. 9. Relationship between the population size (input parameter k) and the run time of ApproxHull on six datasets containing uniformly distributed random samples.

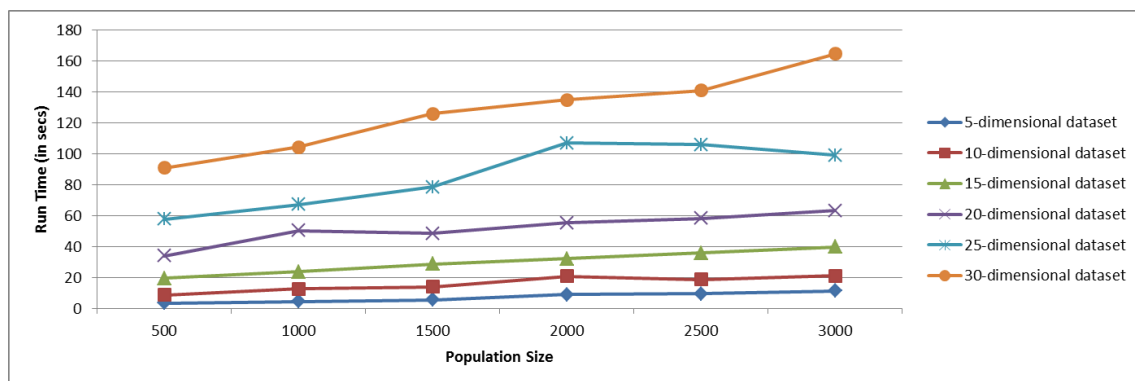


Fig. 10. Relationship between the population size (input parameter k) and the run time of ApproxHull on six datasets containing normally distributed random samples.

As an example, in Fig. 8 for the 25-dimensional dataset, by increasing population size from 1500 to 2000, the run time decreases. This happened because the number of iterations is equal to 44 in the case where the population size is set to 1500, whereas it is equal to 19 in the case where the population size is set to 2000. The corresponding number of iterations for different population sizes shown in Figs. 9 and 10 are given in Fig. 11 and 12.

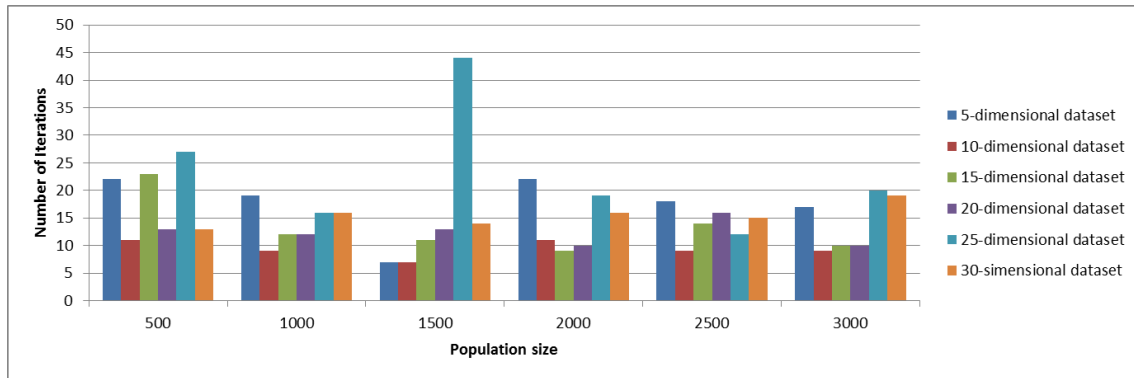


Fig. 11. Number of iterations in ApproxHull with six values for population sizes on six datasets containing 5000 uniformly distributed random samples.

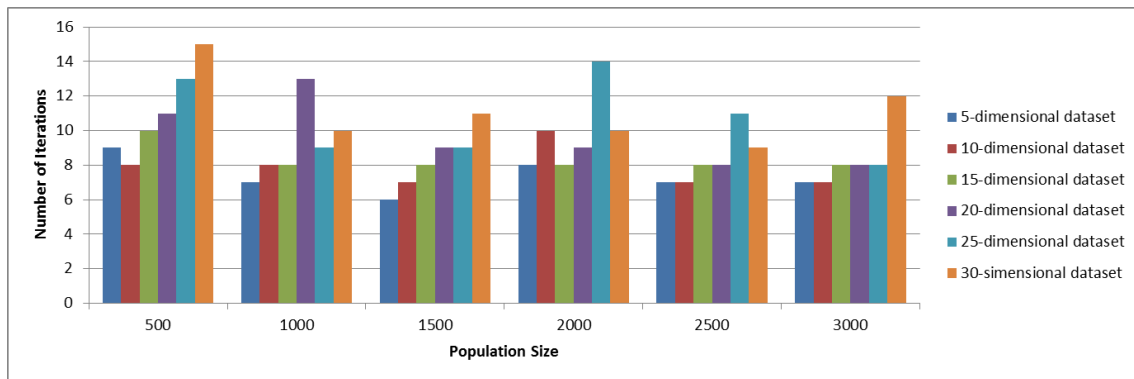


Fig. 12. Number of iterations in ApproxHull with six values for population sizes on six datasets containing 5000 normally distributed random samples.

Similarly, for the second group of datasets, increasing the population size does not always lead to a longer run time. For example, in Fig. 12, for the 20-dimensional dataset, by increasing population size from 1000 to 1500, the run time decreases because the corresponding number of iterations for population size 1000 is equal to 13, while it is equal to 9 for population size 1500.

From Fig. 13 and 14, it may be seen that in both group of datasets, an increase in population size does not always lead to an increase in percentage of samples identified as vertices of the convex hull.

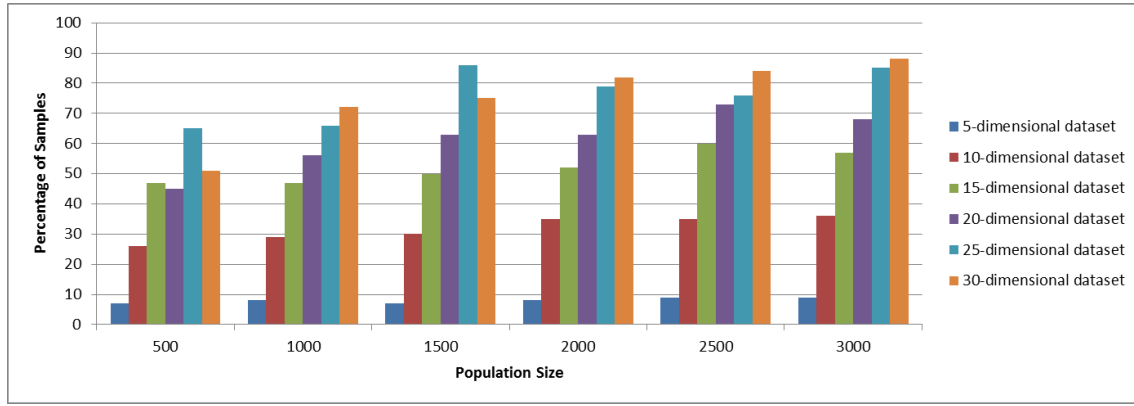


Fig. 13. Percentage of samples identified as vertices of convex hull by employing ApproxHull with six values for population sizes on six datasets containing 5000 uniformly distributed random samples.

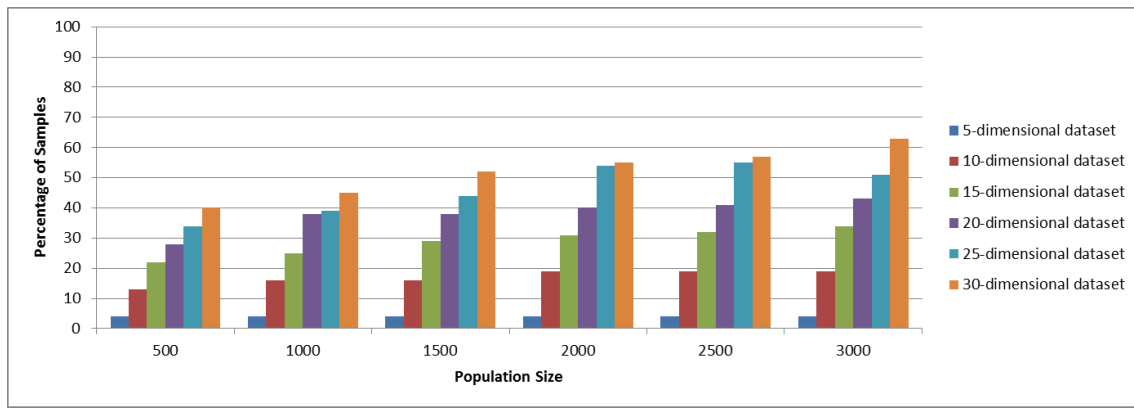


Fig. 14. Percentage of samples identified as vertices of convex hull by employing ApproxHull with six values for population sizes on six datasets containing 5000 normally distributed random samples.

From a data distribution point of view, Table X shows that the minimum and maximum run times of ApproxHull on datasets containing normally distributed random samples (i.e., the first group of datasets) are smaller than those on datasets containing uniformly distributed random samples (i.e., the second group of datasets). Correspondingly, Table XI illustrates that the minimum and the maximum percentages of samples identified as vertices of convex hull from datasets involving normally distributed random samples are less than those from datasets involving uniformly distributed random samples.

Cases	RTMIN_1	RTMIN_2	RTMAX_1	RTMAX_2
Datasets with 1000 samples	7.81	6.31	70.16	46.83
Datasets with 2000 samples	15.82	8.96	128.98	84.73
Datasets with 3000 samples	20.42	9.26	178.17	99.28
Datasets with 4000 samples	13.92	7.64	213.69	117.64
Datasets with 5000 samples	24.87	9.09	295.9	134.82

TABLE X – CORRESPONDING MINIMUM AND MAXIMUM RUN TIME OF APPROXHULL ON DATASETS USED IN FIG. 7 AND 11. RTMIN_1 AND RTMIN_2 DENOTE THE MINIMUM RUN TIME IN THE FIRST AND THE SECOND GROUP OF DATASETS RESPECTIVELY. RTMAX_1 AND RTMAX_2 DENOTE THE MAXIMUM RUN TIME IN THE FIRST AND THE SECOND GROUP OF DATASETS, RESPECTIVELY.

Dataset	PMIN_1	PMIN_2	PMAX_1	PMAX_2
5-dimensional dataset	7	4	9	4
10-dimensional dataset	26	13	36	19
15-dimensional dataset	47	22	60	34
20-dimensional dataset	45	28	73	43
25-dimensional dataset	65	34	86	55
30-dimensional dataset	51	40	88	63

TABLE XI – CORRESPONDING MINIMUM AND MAXIMUM PERCENTAGE OF SAMPLES IDENTIFIED AS VERTICES OF CONVEX HULL FROM DATASETS USED IN FIG. 10 AND 14. PMIN_1 AND PMIN_2 DENOTE THE MINIMUM PERCENTAGE IN FIRST AND SECOND GROUP OF DATASETS. PMAX_1 AND PMAX_2 DENOTE THE MAXIMUM PERCENTAGE IN FIRST AND SECOND GROUP OF DATASETS.

In order to extract an approximate mathematical model specifying the relationship between the run time of ApproxHull as a function of the dataset size, population size, number of iteration and number of convex hull vertices, we employed the ASMOD algorithm [50] on the data obtained in the last experiment described above. The ASMOD algorithm is a design technique for B-spline neural networks. B-spline neural networks belong to the class of networks denoted as lattice-based associative memory networks, where the basis functions are piecewise polynomials. For more information on B-spline networks, please consult [51, 52].

For each group of datasets, we collected 360 records of data by running ApproxHull on the corresponding datasets. According to the mathematical model which was obtained for each group of datasets, the time complexity of ApproxHull for both group of datasets can be approximated as $O(n^2 d^3 v^3 + i^3 p^3)$ where n , d , v , i and p denote the number of samples, dimension, number of convex hull vertices found, number of iterations and population size, respectively. In order to assess the accuracy of the model obtained for the run time of ApproxHull, Fig. 15 presents the run time of ApproxHull (data scaled in a range $[-1, 1]$) and the error obtained by the model in two groups of datasets. As it can be seen, for both groups of datasets, a good accuracy for the model has been obtained.

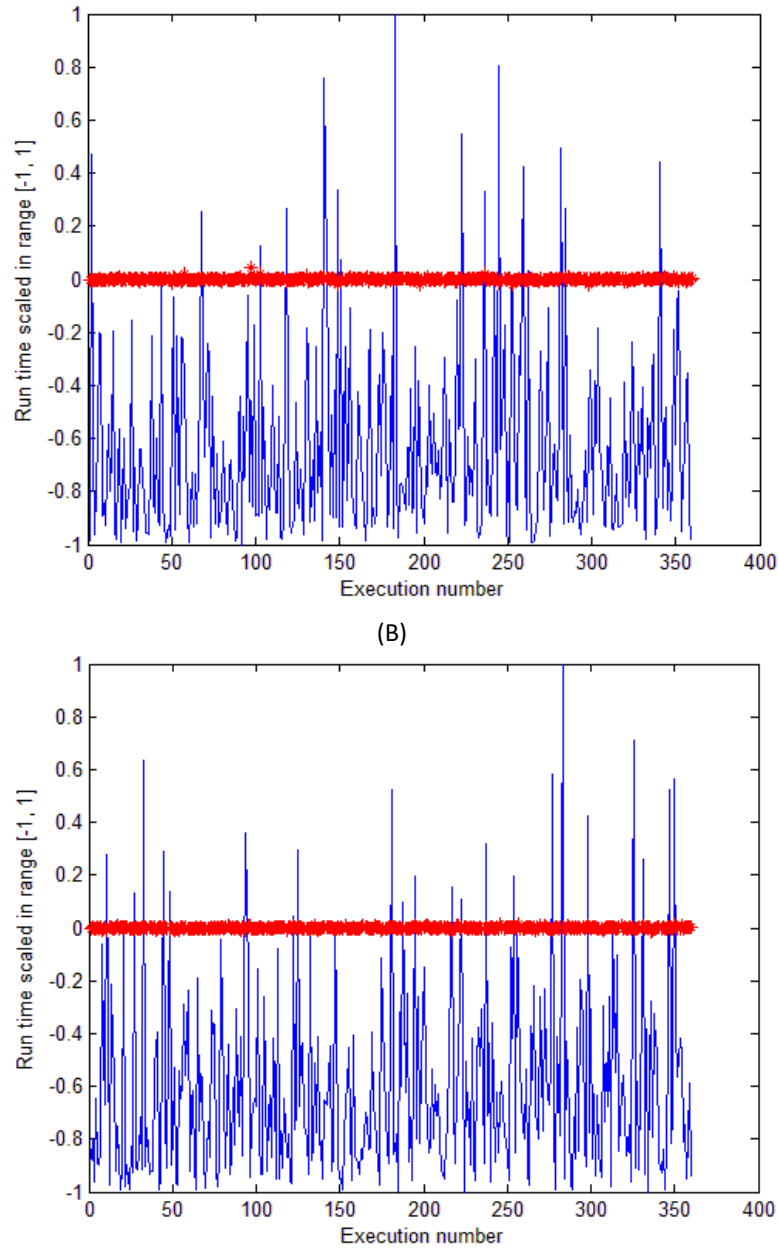


Fig. 15. Run time of ApproxHull and the error obtained by the model on two groups of datasets. (A) First group: uniformly distributed random samples; (B) Second group: normally distributed random samples.

6 Memory Requirement Analysis

As mentioned in Section 2, Quickhull as a standard convex hull algorithm suffers from insufficient memory in high dimensions. In this section, in order to compare empirically ApproxHull with Quickhull in terms of memory requirements, both algorithms were applied to four artificial datasets described in Table XII. All datasets are composed of uniformly distributed random samples.

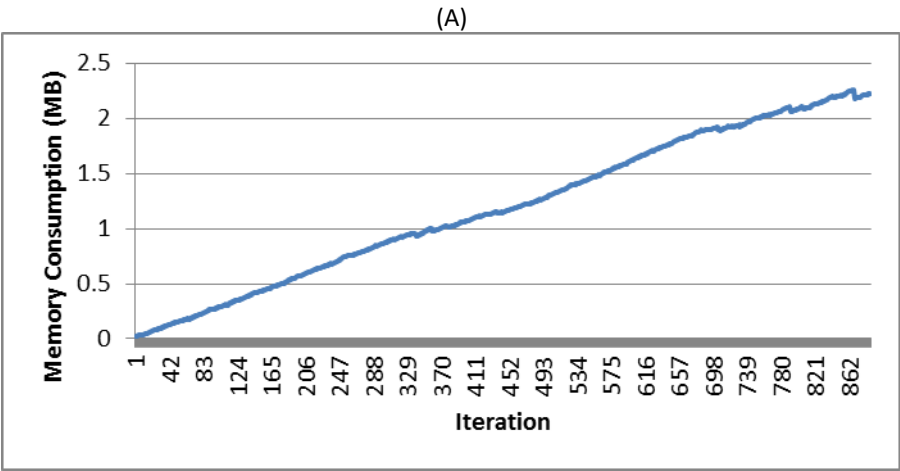
Dataset Name	dim	#S
DS1	5	4000
DS2	6	4000
DS3	7	4000

DS4	8	3500
-----	---	------

TABLE XII - DESCRIPTION OF THE ARTIFICIAL DATASETS CONSISTING OF UNIFORMLY DISTRIBUTED RANDOM SAMPLES. DIM AND #S DENOTE THE NUMBER OF DIMENSION AND SAMPLES RESPECTIVELY.

Since facets in both ApproxHull and Quickhull are the principal objects to which a considerable amount of memory is allocated, this section addresses memory requirements for the generated facets in each iteration for both algorithms. For datasets DS1 to DS3, the sliding window size, w , was set to 10 and for DS4 it was set to 15. The population size, k , was set to 6000, 7000, 8000 and 9000 for datasets DS1, DS2, DS3 and DS4, respectively.

Figs. 16 to 19 show the trend of memory consumption over all iterations in both algorithms on datasets DS1 to DS4. An analysis of these figures shows that memory allocation in Quickhull for all generated facets in each iteration is much larger than that in ApproxHull. As it can be seen in the figures, the trend of memory consumption for all generated facets in ApproxHull is approximately constant in the last iterations. By increasing the dimension, the trend of memory consumption in Quickhull is linearly increasing, translating into a large amount of memory. In Quickhull, in each iteration, only the furthest point to current convex hull is added to the list of vertices and new necessary facets are generated to keep convexity in each iteration. Unlike Quickhull, in each iteration of ApproxHull, a large number of vertices are added into list of vertices of the current convex hull, and that is why the number of iterations of ApproxHull is lower than that of Quickhull. Moreover, in Quickhull, the current convex hull is described in terms of facets and the corresponding vertices so that, in high dimensions, the number of facets which reflect the whole current convex hull is huge. In contrast, the facets in the fixed size population in ApproxHull are only used to detect the furthest point as vertices of real convex hull and they do not describe the whole current convex hull.



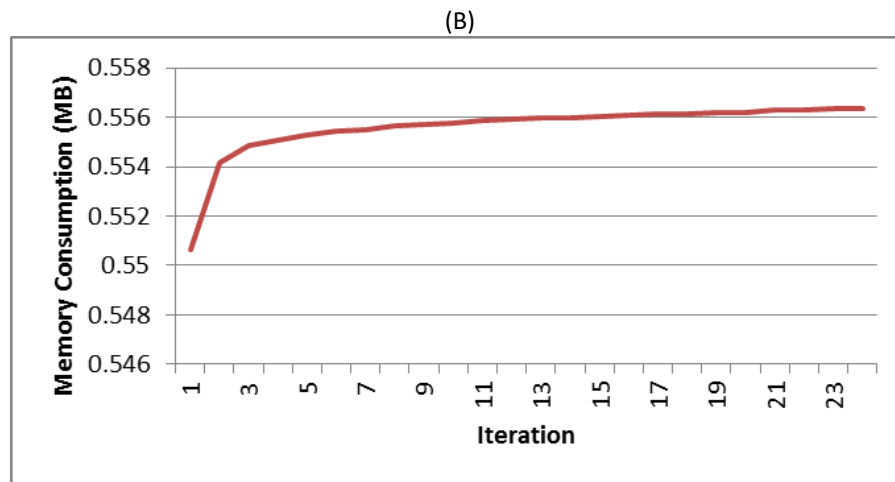


Fig. 16. Trend of memory consumption over iterations on DS1. (A) Quickhull; (B) ApproxHull

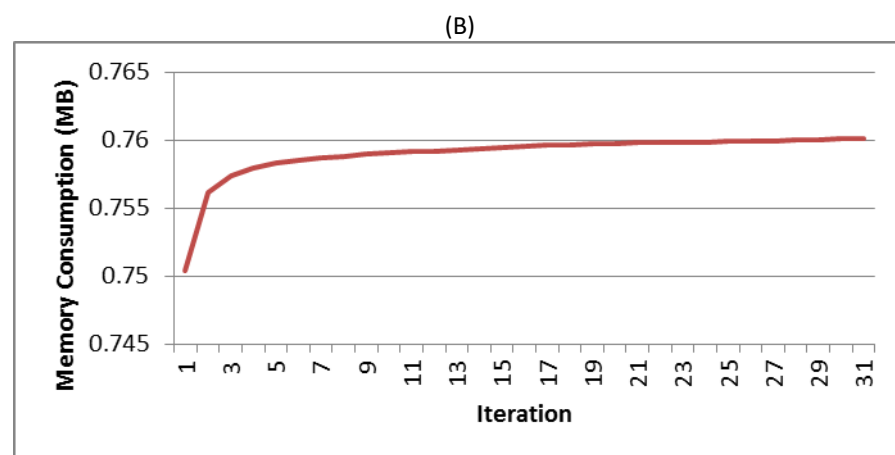
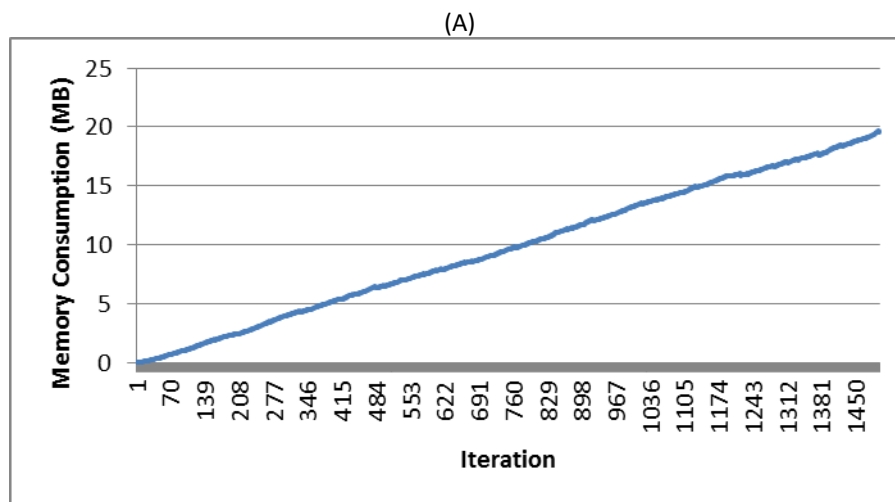


Fig. 17. Trend of memory consumption over iterations on DS2. (A) Quickhull; (B) ApproxHull

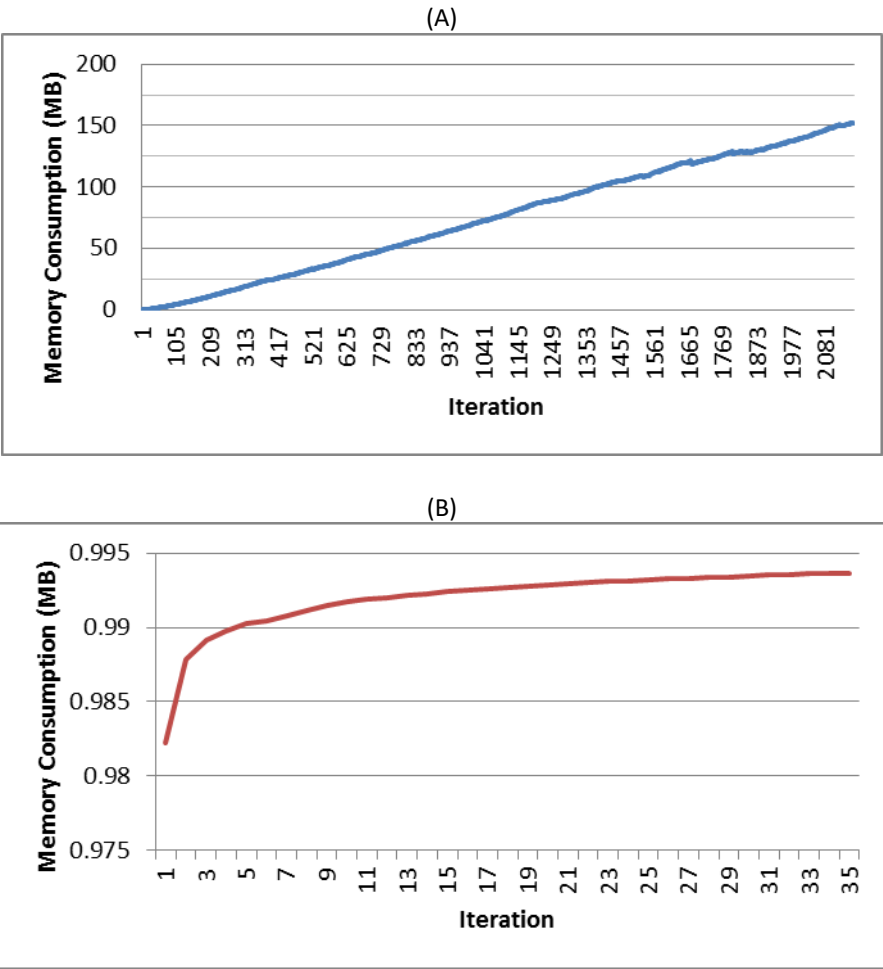
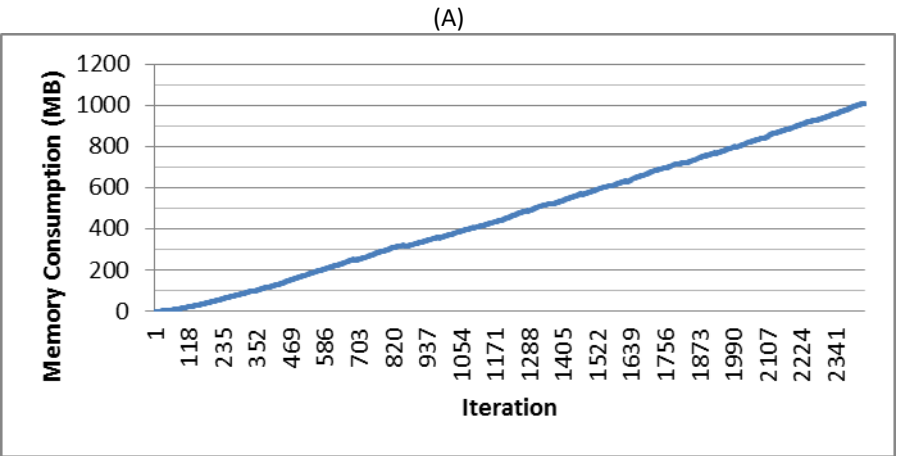


Fig. 18. Trend of memory consumption over iterations on DS3. (A) Quickhull; (B) ApproxHull



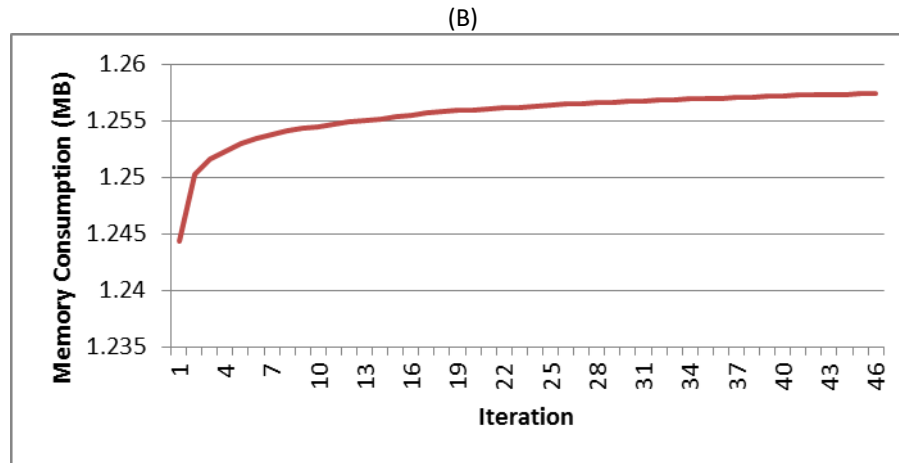


Fig. 19. Trend of memory consumption over iterations on DS4. (A) Quickhull; (B) ApproxHull

According to the explanation above, the amount of memory allocated to the facets for Quickhull is much larger, comparing to ApproxHull. The number of facets and the corresponding amount of memory allocated in both algorithms for the last iteration on datasets DS1 to DS4 are given in Table XIII.

		Quickhull			ApproxHull		
		No. of facets	Total memory size (MB)	Average memory size (B)	No. of facets	Total memory size (MB)	Average memory size (B)
Dataset	DS1	12062	2.222557	193	6000	0.556335	97
	DS2	98801	19.62438	208	7000	0.760101	113
	DS3	712234	152.1882	224	8000	0.993637	130
	DS4	4396390	1006.301	240	9000	1.257355	146

TABLE XIII – NUMBER OF FACETS, TOTAL AND AVERAGE MEMORY SIZE FOR BOTH ALGORITHMS ON DS1 TO DS4 IN THE LAST ITERATION.

7 Conclusions

This paper proposes a novel randomized approximation convex hull algorithm for high-dimensional data, to overcome the limiting memory requirements and time complexity problems found in conventional algorithms. ApproxHull is presented with two policies: stochastic policy and GA-based policy. Simulation results indicate that ApproxHull with Stochastic Policy is faster and its performance is better in comparison to the case where GA-based Policy is applied.

According to the simulation results, ApproxHull can find significantly more vertices of the real convex hull in comparison to Wang's algorithm [27]. Moreover, the obtained results in classification and regression problems show that the use of ApproxHull as a data selection method improves the accuracy of the designed models.

Based on the results obtained from employing ApproxHull with stochastic policy on two groups of datasets, it is revealed that dataset size, population size, number of iterations, number of vertices found as vertices of convex hull and the distribution of samples have influence on the

run time. Based on a mathematical model obtained by using b-spline networks, the approximated time complexity of ApproxHull is $O(n^2 d^3 v^3 + i^3 p^3)$ where n, d, v, i and p denote number of samples, dimension, number of convex hull vertices found, number of iterations and population size, respectively.

From a memory requirements point of view, simulation results reveal that, the memory consumption in ApproxHull is much lower than the Quickhull algorithm, allowing the proposed algorithm to be applied in high-dimensional problems.

Future work will address the use of the proposed approach for online model adaptation purposes, by incorporating ApproxHull in the window management scheme described in [53].

Acknowledgements

The authors would like to acknowledge the support of FCT, through IDMEC, under LAETA, project UID/EMS/50022/2013

REFERENCES

- [1] J. Arturo Olvera-Lopez, J. Ariel Carrasco-Ochoa, J. Francisco Martinez-Trinidad, J. Kittler, A review of instance selection methods, *Artificial Intelligence Review*, 34 (2010) 133-143.
- [2] D.R. Wilson, T.R. Martinez, Reduction techniques for instance-based learning algorithms, *Machine Learning*, 38 (2000) 257-286.
- [3] P. Hart, The condensed nearest neighbor rule (Corresp.), *IEEE Transactions on Information Theory*, 14 (1968) 515-516.
- [4] H. Liu, H. Motoda, *Instance Selection and Construction for Data Mining*, Springer US, 2013.
- [5] G. Ritter, H. Woodruff, S. Lowry, T. Isenhour, An algorithm for a selective nearest neighbor decision rule (Corresp.), *IEEE Transactions on Information Theory*, 21 (1975) 665-669.
- [6] C.-H. Chou, B.-H. Kuo, F. Chang, The generalized condensed nearest neighbor rule as a data reduction method, in: *18th International Conference on Pattern Recognition (ICPR 2006)*, Hong Kong, PEOPLES R CHINA, 2006, pp. 556-559.
- [7] D.L. Wilson, Asymptotic Properties of Nearest Neighbor Rules Using Edited Data, *IEEE Transactions on Systems, Man, and Cybernetics, SMC-2* (1972) 408-421.
- [8] I. Tomek, An Experiment with the Edited Nearest-Neighbor Rule, *IEEE Transactions on Systems, Man, and Cybernetics, SMC-6* (1976) 448-452.
- [9] H. Brighton, C. Mellish, Advances in instance selection for instance-based learning algorithms, *Data Mining and Knowledge Discovery*, 6 (2002) 153-172.
- [10] J.R. Cano, F. Herrera, M. Lozano, Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study, *IEEE Transactions on Evolutionary Computation*, 7 (2003) 561-575.
- [11] L.I. Kuncheva, Editing for the k-nearest neighbors rule by a genetic algorithm, *Pattern Recognition Letters*, 16 (1995) 809-814.
- [12] L.I. Kuncheva, Fitness functions in editing k-NN reference set by genetic algorithms, *Pattern Recognition*, 30 (1997) 1041-1049.
- [13] L.I. Kuncheva, J.C. Bezdek, Nearest prototype classification: Clustering, genetic algorithms, or random search?, *IEEE Transactions on Systems Man and Cybernetics Part C-Applications and Reviews*, 28 (1998) 160-164.

- [14] V. Cerveron, F.J. Ferri, Another move toward the minimum consistent subset: A tabu search approach to the condensed nearest neighbor rule, *Ieee Transactions on Systems Man and Cybernetics Part B-Cybernetics*, 31 (2001) 408-413.
- [15] H.B. Zhang, G.Y. Sun, Optimal reference subset selection for nearest neighbor classification by tabu search, *Pattern Recognition*, 35 (2002) 1481-1490.
- [16] J.A. Olvera-Lopez, J.A. Carrasco-Ochoa, J.F. Martinez-Trinidad, Sequential search for decremental edition, *Intelligent Data Engineering and Automated Learning Ideal 2005*, Proceedings, 3578 (2005) 280-285.
- [17] J.A. Olvera-Lopez, J.F. Martinez-Trinidad, J.A. Carrasco-Ochoa, Restricted sequential floating search applied to object selection, *Machine Learning and Data Mining in Pattern Recognition*, Proceedings, 4571 (2007) 694-702.
- [18] J.C. Riquelme, J.S. Aguilar-Ruiz, M. Toro, Finding representative patterns with ordered projections, *Pattern Recognition*, 36 (2003) 1009-1018.
- [19] T. Raicharoen, C. Lursinsap, A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (POC-NN) algorithm, *Pattern Recognition Letters*, 26 (2005) 1554-1567.
- [20] J.C. Bezdek, L.I. Kuncheva, Nearest prototype classifier designs: An experimental study, *International Journal of Intelligent Systems*, 16 (2001) 1445-1473.
- [21] B. Spillmann, M. Neuhaus, H. Bunke, E. Pekalska, R.P.W. Duin, Transforming strings to vector spaces using prototype selection, *Structural, Syntactic, and Statistical Pattern Recognition*, Proceedings, 4109 (2006) 287-296.
- [22] R.A. Mollineda, F.J. Ferri, E. Vidal, An efficient prototype merging strategy for the condensed 1-NN rule through class-conditional hierarchical clustering, *Pattern Recognition*, 35 (2002) 2771-2782.
- [23] C.J. Veenman, M.J.T. Reinders, The nearest subclass classifier: A compromise between the nearest mean and nearest neighbor classifier, *Ieee Transactions on Pattern Analysis and Machine Intelligence*, 27 (2005) 1417-1429.
- [24] A. Lumini, L. Nanni, A clustering method for automatic biometric template selection, *Pattern Recognition*, 39 (2006) 495-497.
- [25] J.A. Olvera-Lopez, J.F. Martinez-Trinidad, J.A. Carrasco-Ochoa, Mixed data object selection based on clustering and border objects, *Progress in Pattern Recognition, Image Analysis and Applications*, Proceedings, 4756 (2007) 674-683.
- [26] P. Malosek, V. Stopjakova, Pca data preprocessing for neural network-based detection of parametric defects in analog ic, *Proceedings of the 2006 Ieee Workshop on Design and Diagnostics of Electronic Circuits and Systems*, (2006) 131-135.
- [27] D. Wang, H. Qiao, B. Zhang, M. Wang, Online Support Vector Machine Based on Convex Hull Vertices Selection, *Ieee Transactions on Neural Networks and Learning Systems*, 24 (2013) 593-609.
- [28] A. Lopez-Chau, L.L. Garcia, J. Cervantes, X. Li, W. Yu, Data Selection Using Decision Tree for SVM Classification, in: *24th International Conference on Tools with Artificial Intelligence (ICTAI)*, IEEE, 2012, pp. 742-749.
- [29] J. Nalepa, M. Kawulok, A memetic algorithm to select training data for support vector machines, in: *Proceedings of the 2014 conference on Genetic and evolutionary computation*, ACM, Vancouver, BC, Canada, 2014, pp. 573-580.
- [30] A. Lopez Chau, X. Li, W. Yu, Large data sets classification using convex-concave hull and support vector machine, *Soft Computing*, 17 (2013) 793-804.
- [31] V. Bayer, Survey of Algorithms for the Convex Hull Problem, in, *Department of Computer Science*; Oregon State University, 1999.
- [32] C.B. Barber, D.P. Dobkin, H. Huhdanpaa, The Quickhull algorithm for convex hulls, *Acm Transactions on Mathematical Software*, 22 (1996) 469-483.
- [33] R.L. Graham, An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set Inf. Process. Lett., 1 (1972) 2.

- [34] K.L. Clarkson, P.W. Shor, APPLICATIONS OF RANDOM SAMPLING IN COMPUTATIONAL GEOMETRY .2, *Discrete & Computational Geometry*, 4 (1989) 387-421.
- [35] J.L. Bentley, F.P. Preparata, M.G. Faust, Approximation algorithms for convex hulls, *Commun. ACM*, 25 (1982) 64-68.
- [36] H.R. Khosravani, A.E. Ruano, P.M. Ferreira, A simple algorithm for convex hull determination in high dimensions, in: *Intelligent Signal Processing (WISP)*, 2013 IEEE 8th International Symposium on, 2013, pp. 109-114.
- [37] R. Seidel, THE UPPER BOUND THEOREM FOR POLYTOPES - AN EASY PROOF OF ITS ASYMPTOTIC VERSION, *Computational Geometry-Theory and Applications*, 5 (1995) 115-116.
- [38] E.W. Weisstein, Plane, in, *MathWorld--A Wolfram Web Resource*, 2014.
- [39] E.W. Weisstein, Point-Plane Distance., in, *MathWorld--A Wolfram Web Resource*, 2014.
- [40] C.B. Barber, Imprecision in Qhull, in, 1995.
- [41] L.J. Eshelman, Evolutionary Computation 1, in: T. Back, D.B. Fogel, Z. Michalewicz (Eds.) *Genetic algorithms*, Taylor & Francis Group, 2000.
- [42] K. Sastry, D. Goldberg, Search Methodologies, in: E.K.K. Burke, Graham (Ed.) *Introductory Tutorials in Optimization and Decision Support Techniques*, Springer US, 2005.
- [43] A. Frank, A. Asuncion, UCI Machine Learning Repository, in, 2013.
- [44] E. Hajimani, M.G. Ruano, A.E. Ruano, MOGA design for neural networks based system for automatic diagnosis of Cerebral Vascular Accidents, in: *Intelligent Signal Processing (WISP)*, 2015 IEEE 9th International Symposium on, 2015, pp. 1-6.
- [45] M.G. Ruano, E. Hajimani, A.E. Ruano, A Radial Basis Function Classifier for the Automatic Diagnosis of Cerebral Vascular Accidents, in: *2016 Global Medical Engineering Physics Exchanges/Pan American Health Care Exchanges (GMEPE / PAHCE)*, Madrid, Spain, 2016, pp. 188-191.
- [46] J. Demsar, Statistical comparisons of classifiers over multiple data sets, *Journal of Machine Learning Research*, 7 (2006) 1-30.
- [47] F. Wilcoxon, Individual Comparisons by Ranking Methods, in: S. Kotz, N.L. Johnson (Eds.) *Breakthroughs in Statistics: Methodology and Distribution*, Springer-Verlag New York, New York, NY, 1992, pp. 196-202.
- [48] C.E. Rasmussen, R.M. Neal, G.E. Hinton, D.v. Camp, M. Revow, Z. Ghahramani, R. Kustra, R. Tibshirani, Delve datasets, in, 1996.
- [49] A.E. Ruano, P.M. Ferreira, C.M. Fonseca, An Overview of Nonlinear Identification and Control with Neural Networks, in: A.E. Ruano (Ed.) *Intelligent Control Systems using Computational Intelligence Techniques*, Institution of Electrical Engineers, 2005, pp. 37-87.
- [50] T. Kavli, E. Weyer, ASMOD (Adaptive Spline Modelling of Observation Data): some theoretical and experimental results, in: *Advances in Neural Networks for Control and Systems*, IEE Colloquium on, IET, Berlin, 1994.
- [51] A.E. Ruano, C. Cabrita, J.V. Oliveira, L.T. Koczy, Supervised training algorithms for B-Spline neural networks and neuro-fuzzy systems, *Int. J. Syst. Sci.*, 33 (2002) 689-711.
- [52] A.E. Ruano, P.J. Fleming, C. Teixeira, K. Rodriguez-Vazquez, C.M. Fonseca, Nonlinear identification of aircraft gas-turbine dynamics, *Neurocomputing*, 55 (2003) 551-579.
- [53] P.M. Ferreira, A.E. Ruano, Online Sliding-Window Methods for Process Model Adaptation, *IEEE Transactions on Instrumentation and Measurement*, 58 (2009) 3012-3020.