Border Samples Detection for Data Mining Applications Using Non Convex Hulls

Asdrúbal López Chau[1,3], Xiaoou Li[1], Wen Yu,[2],
Jair Cervantes[3], and Pedro Mejía-Álvarez[1]

[1] Computer Science Department, CINVESTAV-IPN, Mexico City, Mexico
achau@computacion.cs.cinvestav.mx, {lixo,pmalavrez}@cs.cinvestav.mx
[2] Automatic Control Department, CINVESTAV-IPN, Mexico City, Mexico
yuw@ctrl.cinvestav.mx
[3] Graduate and Researh, Autonomous University of Mexico State,Texcoco Mexico
chazarra17@gmail.com

Abstract. Border points are those instances located at the outer margin of dense clusters of samples. The detection is important in many areas such as data mining, image processing, robotics, geographic information systems and pattern recognition. In this paper we propose a novel method to detect border samples. The proposed method makes use of a discretization and works on partitions of the set of points. Then the border samples are detected by applying an algorithm similar to the presented in reference [8] on the sides of convex hulls. We apply the novel algorithm on classification task of data mining; experimental results show the effectiveness of our method.

Keywords: Data mining, border samples, convex hull, non-convex hull, support vector machines.

1 Introduction

Geometric notion of *shape* has no associated a formal meaning[1], however intuitively the shape of a set of points should be determined by the borders or boundary samples of the set. The boundary points are very important for several applications such as robotics [2], computer vision [3], data mining and pattern recognition [4]. Topologically, the boundary of a set of points is the closure of it and defines its shape[3]. The boundary does not belong to the interior of the shape.

The computation of border samples that better represent the shape of set of points has been investigated for a long time. One of the first algorithms to compute it is the convex hull (CH). The CH of a set of points is the minimum convex set that contains all points of the set. A problem with CH is that in many cases, it can not represent the shape of a set, i.e., for set of points having interior "corners" or concavities the CH ommits the points that determine the border of those areas. An example of this can be seen in Fig. 1.

I. Batyrshin and G. Sidorov (Eds.): MICAI 2011, Part II, LNAI 7095, pp. 261–272, 2011.

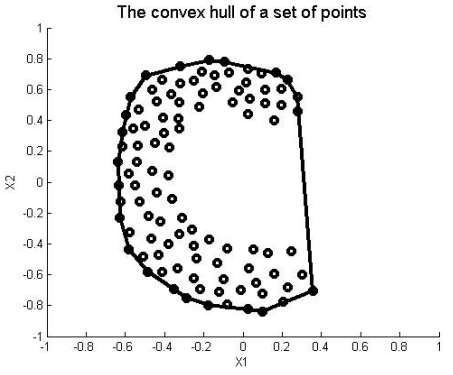

Fig. 1. Convex hull can not represent exactly **the borders** of all sets of points

In order to better characterize the region occupied for a set of points, some proposals have been presented : alpha shapes, conformal alpha shapes, concave hull algorithm and Delaunay-based methods.

In [5] the alpha-shapes as a generalization of the convex hull was presented. Alpha shapes seem to capture the intuitive notions of "fine shape" and "crude shape" of point sets. This algorithm was extended to more than two dimensions in [1]. In [6] is proposed a solution to compute the "footprint" of a set of points. Different from geometric approach was proposed in [7], were the boundary points are recovered based on the observation that they tend to have fewer reverse k-nearest neighbors. An algorithm based on Jarvis march was presented in [8], the algorithm is able to efficiently compute the boundary of a set of points in two dimensions. A problem detected with the algorithm in [8] is that although it can effectively work in almost all scenarios, in some cases it produces a set of elements that does not contain all the samples in a given data set, this is specially notorious if the distribution of samples is not uniform, i.e., if there are "empty" zones, another detail occurs if there are several clusters of points, the algorithm does not compute all border samples.

In this paper we introduce an algorithm to compute border samples. The algorithm is based on the presented in [8] but with the following differences: The algorithm was modified to be able to compute all extreme points, the original algorithm sometimes ignores certain points and the vertexes of convex hull are not included as part of the solution. Instead of using the k nearest neighbors of a point p_i we use the points that are within a hyper-box centered in p_i, this makes the algorithm slightly faster than the original one if the points are previously sorted. The novel algorithm was extended for higher dimensions using a clustering strategy. Finally we use a discretization step and work with groups of adjacent cells from where the border samples are detected.

The rest of the paper is organized as follows. In the section 2 definitions about convexity, convex and non convex hulls are explained. The notion of border samples is also shown. Then in section 3 three useful properties to compute

border samples of a set of points are shown, and proposed algorithms that accomplish the properties are explained. In section 4 the method is applied as a pre-processing step in classification task using Support Vector Machine (SVM) as an application of the algorithms to data mining. The results show the effectiveness of the proposed method. Conclusions and future work are in last part of this paper.

2 Border Points, Convex Hull and Non-convex Hull

The *boundary points (or border points)* of a data set are defined in [7] as those ones that satisfy the following two properties: Given a set of points $P = \{p \in R^n\}$, a $p \in P$ is a border one if

1. It is within a dense region R and
2. $\exists$ region R$'$ near of p such that Density(R $'$) $\gg$ Density(R).

The *convex hull* $\mathcal{CH}$ of a data set X is mathematically defined as in equation (1) and there are several algorithms to compute it [9]: brute force ($\mathcal{O}(n^3)$), Grahamś scan($\mathcal{O}(n \log n)$), divide and conquer $\mathcal{O}(n \log n)$, quick hull (average case $\mathcal{O}(n \log n)$, Jarviś march and Chan's algorithm ($\mathcal{O}(n \log h)$.

$$\mathcal{CH}(X) \left\{ w : w = \sum_{i=1}^{n} a_i x_i, a_i \geq 0, \sum_{i=1}^{n} a_i = 1, x_i \in X \right\} \tag{1}$$

Extreme points are the vertexes of the convex hull at which the interior angle is strictly convex[10]. However as stated before and exemplified in figure 1, $\mathcal{CH}(X)$ can not always capture all border samples of X. Another detail relates to the use of $\mathcal{CH}$ for capturing the border samples occurs when the set of points form several groups or clusters, only extreme borders are computed and outer borders are omitted. For cases like this, the border samples $\mathcal{B}(\cdot)$ usually should define a non-convex set. A convex set is defined as follows[11]: A set S in R^n is said to be convex if for each

$$x_1, x_2 \in S, \alpha x_1 + (1 - \alpha) x_2 \; belongs \; to \; S \tag{2}$$
$$for \; \alpha \in (0, 1).$$

Any set S' that does not hold equation (2) is called a *non-convex*.

We want to compute $\mathcal{B}(X)$ which *envelopes* a set of points, i.e., $\mathcal{B}(X)$ is formed with the borders of X. Because a data set is in general non-convex, we call $B(X)$ *non-convex hull*. The terms non-convex hull and border samples will be used interchangeably in this work.

Although the $\mathcal{CH}(P)$ is unique for each set of points P, the same does not occur with $\mathcal{B}(\mathcal{P})$, there can be more than one valid set of points that define the border for the given P. An example of this is shown in figure 2. The difference of between two border sets $\mathcal{B}(\mathcal{P})$ and $\mathcal{B}(\mathcal{P}')$ is due to the size of each one, which

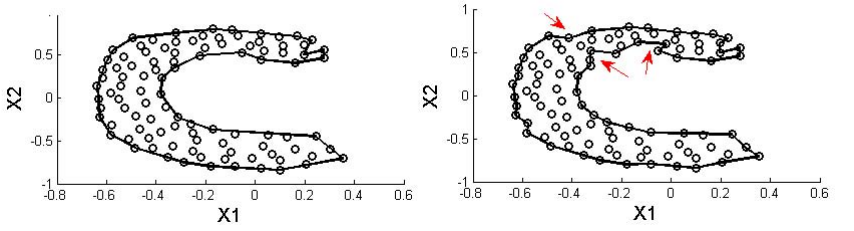

Fig. 2. Two different non-convex hulls for the same set of points. The arrows show some differences.

in turn is related with the degree of detail of the shape. A non-convex hull with a small number of points is faster to compute, but contain less information and vice-verse. This flexibility can be exploited depending on application.

The minimum and maximum size ($|\cdot|$) of a $\mathcal{B}(\mathcal{P})$ for a given set of points P is determined by (3) and (4).

$$min\ |\mathcal{B}(\mathcal{P})| = |\mathcal{CH}(P)|\ \ \forall\ \mathcal{B}(P). \tag{3}$$

$$max|\mathcal{B}(\mathcal{P})| = |P|\ \ \forall\ \mathcal{B}(P). \tag{4}$$

The (3) and (4) are directly derived, the former is from definition of $\mathcal{CH}$, whereas the second happens when $\mathcal{B}(P)$ contains all the points.

Let be P $=\{p \in R^n\}$ and $P^{'}$ a discretization of P by using a grid method. Let be y_i a cell of the grid and let be Y_i a group of adjacent cells with $\bigcap_i Y_i = \varnothing$ and $\bigcup_i Y_i = P^{'}$. The following three properties contribute to detect the border samples of $P^{'}$.

1. $\forall\ \mathcal{B}(P^{'}),\ \mathcal{B}(P^{'}) \supset \mathcal{CH}(P^{'})$.
2. $\bigcup_i \mathcal{B}(Y_i) \supset \mathcal{B}(P^{'})$.
3. *Vertexes of* $\bigcup_i \mathcal{B}(Y_i) \supset$ *vertexes of* $\mathcal{CH}(P^{'})$.

The property 1 obligates that the computed $\mathcal{B}(P^{'})$ contain the vertexes of convex hull of $P^{'}$; it is necessary that all extreme points be include as members of $\mathcal{B}(P^{'})$ in order to explore all space in which points are located, regardless of their distribution.

The property 2 states that border points of $P^{'}$ can be computed on disjoint partitions Y_i of $P^{'}$. The resulting $\mathcal{B}(P^{'})$ contain all border samples of $P^{'}$, this is because border samples are not only searched in exterior borders of the set $P^{'}$, but also within it. The size of $\bigcup_i \mathcal{B}(Y_i)$ is of course greater than the size of $\mathcal{B}(P^{'})$.

Finally the property 3 is similar to the property 1, but here the resulting non-convex hull computed on partitions Y_i of of $P^{'}$ must contain the vertexes of convex hull. If the border samples computed on partitions of $P^{'}$ contain extreme

points, then not only the points in interior corners are detected but also those on the the vertexes of convex hull.

In order the detect border samples and overcome the problems of convex hull approach (interior corners and clusters of points) we propose a strategy based on three properties, if they are accomplished then those points that are not considered in the convex hull but that can be border points (according to definition in past section) can be easily detected.

3 Border Samples Detection

The novel method that we propose is based on the concave hull algorithm presented in [8], with important differences explained in the introduction of this paper, also there are some advantages over [8]: computation of border samples regardless of density distribution of points, extended to more than two dimensions and a easy concurrent implementation is possible. The method consists in three phases: 1) discretization; 2) selection of groups of adjacent boxes; 3) reduction of dimensions and computation of borders.

Firstly, a discretization of a set of points P is done by inserting each $p_i \in P$ into a binary tree $\mathcal{T}$, which represents a grid. The use of a grid helps us to avoid the explicit use of clustering algorithms to get groups of points near among them. This discretization can be seen as the mapping

$$T : P \mapsto P' \tag{5}$$

where $P, P^{'} \in R^n$. Each leaf in $\mathcal{T}$ determine a hyper box b_i, the deeper $\mathcal{T}$ the lesser the volume of b_i. The time required to map all samples in P into the grid is $O(nlog_2(n))$. This mapping is important because it avoids more complicated and computationally expensive operations to artificially create zones of points more equally spaced, also the computation of non-convex hulls requires a set of no repeated points, if two points are repeated in P, then both are mapped to the same hyper box. All this is achieved with the mapping without requiring an additional step $O(|P|)$. During the mapping, the number of points passed trough each node of $\mathcal{T}$ is stored in a integer variable.

The second phase consists in the selection of groups of adjacent boxes in $\mathcal{T}$. The are two main intentions behind this: compute the border of a single cluster of points and control the size of it. We accomplish this two objectives by recursively traversing down $\mathcal{T}$. We stop in a node that contain less than a value of L (predefined by user) in the variable that holds the number of points that have passed through, then we recover the leaves (boxes) below the node. The set of boxes form a partition of $P^{'}$ and are refereed as Y_i. The Algorithm 1 shows the general idea of the method.

For each partition Y_i found, we first reduce the dimension and then compute its border points using algorithm shown in Algorithm 2, which works as follows. First Algorithm 2 computes $\mathcal{CH}(Y_i)$ and then each side of it is explored searching for those points that will form the non-convex hull $\mathcal{B}(P^{'})$ for the partition Y_i. The angle θ in algorithm 2 is computed using the two extreme points of each

Data:
$P \in R^n$: A set of points;
Result:
$\mathcal{B}(P)$: Border samples for P
Map P into P' /* Create a binary tree $\mathcal{T}$ */
Compute partitions Y_i by traversing $\mathcal{T}$ /* Use Algorithm 1 */
Reduce dimension /* Apply Algorithm 4, obtain $cluster_i, i = 1, 2, ...$ */
for *each* $cluster_i$ **do**
  Compute border samples for Y_i within $cluster_i$ /* Algorithm 2 */
  Get back Y_i to original dimension using the centroid of $cluster_i$
  $\mathcal{B}(P') \leftarrow \mathcal{B}(P') \cup$ result of previous step
end
return $\mathcal{B}(\mathcal{P})$

Algorithm 1. Method to compute border samples

side of the convex hull. This part of the method is crucial to compute border samples, because we are searching all points near of each side of convex hull, which are border points. These border points of each side of the convex hull are computed using the algorithm 3.

Data:
Y_i: Partition of P
L: Minimum number of candidates
Result:
$\mathcal{B}(Y_i)$: The border samples for partition Y_i
$\mathcal{CH} \leftarrow \mathcal{CH}(Y_i)$ /* The sides $S = \{S_1, \ldots, S_N\}$ of $\mathcal{CH}$ */
$\theta \leftarrow 0$ /* The initial angle */
$\mathcal{B}(Y_i) \leftarrow \varnothing$
for *each side* $S_i \in S$ *of* $\mathcal{CH}$ **do**
  $\mathcal{BP} \leftarrow$ Compute border points (Y_i, S_i, L, θ)
  $\theta \leftarrow$ get angle $\{s_{i1}, s_{i2}\}$ /* Update the angle */
  $\mathcal{B}(Y_i) \leftarrow \mathcal{B}(Y_i) \cup \mathcal{BP}$
end
return $\mathcal{B}(Y_i)$

Algorithm 2. Detection of border samples for a partition Y_i

The Algorithm 3 shows how each side of $\mathcal{CH}(Y_i)$ is explored. It is similar the presented in [8] which is based on Jarviś march but considering only local candidates, the candidates are those points located inside a box centered at point p_i being analyzed. These local points are computed quickly if Y_i have been previously sorted . The algorithm 3 always include extreme points of Y_i which produces different results from the algorithm in [8]. Also, instead of considering k-nn we use the candidates near to point p_i being analyzed (`currentPoint` in Algorithm 3).

Data:
Y_i: A partition of P
S: Side of a $\mathcal{CH}(Y_i)$
L: (minimum) Number of candidates; θ: Previous angle.
Result:
$\mathcal{BP}$: Border points to side S

```
1  firstPoint ← first element of S
2  stopPoint ← second element of S
3  BP ← {firstPoint}
4  currentPoint ← firstPoint
5  previousAngle ← θ
6  while currentPoint ≠ stopPoint do
7  |  if K > |Y_i| then
8  |  |  L ← |Y_i|
9  |  end
10 |  candidates ← Get Ł elements in the box centered at currentPoint
11 |  Sort candidates by angle considering previousAngle
12 |  currentPoint ← find the first element that do not intersect BP
13 |  if currentPoint is NOT null then
14 |  |  Build a line L with currentPoint and stopPoint
15 |  |  if L intersects BP then
16 |  |  |  BP ← BP ∪ stopPoint
17 |  |  |  return BP
18 |  |  end
19 |  else
20 |  |  BP ← BP ∪ stopPoint
21 |  |  return BP
22 |  end
23 |  BP ← BP ∪ currentPoint
24 |  Remove currentPoint from X
25 |  previousAngle ← angle between last two elements of BP
26 end
27 return BP
```

Algorithm 3. Computation of border points for S_i

For higher than two dimensions we create partitions on them to temporally reduce the dimension of the $P \in R^n$ in several steps. For each dimension we create one dimensional clusters, the number of cluster corresponds to the partitions of the dimension being reduced, then we fixed the value of that partition to be the center of the corresponding cluster. This process is repeated on each dimension. The final bi-dimensional subsets used are formed by considering in decreasing order with respect to the number of partitions of each dimension. We compute border samples and then get them back to their original dimension taking the previously fixed values.

In order to quickly compute the clusters on each feature of data set, we use a similar algorithm to that presented in [12]. The basic idea of the on-line dimensional clustering as follows: if the distance from a sample to the center of a group is less than a previously defined distance L, then the sample belongs to

this group. When new data are obtained, the center and the group should also change. The Euclidean distance at time k is defined by eq. (6)

$$d_{k,x} = \left(\sum_{i=1}^{n} \left[\frac{x_i(k) - \overline{x}^j}{x_{imax} - x_{imin}} \right]^2 \right)^{\frac{1}{2}} \tag{6}$$

Where n is the dimension of sample x, $\overline{x}^j$ is the center of the j^{th} cluster, $x_{imax} = max_k \{x_i(k)\}$ and $x_{imin} = min_k \{x_i(k)\}$.

The center of each cluster can be recursively computed using (7):

$$\overline{x}^j_{i\ k+1} = \frac{k-1}{k} \overline{x}^j_{i\ k} + \frac{1}{k} x_i(k) \tag{7}$$

The Algorithm 4 shows how to compute the partition of one dimension of a given training data set. This algorithm 4 is applied in each dimension, and produces results in linear time with the size of the training data set.

Data:
$\mathcal{X}_m$: The values of the m^{th} feature of training data set $\mathcal{X}$
Result:
$\mathcal{C}_j$: A number of one dimensional clusters(partitions) of m^{th} feature of $\mathcal{X}$ and its corresponding centers.

```
C_1 = x(1)                    /* First cluster is the first arrived sample. */
x̄^1 = x(1)
for each received data x(k) do
    Use eq. (6) to compute distance d_{k,x} from x(k) to cluster C_j
    if d_{k,x} ≤ L then
        x(k) is kept in cluster j
        Update center using eq. (7).
    else
        x(k) belongs to a new cluster C_{j+1}, i.e., C_{j+1} = x(k)
        x̄^{j+1} = x(k)
    end
end
        /* If the distance between two groups centers is more than the
   required distance L */
if Σ_{i=1}^{n} [x̄^p − x̄_i^q]^2 ≤ L then
    The two clusters (C_p and C_q) are combined into one group, the center of the
    new group may be any of the two centers.
end
return SV_1 ⋃ SV_2
```

Algorithm 4. Feature partition algorithm

4 Application on a Data Mining Task

In order to show the effectiveness of the proposed method, we apply the developed algorithms on several data sets and then train a SVM using the detected border points.

All experiments were run on a computer with the following features: Core 2 Duo 1.66 GHz processor, 2.5 GB RAM, linux Fedora 15 operating system installed. The algorithms were implemented in the Java language. The maximum amount of random access memory given to the java virtual machine was set to 1.6 GB for each one of the runs.

For all data sets the training data set was built by randomly choosing the 70% of the whole data set read from disk, the rest of samples were used as testing data set.

The data sets are stored as plain text files in the attribute relation file format. The time used to read the data sets from hard disk was not taken into account for the reported results of all the experiments, as usual in literature, i.e., the measurements were taken from the time when a data set was loaded into memory to the time when the model has been calibrated, i.e., the reported times correspond to the computation of border samples and the training of SVM. The reported results are the average of 10 runs of each experiment.

In order to compare the performance of the proposed algorithm two SVMs are trained using LibSVM library. The first SVM is trained with the entire data set whereas the second SVM is trained using only the border samples recovered using the proposed method. In both cases the corresponding training times and achieved accuracy are measured and compared. The kernel used in all experiments is a radial basis function.

Experiment 1. In this experiment, we use a data set similar to the checkerboard one [13]. Table 1 shows a resume of the data set. The difference with the original is that the data set used in the experiment contains 50000 samples grouped in a similar distribution as shown in figure 4. The squares can overlap in no more than 10%. Note that the number of samples have kept small to clarify the view.

Table 1. Data set Checkerboard2 used in experiment 1

Data set	Features	Size ($y_i = +1/y_i = -1$)
Checkerboard2	2	25000 (12500/12500)

The Checkerboard2 is a linearly inseparable data set. The RBF kernel was used with the parameter $\gamma = 0.055$. Table 2 shows the results of the Experiment1. Column T_{br} in the table refers to the **time for the computation of border samples**, whereas T_{tr} is **the training time**, both are in milliseconds. The column *Time* **is the time elapsed from the load of data set in memory to the time when training of SVM is done**, also it is measured in milliseconds. The column #SV is the number of support vectors and #BS is the number of border samples recovered by the proposed algorithm. The first row of results shows the results using border samples detected with the proposed algorithm whereas the second one is for LibSVM using the entire data set.

Table 2. Results for Checkerboard2 like data set (25000 samples)

T_{bs}	T_{tr}	Time	#SV	#BS	Acc	Training data set
1618	4669	**6287**	2336	2924	**89.9**	*Only Border Samples*
		27943	4931		**90.3**	Whole data set

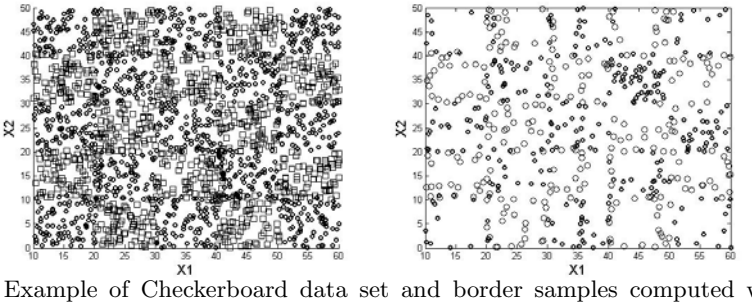

Fig. 3. Example of Checkerboard data set and border samples computed with the proposed algorithm

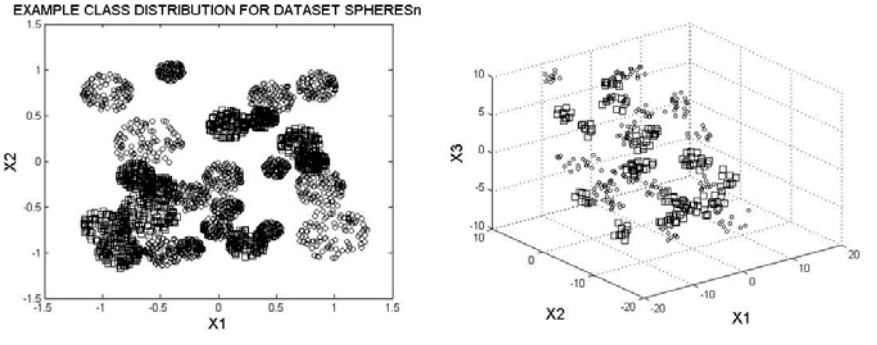

Fig. 4. Example of the class Distribution for data set **Spheres***2* and **Spheres***3*. In higher dimensions a similar random distribution occurs. Circle: $y_i = +1$, Square: $y_i = -1$.

Fig. 4 can be appreciated the border samples detected from data set Checkerboard. The method successfully compute border samples and produces a reduced version of Checkerboard, containing only border points. This samples are used to train SVM, which accelerated the training time as can be seen in table 2.

Experiment 2. In the second experiment, we use a data set of size up to 250000 samples and the number of dimensions is increased up to 4.. The data set is synthetic, composed of dense hyper dimensional balls with random radius and centres. The synthetic data set Spheres$n$ consists in a a number of hyper spheres whose center is randomly located in a n dimensional space. Each sphere has a radius of random length and contains samples having the same label. The hyper spheres can overlap in no more than 10% of the greater radius. Fig. 4

shows an example of data set Spheres$n$ for n=2 and n=3. Again the number of samples have kept small to clarify the view. Similar behaviour occurs in higher dimensions. In the Table 3 can be seen the number of samples and the dimension of data set used in the experiment 2.

Table 3. Data set Spheres$n$ used in experiment 2

Data set	Features	Size ($y_i = +1/y_i = -1$)
Spheres2	2	50000 (16000/34000)
Spheres4	4	200000 (96000/104000)

The training and testing data sets were built by randomly choosing 70% and 30% respectively from the whole data set. For all runs in experiment 2, the parameter $\gamma = 0.07$.

Table 4. Results for Spheres*2* data set (50000 samples)

T_{br}	T_{tr}	Time	#SV	#BS	Acc	Training data set
2635	2887	**5522**	626	2924	**98.4**	*Only Border Samples*
		69009	1495		**98.6**	Whole data set

Table 5. Results for Spheres*4* data set ((200000 samples))

T_{br}	T_{tr}	Time	#SV	#BS	Acc	Training data set
6719	2001	**8720**	627	4632	**98.3**	*Only Border Samples*
		53643	1173		**99.8**	whole data set

Results show that accuracy of the classifier trained using only border samples is slightly degraded but the training times of SVM are reduced considerably. Which agree with the fact that border samples were successfully recognized from training data set.

5 Conclusions

We proposed a method to compute the border samples of a set of points in a multidimensional space. The results of experiments show that the effectiveness of the method on classification task using SVM, the algorithms can quickly obtain border samples that are used to train SVM yielding similar accuracy to the obtained using the whole data set but with the advantage of consuming considerably less time. We are currently working on an incremental version of the algorithm to compute border samples.

References

1. Edelsbrunner, H., Mücke, E.P.: Three-dimensional alpha shapes. ACM Trans. Graph. 13(1), 43–72 (1994)
2. Bader, M.A., Sablatnig, M., Simo, R., Benet, J., Novak, G., Blanes, G.: Embedded real-time ball detection unit for the yabiro biped robot. In: 2006 International Workshop on Intelligent Solutions in Embedded Systems (June 2006)
3. Zhang, J., Kasturi, R.: Weighted boundary points for shape analysis. In: 2010 20th International Conference on Pattern Recognition (ICPR), pp. 1598–1601 (August 2010)
4. Hoogs, A., Collins, R.: Object boundary detection in images using a semantic ontology. In: Conference on Computer Vision and Pattern Recognition Workshop, CVPRW 2006, p. 111 (June 2006)
5. Edelsbrunner, H., Kirkpatrick, D., Seidel, R.: On the shape of a set of points in the plane. IEEE Transactions on Information Theory 29(4), 551–559 (1983)
6. Galton, A., Duckham, M.: What is the Region Occupied by a Set of Points? In: Raubal, M., Miller, H.J., Frank, A.U., Goodchild, M.F. (eds.) GIScience 2006. LNCS, vol. 4197, pp. 81–98. Springer, Heidelberg (2006)
7. Xia, C., Hsu, W., Lee, M., Ooi, B.: Border: efficient computation of boundary points. IEEE Transactions on Knowledge and Data Engineering 18(3), 289–303 (2006)
8. Moreira, J.C.A., Santos, M.Y.: Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points. In: GRAPP (GM/R), pp. 61–68 (2007), http://dblp.uni-trier.de
9. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer, Heidelberg (2008)
10. O'Rourke, J.: Computational Geometry in C. Cambridge University Press (1998), hardback ISBN: 0521640105; Paperback: ISBN 0521649765, http://maven.smith.edu/~orourke/books/compgeom.html
11. Noble, B., Daniel, J.W.: Applied Linear Algebra, 3rd edn. (1988)
12. Yu, W., Li, X.: On-line fuzzy modeling via clustering and support vector machines. Information Sciences 178, 4264–4279 (2008)
13. Ho, T., Kleinberg, E.: Checkerboard data set (1996), http://www.cs.wisc.edu/math-prog/mpml.html