Counter-Example Generation-Based One-Class Classification

András Bánhalmi, András Kocsor, and Róbert Busa-Fekete

Research Group on Artificial Intelligence of the Hungarian Academy of Sciences
and of the University of Szeged,
H-6720 Szeged, Aradi vértanúk tere 1., Hungary
{banhalmi,kocsor,busarobi}@inf.u-szeged.hu

Abstract. For One-Class Classification problems several methods have been proposed in the literature. These methods all have the common feature that the decision boundary is learnt by just using a set of the positive examples. Here we propose a method that extends the training set with a counter-example set, which is generated directly using the set of positive examples. Using the extended training set, a binary classifier (here ν -SVM) is applied to separate the positive and the negative points. The results of this novel technique are compared with those of One-Class SVM and the Gaussian Mixture Model on several One-Class Classification tasks.

1 Introduction

In the field of machine learning there are several problems to which usual multi-class classification methods cannot be applied or do not perform that well. Among these problems an important one is the so-called "One-Class Classification" problem [1], where only positive examples are available for a particular class, or the negative examples are highly under-represented. These problems are also referred to as Data Description, Outlier Detection, and Novelty Detection in different fields. What the methods proposed for this task have in common is that since only positive examples are available from a particular class during the training phase, the decision boundary is learnt not between two or more classes, but it is a set of closed surfaces which surround the majority of the positive training instances. The area of one-class training includes several algorithms like generative probability density estimation methods (Gaussian Mixture Model (GMM), Parzen estimator), reconstruction methods (k-means, Autoencoder Neural Networks), and boundary estimators (k-centers, SVDD [2] [3] [4], NNDD [1]).

We propose here a new method for multidimensional real-valued one-class classification tasks. Firstly, our method selects the boundary points of the positive data. Next, these boundary points are used to generate counter-examples that surround the positives; and finally, a binary classifier (here ν -Support Vector Machine (SVM) [5]) is used to separate the positive and negative data samples. Overall, the goal of the proposed method is to force the binary classifier to learn a "multidimensional contour" at a distance from the positive examples.

2 Generation of Counter-Examples

In order to train the ν -SVM binary classifier by taking just the positive examples of a class, we propose a method for generating negative examples using the positive ones. Essentially, this method first finds the local boundary points of the input data. Then all the positive examples are transformed using the closest boundary point for each. The points obtained after these transformations will form the set of negative examples. This method is described below in more detail.

After, when we have negative examples surrounding the positives, a ν -SVM is trained to separate these two classes. During the testing phase the position of the decision boundary between the positive and negative sets can be controlled by setting a threshold for the acceptance.

2.1 Finding Boundary Points

Our goal is to find those points of the n -dimensional positive examples for which a hyperplane exists that separates this point from the other positive examples in its neighborhood. Here we suggest a method for solving this problem (see the pseudo-code in the Table 1).

Table 1. The Boundary Point Search Method

Input:	A set of N positive examples (X)
Output:	A set of K boundary points (B), and inner points (I)
0	$B = \emptyset$
1	For each x in X do
2	Take x_i , the k closest points (but with a positive distance) to x .
3	Compute the unit vectors: $e_i = \frac{x_i - x}{\ x_i - x\ }$
4	Try to separate e_i points from the origin using a hard margin linear SVM
5	If the optimization of SVM fails
	$I = I \cup \{x\}$
	else
	$B = B \cup \{x\}$, and form the vector: $x_{center} = \sum_{i=1}^k \alpha_i e_i$
	(this vector is needed for counter-example generation)

The explanation of this algorithm is the following. In the 2nd and 3rd rows the k closest points to x are determined (x_i), and unit vectors are computed from $x_i - x$ vectors. Then in the 4th row a linear SVM is applied to separate these normalized vectors from the origin. Next, if the separation process fails, then x is added to the set of inner points. If the points are separable, then x is added to the set of boundary points and an x_{center} vector is computed, which will be needed later for counter-example generation.

Here the center vector will be defined as follows. Let x_b a boundary point, and let x_i the k nearest neighbors of x_b ($i = 1 \dots k$). Let e_i be defined by:

$e_i = \frac{x_i - x_b}{\|x_i - x_b\|}$. Then the *center vector* corresponding to x_b will be defined as the normal vector of the hyperplane separating the origin from the points e_i with maximal margin.

A hard-margin linear SVM is used to find the hyperplane separation with maximal margin, and the normal vector of this hyperplane is computed using the support vectors and the corresponding α_i values computed by the SVM. For more information and for a proof see [6].

2.2 Generating Counter-Examples

The method suggested here is based on the transformation of the positive examples using the boundary points chosen earlier and the corresponding center vectors. For more details about the algorithm, see Table 2 with the following explanation.

Table 2. The base method used to generate counter-examples

Input:	A set of N positive examples (X)
Output:	A set of M negative examples (Y)
0	$Y = \emptyset$, $B = \text{boundaryPoints}(X)$
1	For each x in X do
2	Find x_b , the closest boundary point (but with a positive distance) to x
3	Transform x to x_b using the following formula: $y = v(1 + T(x, x_b, X) / \ v\) + x$, where $v = x_b - x$, T is a function of the X dataset, x , x_b
4	Check y to see if it is an inner point with the algorithm described in Table 1
5	If y is not an inner point, then $Y = Y \cup \{y\}$ else $B = B \setminus \{x_b\}$, and with the next closest boundary point repeat the procedure.

In the 2nd row the closest boundary point x_b to x is selected. The transformation of the point x will be a translation along the $v = x_b - x$ direction, and the distance of the transformed point y from the boundary point x_b depends on the $T(x, x_b, X)$ function. In the 4th and 5th rows the algorithm checks to see if y is an inner point, and if it is, then we do not take it as a negative example, and the boundary point will be deleted from the set. For the T function we suggest the following formula:

$$T(x, x_b, X) = \frac{\text{dist}}{\text{dist} \cdot \text{curv} + \text{CosAngle}(x, x_b, X)}, \quad (1)$$

where

$$\text{CosAngle}(x, x_b, X) = \frac{x_{b, \text{center}}^T \cdot (x - x_b)}{\|x_{b, \text{center}}^T\| \|x - x_b\|}, \quad (2)$$

Fig. 1. These figures show the generated boundary points with different settings. Left to right: 1st: $(dist, curv) = (1, 0)$, 2nd: $(dist, curv) = (0.4, 0)$, 3rd: $(dist, curv) = (1, 0.5)$, 4th: $(dist, curv) = (1, 1)$. One can see that $curv = 0$ will generate points of a hyperplane, while $curv > 0$ will generate points of a better fitting boundary hyper-surface. With the $dist$ parameter the distance between the boundary points and the generated counter-examples can be controlled.

and $x_{b,center}$ is the center vector for the x_b obtained and stored by the boundary point selection method (see Table 1). The constant parameter $dist$ controls the distance between the transformed point and the boundary point, and the $curv$ parameter controls the curvature of the hyper-surface of the counter-examples obtained by the transformation for the same boundary point. Figure 1 provides some examples with different $dist$ and $curv$ parameter values. The method generates N counter examples for a positive dataset of N data samples.

3 Refinements and Variants

The previously described boundary point selection algorithm is based on a local convexity condition. This method can also work on distinct or concave sets, but the condition may exclude many points from the boundary point set. To increase the number of the boundary points, we suggest some refinements.

The first modification we propose to apply can be used to increase the number of boundary points and also to filter extreme distal points. The idea is that we could also add some inner points to the set of the boundary points that are close to the local surface. The modified method iteratively separates new boundary points from the $X \setminus B$ dataset, where B is the set of the boundary points found in the previous iterations. The effect of this method on the dataset is shown in Figure 2.

The second modification of the base method is proposed for the counter-example generation algorithm. The boundary points do not necessarily have the same "popularity". In some cases there are a lot of boundary points which are

Fig. 2. The first picture shows the result of using the base method. The second picture is obtained with the proposed modification after 3 iterations.

Fig. 3. The left picture shows the generated negative examples without applying the proposed modification, while the right picture shows the result when it is applied

rarely chosen for the point-transformation, and this can cause an imbalance in the density of the counter-examples. This problem can be handled by adding extra counter-examples to the original ones in the following way. First, the number of transformations have to be counted for all the boundary points. Then extra boundary points should be added using the following rule: *for each rarely used boundary point, select their k closest points, and after applying the transformation method add the new points to the set of counter-examples*. To define which boundary points the rule can be applied to, a frequency threshold has to be used. The effect of this modification can be seen in Figure 3 above.

4 Time Complexity and Applicability

The time complexity of counter-example generation depends on k and N , where k is the number of nearest neighbors used in the boundary point test, and N is the total number of positive training examples. The boundary point test method uses a linear SVM, so the time complexity is $o(k^3)$. To choose the k -nearest neighbors, $o(N \cdot \log(N))$ time is needed, so altogether for N points the total time complexity is $o(k^3 \cdot N^2 \cdot \log N)$. When generating counter-example for a specified point, we need the closest boundary point, and a test has to be done to see if the transformed point is an inner point. The combined time complexity of this part after summing for each training example is $o(|B| \cdot n \cdot k^3)$, where $|B|$ is the number of boundary points.

The time complexity of the training method depends on which method is chosen for this purpose. Here ν -SVM was applied, which has a time complexity of $o(N^3)$. However, some other methods which have a slightly lower time complexity could be applied for the classification task like the Core Vector Machine [7].

We suggest applying the method proposed here on those problems where the attributes are continuous real-valued variables. Some preprocessing should also be made to eliminate redundant or useless attributes. Since a hyperplane in an n -dimensional space is determined by at least n points (this is important when determining center vectors), we suggest that it also should be guaranteed that the size of the training data is higher than the dimension of the problem. However among our test cases there is an example which does not satisfy this condition, but the proposed method still works well.

5 Experiments and Results

In order to separate the examples from the artificially generated counter-examples ν -SVM with an RBF kernel function was applied using the WEKA system [8] [9]. For the boundary point tests the Matlab SVM Toolbox by Steve Gunn was applied ¹. For a comparison we used GMM (from the GMMBayes Matlab Toolbox [10]) and One-Class SVM (from WEKA) as baselines. All the models were trained using different free parameter settings, and the results with the best scores were taken. The free parameters are the number of clusters and confidence value for the GMM, the ν and γ values for the one-class SVM and the ν -SVM, the number of clusters, the (*curv*, *dist*) parameters, the number of nearest neighbors that need to be considered (k) and the acceptance threshold for the counter-example generation-based method. The ν -SVM with n clusters is a construction where the positive training examples are grouped into n clusters using k-means. After that, the counter examples are generated, and a separate ν -SVM is trained for each cluster (the free parameters were set to the same value). In the decision phase the score of an example is the maximum value of the scores given by the SVM models corresponding to the clusters.

For testing purposes 10 one-class datasets were employed ². The "unvoiced DB", and "silence DB" were constructed by us and contain feature vectors of 25 Mel Filter Bank coefficients created from human speech samples. In the "Unvoiced" one-class problem the task was to train the examples of unvoiced phonemes. The "Silence" one-class database contained feature vectors of silent parts (and also contained the short silences in a plosive). The "Ball-Bearing", "Water Pump" train and test datasets (used in fault detection) were downloaded from [11]. That is, from the Ball-Bearing datasets the "rectangular window" version (containing 26-dimensional vectors), and from the "Water Pump" datasets the "autocorrelation function" version (containing 26-dimensional vectors) were utilized. In [12] these datasets are described in detail, and were used to find

¹ <http://www.isis.ecs.soton.ac.uk/resources/svminfo/>

² All the datasets can be downloaded from:

<http://www.inf.u-szeged.hu/oasis/oneclass/>

Table 3. Best test results using different models in one-class tasks: classification accuracies, and rates of true positives and false positives

Accuracy/TP/ FP rates (%)	train DB size and dimension	one-class SVM	GMM	ν -SVM & counter examples
Silence DB ⁽¹⁾	301/12	82.4/89.7/24.8	85.8 /82.5/36.8	85.6/ 83.8/12.4
Unvoiced DB ⁽²⁾	405/12	78.6/86.3/29.0	78.1/82.9/26.6	82.9 / 80.0/14.4
Ball B. DB ⁽³⁾	56/26	99.3/96.9/ 0.0	96.3/81.3/ 0.0	100.0 /100.0/0.0
Water P. DB ⁽⁴⁾	224/26	94.0/88.5/ 4.4	94.9/87.5/ 4.4	95.7 / 94.8/ 4.1
OC 507 DB ⁽⁵⁾	163/13	67.2/78.8/44.4	60.3/26.9/ 6.3	67.5 / 77.5/42.5
OC 511 DB ⁽⁶⁾	126/5	85.2/88.8/18.4	78.8/80.0/22.4	86.8 / 92.8/19.2
OC 514 DB ⁽⁷⁾	236/278	70.9/82.6/40.9	67.7/53.2/17.9	71.9 / 68.3/34.4
OC 589 DB ⁽⁸⁾	125/34	52.4/ 4.8/ 0.0	52.4/ 4.8/ 0.0	71.6 / 62.4/19.2
OC 598 DB ⁽⁹⁾	300/21	77.8 /76.0/20.3	50.5/ 1.0/ 0.0	77.7/ 86.3/31.0
OC 620 DB ⁽¹⁰⁾	4490/10	86.3 /89.6/17.0	84.9/93.1/23.2	84.6/ 91.1/22.1

the optimal parameter settings for the One-class SVM, and to choose the best preprocessing method. The other 6 datasets (OC xxx) were downloaded from the webpage given in [13]. More information about these datasets can be found there. These datasets did not contain separate train and test sets, so train sets and test sets were generated using a 5-fold cross-validation technique. In the experiments each test dataset contained an equal number of positive and negative examples. Moreover, when preprocessing the positive datasets a normalization and a PCA transformation was applied.

Table 3 below lists the best results of the different classification algorithms applied to one-class tasks³.

6 Conclusions and Future Work

Based on the results of our experiments we can say that the proposed method seems superior to the statistical GMM, and the One-Class SVM in most cases.

³ The settings of the free parameters of the models are the following. Here cn denotes the number of clusters (the default value is 1), $conf$ represents the confidence threshold for GMM (the default value is 1), ν and γ are the parameters of ν -SVM and One-Class SVM, $(dist, curv, k)$ are the parameters for counter-example generation – see Equation 1 – and the default values are $(1, 0.5, 2 \cdot dim)$. The acceptance threshold is represented by ath , whose default value is 0.5. Here G means that the second modification (see Section 3) was applied.

⁽¹⁾ $(\nu, \gamma) = (0.1, 0.01)$, $(conf, cn) = (1, 2)$, $(\nu, \gamma, cn, ath) = (0.1, 0.01, 2, 0.9)$; ⁽²⁾
 $(\nu, \gamma) = (0.3, 0.01)$, $(conf, cn) = (1, 2)$, $(\nu, \gamma, cn, ath) = (0.1, 0.3, 2, 0.9)$ G ; ⁽³⁾
 $(\nu, \gamma) = (0.01, 0.01)$, $(conf, cn) = (0.95, 1)$, $(\nu, \gamma, ath, k) = (0.3, 0.01, 0.3, 56)$;
⁽⁴⁾ $(\nu, \gamma) = (0.05, 0.1)$, $(conf, cn) = (0.95, 1)$, $(\nu, \gamma, dist, curv, ath) =$
 $(0.3, 1, 1.5, 0.1, 0.9)$; ⁽⁵⁾ $(\nu, \gamma) = (0.1, 0.05)$, $(\nu, \gamma, ath) = (0.3, 0.3, 0.1)$ G ; ⁽⁶⁾
 $(\nu, \gamma) = (0.1, 0.5)$, $(\nu, \gamma) = (0.3, 0.3)$; ⁽⁷⁾ $(\nu, \gamma) = (0.1, 0.005)$, $(\nu, \gamma, ath) =$
 $(0.3, 0.01, 0.3)$; ⁽⁸⁾ $(\nu, \gamma) = (0.001, 0.5)$, $(\nu, \gamma) = (0.3, 0.3)$; ⁽⁹⁾ $(\nu, \gamma) = (0.1, 0.3)$,
 $(\nu, \gamma, ath) = (0.3, 0.3, 0.6)$; ⁽¹⁰⁾ $(\nu, \gamma) = (0.1, 0.3)$, $(\nu, \gamma, ath) = (0.3, 0.3, 0.8)$.

However, to make it work better in practice some improvements should be made to reduce the time complexity of our algorithm. Currently we are working on a faster iterative boundary-searching method that begins with a small subset of the whole, then modifies the boundary points by iteratively adding only suitable points to them from the database. In the near future we will implement a faster boundary point search method, and other distance-based classifiers will be tested to separate the positive and the negative examples.

Acknowledgement

We would like to thank György Szarvas, Richárd Farkas and László Tóth for their helpful advice and constructive comments, and also thank David Curley for checking this paper from a linguistic point of view.

References

1. Tax, D.: One-class classification; Concept-learning in the absence of counter-examples. PhD thesis, Delft University of Technology (2001)
2. Schölkopf, B., Platt, J.C., Shawe-Taylor, J., Smola, A.J., Williamson, R.C.: Estimating the support of a high-dimensional distribution. *Neural Computation* 13, 1443–1471 (2001)
3. Tax, D.M.J., Duin, R.P.W.: Support vector domain description. *Pattern Recogn. Lett.* 20, 1191–1199 (1999)
4. Tax, D.M.J., Duin, R.P.W.: Support vector data description. *Mach. Learn.* 54, 45–66 (2004)
5. Chen, P.H., Lin, C.J., Schölkopf, B.: A tutorial on ν -support vector machines: Research articles. *Appl. Stoch. Model. Bus. Ind.* 21, 111–136 (2005)
6. Vapnik, V.N.: *Statistical Learning Theory*. John Wiley and Son, Chichester (1998)
7. Tsang, I.W., Kwok, J.T., Cheung, P.M.: Core vector machines: Fast svm training on very large data sets. *J. Mach. Learn. Res.* 6, 363–392 (2005)
8. EL-Manzalawy, Y., Honavar, V.: WLSVM: Integrating LibSVM into Weka Environment (2005), Software available at <http://www.cs.iastate.edu/~yasser/wlsvm>
9. Witten, I.H., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edn. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers Inc., San Francisco (2005)
10. Paalanen, P.: Bayesian classification using Gaussian mixture model and EM estimation: Implementations and comparisons. Technical report, Department of Information Technology, Lappeenranta University of Technology, Lappeenranta (2004)
11. Unnthorsson, R.: Datasets for model selection in one-class ν -svms using rbf kernels, <http://www.hi.is/~runson/svm/>
12. Unnthorsson, R., Runarsson, T.P., Jonsson, M.T.: Model selection in one-class ν -svms using rbf kernels. In: COMADEM – Proceedings of the 16th international congress (2003)
13. Tax, D.M.: Occ benchmark, <http://www-it.et.tudelft.nl/~davidt/occ/>