ONE-CLASS CLASSIFICATION METHODS VIA AUTOMATIC COUNTER-EXAMPLE GENERATION

András Bánhalmi
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
email: banhalmi@inf.u-szeged.hu

ABSTRACT
Here we propose novel methods for the One-Class Classification task and examine their applicability. Essentially, these methods extend the training set – which contains only positive examples – with artificially generated counter-examples. After, a two-class classifier is used to separate them. In this paper following a description of the existing and the newly proposed methods some problematic issues are investigated theoretically and studied empirically by applying these methods to artificial datasets. Then their efficiency is compared to those of other one-class classification methods.

KEY WORDS
One-class classification, artificial counter-example generation.

1 Introduction

One-Class Classification problems [16] - which we shall focus on in this article - are also referred to as Data Description, Outlier Detection, and Novelty Detection in different fields. What these algorithms have in common is that only positive examples are available from a particular class during the training phase. Thus the classification boundary is defined not between several classes, or between positive and negative examples of a class, 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, Support Vector Data Description, Nearest Neighbor Data Description). Here GMM [3] and the One-Class Support Vector Machine (SVM) [14] [17] will be used for testing purposes in order to compare the performance of our method with a fast generative model and a well-known boundary estimator. In practice, one-class models have been applied with success to numerous problems. Without claiming to be complete, one-class classifiers have been used in the fault detection of machinery [15], in automated currency validation [9], in bioacoustic monitoring [13], and in document classification [10].

In [2] a new methodology is proposed which is different from the current methods in one fundamental way: a binary classifier (ν-SVM) is used for the one-class classification task. However, to achieve this aim, examples of two classes are needed. In [2] a new method was introduced which generates counter-examples around the positives. In other words the goal was to force the binary classifier to learn a "multidimensional contour" at some distance from the positive examples. In this paper after describing the basic methods introduced in [2] we propose a new, approximative boundary point search method. After, we try to apply Radial Basis Neural Network and ν-SVM to separate the examples from the counter-examples.

2 One-Class Classification via Counter-Example Generation

Here the objective is to generate negative examples at a certain distance from the region of the positives [2]. In this section we describe a method which achieves this by finding local boundary points of the set, and then transforms the positive examples into negative examples on the other side of the boundary.

2.1 Determining Local Boundary Points and Center Vectors

Suppose that we have N points in an n-dimensional space, and let us define the neighborhood of a point as the set of its k-nearest neighbors. Let us define "local boundary point" as a point for which a hyperplane exists which hyperplane separates this point from the other positive examples in its neighborhood.

In the following, the "center vector" corresponding to a boundary point will be defined. Let x_b be a boundary point, and let x_i be the k nearest neighbors of x_b ($i = 1 \dots k$). Let e_i be the translated and normalized vectors 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 that separates the origin from the points e_i by a maximal margin.

Input: A set of N positive examples (X)
Output: A set of K boundary points (B), and a set of inner points (I)
Init.: $B = \emptyset$ 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: $x_{center} = \sum_{i=1}^{k} \alpha_i e_i$

Table 1. The SVM-based boundary point search method

Input: A set of N positive examples (X)
Output: A set of K boundary points (B), and a set of inner points (I)
Init: $B = \emptyset$ 1. For each x in X do 2. Take the K set containing the k closest points to x 3. Compute the minimal and maximal values of the components of all the vectors in K: $(x_{\min})_j = \min_{x_i \in K}((x_i - x)/\|x_i - x\|)_j$ $(x_{\max})_j = \max_{x_i \in K}((x_i - x)/\|x_i - x\|)_j$ 4. Form the vector: $x_{center} = x_{\min} + x_{\max}$ 5. For all the vectors in K compute the following values: $\cos(\varphi_i) = \frac{(x_i - x)^T \cdot x_{center}}{\|x_i - x\| \cdot \|x_{center}\|}$ 6. If all the $\cos(\varphi_i)$ values are nonnegative, then $B = B \cup \{x\}$, else $I = I \cup \{x\}$

Table 2. The min/max-based boundary point search method

2.1.1 Exact Solution

The application of the algorithm below was first presented in [2]. The linear Support Vector Machine (SVM) finds the separating hyperplane with a maximal margin, when it exists. Moreover, the normal vector of this hyperplane can be found using the support vectors and the corresponding α_i values computed by SVM. For more information and for a proof see [20]. Making use of this fact, the pseudo-code in the Table 1 shows the corresponding boundary point search algorithm.

2.1.2 Approximated Solution

The idea behind the following algorithm we propose here is that the components of a bisectrix vector (c) between two n-dimensional unit vectors (x, y) can be computed as:

$$c_i = \min(x_i, y_i) + \max(x_i, y_i) \tag{1}$$

Although in this special case a bisectrix vector can be calculated in other ways too (eg. computing the average), when using the min/max formulation the result is influenced by just the extreme-valued components when more than two data samples are available. So let as define the approximated center vector x_c for a boundary point x_b and for the k-nearest neighbors x_i ($i = 1 \ldots k$) of x_b by the following formula:

$$(x_c)_i = \min_{j=1\ldots k}((x_j)_i) + \max_{j=1\ldots k}((x_j)_i) \tag{2}$$

It is evident that the vector computed using the previous min/max formulation differs from the exact center vector. In Section 4.3 we will examine the probability density of the differences by applying both methods to artificially generated datasets.

Using the previously defined center vector, the pseudo-code for the boundary point selection method is shown in Table 2. Since the condition for being a boundary point – that the angles between the vectors to the k-nearest neighbors and the approximated center vector have to be acute angles – is only a sufficient condition, the boundary points found with this method will be a subset of all of them. In Section 4.3 the number of boundary points found will also be examined.

2.2 Generating Counter-Examples

Here the goal is to automatically produce negative examples using just the positive ones [2]. Our approach is to transform the positive examples outside their region. For this purpose the closest boundary point and the corresponding center-vector are utilized. Essentially, each point is transformed somehow along the direction of its nearest boundary point. For the details of our proposed algorithm, see Table 3 with the following explanation.

In the second 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 $T(x, x_b, X)$ functional. In the forth and fifth 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 functional we suggest the following formula:

$$T(x, x_b, X) = \frac{dist}{dist \cdot curv + CosAngle(x, x_b, X)}, \tag{3}$$

Input: A set of N positive examples (X)
Output: A set of N negative examples (Y)
Init.: $Y = \emptyset$, $B = boundaryPoints(X)$ 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 functional of X, x, x_b 4. Check y to see if it is an inner point using the algorithm given in Table 1 or in Table 2 5. If y is not an inner point, then $Y = Y \cup \{y\}$ else $B = B \backslash \{x_b\}$, and with the next closest boundary point repeat the procedure.

Table 3. The method used to generate counter-examples

where

$$CosAngle(x, x_b, X) = \frac{x_{b,center}^T \cdot (x - x_b)}{\left\| x_{b,center}^T \right\| \|x - x_b\|}, \quad (4)$$

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 transformations on 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.

One refinement of the above-described method can be made to achieve a better performance. It springs from the recognition of the fact that the boundary points do not necessarily have the same "popularity". In some cases there are several boundary points which are 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 should 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.

2.3 Training on the Extended Dataset

Here two classical methods are used to separate the positive and negative examples: the ν- Support Vector Machine (ν-SVM) with RBF kernel [4], and the Radial Basis Neural Network (RBN) [6]. The ν-SVM is the WEKA [7] (libsvm) implementation. The RBN comes from the MATLAB Neural Network Toolbox, of which the first layer has radial-bases neurons, while the output layer contains a neuron with a linear activation function. After training, the distance between the decision boundary and the set of the positive examples can be adjusted by setting an acceptance threshold.

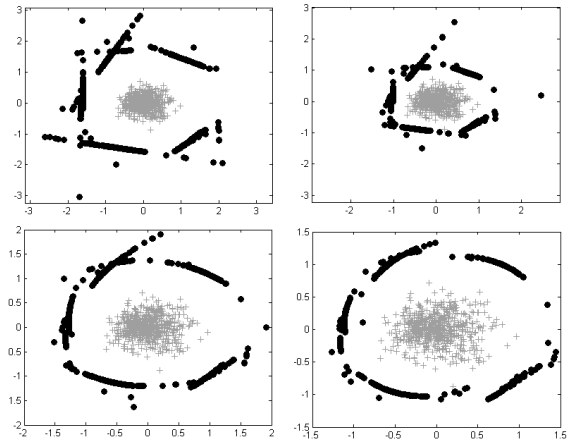

Figure 1. These figures show the generated counter-examples with different settings. Left to right: 1st: $(dist, curv) = (1, 0)$, 2nd: $(dist, curv) = (0.4, 0)$, 3rd: $(dist, curv) = (1, 0.5)$, 4th: $(dist, curv) = (1, 1)$. One can see that $curv = 0$ will generate points of a hyper-plane, while $curv > 0$ 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 adjusted.

Our experiments showed quite clearly that only distance-based classification methods can be used to successfully separate a region in a multidimensional space from its surroundings. That is why we used the above-mentioned classifiers in our experiments. We also plan to test some other Neural Network constructions with a distance-based activation function [6] in the near future.

3 Time Complexity

The time complexity of the counter-example generation – when the dimension is fixed – depends on k and N, where k is the number of nearest neighbors used during the boundary point tests, and N is the total number of positive training examples. The exact boundary point test method uses a linear SVM, so the time complexity is $o(k^3)$. The time complexity of the approximated boundary point test method is $o(k)$. To choose the k-nearest neighbors, a time of $o(N \cdot log(N))$ is needed, thus altogether for N points the total time complexity is $o(k^3 \cdot N^2 \cdot logN)$ or $o(k \cdot N^2 \cdot logN)$. When generating a 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 $o(|B| \cdot n \cdot k^3)$, or $o(|B| \cdot n \cdot k)$, where $|B|$ is the number of boundary points.

4 Investigation of the proposed method

In this part we examine how the results of the proposed methods depend on parameters like the dimension of the space, the number of nearest neighbors, and the number of positive examples. After, the results (the number of boundary points, and the direction of center vectors) of the min/max boundary search method are compared to those of the exact SVM-based method from an approximation point of view. For this purpose artificially generated multidimensional datasets with Gaussian distribution will be used. However, before describing this shall first discuss the curses and blessings of dimensionality.

4.1 Curses and Blessings of Dimensionality

The title of this subsection comes from [5], and tells us that the nature of a multidimensional space is quite strange. The phrase "curse of dimensionality" means that to describe a probability distribution with the "same quality" (or with the same density of the samples) we need data with size that grows exponentially with the number of the dimensions. There are some other important and interesting phenomena associated with high dimensional spaces which are of interest here: the theater effect, the boundary effect, and the measure concentration. For a clear and concise description, see [11]. The "theater effect" denomination refers to the phenomena that if we have a point set with a reasonable number of points distributed normally in a higher dimensional space, then the distances between a point and some other (randomly selected) fixed number of points will be almost equal with a high probability. Moreover, the χ-distribution tells us that the expected value of the distance between a point and the center increases with increasing dimension n, so the points will probably lie farther and farther from the center as n increases. This might be one reason for the "boundary effect" phenomena which can be expressed in our case in the following way: if we have a number of data samples N in an n-dimensional space ($N > n + 1$), then the probability that one of these points is not a boundary point exponentially decreases with the dimension. So a typical training dataset will contain boundary points in a very high ratio. In addition, almost all the *test* points will also be boundary points of the training set, thus the great majority of the test set will not be in the convex hull of the training samples. Approaching the one-class classification task from this point of view, we can say that a good one-class classifier has to learn a decision boundary at a *considerable distance* from the region of the positive examples. Despite this, the "boundary effect" can be useful in some cases – for example for k nearest neighbor searching methods [1], and for us here as well.

4.2 Number of Nearest Neighbors Needed

The above-mentioned "boundary effect" implies that if the dataset contains vectors with a proportionate number of necessary points of exponentially huge size, then all the nearest neighbors have a high probability of being boundary points. In this case as few as possible nearest neighbors are needed to detect boundary points, and the corresponding inner vectors. To include these in our problem: to define a hyperplane in an n-dimensional space, a number of n points are needed (but a hyperplane with a maximal margin exists between two points, respectively). In our experiments we chose $k = 2 \cdot n$ for the boundary point determination. If the size of the database becomes commensurate with the exponentially huge number, then it is still a question of how we should choose the optimal value of k for the k-NN in the boundary point search. We think that k should be chosen as an exponential function of the density of the data, hence k can still be much smaller than the size of the dataset.

4.3 SVM- vs. Min/Max-Based Boundary Point Search

Up until now we have not found a theoretical explanation for why the min/max method can work well. Here some statistical experiments were done to see the differences between the results of the exact and the min/max-approximated boundary point search methods. For this, a comparative study is made on the number of boundary points found, and the directions of the corresponding inner vectors we found using artificially generated normally distributed datasets with different dimensions (2 - 15) and different sizes (100 - 500,1000).

Figure 3 shows the ratio of the sizes of the two boundary point sets determined approximately and exactly, while Figure 2 shows the ratio of the number of the (exact) boundary points, and the total number of positive examples. Figure 4 shows the mean values of the angles between the two kinds of center vectors. The figures tell us that when the "boundary effect" occurs, the number of boundary points detected by the two methods becomes almost equal. On the other hand, the mean of the angles between the two kinds of center vectors does not grow with the dimension n or with the size of the database. Actually, this value is about 0.1 in radians, and is quite a small angle.

5 Experiments and Results

The binary classifiers utilized here for the one-class classification tasks were ν-SVM (WEKA version [7]) and RBN (MATLAB version). For the boundary point tests the Matlab SVM Toolbox by Steve Gunn was used [8]. Both classifiers were applied on two training sets which were created using the exact and the approximated boundary point search method. These test cases will be denoted by "SVM-E", "SVM-A", "RBN-E", and "RBN-A". Here for the sake

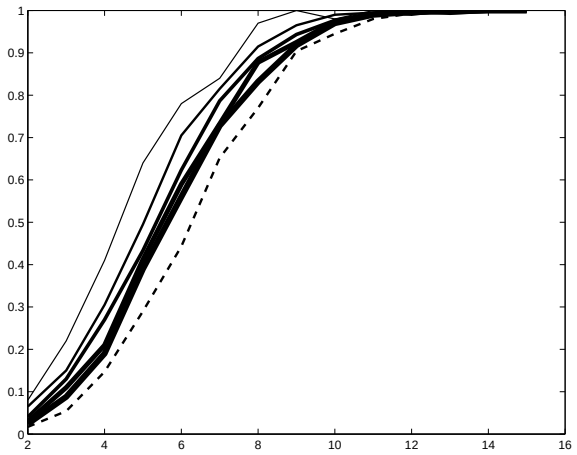

Figure 2. The ratio of the number of the exact boundary points plotted against the total number of examples (y-axis). The width of each solid line indicates the size of the dataset ($100, 200, \ldots, 500$). The dashed line refers to a size of 1000. The x-axis is the dimension of the data vectors. Notice that above a dimension value of 10 the boundary effect occurs, and all the data points become boundary points.

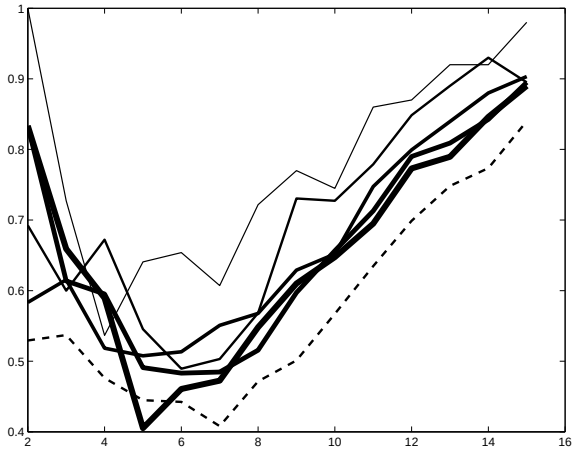

Figure 3. The ratio of the number of the "approximated" boundary points plotted against the number of exact boundary points (y-axis). The x-axis and the with of the lines are the same as before.

of comparison we applied GMM, which is a very fast probabilistic method (we chose the GMMBayes Matlab Toolbox [12]) and the widely used One-Class SVM (or "OC-SVM" for short, from WEKA) as baselines.

All the models were trained using a number of different free parameter settings, and the best accuracy scores are listed in Table 4. The free parameters are the number of clusters and confidence value for GMM, the ν and γ values for the one-class SVM and ν-SVM, the $(curv, dist)$ parameters, the number of nearest neighbors that need to

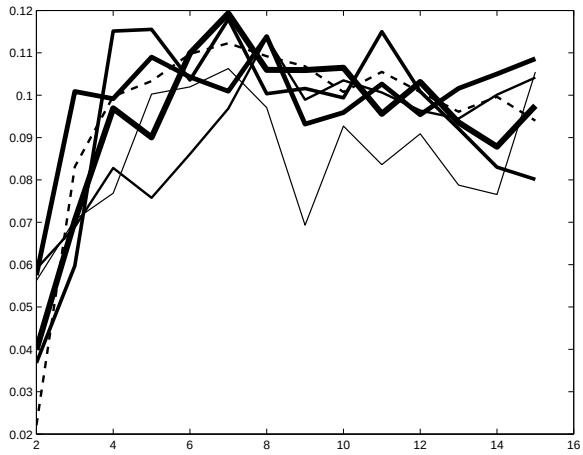

Figure 4. The mean values of the angles between the different center vectors in radians (y-axis). The x-axis and the with of the lines are the same as before.

be considered (k) and the acceptance threshold for the counter-example generation-based methods. In the case of SVM-E and SVM-A the number of clusters can also be set.

For testing purposes 10 one-class datasets were employed. 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 (which also contained the short silences in a plosive). The "Ball-Bearing", "Water Pump" train and test datasets (used in faulty detection) were downloaded from [18]. That is, out of the Ball-Bearing datasets the "rectangular window" version (containing 26-dimensional vectors), and out of the "Water Pump" datasets the "autocorrelation function" version (containing 26-dimensional vectors) were utilized. In [19] these datasets are described in detail, and were used to find the optimal parameter values for the One-class SVM, and to choose the best preprocessing method. The other 6 datasets (OC xxx) were downloaded from a webpage [1]. More information about these datasets can be found there [2]. These datasets did not contain separate train and test sets, so training sets and test sets were generated using a 5-fold cross-validation technique. In our experiments each test dataset contained an equal number of positive and negative examples. To preprocess the positive datasets, a PCA transformation was applied.

Table 4 below lists the best results of the different classification algorithms applied to one-class tasks.

[1] David M.J. Tax: OCC benchmark. `http://www-it.et.tudelft.nl/~davidt/occ/`

[2] The training set in the case of OC-620 contained only the every third element of the original set, because thew RBN implementation did not support more.

DB: train size/dim.	OC-SVM	GMM	SVM-E	SVM-A	RBN-E	RBN-A
Silence: 301/12	82.4/89.7/24.8	**85.8**/82.5/36.8	85.6/83.8/12.4	85.2/86.6/16.1	84.7/85.9/16.5	84.1/86.6/18.3
Unvoiced: 405/12	78.6/86.3/29.0	78.1/82.9/26.6	**82.9**/80.0/14.4	79.6/82.0/22.7	72.4/86.1/41.3	74.4/85.0/35.6
Ball B.: 56/26	99.3/96.9/0.0	96.3/81.3/0.0	**100**/100/0.0	100/100/0.0	100/100/0.0	100/100/0.0
Water P.: 224/26	94.0/88.5/4.4	94.9/87.5/4.4	**95.7**/94.8/4.1	95.7/94.8/4.1	95.4/96.9/9.4	95.7/96.8/8.3
OC507: 163/13	67.2/78.8/44.4	60.3/26.9/6.3	67.5/77.5/42.5	67.5/78.1/43.1	68.4/76.9/40.0	**70.3**/76.8/36.3
OC511: 126/5	85.2/88.8/18.4	78.8/80.0/22.4	**87.2**/93.6/19.2	87.2/94.4/8.0	87.2/82.4/8.0	87.2/83.2/8.8
OC514: 236/278	70.9/82.6/40.9	67.7/53.2/17.9	**71.9**/68.3/34.4	70.0/78.7/39.6	50.0/50.0/50.0	50.0/50.0/50.0
OC589: 125/34	52.4/4.8/0.0	52.4/4.8/0.0	**71.6**/62.4/19.2	56.0/84.8/72.8	52.4/4.8/0.0	52.4/4.8/0.0
OC598: 300/21	**77.8**/76.0/20.3	50.5/1.0/0.0	77.7/86.3/31.0	76.5/81.0/28.0	77.0/78.0/24.0	76.7/64.0/10.7
OC620: 4490/10	87.2/89.4/15.1	89.2/82.1/3.7	85.4/85.8/14.9	87.2/87.9/13.5	88.8/85.5/7.9	**89.2**/83.4/5.0

Table 4. Best test results (accuracy/rates of true positives/rates of false positives) using different models in one-class tasks.

6 Conclusions

Based on the results obtained in the study we think that the counter-example generation-based methods are superior to the statistical GMM, and the One-Class SVM in most cases. Thus the novel methodology for one-class classification which is based on counter-example generation is competitive with the conventional methods which are trained using just the positive examples. However, for better applicability in practice some improvements should be made in the near future to reduce the time complexity of our algorithm.

References

[1] S. Arya, D. M. Mount, and O. Narayan. Accounting for boundary effects in nearest neighbor searching. In *Symposium on Computational Geometry*, pages 336–344, 1995.

[2] A. Bánhalmi, A. Kocsor, and R. Busa-Fekete. Counter-example generation-based one-class classification. In *proc. ECML 2007 LNAI 4701*, pages 543–550, 2007.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] P.-H. Chen, C.-J. Lin, and B. Schölkopf. A tutorial on ν-support vector machines: Research articles. *Appl. Stoch. Model. Bus. Ind.*, 21(2):111–136, 2005.

[5] D. L. Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. 2000.

[6] W. Duch and N. Jankowski. Survey of neural transfer functions, 1999.

[7] Y. EL-Manzalawy and V. Honavar. *WLSVM: Integrating LibSVM into Weka Environment*, 2005. Software available at `http://www.cs.iastate.edu/~yasser/wlsvm`.

[8] I. R. Group. Matlab support vector machine toolbox, `http://www.isis.ecs.soton.ac.uk/resources/svminfo/`.

[9] C. He, M. Girolami, and G. Ross. Employing optimised combinations of one-class classifiers for automated currency validation. *Pattern Recognition*, 37:1085–1096, 2004.

[10] L. M. Manevitz and M. Yousef. One-class SVMs for document classification. *Journal of Machine Learning Research*, 2:139–154, 2001.

[11] J. Marcel and J. jr. Marcel. How does the space where e-golem walks looks like. In *Interdisciplinary Aspects of Human-Machine Co-existence and Cooperation, Prague, CTU 2005*, pages 271–279, 2005.

[12] P. Paalanen. Bayesian classification using Gaussian mixture model and EM estimation: Implementations and comparisons. Technical report, Department of Information Technology, Lappeenranta University of Technology, Lappeenranta, 2004.

[13] A. Sachs, C. Thiel, and F. Schwenker. One-class support-vector machines for the classification of bioacoustic time series. *ICGST International Journal on Artificial Intelligence and Machine Learning (AIML)*, 6(4):29–34, 2006.

[14] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[15] H. J. Shin, D.-H. Eom, and S.-S. Kim. One-class support vector machines: an application in machine fault detection and classification. *Comput. Ind. Eng.*, 48(2):395–408, March 2005.

[16] D. Tax. *One-class classification; Concept-learning in the absence of counter-examples*. PhD thesis, Delft University of Technology, 2001.

[17] D. M. J. Tax and R. P. W. Duin. Support vector domain description. *Pattern Recogn. Lett.*, 20(11-13):1191–1199, 1999.

[18] R. Unnthorsson. Datasets for model selection in one-class ν-svms using rbf kernels, `http://www.hi.is/~runson/svm/`.

[19] R. Unnthorsson, T. P. Runarsson, and M. T. Jonsson. Model selection in one-class ν-svms using rbf kernels. In *COMADEM Proceedings of the 16th international congress*, 2003.

[20] V. N. Vapnik. *Statistical Learning Theory*. John Wiley and Son, 1998.