Incremental One-Class Learning with Bounded Computational Complexity

Rowland R. Sillito and Robert B. Fisher

School of Informatics, University of Edinburgh, UK

Abstract An incremental one-class learning algorithm is proposed for the purpose of outlier detection. Outliers are identified by estimating - and thresholding - the probability distribution of the training data. In the early stages of training a non-parametric estimate of the training data distribution is obtained using kernel density estimation. Once the number of training examples reaches the maximum computationally feasible limit for kernel density estimation, we treat the kernel density estimate as a maximally-complex Gaussian mixture model, and keep the model complexity constant by merging a pair of components for each new kernel added. This method is shown to outperform a current state-of-the-art incremental one-class learning algorithm (Incremental SVDD [5]) on a variety of datasets, while requiring only an upper limit on model complexity to be specified.

1 Introduction

The problem of one-class learning (also known interchangeably as "outlier / novelty / anomaly detection") arises in a wide variety of different application domains. The fundamental goal of one-class learning is to generate a rule that distinguishes between examples of a known class of items and examples from previously-unseen novel classes, on the exclusive basis of training examples from the known class.

This problem presents itself in cases where one wishes to distinguish between members of a class for which examples are abundantly available, and members of another rarely observed class. This often arises when attempting to detect abnormal activity, eg. jet engine failure, computer network intrusions, disease symptoms, etc. In each of these domains, anomalous examples may be scarce or entirely absent during training, but their subsequent identification is of crucial importance. A wide variety of different methods have been proposed to address this problem (see [3] for a review). However, almost all existing one-class classification algorithms require all training examples to be available at once, for a single "batch" learning step: if a new example is presented, the classifier must be retrained from scratch.

Since outliers might only be identifiable by their deviation from a normal model, a key problem in one class learning is the choice of model complexity. In some cases training data may be well described by the parameters of a single Gaussian distribution, while in other cases - eg. where the data has multiple

modes or lies on a non-linear manifold - a more complex model is required. It is important to select the correct level of model complexity: if it is too low, the learned normal model may also include the anomalies that we wish to detect; if it is too high, the model may not include the majority of normal examples.

In many cases it would be useful to be able to incrementally train a classifier as data became available, without needing to pre-specify the level of model complexity. In this paper we propose a new technique for performing incremental one-class learning, where only an upper limit on model complexity needs to be specified. While computationally feasible, our algorithm attempts to estimate the underlying p.d.f. (probability density function) of the training data using non-parametric kernel density estimation (with Gaussian kernels), thereby generating a maximally complex one-component-per-example Gaussian mixture model. Once a maximum number of mixture components has been reached, it is kept constant by merging a pair of components for every new component added. We choose pairs of components for merging based on an information theoretic merging-cost function originally proposed by Goldberger and Roweis in [2].

Currently, at least two related techniques exist in the literature. In [9] Yamanishi et al. propose an unsupervised outlier detection procedure "SmartSifter" in which a Gaussian mixture model is trained using an on-line adaptation of the EM (Expectation Maximization) algorithm. A key aspect of their adaptation is the inclusion of "discounting" parameters which ensure that the effect of older training examples on the model parameters is rapidly displaced by new examples. Each new training example is given a score based on the extent to which it changes the model parameters: a comparatively high score, indicating a large change in model parameters, indicates the possibility of an outlier. This algorithm seems inappropriate for comparison with the proposed algorithm as it models a finite window of training data preceding each new example, rather than attempting to incrementally build a complete normal class description.

A more closely related algorithm has been proposed in [5], where Tax and Laskov present an incremental training procedure for the SVDD (Support Vector Data Description) algorithm originally proposed by Tax and Duin in [7]. The SVDD algorithm attempts to find the smallest hypersphere that encloses the training data, and allows more complex hyper-volumes (which may fit the data better) to be obtained by introducing kernel functions which map the training data to a higher dimensional space [7]. In both batch and incremental forms, the SVDD algorithm relies crucially on the correct choice of model complexity parameters. Various methods have been proposed to address this issue in the absence of example outliers, including: procedures for generating synthetic outliers (Tax and Duin [4]) and, more recently, a consistency based approach which takes the simplest possible classifier and increases its complexity parameter until the proportion of correctly recognized training data starts to fall (Tax and Muller [8]). The incremental variant of SVDD does not include any on-line model complexity selection, but it is conceivable that the batch optimization methods could be applied to a pre-existing dataset to optimally parametrize the classifier before using it for on-line training.

We provide a detailed description of the proposed algorithm in Section 2, and then illustrate its performance on a variety of datasets in Section 3, where we also compare its performance with the incremental SVDD algorithm [5] (optimized using the consistency criterion proposed in [8]). Our algorithm is shown to yield equivalent, often better, performance than the incremental SVDD algorithm without requiring any time-consuming parameter optimization.

2 Algorithm

The proposed algorithm is designed to receive a sequence of labeled multivariate training data, and to determine - at any stage in the training process - whether or not new data are outliers. This is achieved by estimating the underlying probability density function that gave rise to the data, and setting a threshold on this density: if a new example has a probability lower than the threshold, it is classified as an outlier.

2.1 Density Estimation

Phase 1: Kernel density estimation Initially the probability density of the data is determined using Kernel Density Estimation. This technique allows us to evaluate the probability of a new example by taking a uniformly weighted combination of a set of Gaussian kernels (with identical covariance matrices Σ) centered on each of the training data. Thus, finding the probability of a new data point z, given a set of N training data $X = \{x_1, \ldots, x_N\}$, simply consists of evaluating the following function:

$$p(z) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} \cdot \frac{1}{N} \cdot \sum_{n=1}^{N} e^{-\frac{1}{2}(z-x_n)^T \Sigma^{-1}(z-x_n)} \tag{1}$$

The factor $\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}}$ (where d refers to the dimensionality of the data) ensures that the resulting probability distribution integrates to 1. Training the model is straight-forward: when a new training example x_{NEW} is received, it is simply added to the set X:

$$X \rightarrow X \cup \{x_{NEW}\} \tag{2}$$

The only remaining problem is the choice of the covariance matrix Σ. To reduce computational cost, we use a uniform covariance matrix $\Sigma(\sigma) = I^d \cdot \sigma^2$, which leaves only a single parameter σ to be determined. The value of σ is chosen using the "leave-one-out" likelihood criterion proposed by Duin in [1] (recently shown to be a good model selection criterion for multivariate kernel density estimation by Zhang et al. in [10]). This technique allows us to find a parameter that maximizes the likelihood of the dataset, while avoiding the problem of the data likelihood tending towards infinity as $\sigma \rightarrow 0$. For a given value of σ, the "leave-one-out" likelihood function combines the log-likelihoods for every

individual example x_n given a model constructed from all others $\forall x \neq x_n$, as follows:

$$LL(\sigma) = \sum_{n=1}^{N} \log \left(\frac{1}{(2\pi\sigma)^{\frac{d}{2}}} \cdot \frac{1}{N-1} \cdot \sum \forall x \neq x_n e^{-\frac{1}{2\sigma^2}(x_n - x)^T (x_n - x)} \right) \quad (3)$$

Every time the training dataset is updated (during the kernel density estimation phase) we evaluate (3) for a range of values surrounding the previous σ, and choose $\sigma = \arg\max_\sigma (LL(\sigma))$.

Phase 2: Mixture model merging Since the computational cost of evaluating (1) scales linearly with the quantity of training data, it eventually becomes infeasible to estimate the p.d.f. of the data in this fashion. Noting that the kernel density estimate is essentially a maximally-complex Gaussian mixture model, we adapt a method proposed by Goldberger and Roweis for reducing the complexity of Gaussian mixture models in [2]: once the maximum feasible model complexity has been reached, we keep it constant by merging a pair of components for each new component added.

Initialization Once the maximum model complexity has been reached $N = N_{max}$ we initialize a data structure to store a Gaussian mixture model with weights (initially uniform), and covariances (set to the final value estimated in the kernel density phase), and means (the training data) as follows:

$$\begin{aligned} w_{1\ldots N} &= \tfrac{1}{N} \\ \Sigma_{1\ldots N} &= I^d \cdot \sigma^2_{final} \\ \mu_{1\ldots N} &= x_{1\ldots N} \end{aligned} \quad (4)$$

For each pair of components $G_i = \{w_i, \mu_i, \Sigma_i\}$ and $G_j = \{w_j, \mu_j, \Sigma_j\}$ a merging cost is then calculated using (7) - explained in the next section - forming an $N \times N$ matrix C. This one-off[1] calculation of $\frac{N_{max}(N_{max}-1)}{2}$ different cost values is computationally feasible for values of N_{max} where it is still possible to evaluate (1) in reasonable time.

Merging Strategy In this stage every new training example still contributes a Gaussian kernel. However the covariance matrix is now fixed to the final estimate obtained in the preceding stage, and we employ a merging strategy to keep the number of mixture components constant. For every new component added, a pair of components (which may include the new one) is merged as follows:

$$\begin{aligned} w_{merge(i,j)} &= w_i + w_j \\ \mu_{merge(i,j)} &= \tfrac{w_i}{w_i+w_j}\mu_i + \tfrac{w_j}{w_i+w_j} \cdot \mu_j \\ \Sigma_{merge(i,j)} &= \tfrac{w_i}{w_i+w_j}\left(\Sigma_i + (\mu_i - \mu_{merge(i,j)})(\mu_i - \mu_{merge(i,j)})^T\right) \\ &\quad + \tfrac{w_j}{w_i+w_j}\left(\Sigma_j + (\mu_j - \mu_{merge(i,j)})(\mu_j - \mu_{merge(i,j)})^T\right) \end{aligned} \quad (5)$$

[1] Subsequently maintaining this cost matrix only requires a fixed number of $N_{max}+1$ cost evaluations for each new training example.

We wish to choose a pair of components to merge in a way that minimizes the resulting change in the p.d.f. encoded by the model. The Kullback-Leibler divergence provides a means of assessing the "damage" caused by replacing a particular pair of components with a single merged component. Essentially, the KL divergence $\mathcal{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx$ quantifies the expected information loss per sample when an approximating distribution Q is substituted for a true distribution P. For a pair of Gaussian distributions $G_p = \{\mu_p, \Sigma_p\}$ and $G_q = \{\mu_q, \Sigma_q\}$, it can be calculated as follows [2]:

$$\mathcal{KL}(G_p||G_q) = \frac{1}{2}\left(\log\frac{|\Sigma_q|}{|\Sigma_p|} + \mathrm{Tr}(\Sigma_q^{-1}\Sigma_p) + (\mu_p - \mu_q)\Sigma_q^{-1}(\mu_p - \mu_q)^T - d\right) \quad (6)$$

This allows us to quantify the cost of replacing components G_i and G_j (where $i \neq j$) with their merged counterpart $G_{merge(i,j)}$ by calculating a weighted combination (as proposed by Goldberger and Roweis in [2]) of their respective Kullback-Leibler divergences from $G_{merge(i,j)}$ as follows:

$$cost(G_i, G_j) = w_i \mathcal{KL}(G_i||G_{merge(i,j)}) + w_j \mathcal{KL}(G_j||G_{merge(i,j)}) \quad (7)$$

Updating Procedure When a new training example x_{NEW} arrives, a temporary new component $G_{N_{max}+1} = \{\frac{1}{N_{ex}+1}, x_{NEW}, I^d \cdot \sigma^2_{final}\}$ is created, and the weights of existing components are rescaled by a factor of $\frac{N_{ex}}{N_{ex}+1}$, where N_{ex} is the total number of training examples received before the new one. The cost matrix is augmented with a new row/column for the new component, and a pair of components is chosen such that $\{G_i, G_j\} = \arg\min_{G_i,G_j}(cost(G_i, G_j))$. If $\{G_i, G_j\}$ are both existing components, then G_i is replaced with $G_{merge(i,j)}$ and G_j is replaced with the new component; alternatively if G_j is the new component then G_i is simply replaced with $G_{merge(i,j)}$. The temporary component $G_{N_{max}+1}$ is then removed, and the merging cost matrix C updated accordingly. This procedure requires a fixed total of $N_{max} + 1$ evaluations of (7) for every new training example, as the cost matrix only needs to be updated for entries corresponding to merged/new components.

2.2 Classification Threshold

Given the proposed density estimation method, an important remaining issue is the choice of classification threshold. A naive approach would be to set the threshold at that the level of the least probable (given the current model) training example, thereby correctly classifying all training data as normal. However, it is quite possible that the least probable training example - which will be located in the most sparsely populated region of training data - may have a probability value equivalent to that of the outliers we wish to detect. To avoid this problem, and to make the method robust to potential outliers in the training set, we set the threshold at a value that deliberately misclassifies a certain proportion of the training data as outliers. In the experiments described in the following section

we choose a value of 10%, aiming to learn a classifier that filters out 90% of normal data.

3 Experiments

In this section we measure the classification performance of the proposed algorithm on a variety of datasets, showing how classification performance changes as the model is trained on more training examples. To place the performance of the proposed algorithm in context we compare its performance to that of the incremental SVDD algorithm [5], making comparisons at the point where both algorithms have been trained on all training examples in a given dataset.

We use a freely available implementation of the incremental SVDD algorithm, *incsvdd*, contained in the *DDtools* MATLAB toolbox [6]. In all tests we use the radial basis kernel function, and optimize the kernel parameter (for the whole training dataset) using the *consistent_occ* function (also from [6]) which implements the consistency-based model selection criterion proposed in [8]. We initially apply this criterion to a range of 20 linearly spaced values between the shortest and longest Euclidean distances observed within the dataset; to search for potentially better parameter values on a finer scale, we then run a second parameter optimization for a further 20 values surrounding the optimal parameter from the first set. As for the proposed algorithm, we set the SVDD threshold parameter at a level that aims to reject to 10% of the training data.

Synthetic Dataset An initial experiment was carried out on a synthetic 2 dimensional dataset: we defined a spiral shaped region which we used to divide a set of uniformly distributed random datapoints into a hypothetical normal class of datapoints (points in the spiral region) and outliers (all other points). We used 2500 spiral points for training the algorithm, and a further 2500 points from the spiral along with 2500 outlier points for testing it, as shown in Figure 1.

For this test (as for all subsequent tests) we set the upper limit on the number of mixture components N_{max} to be 100. The middle section of figure 1 shows the configuration of the 100 Gaussian components before the merging phase commences, and at the end of the training process. The resulting model organization appears to accurately reflect the shape of the spiral: indeed, at the end of training the algorithm correctly classifies 88.13% of all test data, with a True Positive rate[2] of $TP = 86.1\%$ and a False Positive rate[3] of $FP = 0.0984\%$. The TP and FP curves shown in the lower left hand section of Figure 1 indicate that the classification performance increased in a stable fashion as more training examples were processed. In this plot, and in subsequent plots of this type, the vertical dotted line indicates the start of the merging phase.

At the end of training, the incremental SVDD algorithm correctly classified 79.44% of the test data (with $TP = 89.69\%$ and $FP = 0.308\%$), misclassifying

[2] Indicating normal examples correctly identified as normal.
[3] Indicating outliers incorrectly classified as normal.

a much larger number of outliers as normal. The ROC[4] curve in the lower right hand section of Figure 1 shows the different TP and FP values obtained as the (training data rejection) threshold is varied for each classifier, indicating that the proposed algorithm outperforms incremental SVDD algorithm across the range of possible thresholds. Both plots in Figure 1 show the mean performance for 10 different random orderings of the training data.

Real Datasets A series of subsequent experiments were then carried out on three different real-world datasets obtained from the UCI Machine Learning Repository[5]:

1. The *Wisconsin Breast Cancer Database*, which contains 699 (9-dimensional) datapoints, containing 458 normal examples and 241 cases of cancer.
2. The *Letter Recognition Database,* which contains 20,000 (16-dimensional) parametrizations of examples of printed letters, with 26 classes corresponding to the alphabet. We use the 789 examples of the letter 'A' as a hypothetical normal class, and all other classes as outliers.
3. The *STATLOG Landsat Satellite Database*, which contains 6435 (36 dimensional) vectors corresponding multispectral images of 6 different types of ground coverage: we use the 1533 examples of 'red soil' as the normal class.

For each of these datasets we use 90% of examples of the chosen normal class as training data, and the remaining 10% for testing. All subsequent experiments are performed for 10 different testing/training permutations of the normal class. Again, we test our algorithm with a maximum complexity level of 100 components, and compare it to the consistency-optimized incremental SVDD algorithm. The classification results illustrated by the ROC curves in Figure 2, indicate that the proposed algorithm consistently outperforms the incremental SVDD algorithm, although the performance obtained on the Cancer and Satellite datasets is very similar.

Computational Complexity To confirm the assertion that the proposed algorithm has bounded computational complexity, we recorded the time taken to train our algorithm on each datapoint during the tests on the 36 dimensional Satellite dataset. This is plotted (excluding the point where the merging matrix is first initialized) in Figure 3, indicating that a fixed processing time per example is indeed reached soon after the merging phase commences. Our algorithm takes an average time of 565.56 ± 0.32 seconds to train on the 1379 examples, while the SVDD algorithm takes a significantly shorter time of 6.25 ± 0.34 seconds to train, albeit after a parameter optimization step which takes 482.53 ± 60.21 seconds. Evaluation times for the two algorithms are similar: our algorithm takes 2.88 ± 0.02 seconds to classify 5056 testing examples, while the incremental SVDD algorithm takes 2.01 ± 0.102 seconds to classify the same examples.

[4] Receiver Operating Characteristic
[5] http://www.ics.uci.edu/~mlearn/MLRepository.html

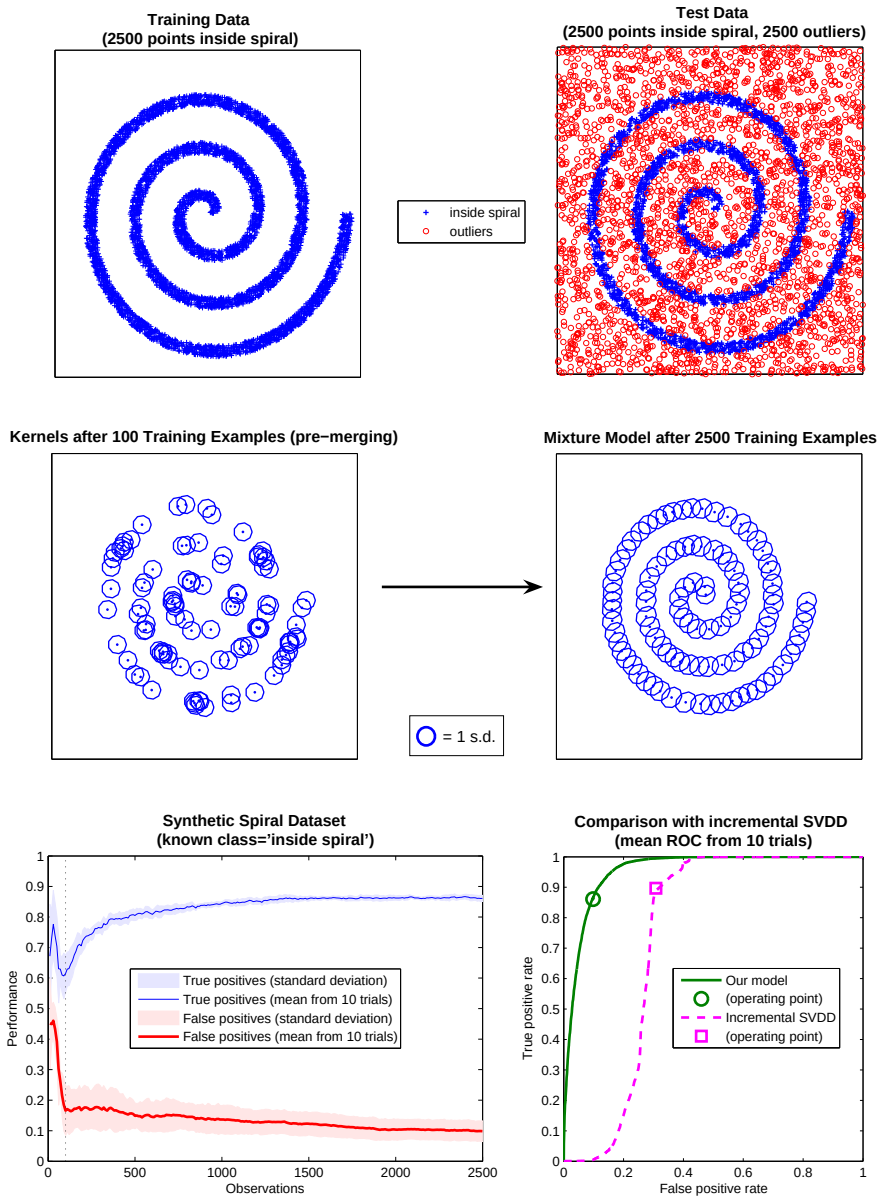

Figure1. Results for the synthetic spiral dataset. See text for description.

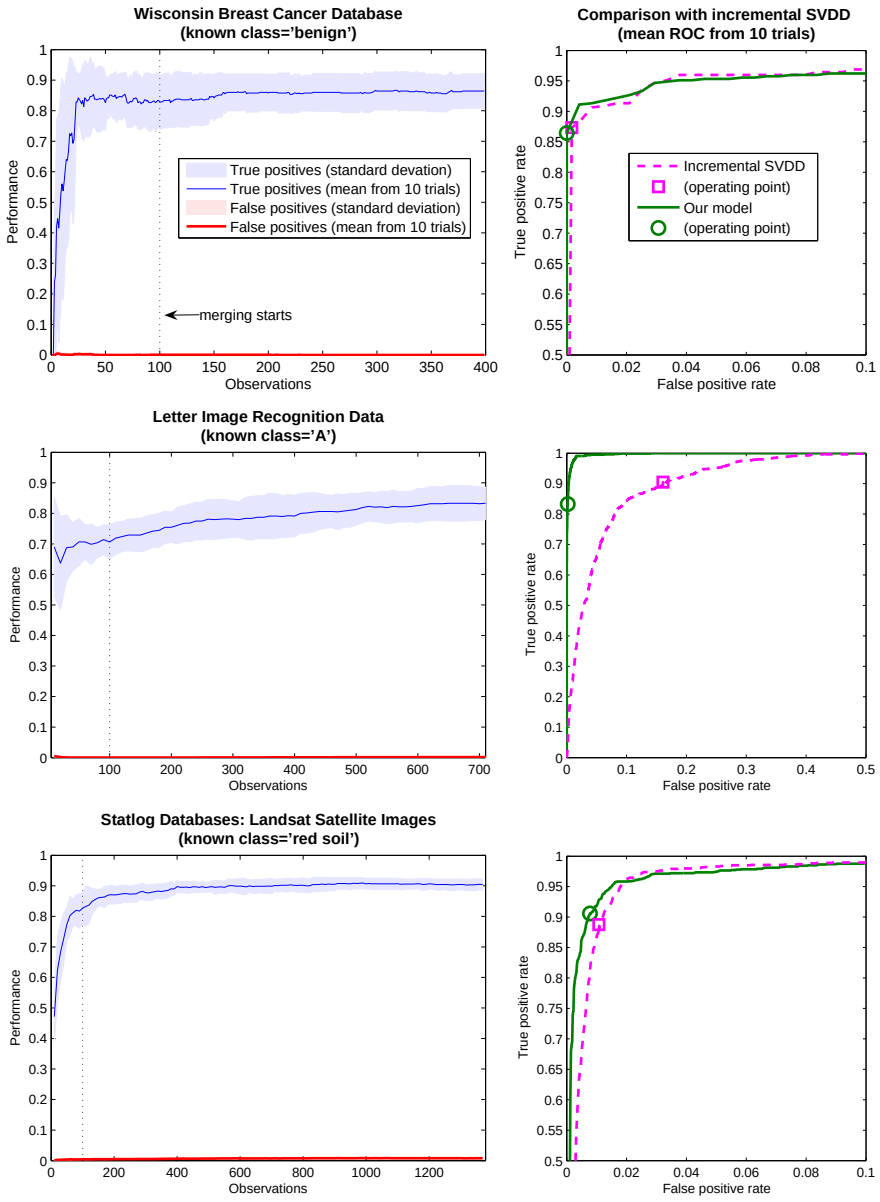

Figure2. Results for real datasets. See text for description.

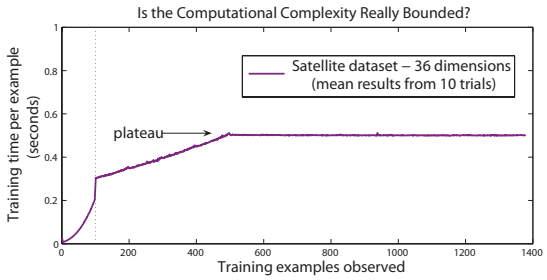

Figure3. Measuring computational complexity.

4 Discussion

We have proposed a simple procedure for incrementally training a one-class classifier to perform outlier detection, without the need for any time-consuming model optimization procedures. Despite its simplicity, the proposed algorithm appears to perform better than the incremental SVDD algorithm, even though the parameters of the latter were being chosen through a lengthy optimization process. The fact that the optimization process proposed in [8] did not find parameters that allowed incremental SVDD to outperform our algorithm does not mean that such parameters could not be found in principle: it does, however, illustrate the the key strength of our algorithm - the fact that it automatically generates models that achieve a useful level of outlier detection performance.

References

1. R.P.W. Duin. On the choice of smoothing parameters for Parzen estimators of probability density functions. *IEEE Trans. Computers*, C-25:1175–1179, 1976.
2. J. Goldberger and S. Roweis. Hierarchical clustering of a mixture model. In *Advances in Neural Information Processing Systems 17*, pages 505–512. 2005.
3. V.J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:85–126, 2004.
4. D. M. J. Tax and R. P. W. Duin. Uniform object generation for optimizing one-class classifiers. *Journal of Machine Learning Research*, 2:155–173, 2001.
5. D. M. J. Tax and P. Laskov. Online SVM learning: from classification to data description and back. In *Proc. 13th IEEE NNSP Workshop*, 2003.
6. D.M.J. Tax. DDtools, the Data description toolbox for Matlab. version 1.5.5.
7. D.M.J. Tax and R.P.W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20:1191–1199, 1999.
8. D.M.J. Tax and K.-R. Muller. A consistency-based model selection for one-class classification. In *Proc. ICPR 2004*, volume 3, pages 363–366, 2004.
9. K. Yamanishi, J. Takeuchi, G. Williams, and P. Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Data Mining and Knowledge Discovery*, 8:275–300, 2004.
10. X. Zhang, M. King, and R. J. Hyndman. A Bayesian approach to bandwidth selection for multivariate kernel density estimation. *Computational Statistics & Data Analysis*, 50:3009–3031, 2006.