Scaling up Support Vector Data Description by Using Core-Sets

Calvin S. Chu Ivor W. Tsang James T. Kwok
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay
Hong Kong
E-mail: {cscalvin,ivor,jamesk}@cs.ust.hk

***Abstract*— Support vector data description (SVDD) is a powerful kernel method that has been commonly used for novelty detection. While its quadratic programming formulation has the important computational advantage of avoiding the problem of local minimum, this has a runtime complexity of $O(N^3)$, where N is the number of training patterns. It thus becomes prohibitive when the data set is large. Inspired from the use of core-sets in approximating the minimum enclosing ball problem in computational geometry, we propose in this paper an approximation method that allows SVDD to scale better to larger data sets. Most importantly, the proposed method has a running time that is only linear in N. Experimental results on two large real-world data sets demonstrate that the proposed method can handle data sets that are much larger than those that can be handled by standard SVDD packages, while its approximate solution still attains equally good, or sometimes even better, novelty detection performance.**

I. Introduction

In recent years, there has been a lot of interest on using kernels in various aspects of machine learning, such as classification, regression, clustering, ranking and principal component analysis [1], [2], [3]. A well-known example in supervised learning is the support vector machines (SVMs). The basic idea of kernel methods is to map the data from an input space to a feature space $\mathcal{F}$ via some map φ, and then apply a linear procedure there. It is now well-known that the computations do not involve φ explicitly, but depend only on the inner product defined in $\mathcal{F}$, which in turn can be obtained efficiently from a suitable *kernel* function (the "kernel trick").

In this paper, we will focus on the use of kernel methods in novelty detection, in which one aims at differentiating known objects (or normal patterns) from unknown objects (or outliers) [4], [5], [6]. There are a large number of real-world novelty detection applications, such as the detection of unusual vibration signatures in jet engines [7] or the detection of new events from newswire stories in text mining [8]. As only the positive information is available, novelty detection is more challenging than supervised learning. Traditionally, novel patterns are detected by either estimating the density function of the normal patterns, or by finding a small set $\mathcal{Q}$ such that $P(\mathbf{x} \in \mathcal{Q}) = \alpha$ for some fixed $\alpha \in (0, 1]$ (quantile estimation). However, both depend critically on the parametric form of the density function, and can fail miserably when the parametric form is incorrect.

Instead of estimating the density or quantile, a simpler task is to model the support of the data distribution directly. Tax and Duin proposed the *support vector data description* (SVDD) [9], which uses a ball with minimum volume to enclose most of the data. Computationally, this leads to a quadratic programming (QP) problem, which has the important advantage that the solution obtained is always globally optimal. Moreover, as with other kernel methods, SVDD works well with high-dimensional data and can be easily kernelized by replacing the dot product between patterns with the corresponding kernel evaluation.

Besides using a ball, one can also use a hyperplane. Schölkopf *et al.* proposed the *one-class SVM* that separates the normal patterns from the outliers (represented by the origin) with maximum margin [10], [11]. Again, computationally, this leads to a QP problem. Moreover, when Gaussian kernels are used, the one-class SVM solution is equivalent to that of the SVDD.

Recently, Lanckriet *et al.* proposed the *single-class minimax probability machine* (MPM) [12] that is also based on the use of hyperplanes. A distinctive aspect of the single-class MPM is that it can provide a distribution-free probability bound. Specifically, given only the mean and covariance matrix of a distribution and without making any other distributional assumption, it seeks the smallest half-space $\mathcal{Q}(\mathbf{w}, b) = \{\mathbf{z} \mid \mathbf{w}'\mathbf{z} \geq b\}$, not containing the origin, that minimizes the worst-case probability of a data point falling outside of $\mathcal{Q}$. However, despite its interesting theoretical properties, the single-class MPM has high false negative rate in practice [13], and some uncertainty information on the covariance matrix is required to alleviate this problem.

While the QP formulations in both SVDD and one-class SVM have the computational advantage of avoiding the problem of local minimum, their runtime complexities are of $O(N^3)$, where N is the number of training patterns. In order to allow the QP to scale better to larger data sets, Schölkopf *et al.* [11] suggested the use of a modified version of the sequential minimal optimization (SMO) algorithm [14]. However, as will be demonstrated experimentally in Section IV, a SMO-based implementation of the one-class SVM can still suffer from scale-up problems.

A more radical possibility to improve the scale-up behavior

is by changing the formulation altogether, such as from quadratic programming to linear programming (LP) [15]. In general, LPs can handle large data sets, especially with the use of column generation algorithms. However, the LP formulation in [15] is based on minimizing the mean value of the outputs on the normal patterns, which is less intuitive than minimizing the size of the ball as in SVDD or maximizing the margin as in one-class SVM.

Viewing from a broader perspective, the ball-finding problem in SVDD is related to the *minimum enclosing ball* (MEB) problem in computational geometry [16], [17], [18]. Given a set S of points, the MEB of S, denoted by MEB(S), is the unique minimum radius ball that contains all of S. Recently, Badoiu *et al.* [16] showed that there exist efficient algorithms for finding its $(1+\epsilon)$-approximation[1]. The main idea is to use only a small subset of points, called the *core-set*[2], in computing the approximate solution. The resultant core-set can be shown to be of size $O(1/\epsilon)$. Subsequently, the algorithm has a running time that is only linear in the number of points, and is thus readily scalable.

However, despite the apparent similarity between the MEB problem and SVDD, these MEB algorithms cannot be readily applied to SVDD. A crucial distinction is that the MEB is required to enclose all data points in S, including even outliers. SVDD, on the other hand, allows outliers to remain outside the ball with the use of slack variables. Moreover, most existing algorithms for finding the MEB can only handle low-dimensional data, whereas SVDD has to operate in the possibly infinite-dimensional kernel-induced feature space.

Inspired by the MEB algorithms and the use of core-sets, we propose in this paper a procedure for speeding up SVDD. Most importantly, we will show that its running time is only linear in the number of training patterns (N), instead of the $O(N^3)$ complexity for standard SVDD. The rest of this paper is organized as follows. Section II first introduces SVDD. Section III then describes our proposed speed-up procedure. Experimental results on two large, real-world data sets are presented in Section IV, and the last section gives some concluding remarks.

II. Support Vector Data Description

Given a set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, SVDD attempts to find a small ball, with center $\mathbf{c}$ and radius R, that contains most of the patterns in S. To allow for the presence of outliers, slack variables ξ_i's are introduced as in other kernel methods. The (primal) optimization problem is then

$$\begin{aligned} \min_{R,\xi_i \geq 0, \mathbf{c}} \quad & R^2 + \frac{1}{\nu N}\sum_{i=1}^{N}\xi_i \\ \text{s.t.} \quad & \|\mathbf{c}-\mathbf{x}_i\|^2 \leq R^2 + \xi_i,\ \xi_i \geq 0,\ \ i = 1, \ldots, N, \end{aligned}$$

Here, $\nu \in (0,1)$ is a user-provided parameter specifying an upper bound on the fraction of outliers. The corresponding dual problem is:

$$\begin{aligned} \max_{\alpha_i} \quad & \sum_{i=1}^{N}\alpha_i \mathbf{x}_i'\mathbf{x}_i - \sum_{i,j=1}^{N}\alpha_i\alpha_j\mathbf{x}_i'\mathbf{x}_j \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq \frac{1}{\nu N},\ \ i = 1, \ldots, N, \\ & \sum_{i=1}^{N}\alpha_i = 1. \end{aligned}$$

This is a quadratic programming problem in the N variables $\alpha_1, \ldots, \alpha_N$. As mentioned in Section I, its solution is guaranteed to be globally optimal. By using the Karush-Kuhn-Tucker (KKT) condition, the center can be obtained from the α_i's as $\mathbf{c} = \sum_{i=1}^{N}\alpha_i\mathbf{x}_i$. Moreover, the radius R can also be computed by calculating the distance between $\mathbf{c}$ and any support vector $\mathbf{x}_i$ on the boundary of the ball.

On testing, a new pattern $\mathbf{z}$ will be predicted to be an outlier if its distance from the center $\mathbf{c}$ is larger than the radius; otherwise, it will be predicted as normal. Finally, notice that SVDD can be easily kernelized by simply replacing $\mathbf{x}_i'\mathbf{x}_j$ in the computations by $k(\mathbf{x}_i, \mathbf{x}_j)$, where $k(\cdot,\cdot)$ is some suitable kernel function.

III. Scaling up SVDD

In this Section, we borrow the idea of core-sets in the MEB algorithms to scale up SVDD. The basic procedure is as follows. First, we construct an initial core-set containing only one normal pattern (Section III-A), and patterns are then added to it incrementally. Instead of using all N training patterns in SVDD's QP, we only use patterns in this core-set to form the QP (Section III-B). By keeping the size of the core-set small (say, of size $n << N$), the computational complexity of each QP will be of $O(n^3) << O(N^3)$. Moreover, as will be shown in Section III-C, the number of iterations is independent of N, which then enables the proposed procedure to have a runtime complexity that is only linear in the number of training patterns, which is similar to the MEB algorithms discussed in Section I.

A. Initialization

There are two issues that have to be tackled on initialization. First, we have to find a pattern that is very likely to be normal. A natural choice is to use the pattern in S that is closest to the sample mean. In the input space, this sample mean can be easily obtained as an explicit data vector. However, in the kernel-induced feature space, the sample mean can only be expressed as a linear combination of the N φ-mapped patterns (where φ is the nonlinear mapping corresponding to the kernel function). Computing the distance between any φ-mapped pattern and this sample mean thus takes $O(N)$ time. Consequently, finding the pattern closest to the sample mean will already take $O(N^2)$ time, defeating our goal of obtaining a procedure whose runtime is only linear in N.

[1]Denote the ball with center $\mathbf{c}$ and radius r by $B_{\mathbf{c},r}$. A ball $B_{\mathbf{c},r} \supset S$ is a $(1+\epsilon)$-approximation of MEB(S) if $r \leq (1+\epsilon)r^*$, where r^* is the radius of MEB(S) and $\epsilon > 0$.

[2]To be more specific, a subset $X \subseteq S$ is a core-set of S if $B_{\mathbf{c},(1+\epsilon)r} \supset S$, where $B_{\mathbf{c},r} = \text{MEB}(X)$.

Thus, instead, we first randomly sample a fixed number (say, n_0) of patterns from S. Standard SVDD is then run on these n_0 patterns to obtain a ball with center $\tilde{\mathbf{c}}$. Among these n_0 patterns, the pattern $\mathbf{z}$ that is closest to $\tilde{\mathbf{c}}$ is picked. Intuitively, this $\mathbf{z}$ is unlikely to be an outlier and so will be used in constructing the initial core-set. Moreover, as will be shown in Section III-C, such initialization only takes linear time.

The second issue is on how to set the initial radius R_1 of the ball. A small R_1 will be desirable so that the initial ball does not contain any outlier. Hence, we first randomly pick a pattern $\mathbf{x}$ from among those n_0 patterns above and then find the pattern $\mathbf{y} \in S$ that is furthest from $\mathbf{x}$. Define $D = \|\mathbf{x}-\mathbf{y}\|$. It is obvious that $D \geq R_{MEB(S)}$, where $R_{MEB(S)}$ is the radius of MEB(S). We then initialize $R_1 = D/k$, where $k > 1$, such that R_1 is a small number. Moreover, note that

$$R_1 \geq R_{MEB(S)}/k. \tag{1}$$

B. Iterative Procedure

After initialization, patterns will be added to the core-set incrementally. In the following, we denote the center, radius and the core-set at the ith iteration by $\mathbf{c}_i, R_i$ and S_i respectively. Moreover, as in traditional SVDD, we assume that the user will supply the value of ν, which is an upper bound on the fraction of outliers.

The following iterative procedure is then taken:

1) Initialize R_1 and $\mathbf{z}$ as mentioned in Section III-A. Set $S_1 = \{\mathbf{z}\}, \mathbf{c}_1 = \mathbf{z}$ and $i = 1$.
2) Find the set P_i of patterns in S that fall outside the $(1+\epsilon)$-ball $B_{\mathbf{c}_i,(1+\epsilon)R_i}$. In other words,
$$P_i = \{\mathbf{x} \in S \mid \|\mathbf{x} - \mathbf{c}_i\| > (1+\epsilon)R_i\}.$$
3) If the size of P_i is smaller than νN, the expected number of outliers, then terminate.
4) Otherwise, expand the core-set by including the pattern in $P_i \setminus S_i$ that is closest to $\mathbf{c}_i$. Denote the expanded core-set by S_{i+1}.
5) Run SVDD on S_{i+1}, and obtain the new center $\mathbf{c}_{i+1}$ and radius R_{i+1}.
6) Enforce the constraint that
$$R_{i+1} \geq (1+\delta\epsilon)R_i, \tag{2}$$
where δ is a small, user-defined constant. In other words, the radius must increase by at least $\delta\epsilon R_i$ at the ith iteration. As will be discussed in section III-C, this constraint is crucial for bounding the time complexity.
7) Increment i by 1 and go back to Step 2.

C. Time Complexity

In the MEB problem, it can be shown that the number of iterations in a similar procedure as above is of $O(1/\epsilon^2)$ [16] (or even $O(1/\epsilon)$ when the furthest pattern is used in each iteration [17]). However, as mentioned in Section I, these results cannot be directly applied here because of the presence of slack variables in the SVDD formulation. Nevertheless, in this Section, we will still be able to show that the algorithm in Section III-B has a time complexity that is only linear in the number of training patterns N.

Consider first the initialization step. As n_0 is fixed, both running the initial SVDD and the finding of $\mathbf{z}$ only take $O(1)$ time. In determining the initial radius R_1, the finding of $\mathbf{y}$ takes $O(N)$ time. So, the total time required for initialization is $O(N)$.

At the ith iteration, R_i is increased by at least

$$\delta\epsilon R_i > \delta\epsilon R_{i-1} > \cdots > \delta\epsilon R_1 \geq (\frac{\delta\epsilon}{k})R_{MEB(S)},$$

on using (1) and (2). Obviously, $R_{MEB(S)}$ is an upper bound on the radius of the ball required. Therefore, the total number of iterations is no more than $\frac{k}{\delta\epsilon} = O(1/\epsilon)$.

At each iteration, one pattern will be added to the core-set in step 4. Hence, the size of S_i is i and consequently $\mathbf{c}_i$ is a linear combination of i φ-mapped patterns. Thus, at the ith iteration, step 4 requires takes $O(iN)$ time, while running SVDD in step 5 takes $O((i+1)^3) = O(i^3)$ time. The other steps take only constant time. And so the total time for the ith iteration is $O(iN + i^3)$.

The total time for the whole procedure, including initialization and $M = O(1/\epsilon)$ iterations, is then:

$$\begin{aligned} T &= O(N) + \sum_{i=1}^{M} O(iN + i^3) \\ &= O(N) + \left(\sum_{i=1}^{M} i\right) O(N) + \sum_{i=1}^{M} i^3 \\ &= O(M^2N + M^4) \\ &= O(\frac{1}{\epsilon^2}N + \frac{1}{\epsilon^4}). \end{aligned} \tag{3}$$

For a fixed ϵ, T is thus linear in N, instead of $O(N^3)$ in traditional SVDD.

IV. Experiment

In this Section, we perform experiments on two real-world datasets, the BioID Face Database[3] (Section IV-A) and the MNIST handwritten digits database[4] (Section IV-B). In the sequel, we denote the proposed method by CSVDD, which stands for "Core-set Support Vector Data Description". It will be compared with two standard SVDD packages: the data description toolbox (dd_tools) and the SMO-based LIBSVM[5] [19]. All these are implemented in MATLAB, with MOSEK[6] (version 3) being used as the underlying QP solver in both dd_tools and CSVDD. Experiments are run on a P4-2.24GHz machine, with 1G RAM, running Windows XP. Moreover, in

[3]The BioID Face Database can be downloaded from `http://www.humanscan.de/support/downloads/facedb.php`.

[4]The MNIST database can be downloaded from `ftp://ftp.kyb.tuebingen.mpg.de/pub/bs/data`.

[5]dd_tools and LIBSVM can be downloaded from `http://www.ph.tn.tudelft.nl/~davidt/dd_toolbox.html` and `http://www.csie.ntu.edu.tw/~cjlin/libsvm` respectively.

[6]MOSEK can be downloaded from `http://www.mosek.com`.

all the experiments, we set n_0 in the initialization step to 20, k in (1) to 10, and δ in (2) to 0.01ϵ.

Several criteria are used to compare the novelty detectors. The first set of criteria, namely the AUC error, FP and FN, are based on the ROC (receiver operating characteristic) graph [20], which plots the true positive (TP) rate on the Y-axis and the false positive (FP) rate on the X-axis. TP and FP are defined by

$$\begin{aligned} \text{TP} &= \frac{\text{positives correctly classified}}{\text{total positives}}, \\ \text{FP} &= \frac{\text{negatives incorrectly classified}}{\text{total negatives}}, \end{aligned}$$

respectively. Here, outliers are treated as positives while normal patterns as negatives. Similarly, the false negative rate (FN) is defined as

$$\text{FN} = \frac{\text{positives incorrectly classified}}{\text{total positives}}.$$

AUC stands for the area under the ROC curve, and the AUC error is defined as $(1 - \text{AUC})$. The AUC error is always between 0 and 1. A perfect novelty detector will have zero AUC error, while random guessing will have an AUC error of 0.5. The second criterion is the CPU time (in seconds) required by Matlab for the whole procedure. To reduce statistical variability, results here are based on averages over 20 random repetitions.

A. Face Database

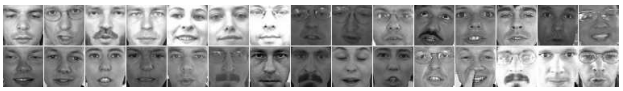
This database contains 1,521 gray level images, each showing the frontal view of a face of one out of 23 different test persons. Because the images are taken under a large variety of background, the faces need to be first manually aligned and cropped, producing images with a resultant size of 25×25. Moreover, in order to better evaluate the scale-up capabilities, we use tensor voting [21] to expand this database to a total of 75,787 face images, with grey scale from 0 to 255. No preprocessing, such as illumination correction, mean value normalization and histogram equalization are performed. Moreover, to provide outliers in the experiments, we gather some non-face images from images of natural scenes, buildings and textures. Sample images are shown in Figure 1.

During training, we use 100 to 30,000 face images sampled from the 75,787 images generated above, and no non-face image is used. The test set consists of 2,000 face and 2,000 non-face images. We use the Gaussian kernel

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right), \tag{4}$$

with $\sigma = 500$. Moreover, ν is always set to 0.05.

Table I compares CSVDD (with $\epsilon = 0.3$) with the standard SVDD packages of dd_tools and LIBSVM. As can be seen, the AUC errors of all three are comparable for this value of ϵ, and CSVDD has a lower FN but higher FP. Thus, although CSVDD is only an approximate solution, results here indicated that its solution is still close to optimal.



(a) Images cropped from the database.

(b) Images generated by tensor voting.

(c) Non-face images.

Fig. 1. Sample images used in the experiment in Section IV-A.

TABLE I
RESULTS ON THE FACE DATA SET (NUMBERS IN BRACKETS ARE THE STANDARD DEVIATIONS).

size of tr set	method	AUC error	FN	FP
100	dd_tools	0.0489 (0.003)	0.28 (0.04)	0.07 (0.01)
	LIBSVM	0.0489 (0.003)	0.28 (0.04)	0.07 (0.01)
	CSVDD	0.0484 (0.004)	0.02 (0.02)	0.28 (0.19)
300	dd_tools	0.0499 (0.004)	0.14 (0.01)	0.09 (0.01)
	LIBSVM	0.0499 (0.004)	0.14 (0.01)	0.09 (0.01)
	CSVDD	0.0507 (0.007)	0.01 (0.01)	0.30 (0.18)
600	dd_tools	0.0488 (0.003)	0.10 (0.01)	0.11 (0.01)
	LIBSVM	0.0488 (0.003)	0.10 (0.01)	0.11 (0.01)
	CSVDD	0.0504 (0.005)	0.02 (0.01)	0.24 (0.06)
1000	dd_tools	0.0492 (0.002)	0.08 (0.00)	0.12 (0.01)
	LIBSVM	0.0492 (0.002)	0.08 (0.00)	0.12 (0.01)
	CSVDD	0.0468 (0.005)	0.01 (0.01)	0.27 (0.09)
3000	dd_tools	0.0497 (0.001)	0.06 (0.01)	0.14 (0.01)
	LIBSVM	0.0497 (0.001)	0.06 (0.01)	0.14 (0.01)
	CSVDD	0.0478 (0.005)	0.01 (0.01)	0.23 (0.06)
6000	CSVDD	0.0531 (0.006)	0.01 (0.01)	0.25 (0.06)
10000	CSVDD	0.0499 (0.004)	0.02 (0.01)	0.21 (0.05)
20000	CSVDD	0.0492 (0.005)	0.01 (0.01)	0.26 (0.08)
30000	CSVDD	0.0498 (0.005)	0.02 (0.01)	0.23 (0.07)

Figure 2 compares their speeds. When the training set is small, both dd_tools and LIBSVM are faster than CSVDD. This, nevertheless, is not surprising as CSVDD has to run the QP multiple times. In fact, under this situation, there is no scale-up problem and the standard SVDD packages should be the preferred choice. The real power of CSVDD, however, can be seen as the training set gets larger. This can be seen more clearly in Figure 2, which shows that CSVDD then becomes significantly faster than the other two traditional SVDD approaches. With 6,000 or more training patterns, both dd_tools and LIBSVM cannot be run on the machine used in the experiment.

We then vary the value of ϵ used in CSVDD. Table II shows that there is only very small difference in terms of novelty

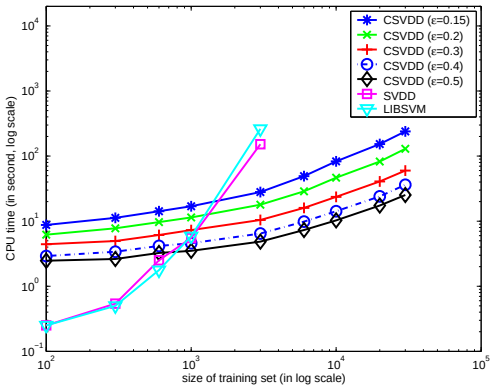


Fig. 2. CPU time vs number of training patterns (Face database).

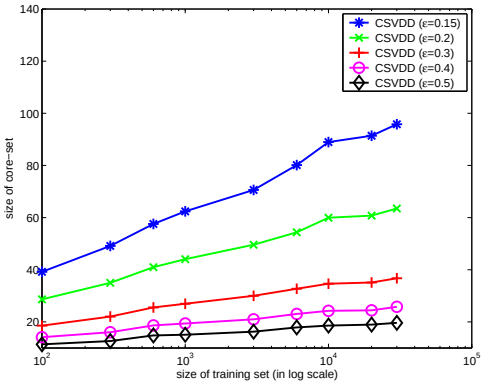


Fig. 3. Size of the core-set at different values of ϵ (Face database).

detection performance when ϵ is changed in the range tested. However, as can be seen in (3), a smaller value of ϵ leads to longer training time (Figure 2). Figure 3 plots the size of the resultant core-set, which is the same as the number of iterations, with the value of ϵ. As expected, the resultant core-set is only a very small subset of the training set, and its size also grows slowly with increasing training set size. Moreover, as discussed in Section III-C, the number of iterations is of $O(1/\epsilon)$, and this rising trend with $1/\epsilon$ can also be clearly observed.

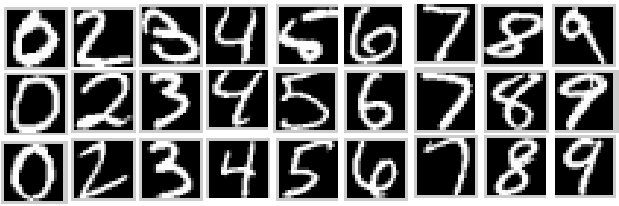
TABLE II
CSVDD RESULTS AT DIFFERENT VALUES OF ϵ (FACE DATABASE).

ϵ	size of tr set	AUC error	ϵ	size of tr set	AUC error
0.5	100	0.0526 (0.007)	0.2	100	0.0484 (0.004)
	300	0.0496 (0.007)		300	0.0490 (0.006)
	600	0.0518 (0.006)		600	0.0524 (0.006)
	1000	0.0492 (0.009)		1000	0.0492 (0.005)
	3000	0.0535 (0.006)		3000	0.0503 (0.006)
	6000	0.0510 (0.006)		6000	0.0490 (0.004)
	10000	0.0488(0.008)		10000	0.0484 (0.005)
	20000	0.0527 (0.008)		20000	0.0482 (0.004)
	30000	0.0496 (0.005)		30000	0.0477 (0.006)
0.4	100	0.0528 (0.007)	0.15	100	0.0496 (0.004)
	300	0.0510 (0.007)		300	0.0482 (0.004)
	600	0.0514 (0.007)		600	0.0495 (0.004)
	1000	0.0476 (0.005)		1000	0.0472 (0.005)
	3000	0.0510 (0.005)		3000	0.0483 (0.003)
	6000	0.0509 (0.005)		6000	0.0490 (0.005)
	10000	0.0539 (0.007)		10000	0.0483 (0.003)
	20000	0.0502 (0.008)		20000	0.0488 (0.005)
	30000	0.0518 (0.006)		30000	0.0460 (0.004)

B. MNIST Database

The second set of experiments is performed on the MNIST database, which contains 20×20 grey-level images of the digits $0, 1, \ldots, 9$. In this experiment, we treat the digit one images as normal patterns and all others as outliers. Sample images are shown in Figure 4. A total of 6,742 digit one images are used to form the training set, while the test set has 10,000 images, with 1,135 of them belonging to the digit one. Again, we use the same Gaussian kernel in (4), with $\sigma = 8$, and also ν is set to 0.9. On this database, the LIBSVM package cannot converge to the correct solution, and so only dd_tools and CSVDD can be compared.

(a) Digit 1 images.



(b) Outlier images.

Fig. 4. Sample images used in the experiment in Section IV-B.

Table III compares the novelty detection performance. It can be seen that CSVDD is even more accurate than dd_tools in terms of AUC error. Variations of the CPU time and size of the core-set with the number of training patterns are shown in Figures 5 and 6 respectively. Again, standard SVDD is faster than CSVDD when the data set is small. However, CSVDD becomes significantly faster on large data sets. With 3,000 or more training patterns, dd_tools cannot be run on the machine used in the experiment. Finally, other trends as discussed in Section IV-A can also be observed here.

V. CONCLUSION

In this paper, we proposed a scale-up algorithm for SVDD based on the idea of core-sets in computational geometry. Unlike the standard SVDD which has a runtime complexity of $O(N^3)$, where N is the number of training patterns, the proposed procedure has a running time which is only linear in N. Experimental results show that this allows SVDD to be

TABLE III

RESULTS ON THE MNIST DATA SET (NUMBERS IN BRACKETS ARE THE STANDARD DEVIATIONS).

size of tr set	method	AUC error	FN	FP
100	dd_tools	0.145 (0.057)	0.20 (0.12)	0.31 (0.22)
	CSVDD ($\epsilon = 0.3$)	0.033 (0.038)	0.15 (0.15)	0.28 (0.35)
	CSVDD ($\epsilon = 0.4$)	0.039 (0.074)	0.16 (0.17)	0.33 (0.41)
	CSVDD ($\epsilon = 0.5$)	0.035 (0.035)	0.15 (0.17)	0.31 (0.41)
300	dd_tools	0.162 (0.040)	0.13 (0.09)	0.42 (0.19)
	CSVDD ($\epsilon = 0.3$)	0.036 (0.041)	0.09 (0.10)	0.40 (0.42)
	CSVDD ($\epsilon = 0.4$)	0.054 (0.110)	0.11 (0.12)	0.41 (0.43)
	CSVDD ($\epsilon = 0.5$)	0.078 (0.148)	0.14 (0.13)	0.31 (0.41)
600	dd_tools	0.159 (0.030)	0.09 (0.06)	0.50 (0.19)
	CSVDD ($\epsilon = 0.3$)	0.126 (0.148)	0.09 (0.10)	0.49 (0.35)
	CSVDD ($\epsilon = 0.4$)	0.176 (0.219)	0.19 (0.23)	0.44 (0.33)
	CSVDD ($\epsilon = 0.5$)	0.175 (0.218)	0.13 (0.18)	0.54 (0.38)
1000	dd_tools	0.161 (0.020)	0.10 (0.05)	0.49 (0.14)
	CSVDD ($\epsilon = 0.3$)	0.084 (0.114)	0.10 (0.11)	0.37 (0.37)
	CSVDD ($\epsilon = 0.4$)	0.097 (0.145)	0.12 (0.15)	0.37 (0.36)
	CSVDD ($\epsilon = 0.5$)	0.119 (0.175)	0.12 (0.13)	0.39 (0.38)
2000	dd_tools	0.162 (0.020)	0.08 (0.04)	0.53 (0.15)
	CSVDD ($\epsilon = 0.3$)	0.053 (0.078)	0.07 (0.09)	0.28 (0.30)
	CSVDD ($\epsilon = 0.4$)	0.063 (0.096)	0.08 (0.09)	0.28 (0.31)
	CSVDD ($\epsilon = 0.5$)	0.086 (0.139)	0.10 (0.11)	0.27 (0.30)
3000	CSVDD ($\epsilon = 0.3$)	0.067 (0.104)	0.09 (0.11)	0.24 (0.26)
	CSVDD ($\epsilon = 0.4$)	0.091 (0.144)	0.10 (0.13)	0.24 (0.26)
	CSVDD ($\epsilon = 0.5$)	0.090 (0.131)	0.10 (0.09)	0.26 (0.31)
6000	CSVDD ($\epsilon = 0.3$)	0.067 (0.091)	0.09 (0.05)	0.18 (0.20)
	CSVDD ($\epsilon = 0.4$)	0.091 (0.140)	0.12 (0.12)	0.19 (0.20)
	CSVDD ($\epsilon = 0.5$)	0.106 (0.170)	0.15 (0.18)	0.18 (0.18)

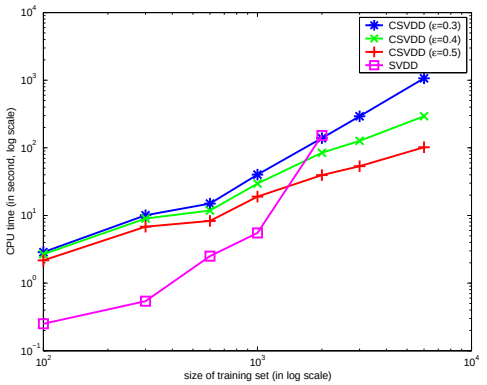


Fig. 5. CPU time vs number of training patterns (MNIST database).

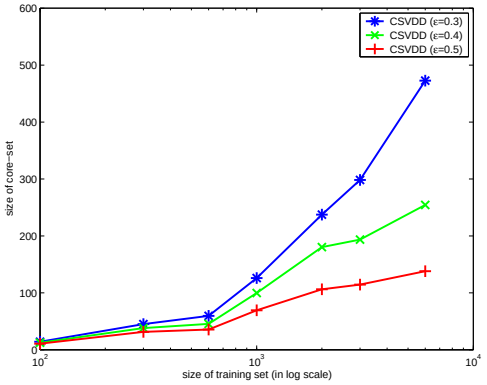


Fig. 6. Size of the core-set at different values of ϵ (MNIST database).

performed on much larger training sets with no degradation, or even improvement, in novelty detection performance.

Because of the close resemblance between SVDD and one-class SVM, we will also explore extending our method to one-class SVM in the future.

ACKNOWLEDGMENT

The authors would like to thank Wing Yu for providing the extended BioID Face database and Ka-Chun Wong for useful discussions. This research has been partially supported by the Research Grants Council of the Hong Kong Special Administrative Region under grant HKUST6195/02E.

REFERENCES

[1] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.
[2] B. Schölkopf and A. Smola, *Learning with Kernels*. MIT, 2002.
[3] V. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.
[4] M. Markou and S. Singh, "Novelty detection: A review, Part I: Statistical approaches," *Signal Processing*, vol. 83, no. 12, pp. 2481–2497, 2003.
[5] ——, "Novelty detection: A review, Part II: Neural network based approaches," *Signal Processing*, vol. 83, no. 12, pp. 2499–2521, 2003.
[6] S. Marsland, "Novelty detection in learning systems," *Neural Computing Surveys*, vol. 3, pp. 157–195, 2003.
[7] P. Hayton, B. Schölkopf, L. Tarassenko, and P. Anuzis, "Support vector novelty detection applied to jet engine vibration spectra," in *Advances in Neural Information Processing Systems 13*, T. Leen, T. Dietterich, and V. Tresp, Eds. Cambridge, MA: MIT Press, 2001.
[8] Y. Yang, J. Zhang, J. Carbonell, and C. Jin, "Topic-conditioned novelty detection," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
[9] D. Tax and R. Duin, "Support vector domain description," *Pattern Recognition Letters*, vol. 20, pp. 1991–1999, 1999.
[10] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," in *Advances in Neural Information Processing Systems 12*, S. Solla, T. Leen, and K.-R. Müller, Eds. San Mateo, CA: Morgan Kaufmann, 2000, pp. 582–588.
[11] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, July 2001.
[12] G. Lanckriet, L. El Ghaoui, and M. Jordan, "Robust novelty detection with single-class MPM," in *Advances in Neural Information Processing Systems 15*, S. Becker, S. Thrun, and K. Obermayer, Eds. Cambridge, MA: MIT Press, 2003.
[13] ——, "Robust novelty detection with single-class MPM," in *IMA Workshop on Semidefinite Programming and Robust Optimization*, 2003.
[14] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods – Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. Cambridge, MA: MIT Press, 1999, pp. 185–208.
[15] C. Campbell and K. Bennet, "A linear programming approach to novelty detection," in *Advances in Neural Information Processing Systems 14*, T. Dietterich, S. Becker, and Z. Ghahramani, Eds. Cambridge, MA: MIT Press, 2002.
[16] M. Badoiu, S. Har-Peled, and P. Indyk, "Approximate clustering via core-sets," in *Proceedings of 34th Annual ACM Symposium on Theory of Computing*, Montréal, Québec, Canada, 2002, pp. 250–257.
[17] M. Badoiu and K. Clarkson, "Smaller core-sets for balls," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Baltimore, Maryland, USA, 2002, pp. 801–802.
[18] P. Kumar, J. Mitchell, and A. Yildirim, "Computing core-sets and approximate smallest enclosing hyperspheres in high dimensions," in *5th Workshop on Algorithm Engineering and Experiments*, 2003.
[19] C.-C. Chang and C.-J. Lin, *LIBSVM: a Library for Support Vector Machines*, 2001.
[20] A. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.
[21] G. Medioni, M.-S. Lee, and C.-K. Tang, *A Computational Framework for Feature Extraction and Segmentation*. Elsevier Science, 2000.