# Anomaly Detection in Vertically Partitioned Data by Distributed Core Vector Machines

Marco Stolpe[1], Kanishka Bhaduri[2], Kamalika Das[3], and Katharina Morik[1]

[1] TU Dortmund, Computer Science, LS 8, 44221 Dortmund, Germany
[2] Netflix Inc., Los Gatos, CA 94032, USA
[3] UARC, NASA Ames, CA 94035, USA

**Abstract.** Observations of physical processes suffer from instrument malfunction and noise and demand data cleansing. However, rare events are not to be excluded from modeling, since they can be the most interesting findings. Often, sensors collect features at different sites, so that only a subset is present (vertically distributed data). Transferring all data or a sample to a single location is impossible in many real-world applications due to restricted bandwidth of communication. Finding interesting abnormalities thus requires efficient methods of distributed anomaly detection.

We propose a new algorithm for anomaly detection on vertically distributed data. It aggregates the data directly at the local storage nodes using RBF kernels. Only a fraction of the data are communicated to a central node. Through extensive empirical evaluation on controlled datasets, we demonstrate that our method is an order of magnitude more communication efficient than state of the art methods, achieving a comparable accuracy.

**Keywords:** 1-class learning, core vector machine, distributed features, communication efficiency

## 1 Introduction

Outlier or anomaly detection [8] refers to the task of identifying abnormal or inconsistent patterns in a dataset. It is a well studied problem in the literature of data mining, machine learning, and statistics. While outliers are, in general, deemed as undesirable entities, their identification and further analysis can be crucial to many tasks such as fraud and intrusion detection [7], climate pattern discovery in Earth sciences [29], quality control in manufacturing processes [15], and adverse event detection in aviation safety applications [11].

Large amounts of data are accumulated and stored in an entirely decentralized fashion. In cases where storage, bandwidth, or power limitations prohibit the transfer of all data to a central node for analysis, distributed algorithms are needed that are communication efficient, but nevertheless accurate. For example, the high amount of data in large scale applications such as Earth sciences makes it infeasible to store all the data at a central repository. Communication is one of the most expensive operations in wireless networks of battery-powered sensors [3] and mobile devices [6].

Research has focused on *horizontally partitioned data* [24], where each node stores a subset of all observations. In contrast, many applications such as the detection of outliers in spatio-temporal data have the observations *vertically partitioned*, *i.e.* each node stores different feature sets of the (same set of) observations. For instance, at the NASA's Distributed Active Archive Centers (DAAC), all precipitation data for all locations on the earth's surface can be at one data center while humidity observations for the same locations can be at another data center. Another example are oceanic water levels and weather conditions recorded by spatially distributed sensors over one day before a Tsunami. Only few communication efficient algorithms have been proposed for the vertically partitioned scenario. This task is particularly challenging, if the analysis depends on a combination of features from more than a single data location.

In this paper, we introduce a distributed method for 1-class learning which works in the vertically partitioned data scenario and has low communication costs. In particular, the contributions of this paper are:

- A new method for distributed 1-class learning on vertically partitioned data is proposed, the Vertically Distributed Core Vector Machine (VDCVM).
- It is theoretically proven and empirically demonstrated that the VDCVM can have an order of magnitude lower communication cost compared to a current state of the art method for distributed 1-class learning [10].
- The anomaly detection accuracy of VDCVM is systematically assessed on synthetic and real world datasets of varying difficulty. It is demonstrated that VDCVM provides comparable accuracy as the state of the art [10] while reducing communication cost.

The rest of this paper is organized as follows. In the next section (Section 2) we discuss some work related to the task of outlier detection. Relevant background material concerning traditional and distributed 1-class learning is discussed in Section 3. Details about the core algorithmic contribution of VDCVM are presented in Section 4. We demonstrate the performance of VDCVM in Section 5. Finally, we conclude the paper in Section 6.

## 2 Related Work

Several researchers have developed methods for parallelizing or distributing the optimization problem in SVMs. This is particularly useful when the datasets are large and the computation cannot be executed on a single machine. The cascade SVM by Graf *et al.* [14] uses a cascade of binary SVMs arranged in an inverted binary tree topology to train a global model. This method is guaranteed to reach a global optimum. In a different method, Chang *et al.* [9] present a parallel SVM formulation which reduces memory use through performing a row-based, approximate matrix factorization, and which loads only essential data to each machine to perform parallel computation. The solution to the optimization problem is achieved using a parallel interior point method (IPM) which computes the update rules in a distributed fashion. Hazan *et al.* [16] presents a method for parallel SVM learning based on the parallel Jacobi block update scheme derived from the convex conjugate Fenchel duality. Unfortunately, this method cannot guarantee optimality. In a related method, Flouri *et al.* [12] and Lu *et al.* [22] have proposed techniques in which the computation is done by the local nodes and then the central node performs aggregation of the results. In their method, SVMs are learned at each node independently and then the SVs are passed onto the other nodes for updating the models of the other nodes. This process has to be repeated for a few iterations to ensure convergence. Another interesting technique is the ADMM-based consensus SVM method proposed by Forero *et al.* [13] where the authors build a global SVM model in a sensor network without any central authority. The proposed algorithm is asynchronous in which messages are exchanged only among the neighboring nodes.

The aforementioned methods distribute the SVM problem, but do not focus on outlier detection. Those can be detected using *unsupervised*, *supervised*, or *semi-supervised* techniques [17][8]. In the field of distributed anomaly detection, researchers have mainly focused on the horizontally distributed scenario. In the PBay algorithm by Lozano and Acuna [21], a master node first splits the data into separate chunks for each processor. Then the master node loads each block of test data and broadcasts it to each of the worker nodes. Each worker node then executes a distance based outlier detection technique using its local database and the test block. Hung and Cheung [18] present a parallel version of the basic nested loop algorithm which is not suitable for distributed computation since it requires all the dataset to be exchanged among all the nodes. Otey *et al.* [24] present a distributed algorithm for mixed attribute datasets. Angiulli *et al.* [1] present a distributed distance-based outlier detection algorithm based on the concept of solving set which can be viewed as a compact representation of the original dataset. The solving set is such that by comparing any data point to only the elements of the solving set, it can be concluded if the point is an outlier or not. More recently, Bhaduri *et al.* [2] have developed a distributed method by using an efficient parallel pruning rule. For the vertically partitioned scenario, Brefeld *et al.* [4] use co-regularisation and block coordinate descent for least squares regression. Lee *et al.* [20] have proposed a separable primal formulation of the SVM training local SVM models and combining their predictions. While their algorithm can be extended to anomaly detection, their main focus is on supervised learning. A technique more focused on anomaly detection in the vertically partitioned scenario is proposed by Das *et al.* [10] for Earth science datasets. It trains local 1-class models and reduces communication by a pruning rule. This technique is used for comparisons and discussed in details in Sec. 3.3.

## 3 The Problem of 1-class Learning

The next sections introduce important notations by giving some background information on 1-class learning and the problem of vertically partitioned data.

### 3.1 Support Vector Data Description

The task of data description, or 1-class classification [23], is to find a model that well describes a training set of observations. The model can then be used to check whether new observations are similar or dissimilar to the previously seen data points and mark dissimilar points as anomalies or outliers. It has been shown by Tax and Duin [27] that instead of estimating the data distribution based on a training set, it is more efficient to compute a spherical boundary around the data. This method, called the Support Vector Data Description (SVDD), allows for choosing the diameter of an enclosing ball in order to control the volume of the training data that falls within the ball. Observations inside the ball are then classified as normal whereas those outside the ball are treated as outliers.

Given a vector space $X$ and a set $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq X$ of training instances, the primal problem is to find a minimum enclosing ball (MEB) with radius $R$ and center $\mathbf{c}$ around all data points $\mathbf{x}_i \in \mathcal{S}$:

$$\min_{R,\mathbf{c}} R^2 : ||\mathbf{c} - \mathbf{x}_i||^2 \leq R^2,\ i = 1, \ldots, n$$

When the input space is not a vector space or the decision boundary is non-spherical, observations may be mapped by $\varphi : X \rightarrow \mathcal{F}$ to a feature space $\mathcal{F}$ for which an inner product is defined. The explicit computation of this mapping to an (possibly) infinite dimensional space can be avoided by use of a kernel function $k : X \times X \rightarrow \mathbb{R}$, which computes the inner product in $\mathcal{F}$ between the input observations. The dual problem after the kernel transformation then becomes

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,j=1}^{n} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{1}$$
$$\text{s.t.} \quad \alpha_i \geq 0, \quad i = 1, \ldots, n, \quad \sum_{i=1}^{n} \alpha_i = 1$$

Here $\alpha = (\alpha_1, \ldots, \alpha_n)^T$ is a vector of Lagrange multipliers and the primal variables can be recovered using

$$\mathbf{c} = \sum_{i=1}^{n} \alpha_i \varphi(\mathbf{x}_i), \quad R = \sqrt{\alpha^T \text{diag}(\mathbf{K}) - \alpha^T \mathbf{K} \alpha}$$

where $\mathbf{K}_{n \times n} = (k(\mathbf{x}_i, \mathbf{x}_j))$ is the kernel matrix. After optimal $\alpha$s are found, the model consists of all data points for which $\alpha_i > 0$, called the support vectors (set $SV$), and the corresponding $\alpha$s. An observation $\mathbf{x}$ is said to belong to the training set distribution if its distance from the center $\mathbf{c}$ is smaller than the radius $R$, where the distance is expressed in terms of the support vectors and the kernel function as

$$||\mathbf{c} - \varphi(\mathbf{x})||^2 = k(\mathbf{x}, \mathbf{x}) - 2 \sum_{i=1}^{|SV|} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + \sum_{i,j=1}^{|SV|} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \leq R^2$$

It can be shown [28] that for kernels $k(\mathbf{x},\mathbf{x}) = \kappa$ ($\kappa$ constant), that map all input patterns to a sphere in feature space, (1) can be simplified to the optimization problem

$$\max_{\alpha} -\alpha^T \mathbf{K} \alpha : \alpha \geq \mathbf{0}, \alpha^T \mathbf{1} = 1 \tag{2}$$

where $\mathbf{0} = (0, \ldots, 0)^T$ and $\mathbf{1} = (1, \ldots, 1)^T$.

Whenever the kernel satisfies $k(\mathbf{x},\mathbf{x}) = \kappa$, any problem of the form (2) is an MEB problem. For example, Schölkopf [25] proposed the 1-class $\nu$-SVM that, instead of minimizing an enclosing ball, separates the normal data by a hyperplane with maximum margin from the origin in feature space. If $k(\mathbf{x},\mathbf{x}) = \kappa$, the optimization problems of the SVDD and the 1-class $\nu$-SVM with $C = 1/(\nu n)$ are equivalent and yield identical solutions.

### 3.2 Core Vector Machine (CVM)

Bădoiu and Clarkson [5] have shown that a $(1+\varepsilon)$-approximation of the MEB can be computed with constant time and space requirements. Their algorithm only depends on $\varepsilon$, but not on the dimension $m$ or the number of training examples $n$. Tsang *et al.* [28] have adopted this algorithm for kernel methods like the SVDD.

Let $\mathcal{S}$ be the training data as described in Section 3.1. For an $\varepsilon > 0$, the ball $B(\mathbf{c}, (1+\varepsilon)R)$ with center $\mathbf{c}$ and radius $R$ is an $(1+\varepsilon)$-approximation of the MEB($\mathcal{S}$), the minimum enclosing ball that contains all data points of $\mathcal{S}$. A subset $Q \subseteq \mathcal{S}$ is called the *core set* of $\mathcal{S}$ if the expansion of MEB($Q$) by the factor $(1+\varepsilon)$ contains $\mathcal{S}$.

The Core Vector Machine (CVM) algorithm shown in Figure 1 starts with an empty core set and extends it consecutively by the furthest point from the current center in feature space until all data is contained in an approximate MEB. The algorithm uses a modified kernel function $\tilde{k}$ for the reason that optimization problem (1) yields a hard margin solution, but can be transformed into a soft margin problem [19] by introducing a 2-norm error on the slack variables, *i.e.* by replacing $C\sum_{i=1}^{n} \xi_i$ with $C\sum_{i=1}^{n} \xi_i^2$, and replacing the original kernel function $k$ with a new kernel function $\tilde{k} : \tilde{\varphi} \to \tilde{\mathcal{F}}$, where

$$\tilde{k}(\mathbf{x}_i, \mathbf{x}_i) = k(\mathbf{x}_i, \mathbf{x}_j) + \frac{\delta_{ij}}{C}, \quad \delta_{ij} = \begin{cases} 1 : i = j \\ 0 : i \neq j \end{cases} \tag{3}$$

The new kernel again satisfies $\tilde{k}(\mathbf{z},\mathbf{z}) = \tilde{\kappa}$ with $\tilde{\kappa}$ being constant.

The furthest point calculation in step 2 takes $O(|\mathcal{S}_t|^2 + n|\mathcal{S}_t|)$ time for the $t^{th}$ iteration. However, as is mentioned by Schölkopf [26], the furthest point obtained from a randomly sampled subset $\mathcal{S}' \subset \mathcal{S}$ of size 59 already has a probability of 95% to be among the furthest 5% points in the whole dataset $\mathcal{S}$. By using this *probabilistic speed-up* strategy, *i.e.* determining the furthest point on a small sampled subset of points in each iteration, the running time for the furthest point calculation can be reduced to $O(|\mathcal{S}_t|^2)$. As shown by Tsang *et al.* [28], with probabilistic speed-up and a standard QP solver, the CVM reaches

**Core Vector Machine (CVM)**

$\mathcal{S}$: training set, consisting of $n$ examples
$\mathcal{S}_t \subseteq \mathcal{S}$: core set of $\mathcal{S}$ at iteration $t$
$\mathbf{c}_t$: center of the MEB around $\mathcal{S}_t$ in feature space
$R_t$: radius of the MEB

1. **Initialization:** Uniformly at random choose a point $\mathbf{z} \in \mathcal{S}$. Determine a point $\mathbf{z}_a \in \mathcal{S}$ that is furthest away from $\mathbf{z}$ in feature space, then a point $\mathbf{z}_b \in \mathcal{S}$ that is furthest away from $\mathbf{z}_a$. Set $\mathcal{S}_0 := \{\mathbf{z}_a, \mathbf{z}_b\}$ and the initial radius
$$R_0 := \frac{1}{2}\sqrt{2\tilde{\kappa} - 2\tilde{\kappa}(\mathbf{z}_a, \mathbf{z}_b)}$$
2. **Furthest point calculation:** Find $\mathbf{z} \in \mathcal{S}$ such that $\tilde{\phi}(\mathbf{z})$ is furthest away from $\mathbf{c}_t$. The new core set becomes $\mathcal{S}_{t+1} = \mathcal{S}_t \cup \{\mathbf{z}\}$. The squared distance of any point from the center in $\tilde{\mathcal{F}}$ can be calculated using the kernel function
$$||\mathbf{c}_t - \tilde{\phi}(\mathbf{z}_\ell)||^2 = \sum_{z_i, z_j \in \mathcal{S}_t} \alpha_i \alpha_j \tilde{k}(\mathbf{z}_i, \mathbf{z}_j) - 2 \sum_{z_i \in \mathcal{S}_t} \alpha_i \tilde{k}(\mathbf{z}_i, \mathbf{z}_\ell) + \tilde{k}(\mathbf{z}_\ell, \mathbf{z}_\ell)$$
3. **Termination check:** Terminate if all training points are inside the $(1+\varepsilon)$-ball $B(\mathbf{c}_t, (1+\varepsilon)R_t)$ in feature space, *i.e.* $||\mathbf{c}_t - \tilde{\phi}(\mathbf{z})|| \leq R_t(1+\varepsilon)$.
4. **MEB calculation:** Find a new MEB($\mathcal{S}_{t+1}$) by solving the QP problem
$$\max_{\alpha} -\alpha^T \tilde{\mathbf{K}} \alpha \; : \; \alpha \geq \mathbf{0}, \alpha^T \mathbf{1} = 1, \; \tilde{\mathbf{K}} = [\tilde{k}(\mathbf{z}_i, \mathbf{z}_j)]$$
on all points of the core set. Set $R_{t+1} := \sqrt{\tilde{\kappa} - \alpha^T \tilde{\mathbf{K}} \alpha}$.
5. $t := t + 1$, then go to step 2.

**Fig. 1.** Core Vector Machine (CVM) algorithm by Tsang *et al.* [28]

a $(1+\varepsilon)^2$-approximation of the MEB with high probability. The total number of iterations is bounded by $O(1/\varepsilon^2)$, the running time by $O(1/\varepsilon^8)$, and the space complexity by $O(1/\varepsilon^4)$. The running time and resulting core set size are thus constant and *independent* of the size of the whole dataset.

The CVM already seems to be better suited for a network setting than the 1-class $\nu$-SVM, as it works incrementally and could sample only as much data as needed from the storage nodes. However, it is no distributed algorithm. Section 4 discusses how the CVM can be turned into a distributed algorithm that is even more communication efficient.

### 3.3 Vertically Distributed 1-class Learning

In the vertically partitioned data scenario, each data site has all observations, but only a subset of the features. Let $P_0, ..., P_k$ be a set of nodes where $P_0$ is designated as the central node and the others are denoted as the data nodes. For the rest of the paper, it is assumed that all nodes can also be used for computations. Let the dataset at node $P_i$ ($\forall i > 0$) be denoted by $\mathcal{S}_i = [\mathbf{x}_1^{(i)} \; \ldots \; \mathbf{x}_n^{(i)}]^T$ consisting

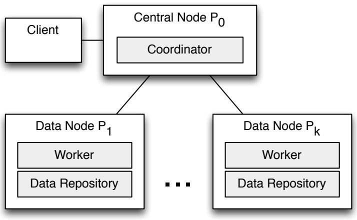

**Fig. 2.** Components of the VDCVM

of $n$ rows where $\mathbf{x}_j^{(i)} \in \mathbb{R}^{m_i}$ and $m_i$ is the number of variables in the $i^{th}$ site. Here, each row corresponds to an observation and each column corresponds to a variable (feature). There should be a one-to-one mapping between the rows across the different nodes. There exist crossmatching techniques that can be used to ensure that. The global set of features $A$ is the vertical concatenation of all the $m = \sum_{i=1}^{k} m_i$ features over all nodes and is defined as $A = [A_1\, A_2\, \ldots\, A_k]$ (using Matlab notation). Hence, the global data $\mathcal{S}$ is the $n \times m$ matrix defined as the union of all data over all nodes *i.e.* $\mathcal{S} = [\mathbf{x}_1\, \ldots\, \mathbf{x}_n]^T$ with $\mathbf{x}_j \in \mathbb{R}^m$. The challenge is to learn an accurate 1-class model without transferring all data to a central node.

Das *et al.* [10] have proposed a synchronized distributed anomaly detection algorithm (called VD**S**VM in this paper) for vertically partitioned data based on the 1-class $\nu$-SVM. At each data node $P_i$, a local 1-class model is trained. Points identified as local outliers are sent to the central node $P_0$, together with a small sample of all observations. At the central node, a global model is trained on the sample and used to decide if the outlier candidates sent from the data nodes are global outliers or not. The VDSVM cannot detect outliers which are global due to a combination of attributes. However, the algorithm shows good performance if global outliers are also local outliers. Moreover, in the application phase, the algorithm is highly communication efficient, since the number of outlier candidates is often only a small fraction of the data. A major drawback is that the fixed-size sampling approach gives no guarantees or bounds on the correctness of the global model. For a user, it is therefore difficult to set the sampling size correctly, in advance. Moreover, during training, no other strategies than sampling are used for reducing communication costs. We address these issues in this paper by developing a distributed version of the Core Vector Machine (VD**C**VM) which is more communication-efficient in the training phase and samples only as many points as needed, with known bounds for the correctness of the global model.

## 4 Vertically Distributed CVM (VDCVM)

In this section, we introduce the Vertically Distributed Core Vector Machine (VDCVM). It consists of the components shown in Figure 2. The *Coordinator* communicates with *Worker* components that reside locally on the data nodes and can access the values of the local feature subsets directly, without any network communication. While the termination check and the QP optimization (steps 3 and 4 of the CVM algorithm in Figure 1) are still done centrally by the *Coordinator*, the sampling and furthest point calculations are combined in a single step and done in parallel by the *Worker* components, as described in the next sections.

### 4.1 Distributed Furthest Point Calculation

In any iteration $t$, the original CVM algorithm (Figure 1) with probabilistic speedup draws a fixed-sized sample of data points from the whole dataset. Let $V_t$ denote the sample drawn at iteration $t$ and $|V_t|$ denote its size. From the sample, the CVM determines the point $\mathbf{z}_t$ furthest away from the current center $\mathbf{c}_t$ in feature space by calculating the squared distance $||\mathbf{c}_t - \tilde{\phi}(\mathbf{z}_\ell)||^2$ for each sample point $\mathbf{z}_\ell \in V_t$. Since $\tilde{k}(\mathbf{z}_\ell, \mathbf{z}_\ell) = \tilde{\kappa}$ is constant and the sum $\sum_{z_i, z_j \in \mathcal{S}_t} \alpha_i \alpha_j \tilde{k}(\mathbf{z}_i, \mathbf{z}_j)$ does not depend on the sampled points, the furthest point calculation at iteration $t$ can be simplified to

$$\mathbf{z}_t = \mathrm{argmax}_{\mathbf{z}_\ell \in V_t} \left[ - \sum_{z_j \in \mathcal{S}_t} \alpha_j \tilde{k}(\mathbf{z}_j, \mathbf{z}_\ell) \right] \tag{4}$$

Let $\mathbf{z}[i]$ denote the $i$th component of vector $\mathbf{z}$. With the linear dot product kernel $k(\mathbf{z}_i, \mathbf{z}_j) = \langle \mathbf{z}_i, \mathbf{z}_j \rangle$, the sum in (4) could be written as

$$\mathbf{z}_t = \mathrm{argmin}_{\mathbf{z}_\ell \in V_t} \sum_{z_j \in \mathcal{S}_t} \alpha_j \langle \mathbf{z}_j, \mathbf{z}_\ell \rangle = \mathrm{argmin}_{\mathbf{z}_\ell \in V_t} \sum_{i=1}^{m} \sum_{z_j \in \mathcal{S}_t} \alpha_j \mathbf{z}_j[i] \mathbf{z}_\ell[i] \tag{5}$$

Since the dot product kernel multiplies each component $i$ of the $\mathbf{z}_\ell$ and $\mathbf{z}_j$ vectors independently, we only need the index $i$ of the vector component and all $\alpha$ values for calculating the partial inner sums separately on each node

$$v_\ell^{(p)} = \sum_{i=1}^{m_p} \sum_{z_j^{(p)} \in \mathcal{S}_t} \alpha_j \, \mathbf{z}_j^{(p)}[i] \, \mathbf{z}_\ell^{(p)}[i] \tag{6}$$

over its subset of attributes $A_p$ for all random indices $\ell \in I_t$ and send these partial sums back to the coordinator. The coordinator then aggregates the sums and determines the index $\ell_{\mathrm{max}} \in I_t$ of the furthest point:

$$\ell_{\mathrm{max}} = \mathrm{argmin}_{\ell \in I_t} \sum_{p=1}^{k} v_\ell^{(p)} \tag{7}$$

Each data node thus only transmits a *single* numerical value for each point of the random sample, instead of sending *all* attribute values of the sampled points to the central node.

```
VDCVM Coordinator
on workerInitialized():
    if received message from all workers then
        Determine random index set $I_0$ for data points.
        Send getPartialSums( $I_0$ ) to all workers.
on getPartialSumsAnswer( $v_\ell^{(p)}\ \forall \ell \in I_t$ ):
    Store partial sums received from worker $p$.
    if received message from all workers then
        $\ell_{\max} = \operatorname{argmin}_{\ell \in I_t} \sum_{p=1}^{k} v_\ell^{(p)}$
        Send getData( $\{\ell\}$ ) to all repositories.
on getDataAnswer( $\{\mathbf{z}_t[1 \dots m_p]\}$ ):
    Store attribute values received from Data Repository $p$.
    if received message from all repositories then
        Construct furthest point $\mathbf{z}_t$ from attribute values.
        if $||\mathbf{c}_t - \phi(\mathbf{z}_t)|| \leq (1+\varepsilon) \cdot R_t$ then
            Broadcast stop and return model.
        else
            $\mathcal{S} := \mathcal{S} \cup \{\mathbf{z}_t\}$.
            Calculate new MEB($\mathcal{S}_{t+1}$). (Solve QP problem.)
            $R_{t+1} := \sqrt{\tilde{\kappa} - \alpha^T \tilde{\mathbf{K}} \alpha}$.
            Determine random index set $I_t$ for data points.
            Send getPartialSums( $I_t$, $\alpha s$ ) to workers.
            $t := t + 1$.
```

**Fig. 3.** Operations of the VDCVM *Coordinator*

Splitting (4) into partial sums can be done for the linear kernel, but it is impossible or at least non-trivial for non-linear mappings. For example, the SVDD is usually used with the RBF kernel $k(\mathbf{z}_i, \mathbf{z}_j) = e^{-\gamma||\mathbf{z}_i - \mathbf{z}_j||^2}$ and the CVM requires $k(\mathbf{x}, \mathbf{x})$ to be constant, which holds for the RBF kernel.One possible choice for a kernel is the summation of kernels defined on the local attributes only, like a combination of RBF kernels (see also Lee *et al.* [20]):

$$k(\mathbf{z}_i, \mathbf{z}_j) = \sum_{p=1}^{k} e^{-\gamma_p||\mathbf{z}_i^{(p)} - \mathbf{z}_j^{(p)}||^2} \qquad (8)$$

In Section 5 it is empirically shown that such a combination yields a similar accuracy as VD**S**VM on most of the tested datasets.

### 4.2 The VDCVM Algorithm

The *Coordinator* retrieves meta information attached to the datasets from all *Data Repository* components. As initial point $\mathbf{z}$, the algorithm takes the mean vector of the minimum and maximum attribute values. Thereby, $\mathbf{z}$ does not need to be sampled from the network. The constant $\tilde{\kappa}$ is calculated and all data

**VDCVM Worker**

```
on initializeWorker( parameters, $\mathbf{z}$, $\tilde{\kappa}$ ):
    Store $C$, $\gamma$, $\mathbf{z}$ and $\tilde{\kappa}$.
    Set $\mathcal{S} = \emptyset$, $t := 0$.
    Send workerInitialized to Coordinator.
on getPartialSums( $I_t$, $\alpha s$ ):
    if $t \geq 2$ then store new $\alpha$s.
    $\mathcal{S} := \mathcal{S} \cup \{\mathbf{z}_{\ell_{\max}}[1 \dots m_p]\}$.
    Calculate $v_\ell^{(p)}$ $\forall \ell \in I_t$ — see (6) and (7)
on stop():
    Free all resources.
```

**Fig. 4.** Operations of the VDCVM *Worker*

structures (the core set) are initialized. These are transmitted together with the parameters $C$ and $\gamma_1, \dots, \gamma_p$ to all *Worker* components.

The main part of the *Coordinator* is shown in Figure 3. The indices of $|V_t|$ random data points are sampled and sent to the workers in a request for the partial sums $v_\ell$. When the *Coordinator* has received all partial sums, it can calculate the index $\ell_{\max}$ of the furthest point $\mathbf{z}_t$ and ask the repositories for their feature values. If the termination criterion is not fulfilled, the coordinator goes on with solving the QP problem and calculates the new radius $R_{t+1}$. It then determines a new random index set $I_t$ and requests the next partial sums from the workers. It furthermore transmits all updated $\alpha$ values.

Based on the updated $\alpha$s, each *Worker* gets the local components of points $\mathbf{z}_t$ by its furthest index $\ell_{\max}$. It then calculates $v_\ell^{(p)}$ for all random indices $\ell \in I_t$ received from the *Coordinator*, according to Equation (7). The partial sums are then sent back to the *Coordinator* which continues with the main algorithm.

### 4.3 Analysis of Running Time and Communication Costs

The VDCVM performs exactly the same calculations as the original CVM algorithm. It therefore inherits all properties of the CVM, including the constant bound on the total number of iterations (see Section 3.2) and the $(1+\varepsilon)^2$-approximation guarantee for the calculated MEB.

Regarding communication costs, we assume that messages can be broadcast to all workers, that training point indices are represented by 4 bytes and real numbers by 8 bytes. The total number of bytes transferred (excluding initialization and message headers) when sending all $m$ attributes of $n$ points in a sample to a central server for training (as did the VD**S**VM) is

$$B_{\text{central}}(n) = n \cdot 4 + n \cdot m \cdot 8$$

**Table 1.** Numbers of iterations up to which VDCVM is more communication efficient than VD**S**VM, for different numbers of nodes $k$ and attributes $m$, $s = 59$.

| $m$ | $k$=1 | $k$=2 | $k$=5 | $k$=10 | $k$=25 |
|---|---|---|---|---|---|
| 10 | 1,042 | 924 | 570 | 0 | - |
| 25 | 2,782 | 2,664 | 2,310 | 1,720 | 0 |
| 50 | 5,682 | 5,564 | 5,210 | 4,620 | 2,850 |
| 100 | 11,482 | 11,364 | 11,010 | 10,420 | 8,650 |

In contrast, the bytes transferred by VD**C**VM up to iteration $T$ are

$$B_{\text{VDCVM}}(T) = [T \cdot s \cdot 4 + T \cdot s \cdot k \cdot 8] + [T \cdot 4 + T \cdot m \cdot 8] + \left[\frac{T(T+1)}{2} \cdot 8\right]$$

The coordinator at the central node first broadcasts $s$ index values to all data nodes and receives partial kernel sums for each, from $k$ workers (first term in brackets). Then, the index value of the furthest point is broadcast to all data nodes and the coordinator receives its $m$ feature values (second term). The total number of $\alpha$s transmitted is quadratic in the number of iterations (last term).

The break even point $T_{\text{worse}}$, *i.e.* the iteration from when on the VDCVM has worse communication costs than central sampling, can be calculated by setting $B_{\text{central}}(n)$ with $n = Ts$ equal to $B_{\text{VDCVM}}(T)$ and solving for $T$:

$$T_{\text{worse}} = 2 \cdot m \cdot (s-1) - 2 \cdot k \cdot s \tag{9}$$

According to (9), the communication efficiency of VDCVM depends on the number of attributes per node. Table 1 contains values of $T_{\text{worse}}$ for different numbers of data nodes $k$ and attributes $m$. The number of iterations occuring in practice is often much lower than those in Table 1 (cf. Section 5).

## 5 Experimental Evaluation

In this section we demonstrate the performance of VDCVM on a variety of datasets and compare it to VDSVM and a single central model. In 1-class learning, the ground truth about the outliers is often not available. For a systematic performance evaluation of the algorithms, synthetic data containing known outliers was therefore generated. In addition, the methods also have been evaluated on three real world datasets with known binary class labels.

**Synthetic Data** Figure 5 visualizes the generated datasets for two dimensions. The data was generated in a unit hypercube of $m$ dimensions (for $m = 2, 4, 8, 16, 32, 64$). The different types of data pose varying challenges to the algorithms when vertically partitioned among network nodes. The easiest scenario is the one in which each attribute reveals all information about the label, represented by `SepBox`. For `Gaussian`, the means $\mu_{+,-}$ and standard deviations $\sigma_{+,-}$ of two Gaussians were chosen randomly and independently for each attribute

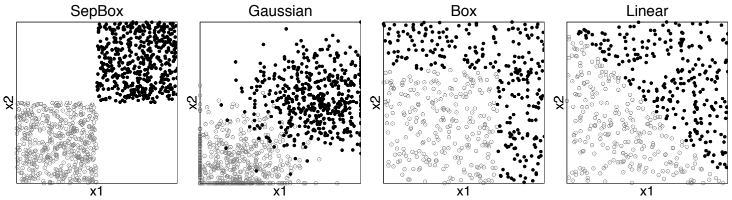

**Fig. 5.** Generated normal data (grey) and outliers (black) in two dimensions.

**Table 2.** Splitting of datasets into training, test and validation sets.

| Dataset | Total | Training | Test | | Validation | |
|---|---|---|---|---|---|---|
| | | | normal | outliers | normal | outliers |
| `random` | 60,000 | 20,000 | 10,000 | 10,000 | 10,000 | 10,000 |
| `letter` | 1,000 | 400 | 150 | 150 | 150 | 150 |
| `kddcup-99` | 60,000 | 20,000 | 10,000 | 10,000 | 10,000 | 10,000 |
| `face` | 20,000 | 10,000 | 2,500 | 2,500 | 2,500 | 2,500 |

(with $\mu_{+,-} \in [0.1, 0.9]$ and $\sigma_{+,-} \in [0, 0.25]$). If the Gaussians overlap in each single dimension, they may nevertheless become separable by a combination of attributes. In the `Box` dataset, an outlier is a point for which $\exists\, \mathbf{x}[i] > \rho$ with $\rho = 0.5^{(1/m)}$ (*i.e.* the normal data lies in half the volume of the $m$-dimensional unit hypercube). Separation is only given by all dimensions in conjunction. The same is true for the `Linear` dataset, where the normal data is separated from the outliers by the hyperplane $h = \{\mathbf{x} \,|\, \mathbf{x}/||m|| - 0.5||m|| = 0\}$.

**Real World Data** All real world data was taken from the CVM authors' web site[4]. The `letter` dataset consists of 20,000 data points for the 26 letters of the latin alphabet, each represented by 16 attributes. The KDDCUP-99 data consists of 5,209,460 examples of network traffic described by 127 features. The task is to differentiate between normal and bad traffic patterns. The extended MIT face dataset contains 513,455 images consisting of 19x19 (361) grey scale values. The task is to decide if the image contains a human face or not.

### 5.1 Experimental Setup

VDCVM was implemented in Java using the Spread Toolkit[5]. VDSVM was implemented in Python using LibSVM.

For the generated data, the central and local VDSVM models were trained on a fixed sample of data points and tuned and validated on independent test and validation sets. See Table 2 for the exact splitting (dataset `random`). VD-CVM was allowed to sample up to the same amount of training data. The $C$,

[4] `http://c2inet.sce.ntu.edu.sg/ivor/cvm.html`

[5] `http://www.spread.org`

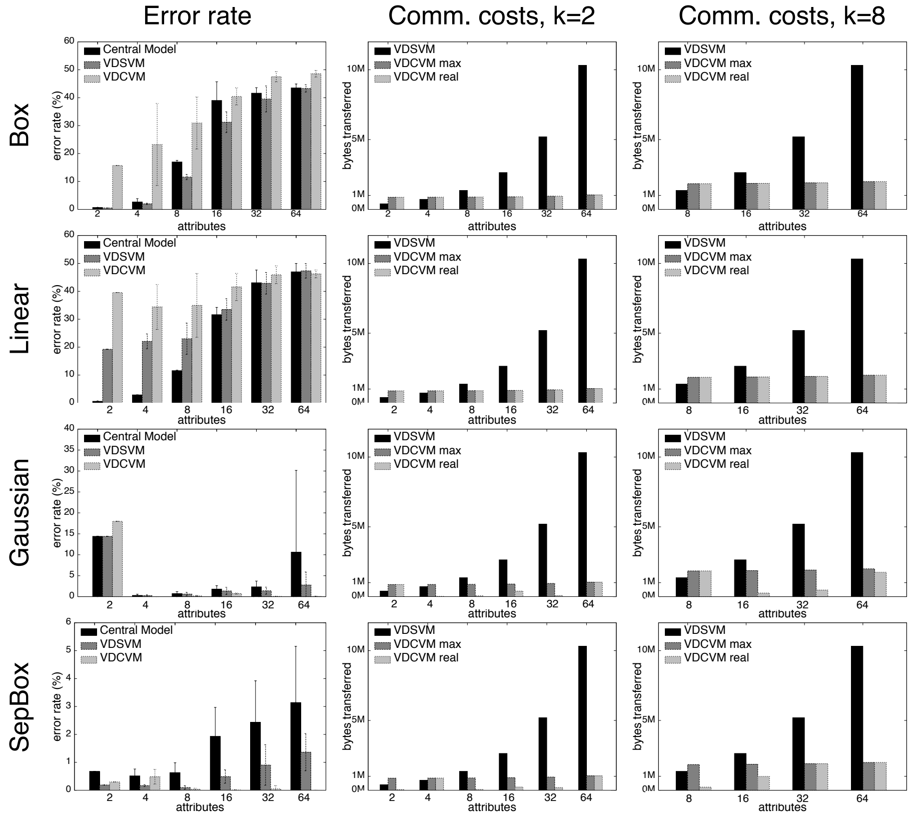

**Fig. 6.** Performance of VDCVM, VDSVM and a central model with standard RBF kernel on the generated datasets.

$\nu$ and $\gamma$ parameters were tuned on test sets. In call cases, for VDSVM, 75 random parameter combinations were tried. For VDCVM, 100 combinations were tried, alternatingly conducting a local and global random search. All values in Section 5.2 are the results of a single validation run with tuned parameters.

From the `letter` dataset, 773 examples for the letter `G` were taken as normal data and 796 for `T` extracted as outliers. For the KDDCUP-99 intrusion detection and the face recognition task, also only a subset of all data was taken (see column *Total* in Table 2).

### 5.2 Results

The plots in Figure 6 compare the performance of VDCVM to a single central 1-class model with standard RBF kernel and to VDSVM, *i.e.* local 1-class models which communicate only outlier candidates to the central model for testing. The error rates are averaged over the results obtained for different numbers of

**Table 3.** Results on real world datasets (*m*: attributes, *k*: nodes, *err*: error rate in %, *bytes*: amount of bytes transferred).

| Dataset | m | k | Central model | | VDSVM | | VDCVM | |
|---|---|---|---|---|---|---|---|---|
| | | | err | Kbytes | err | Kbytes | err | Kbytes |
| `letter` | 16 | 2 | 10.000 | 54 | 9.333 | 54 | 6.500 | **9** |
| | | 4 | 11.000 | 54 | 9.333 | 54 | 10.500 | **16** |
| `kddcup-99` | 127 | 2 | 0.285 | 20,401 | 0.220 | 20,401 | 0.000 | **1,206** |
| | | 4 | 0.450 | 20,401 | 0.290 | 20,401 | 0.002 | **1,526** |
| `face` | 361 | 2 | 6.220 | 29,006 | 7.900 | 29,006 | 4.940 | **808** |
| | | 4 | 5.580 | 29,006 | 6.880 | 29,006 | 5.100 | **969** |

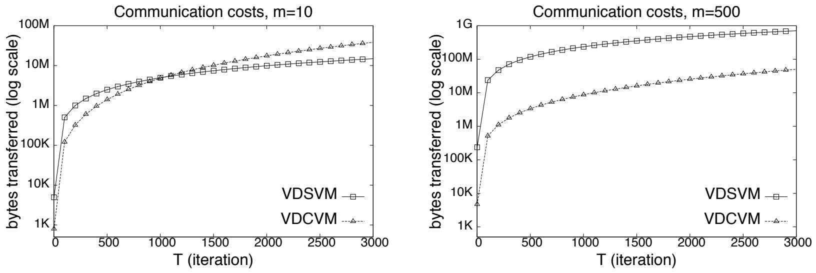

**Fig. 7.** Bytes transferred (log scale) by VDSVM and VDCVM with a growing number of iterations (T), for 10 (left) and 500 (right) attributes, $s = 59$, $k = 1$.

nodes (2, 4, 8, 16, 32). All methods, including the central 1-class model, have difficulties to separate the `Linear` and `Box` datasets in higher dimensions. In low dimensions, the combined RBF kernel has worse error rates than a standard RBF kernel and VDSVM's ensemble of local classifiers. VDCVM shows similar or even slightly *better performance on the* `Gaussian` *and* `SepBox` *datasets*, whose attribute values provide more information about the label locally. Even for the maximum number of 20,000 points allowed to sample ( *VDCVM max*), it already has much *lower communication costs* than VDSVM in most cases. When the data is easy to separate (`Gaussian` and `SepBox`), the real number of sampled points is often lower, resulting in *even less communication* (see *VDCVM real*). All methods achieve similar error rates on the real world datasets (see Table 3), with VDCVM being more *communication efficient*. The central 1-class model performing worse in some cases can be explained by VDSVM using an ensemble of (local and global) classifiers, increasing chances by at least one of them making the correct prediction. The better performance of VDCVM might be explained by better parameters found with the random tuning strategy.

The plots in Figure 7 show how the number of transmitted bytes grows with the number of iterations, for a fixed number of features. As shown in the left figure for 10 features, the crossover occurs at 1,000 iterations. The right figure plots the transmitted bytes for 500 features. Here, the VDCVM is at least an order of magnitude more communication efficient for all plotted iterations of the algorithm. In general, the more attributes are stored at each data node, the more can be saved in comparison to transmitting all data to the central node.

## 6 Conclusion

In this paper we have developed VDCVM – a distributed algorithm for anomaly detection from vertically distributed datasets. VDCVM is based on the recently proposed Core Vector Machine algorithm which uses a minimum enclosing ball formulation to describe the data. The algorithm solves local problems which are iteratively coordinated with the central server to compute the final solution. The proposed algorithm is as accurate as state of the art methods, but can have an order of magnitude less communication overhead in the training phase. Extensive experiments on datasets containing ground truth demonstrate the validity of the claims. In future work, we want to explore other combinations of local kernels and learning tasks such as distributed support vector clustering using VDCVM. Moreover, an implementation on embedded devices remains to be done, together with measuring real energy consumption.

## Acknowledgment

This work has been supported by the DFG, Collaborative Research Center SFB876 (`http://sfb876.tu-dortmund.de/`), project B3.

## References

1. Angiulli, F., Basta, S., Lodi, S., Sartori, C.: A Distributed Approach to Detect Outliers in Very Large Data Sets. In: Proc. of Euro-Par'10. pp. 329–340 (2010)
2. Bhaduri, K., Matthews, B.L., Giannella, C.: Algorithms for speeding up distance-based outlier detection. In: Proc. of KDD'11. pp. 859–867 (2011)
3. Bhaduri, K., Stolpe, M.: Distributed data mining in sensor networks. In: Aggarwal, C.C. (ed.) Managing and Mining Sensor Data. Springer, Berlin, Heidelberg (2013)
4. Brefeld, U., Gärtner, T., Scheffer, T., Wrobel, S.: Efficient co-regularised least squares regression. In: Proc. of the 23rd int. conf. on Machine Learning. pp. 137–144. ICML '06, ACM, New York, NY, USA (2006)
5. Bădoiu, M., Clarkson, K.: Optimal core sets for balls. In: DIMACS Workshop on Computational Geometry (2002)
6. Carroll, A., Heiser, G.: An analysis of power consumption in a smartphone. In: Proc. of the 2010 USENIX conf. on USENIX ann. technical conf. USENIXATC'10, USENIX Association, Berkeley, CA, USA (2010)
7. Chan, P., Fan, W., Prodromidis, A., Stolfo, S.: Distributed Data Mining in Credit Card Fraud Detection. IEEE Intelligent Systems 14, 67–74 (1999)
8. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Comp. Surveys 41(3), 1–58 (2009)
9. Chang, E.Y., Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z., Cui, H.: Psvm: Parallelizing support vector machines on distributed computers. In: NIPS (2007)

10. Das, K., Bhaduri, K., Votava, P.: Distributed anomaly detection using 1-class SVM for vertically partitioned data. Stat. Anal. Data Min. 4(4), 393–406 (2011)
11. Das, S., Matthews, B., Srivastava, A., Oza, N.: Multiple kernel learning for heterogeneous anomaly detection: algorithm and aviation safety case study. In: Proc. of KDD'10. pp. 47–56 (2010)
12. Flouri, K., Beferull-Lozano, B., Tsakalides, P.: Optimal gossip algorithm for distributed consensus svm training in wireless sensor networks. In: Proceedings of DSP'09. pp. 886–891 (2009)
13. Forero, P.A., Cano, A., Giannakis, G.B.: Consensus-based distributed support vector machines. J. Mach. Learn. Res. 99, 1663–1707 (2010)
14. Graf, H., Cosatto, E., Bottou, L., Durdanovic, I., Vapnik, V.: Parallel support vector machines: The cascade svm. In: NIPS (2004)
15. Harding, J., Shahbaz, M., Srinivas, Kusiak, A.: Data mining in manufacturing: A review. Manufacturing Science and Engineering 128(4), 969–976 (2006)
16. Hazan, T., Man, A., Shashua, A.: A parallel decomposition solver for svm: Distributed dual ascend using fenchel duality. In: CVPR 2008. pp. 1–8 (2008)
17. Hodge, V., Austin, J.: A survey of outlier detection methodologies. A. I. Review 22(2), 85–126 (2004)
18. Hung, E., Cheung, D.: Parallel Mining of Outliers in Large Database. Distrib. Parallel Databases 12, 5–26 (2002)
19. Keerthi, S., Shevade, S., Bhattacharyya, C., Murthy, K.: A fast iterative nearest point algorithm for support vector machine classifier design. IEEE Transactions on Neural Networks 11(1), 124–136 (2000)
20. Lee, S., Stolpe, M., Morik, K.: Separable approximate optimization of support vector machines for distributed sensing. In: Proc. of ECML/PKDD'12. LNCS, vol. 7524, pp. 387–402. Springer-Verlag, Berlin, Heidelberg (2012)
21. Lozano, E., Acuna, E.: Parallel algorithms for distance-based and density-based outliers. In: ICDM'05. pp. 729–732 (2005)
22. Lu, Y., Roychowdhury, V.P., Vandenberghe, L.: Distributed Parallel Support Vector Machines in Strongly Connected Networks. IEEE Transactions on Neural Networks 19(7), 1167–1178 (2008)
23. Moya, M., Koch, M., Hostetler, L.: One-class classifier networks for target recognition applications. In: Proc. World Congress on Neural Networks. pp. 797–801. International Neural Network Society (1993)
24. Otey, M., Ghoting, A., Parthasarathy, S.: Fast Distributed Outlier Detection in Mixed-Attribute Data Sets. Data Min. Knowl. Discov. 12, 203–228 (2006)
25. Schölkopf, B., Platt, J.C., Shawe-Taylor, J.C., Smola, A.J., Williamson, R.C.: Estimating the support of a high-dimensional distribution. Neural Comp. 13(7), 1443–1471 (2001)
26. Schölkopf, B., Smola, A.J.: Learning with Kernels. MIT Press (2002)
27. Tax, D.M.J., Duin, R.P.W.: Support vector data description. Mach. Learn. 54, 45–66 (2004)
28. Tsang, I., Kwok, J., Cheung, P.: Core Vector Machines: Fast SVM Training on Very Large Data Sets. J. Mach. Learn. Res. 6, 363–392 (December 2005)
29. Zhang, J., Roy, D., Devadiga, S., Zheng, M.: Anomaly detection in MODIS land products via time series analysis. Geo-Spat. Inf. Science 10, 44–50 (2007)