Fast SVM Training Using Approximate Extreme Points

Manu Nandan MNANDAN@UFL.EDU
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, FL 32611, USA

Pramod P. Khargonekar PPK@ECE.UFL.EDU
Department of Electrical and Computer Engineering
University of Florida,
Gainesville, FL 32611, USA

Sachin S. Talathi TALATHI@UFL.EDU
Department of Pediatrics, Division of Neurology
Department of Biomedical Engineering
Department of Neuroscience
University of Florida
Gainesville, FL 32611, USA

Editor:

Abstract

Applications of non-linear kernel Support Vector Machines (SVMs) to large datasets is seriously hampered by its excessive training time. We propose a modification, called the approximate extreme points support vector machine (AESVM), that is aimed at overcoming this burden. Our approach relies on conducting the SVM optimization over a carefully selected subset, called the representative set, of the training dataset. We present analytical results that indicate the similarity of AESVM and SVM solutions. A linear time algorithm based on convex hulls and extreme points is used to compute the representative set in kernel space. Extensive computational experiments on nine datasets compared AESVM to LIBSVM (**?**), CVM (**?**) , BVM (**?**), LASVM (**?**), SVM$^{\text{perf}}$ (**?**), and the random features method (**?**). Our AESVM implementation was found to train much faster than the other methods, while its classification accuracy was similar to that of LIBSVM in all cases. In particular, for a seizure detection dataset, AESVM training was almost 10^3 times faster than LIBSVM and LASVM and more than forty times faster than CVM and BVM. Additionally, AESVM also gave competitively fast classification times.

Keywords: support vector machines, convex hulls, large scale classification, non-linear kernels, extreme points

1. Introduction

Several real world applications require solutions of classification problems on large datasets. Even though SVMs are known to give excellent classification results, their application to problems with large datasets is impeded by the burdensome training time requirements. Recently, much progress has been made in the design of fast training algorithms (**??**) for SVMs with the linear kernel (linear SVMs). However, many applications require SVMs with non-linear kernels for accurate classification. Training time complexity for SVMs with non-linear kernels is typically quadratic in the size of the training dataset (**?**). The difficulty of the long training time is exacerbated when grid search with cross-validation is used to derive the optimal hyper-parameters, since this requires multiple SVM training runs. Another problem that sometimes restricts the applicability of SVMs is the long

classification time. The time complexity of SVM classification is linear in the number of support vectors and in some applications the number of support vectors is found to be very large (**?**).

In this paper, we propose a new approach for fast SVM training. Consider a two class dataset of N data vectors, $\mathbf{X} = \{\mathbf{x}_i : \mathbf{x}_i \in \mathbb{R}^D, i = 1, 2, ..., N\}$, and the corresponding target labels $\mathbf{Y} = \{y_i : y_i \in [-1, 1], i = 1, 2, ..., N\}$. The SVM primal problem can be represented as the following unconstrained optimization problem (**??**):

$$\min_{\mathbf{w},b} F_1(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{N}\sum_{i=1}^{N} l(\mathbf{w}, b, \phi(\mathbf{x}_i)) \tag{1}$$
$$\text{where } l(\mathbf{w}, b, \phi(\mathbf{x}_i)) = max\{0, 1 - y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b)\}, \forall \mathbf{x}_i \in \mathbf{X}$$
$$\text{and } \phi : \mathbb{R}^D \to \mathbb{H}, b \in \mathbb{R}, \text{ and } \mathbf{w} \in \mathbb{H}, \text{ a Hilbert space}$$

Here $l(\mathbf{w}, b, \phi(\mathbf{x}_i))$ is the hinge loss of $\mathbf{x}_i$. Note that SVM formulations where the penalty parameter C is divided by N have been used extensively (**???**). These formulations enable better analysis of the scaling of C with N (**?**). The problem in (**??**) requires optimization over N variables. In general, for SVM training algorithms the training time will reduce if the size of the training dataset is reduced.

*In this paper, we present an alternative to (**??**), called approximate extreme points support vector machines (AESVM), that requires optimization over only a subset of the training dataset.* The AESVM formulation is:

$$\min_{\mathbf{w},b} F_2(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{N}\sum_{t=1}^{M} \beta_t l(\mathbf{w}, b, \phi(\mathbf{x}_t)) \tag{2}$$
$$\text{where } \mathbf{x}_t \in \mathbf{X}^*, \mathbf{w} \in \mathbb{H}, \text{ and } b \in \mathbb{R}$$

Here M is the number of vectors in the selected subset of $\mathbf{X}$, called the representative set $\mathbf{X}^*$. The constants β_t are defined in (**??**). We will prove in Section **??** that:

- $F_1(\mathbf{w}_1^*, b_1^*) - F_2(\mathbf{w}_2^*, b_2^*) \leq C\sqrt{C\epsilon}$, where $(\mathbf{w}_1^*, b_1^*)$ and $(\mathbf{w}_2^*, b_2^*)$ are the solutions of (**??**) and (**??**) respectively
- Under the assumptions given in corollary 4, $F_1(\mathbf{w}_2^*, b_2^*) - F_1(\mathbf{w}_1^*, b_1^*) \leq 2C\sqrt{C\epsilon}$
- The AESVM problem minimizes an upper bound of a low rank Gram matrix approximation of the SVM objective function

Based on these results we claim that solving the problem in (**??**) yields a solution close to that of (**??**). As a by-product of the reduction in size of the training set, AESVM is also observed to result in fast classification. Considering that the representative set will have to be computed several times if grid search is used to find the optimum hyper-parameter combination, we also propose fast algorithms to compute $\mathbf{Z}^*$. In particular, we present an algorithm of time complexity $O(N)$ and an alternative algorithm of time complexity $O(N \log_2 \frac{N}{P})$ to compute $\mathbf{Z}^*$, where P is a predefined large integer.

The main contributions of this work can be summarized as follows:

- *Theoretical:* Theorems 1 and 2, and Corollaries 3 to 5 give rationale for the use of AESVM as a computationally less demanding alternative to the SVM formulation.
- *Algorithmic:* The algorithm DeriveRS, described in Section **??**, computes the representative set in linear time.
- *Experimental:* Our extensive experiments on nine datasets of varying characteristics, illustrate the suitability of applying AESVM to classification on large datasets.

This paper is organized as follows: in Section 2, we briefly discuss recent research on fast SVM training that is closely related to this work. Next, we provide the definition of the representative set and discuss properties of AESVM. In section 4, we present efficient algorithms to compute the representative set and analyze its computational complexity. Section 5 describes the results of our computational experiments. We compared AESVM to the widely used LIBSVM library, core vector machines (CVM), ball vector machines (BVM), LASVM, SVM$^{\text{perf}}$, and the random features method by **?**. Our experiments used eight publicly available datasets and a data set on EEG from an animal model of epilepsy (**??**). We conclude with a discussion of the results of this paper in Section 6.

2. Related Work

Several methods have been proposed to efficiently solve the SVM optimization problem. SVMs require special algorithms, as standard optimization algorithms such as interior point methods (**??**) have large memory and training time requirements that make it infeasible for large datasets. In the following sections we discuss the most widely used strategies to solve the SVM optimization problem. We present a comparison of some of these methods to AESVM in Section **??**. SVM solvers can be broadly divided into two categories as described below.

2.1 Dual optimization

The SVM primal problem is a convex optimization problem with strong duality (**?**). Hence its solution can be arrived at by solving its dual formulation given below:

$$\max_{\alpha} \; L_1(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{3}$$

$$\text{subject to } 0 \leq \alpha_i \leq \frac{C}{N} \text{ and } \sum_{i=1}^{N} \alpha_i y_i = 0$$

Here $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$, is the kernel product (**?**) of the data vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, and α is a vector of all variables α_i. Solving the dual problem is computationally simpler, especially for non-linear kernels and a majority of the SVM solvers use dual optimization. Some of the major dual optimization algorithms are discussed below.

Decomposition methods (**?**) have been widely used to solve (**??**). These methods optimize over a subset of the training dataset, called the 'working set', at each algorithm iteration. SVM^{light} (**?**) and SMO (**?**) are popular examples of decomposition methods. Both these methods have a quadratic time complexity for linear and non-linear SVM kernels (**?**). Heuristics such as shrinking and caching (**?**) enable fast convergence of decomposition methods and reduce their memory requirements. LIBSVM (**?**) is a very popular implementation of SMO. A *dual coordinate descent* (**?**) SVM solver computes the optimal α value by modifying one variable α_i per algorithm iteration. Dual coordinate descent SVM solvers, such as LIBLINEAR (**?**), have been proposed primarily for the linear kernel.

Approximations of the Gram matrix (**??**), have been proposed to increase training speed and reduce memory requirements of SVM solvers. The Gram matrix is the NxN square matrix composed of the kernel products $K(\mathbf{x}_i, \mathbf{x}_j)$, $\forall \mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$. *Training set selection* methods attempt to reduce the SVM training time by optimizing over a selected subset of the training set. Several distinct approaches have been used to select the subset. Some methods use clustering based approaches (**?**) to select the subsets. In **?**, hierarchical clustering is performed to derive a dataset that has more data vectors near the classification boundary than away from it. Minimum enclosing ball clustering is used in **?** to remove data vectors that are unlikely to contribute to the SVM training.

Random sampling of training data is another approach followed by approximate SVM solvers. **?** proposed reduced support vector machines (RSVM), in which only a random subset of the training

dataset is used. They solve a modified formulation of the L2-SVM that minimizes the l^2-norm of ξ instead of its l^1-norm. **?** proposed the LASVM algorithm that uses *active selection* techniques to train SVMs on a subset of the training dataset.

A *core set* (**?**) can be loosely defined as the subset of $\mathbf{X}$ for which the solution of an optimization problem such as (**??**) has a solution similar to that for the entire dataset $\mathbf{X}$. **?** proved that the L2-SVM is a reformulation of the minimum enclosing ball problem for some kernels. They proposed core vector machine (CVM) that approximately solves the L2-SVM formulation using core sets. A simplified version of CVM called ball vector machine (BVM) was proposed in **?**, where only an enclosing ball is computed. **?** proposed an algorithm to solve the L1-SVM problem, by computing the shortest distance between two polytopes (**?**) using core sets. However, there are no published results on solving L1-SVM with non-linear kernels using their algorithm.

Another method used to approximately solve the SVM problem is to map the data vectors into a *randomized feature space* that is relatively low dimensional compared to the kernel space $\mathbb{H}$ (**?**). Inner products of the projections of the data vectors are approximations of their kernel product. This effectively reduces the non-linear SVM problem into the simpler linear SVM problem, enabling the use of fast linear SVM solvers. This method is referred as RfeatSVM in the following sections of this document.

2.2 Primal optimization

In recent years, linear SVMs are finding increased use in applications with high-dimensional datasets. This has led to a surge in publications on efficient primal SVM solvers, which are mostly used for linear SVMs. To overcome the difficulties caused by the non-differentiability of the primal problem, the following methods are used.

Stochastic sub-gradient descent (**?**) uses the sub-gradient computed at some data vector $\mathbf{x}_i$ to iteratively update $\mathbf{w}$. **?** proposed a stochastic sub-gradient descent SVM solver, Pegasos, that is reported to be among the fastest linear SVM solvers. *Cutting plane algorithms* (**?**) solve the primal problem by successively tightening a piecewise linear approximation. It was employed by **?** to solve linear SVMs with their implementation SVM^{perf}. This work was generalized in **?** to include non-linear SVMs by approximately estimating $\mathbf{w}$ with arbitrary basis vectors using the fix-point iteration method (**?**). **?** proposed a related method for linear SVMs, that corrected some stability issues in the cutting plane methods.

3. Analysis of AESVM

As mentioned in the introduction, AESVM is an optimization problem on a subset of the training dataset called the representative set. In this section we first define the representative set. Then we present some properties of AESVM. These results are intended to provide theoretical justifications for the use of AESVM as an approximation to the SVM problem (**??**). We denote the cardinality of a set S by $|S|$.

3.1 Definition of the representative set

The convex hull of a set $\mathbf{X}$ is the smallest convex set containing $\mathbf{X}$ (**?**) and can be obtained by taking all possible convex combinations of elements of $\mathbf{X}$. Assuming $\mathbf{X}$ is finite, the convex hull is a polygon. The extreme points of $\mathbf{X}$, $EP(\mathbf{X})$, are defined to be the vertices of the convex polygon formed by the convex hull of $\mathbf{X}$. Any vector $\mathbf{x}_i$ in $\mathbf{X}$ can be represented as a convex combination of vectors in $EP(\mathbf{X})$:

$$\mathbf{x}_i = \sum_{\mathbf{x}_t \in EP(\mathbf{X})} \pi_t^i \mathbf{x}_t, \text{ where } 0 \leq \pi_t^i \leq 1, \text{ and } \sum_{\mathbf{x}_t \in EP(\mathbf{X})} \pi_t^i = 1$$

We can see that functions of any data vector in $\mathbf{X}$ can be computed using only $EP(\mathbf{X})$ and the convex combination weights $\{\pi_t^i\}$. The design of AESVM is motivated by the intuition that the use of extreme points may provide computational efficiency. However, extreme points are not useful in all cases, as for some kernels all data vectors are extreme points in kernel space. For example, for the Gaussian kernel, $K(\mathbf{x}_i, \mathbf{x}_i) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) = 1$. This implies that all the data vectors lie on the surface of the unit ball in the Gaussian kernel space and therefore are extreme points. Hence, we introduce the concept of *approximate extreme points*.

Consider the set of transformed data vectors:

$$\mathbf{Z} = \{\mathbf{z}_i : \mathbf{z}_i = \phi(\mathbf{x}_i), \forall \mathbf{x}_i \in \mathbf{X}\} \tag{4}$$

Here, the explicit representation of vectors in kernel space is only for the ease of understanding and all the computations are performed using kernel products. Let V be a positive integer that is much smaller than N and ϵ be a small positive real number. For notational simplicity, we assume N is divisible by V. Let $\mathbf{Z}_l$ be subsets of $\mathbf{Z}$ for $l = 1, 2, ..., (\frac{N}{V})$, such that $\mathbf{Z} = \underset{l}{\cup}\mathbf{Z}_l$ and $\mathbf{Z}_l \cap \mathbf{Z}_m = \emptyset$ for $l \neq m$, where $m = 1, 2, ..., (\frac{N}{V})$. We require that the subsets $\mathbf{Z}_l$ satisfy $|\mathbf{Z}_l| = V, \forall l$ and

$$\forall \mathbf{z}_i, \mathbf{z}_j \in \mathbf{Z}_l, \text{ we have } y_i = y_j \tag{5}$$

Let $\mathbf{Z}_l^q$ denote an arbitrary subset of $\mathbf{Z}_l$. Next, for any $\mathbf{z}_i \in \mathbf{Z}_l$ we define:

$$f(\mathbf{z}_i, \mathbf{Z}_l^q) = \underset{\mu^i}{\mathbf{min}} \|\mathbf{z}_i - \sum_{\mathbf{z}_t \in \mathbf{Z}_l^q} \mu_t^i \mathbf{z}_t\|^2 \tag{6}$$
$$\text{s.t. } 0 \leq \mu_t^i \leq 1, \text{ and } \sum_{\mathbf{z}_t \in \mathbf{Z}_l^q} \mu_t^i = 1$$

Consider the collection of subsets

$$\mathcal{Z}_\epsilon := \{\mathbf{Z}_l^q : \max_{\mathbf{z}_i \in \mathbf{Z}_l} f(\mathbf{z}_i, \mathbf{Z}_l^q) \leq \epsilon\}$$

A set of approximate extreme points of $\mathbf{Z}_l$ is denoted by $\mathbf{Z}_l^*$, and is defined as follows [1]

$$\mathbf{Z}_l^* \in \underset{\mathbf{Z}_l^q \in \mathcal{Z}_\epsilon}{\mathbf{argmin}} |\mathbf{Z}_l^q| \tag{7}$$

It can be seen that μ_t^i for $\mathbf{z}_t \in \mathbf{Z}_l^*$ are analogous to the convex combination weights π_t^i for $\mathbf{x}_t \in EP(\mathbf{X})$. The *representative set* $\mathbf{Z}^*$ of $\mathbf{Z}$ is the union of the sets of approximate extreme points of its subsets $\mathbf{Z}_l$.

$$\mathbf{Z}^* = \overset{\frac{N}{V}}{\underset{l=1}{\cup}} \mathbf{Z}_l^*$$

The representative set has properties that are similar to $EP(\mathbf{X})$. Given any $\mathbf{z}_i \in \mathbf{Z}$, we can find $\mathbf{Z}_l$ such that $\mathbf{z}_i \in \mathbf{Z}_l$. Let $\gamma_t^i = \{\mu_t^i$ for $\mathbf{z}_t \in \mathbf{Z}_l^*$ and $\mathbf{z}_i \in \mathbf{Z}_l$, and 0 otherwise$\}$. Now using (**??**), we can write:

$$\mathbf{z}_i = \sum_{\mathbf{z}_t \in \mathbf{Z}^*} \gamma_t^i \mathbf{z}_t + \tau_i \tag{8}$$

Here τ_i is a vector that accounts for the approximation error $f(\mathbf{z}_i, \mathbf{Z}_l^q)$ in (**??**). From (**??**)-(**??**) we can conclude that:

$$\|\tau_i\|^2 \leq \epsilon \; \forall \; \mathbf{z}_i \in \mathbf{Z} \tag{9}$$

1. The properties derived for AESVM in Section **??** are valid for any $\mathbf{Z}_l^q$. The requirement for the smallest $\mathbf{Z}_l^q$ is made only for the sake of a computationally simpler AESVM problem

Since ϵ will be set to a very small positive constant, we can infer that τ_i is a very small vector. The weights γ_t^i are used to define β_t in (**??**) as:

$$\beta_t = \sum_{i=1}^{N} \gamma_t^i \tag{10}$$

For ease of notation, we refer to the set $\mathbf{X}^* := \{\mathbf{x}_t : \mathbf{z}_t \in \mathbf{Z}^*\}$ as the representative set of $\mathbf{X}$ in the remainder of this paper. For the sake of simplicity, we assume that all $\gamma_t^i, \beta_t, \mathbf{X}$, and $\mathbf{X}^*$ are arranged so that $\mathbf{X}^*$ is positioned as the first M vectors of $\mathbf{X}$, where $M = |\mathbf{Z}^*|$.

3.2 Properties of AESVM

Consider the following optimization problem.

$$\min_{\mathbf{w},b} F_3(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{N}\sum_{i=1}^{N} l(\mathbf{w}, b, \mathbf{u}_i) \tag{11}$$

$$\text{where } \mathbf{u}_i = \sum_{t=1}^{M} \gamma_t^i \mathbf{z}_t, \mathbf{z}_t \in \mathbf{Z}^*, \mathbf{w} \in \mathbb{H}, \text{ and } b \in \mathbb{R}$$

We use the problem in (**??**) as an intermediary between (**??**) and (**??**). The intermediate problem (**??**) has a direct relation to the AESVM problem, as given in the following theorem. The properties of the *max* function given below are relevant to the following discussion:

$$max(0, A + B) \leq max(0, A) + max(0, B) \tag{12}$$

$$max(0, A - B) \geq max(0, A) - max(0, B) \tag{13}$$

$$\sum_{i=1}^{N} max(0, c^i A) = max(0, A) \sum_{i=1}^{N} c^i \tag{14}$$

for $A, B, c^i \in \mathbb{R}$ and $c^i \geq 0$.

Theorem 1 *Let $F_3(\mathbf{w}, b)$ and $F_2(\mathbf{w}, b)$ be as defined in (**??**) and (**??**) respectively. Then,*

$$F_3(\mathbf{w}, b) \leq F_2(\mathbf{w}, b) \ , \forall \mathbf{w} \in \mathbb{H} \text{ and } b \in \mathbb{R}$$

Proof Let $\mathcal{L}_2(\mathbf{w}, b, \mathbf{X}^*) = \frac{C}{N}\sum_{t=1}^{M} l(\mathbf{w}, b, \mathbf{z}_t)\sum_{i=1}^{N}\gamma_t^i$ and $\mathcal{L}_3(\mathbf{w}, b, \mathbf{X}^*) = \frac{C}{N}\sum_{i=1}^{N} l(\mathbf{w}, b, \mathbf{u}_i)$, where $\mathbf{u}_i = \sum_{t=1}^{M}\gamma_t^i \mathbf{z}_t$. From the properties of γ_t^i in (**??**), and from (**??**) we get:

$$\begin{aligned} \mathcal{L}_3(\mathbf{w}, b, \mathbf{X}^*) &= \frac{C}{N}\sum_{i=1}^{N} max\left[0, \left\{1 - y_i(\mathbf{w}^T \sum_{t=1}^{M}\gamma_t^i \mathbf{z}_t + b)\right\}\right] \\ &= \frac{C}{N}\sum_{i=1}^{N} max\left[0, \sum_{t=1}^{M}\gamma_t^i \left\{1 - y_t(\mathbf{w}^T \mathbf{z}_t + b)\right\}\right] \end{aligned} \tag{15}$$

Using properties (**??**) and (**??**) we get:

$$\begin{aligned} \mathcal{L}_3(\mathbf{w}, b, \mathbf{X}^*) &\leq \frac{C}{N}\sum_{i=1}^{N}\sum_{t=1}^{M} max\left[0, \gamma_t^i \left\{1 - y_t(\mathbf{w}^T \mathbf{z}_t + b)\right\}\right] \\ &= \frac{C}{N}\sum_{t=1}^{M} max\left[0, 1 - y_t(\mathbf{w}^T \mathbf{z}_t + b)\right] \sum_{i=1}^{N}\gamma_t^i \\ &= \mathcal{L}_2(\mathbf{w}, b, \mathbf{X}^*) \end{aligned}$$

Adding $\frac{1}{2}\|\mathbf{w}\|^2$ to both sides of the inequality above we get

$$F_3(\mathbf{w}, b) \leq F_2(\mathbf{w}, b)$$

■

The following theorem gives a relationship between the SVM problem and the intermediate problem.

Theorem 2 *Let $F_1(\mathbf{w}, b)$ and $F_3(\mathbf{w}, b)$ be as defined in (**??**) and (**??**) respectively. Then,*

$$-\frac{C}{N}\sum_{i=1}^{N} max\left\{0, y_i \mathbf{w}^T \tau_i\right\} \leq F_1(\mathbf{w}, b) - F_3(\mathbf{w}, b) \leq \frac{C}{N}\sum_{i=1}^{N} max\left\{0, -y_i \mathbf{w}^T \tau_i\right\}$$

*$\forall \mathbf{w} \in \mathbb{H}$ and $b \in \mathbb{R}$, where $\tau_i \in \mathbb{H}$ is the vector defined in (**??**).*

Proof Let $\mathcal{L}_1(\mathbf{w}, b, \mathbf{X}) = \frac{C}{N}\sum_{i=1}^{N} l(\mathbf{w}, b, \mathbf{z}_i)$, denote the average hinge loss that is minimized in (**??**) and $\mathcal{L}_3(\mathbf{w}, b, \mathbf{X}^*)$ be as defined in Theorem 1. Using (**??**) and (**??**) we get:

$$\begin{aligned}
\mathcal{L}_1(\mathbf{w}, b, \mathbf{X}) &= \frac{C}{N}\sum_{i=1}^{N} max\left\{0, 1 - y_i(\mathbf{w}^T \mathbf{z_i} + b)\right\} \\
&= \frac{C}{N}\sum_{i=1}^{N} max\left\{0, 1 - y_i(\mathbf{w}^T(\sum_{t=1}^{M}\gamma_t^i \mathbf{z}_t + \tau_i) + b)\right\}
\end{aligned}$$

From the properties of γ_t^i in (**??**), and from (**??**) we get:

$$\mathcal{L}_1(\mathbf{w}, b, \mathbf{X}) = \frac{C}{N}\sum_{i=1}^{N} max\left\{0, \sum_{t=1}^{M}\gamma_t^i(1 - y_t(\mathbf{w}^T \mathbf{z}_t + b)) - y_i \mathbf{w}^T \tau_i\right\} \tag{16}$$

Using (**??**) on (**??**), we get:

$$\begin{aligned}
\mathcal{L}_1(\mathbf{w}, b, \mathbf{X}) &\leq \frac{C}{N}\sum_{i=1}^{N} max\left[0, \sum_{t=1}^{M}\gamma_t^i\left\{1 - y_t(\mathbf{w}^T \mathbf{z}_t + b)\right\}\right] + \frac{C}{N}\sum_{i=1}^{N} max\left\{0, -y_i \mathbf{w}^T \tau_i\right\} \\
&= \mathcal{L}_3(\mathbf{w}, b, \mathbf{X}^*) + \frac{C}{N}\sum_{i=1}^{N} max\left\{0, -y_i \mathbf{w}^T \tau_i\right\}
\end{aligned}$$

Using (**??**) on (**??**), we get:

$$\begin{aligned}
\mathcal{L}_1(\mathbf{w}, b, \mathbf{X}) &\geq \frac{C}{N}\sum_{i=1}^{N} max\left[0, \sum_{t=1}^{M}\gamma_t^i\left\{1 - y_t(\mathbf{w}^T \mathbf{z}_t + b)\right\}\right] - \frac{C}{N}\sum_{i=1}^{N} max\left\{0, y_i \mathbf{w}^T \tau_i\right\} \\
&= \mathcal{L}_3(\mathbf{w}, b, \mathbf{X}^*) - \frac{C}{N}\sum_{i=1}^{N} max\left\{0, y_i \mathbf{w}^T \tau_i\right\}
\end{aligned}$$

From the two inequalities above we get,

$$\mathcal{L}_3(\mathbf{w}, b, \mathbf{X}^*) - \frac{C}{N}\sum_{i=1}^{N} max\left\{0, y_i \mathbf{w}^T \tau_i\right\} \leq \mathcal{L}_1(\mathbf{w}, b, \mathbf{X}) \leq \mathcal{L}_3(\mathbf{w}, b, \mathbf{X}^*) + \frac{C}{N}\sum_{i=1}^{N} max\left\{0, -y_i \mathbf{w}^T \tau_i\right\}$$

Adding $\frac{1}{2}\|\mathbf{w}\|^2$ to the inequality above we get

$$F_3(\mathbf{w},b) - \frac{C}{N}\sum_{i=1}^{N} max\left\{0, y_i\mathbf{w}^T\tau_i\right\} \leq F_1(\mathbf{w},b) \leq F_3(\mathbf{w},b) + \frac{C}{N}\sum_{i=1}^{N} max\left\{0, -y_i\mathbf{w}^T\tau_i\right\}$$

■

Using the above theorems we derive the following corollaries. These results provide the theoretical justification for AESVM.

Corollary 3 *Let* $(\mathbf{w}_1^*, b_1^*)$ *be the solution of (??) and* $(\mathbf{w}_2^*, b_2^*)$ *be the solution of (??). Then,*

$$F_1(\mathbf{w}_1^*, b_1^*) - F_2(\mathbf{w}_2^*, b_2^*) \leq C\sqrt{C\epsilon}$$

Proof It is known that $\|\mathbf{w}_1^*\| \leq \sqrt{C}$ (refer Theorem 1 in **?**). It is straight forward to see that the same result also applies to AESVM, $\|\mathbf{w}_2^*\| \leq \sqrt{C}$. Based on (**??**) we know that $\|\tau_i\| \leq \sqrt{\epsilon}$. From Theorem 2 we get:

$$\begin{aligned} F_1(\mathbf{w}_2^*, b_2^*) - F_3(\mathbf{w}_2^*, b_2^*) &\leq \frac{C}{N}\sum_{i=1}^{N} max\left\{0, -y_i\mathbf{w}_2^{*T}\tau_i\right\} \leq \frac{C}{N}\sum_{i=1}^{N}\|\mathbf{w}_2^*\|\|\tau_i\| \\ &\leq \frac{C}{N}\sum_{i=1}^{N}\sqrt{C\epsilon} = C\sqrt{C\epsilon} \end{aligned}$$

Since $(\mathbf{w}_1^*, b_1^*)$ is the solution of (**??**), $F_1(\mathbf{w}_1^*, b_1^*) \leq F_1(\mathbf{w}_2^*, b_2^*)$. Using this property and Theorem 1 in the inequality above, we get:

$$\begin{aligned} F_1(\mathbf{w}_1^*, b_1^*) - F_2(\mathbf{w}_2^*, b_2^*) &\leq F_1(\mathbf{w}_1^*, b_1^*) - F_3(\mathbf{w}_2^*, b_2^*) \\ &\leq F_1(\mathbf{w}_2^*, b_2^*) - F_3(\mathbf{w}_2^*, b_2^*) \leq C\sqrt{C\epsilon} \end{aligned} \tag{17}$$

■

Now we demonstrate some properties of AESVM using the dual problem formulations of AESVM and the intermediate problem. The dual form of AESVM is given by:

$$\begin{aligned} \max_{\hat{\alpha}} \; L_2(\hat{\alpha}) &= \sum_{t=1}^{M}\hat{\alpha}_t - \frac{1}{2}\sum_{t=1}^{M}\sum_{s=1}^{M}\hat{\alpha}_t\hat{\alpha}_s y_t y_s \mathbf{z}_t^T\mathbf{z}_s \\ \text{subject to } 0 \leq \hat{\alpha}_t &\leq \frac{C}{N}\sum_{i=1}^{N}\gamma_t^i \text{ and } \sum_{t=1}^{M}\hat{\alpha}_t y_t = 0 \end{aligned} \tag{18}$$

The dual form of the intermediate problem is given by:

$$\begin{aligned} \max_{\breve{\alpha}} \; L_3(\breve{\alpha}) &= \sum_{i=1}^{N}\breve{\alpha}_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\breve{\alpha}_i\breve{\alpha}_j y_i y_j \mathbf{u}_i^T\mathbf{u}_j \\ \text{subject to } 0 \leq \breve{\alpha}_i &\leq \frac{C}{N} \text{ and } \sum_{i=1}^{N}\breve{\alpha}_i y_i = 0 \end{aligned} \tag{19}$$

Consider the mapping function $h : \mathbb{R}^N \to \mathbb{R}^M$, defined as

$$h(\breve{\alpha}) = \{\tilde{\alpha}_t : \tilde{\alpha}_t = \sum_{i=1}^{N} \gamma_t^i \breve{\alpha}_i\} \tag{20}$$

It can be seen that the objective functions $L_2(h(\breve{\alpha}))$ and $L_3(\breve{\alpha})$ are identical.

$$\begin{aligned} L_2(h(\breve{\alpha})) &= \sum_{t=1}^{M} \tilde{\alpha}_t - \frac{1}{2}\sum_{t=1}^{M}\sum_{s=1}^{M} \tilde{\alpha}_t \tilde{\alpha}_s y_t y_s \mathbf{z}_t^T \mathbf{z}_s \\ &= \sum_{i=1}^{N} \breve{\alpha}_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \breve{\alpha}_i \breve{\alpha}_j y_i y_j \mathbf{u}_i^T \mathbf{u}_j \\ &= L_3(\breve{\alpha}) \end{aligned}$$

It is also straight forward to see that, for any feasible $\breve{\alpha}$ of (**??**), $h(\breve{\alpha})$ is a feasible point of (**??**) as it satisfies the constraints in (**??**). However, the converse is not always true. With that clarification, we present the following corollary.

Corollary 4 *Let $(\mathbf{w}_1^*, b_1^*)$ be the solution of (**??**) and $(\mathbf{w}_2^*, b_2^*)$ be the solution of (**??**). Let $\hat{\alpha}_2$ be the dual variable corresponding to $(\mathbf{w}_2^*, b_2^*)$. Let $h(\breve{\alpha}_2)$ be as defined in (**??**). If there exists an $\breve{\alpha}_2$ such that $h(\breve{\alpha}_2) = \hat{\alpha}_2$ and $\breve{\alpha}_2$ is a feasible point of (**??**), then,*

$$F_1(\mathbf{w}_2^*, b_2^*) - F_1(\mathbf{w}_1^*, b_1^*) \leq 2C\sqrt{C\epsilon}$$

Proof Let $(\mathbf{w}_3^*, b_3^*)$ be the solution of (**??**) and $\breve{\alpha}_3$ the solution of (**??**). We know that $L_3(\breve{\alpha}_2) = L_2(\hat{\alpha}_2) = F_2(\mathbf{w}_2^*, b_2^*)$ and $L_3(\breve{\alpha}_3) = F_3(\mathbf{w}_3^*, b_3^*)$. Since $L_3(\breve{\alpha}_3) \geq L_3(\breve{\alpha}_2)$, we get

$$F_3(\mathbf{w}_3^*, b_3^*) \geq F_2(\mathbf{w}_2^*, b_2^*)$$

But, from Theorem 1 we know $F_3(\mathbf{w}_3^*, b_3^*) \leq F_3(\mathbf{w}_2^*, b_2^*) \leq F_2(\mathbf{w}_2^*, b_2^*)$. Hence

$$F_3(\mathbf{w}_3^*, b_3^*) = F_3(\mathbf{w}_2^*, b_2^*)$$

From the above result we get

$$F_3(\mathbf{w}_2^*, b_2^*) - F_3(\mathbf{w}_1^*, b_1^*) \leq 0 \tag{21}$$

From Theorem 2 we have the following inequalities:

$$-\frac{C}{N}\sum_{i=1}^{N} max\left\{0, y_i \mathbf{w}_1^{*T} \tau_i\right\} \leq F_1(\mathbf{w}_1^*, b_1^*) - F_3(\mathbf{w}_1^*, b_1^*) \tag{22}$$

$$F_1(\mathbf{w}_2^*, b_2^*) - F_3(\mathbf{w}_2^*, b_2^*) \leq \frac{C}{N}\sum_{i=1}^{N} max\left\{0, -y_i \mathbf{w}_2^{*T} \tau_i\right\} \tag{23}$$

Adding (**??**) and (**??**) we get:

$$F_1(\mathbf{w}_2^*, b_2^*) - F_1(\mathbf{w}_1^*, b_1^*) \leq R + \frac{C}{N}\sum_{i=1}^{N}\left[max\left\{0, -y_i \mathbf{w}_2^{*T} \tau_i\right\} + max\left\{0, y_i \mathbf{w}_1^{*T} \tau_i\right\}\right] \tag{24}$$

where $R = F_3(\mathbf{w}_2^*, b_2^*) - F_3(\mathbf{w}_1^*, b_1^*)$. Using (**??**) and the properties $\|\mathbf{w}_2^*\| \leq \sqrt{C}$ and $\|\mathbf{w}_1^*\| \leq \sqrt{C}$ in (**??**):

$$\begin{aligned} F_1(\mathbf{w}_2^*, b_2^*) - F_1(\mathbf{w}_1^*, b_1^*) &\leq \frac{C}{N}\sum_{i=1}^{N}\left[max\left\{0, -y_i\mathbf{w}_2^{*T}\tau_i\right\} + max\left\{0, y_i\mathbf{w}_1^{*T}\tau_i\right\}\right] \\ &\leq \frac{C}{N}\sum_{i=1}^{N}\|\mathbf{w}_2^*\|\|\tau_i\| + \|\mathbf{w}_1^*\|\|\tau_i\| \\ &\leq \frac{C}{N}\sum_{i=1}^{N}2\sqrt{C\epsilon} = 2C\sqrt{C\epsilon} \end{aligned}$$

■

Now we prove a relationship between AESVM and the Gram matrix approximation methods mentioned in Section **??**.

Corollary 5 *Let $L_1(\alpha)$, $L_3(\breve{\alpha})$, and $F_2(\mathbf{w}, b)$ be the objective functions of the SVM dual (**??**), intermediate dual (**??**) and AESVM (**??**) respectively. Let $\mathbf{z}_i$, τ_i, and $\mathbf{u}_i$ be as defined in (**??**), (**??**), and (**??**) respectively. Let $\mathbf{G}$ and $\tilde{\mathbf{G}}$ be the NxN matrices with $\mathbf{G}_{ij} = \mathbf{y}_i\mathbf{y}_j\mathbf{z}_i^T\mathbf{z}_j$ and $\tilde{\mathbf{G}}_{ij} = \mathbf{y}_i\mathbf{y}_j\mathbf{u}_i^T\mathbf{u}_j$ respectively. Then for any feasible $\breve{\alpha}, \alpha, \mathbf{w}$, and b:*

1. *Rank of $\tilde{\mathbf{G}} = M, L_1(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\alpha\mathbf{G}\alpha^T, L_3(\breve{\alpha}) = \sum_{i=1}^{N}\breve{\alpha}_i - \frac{1}{2}\breve{\alpha}\tilde{\mathbf{G}}\breve{\alpha}^T$, and*

$$\text{Trace}(\mathbf{G} - \tilde{\mathbf{G}}) \leq N\epsilon + 2\sum_{t=1}^{M}\mathbf{z}_t^T\sum_{i=1}^{N}\gamma_t^i\tau_i$$

2. *$F_2(\mathbf{w}, b) \geq L_3(\breve{\alpha})$*

Proof Using $\mathbf{G}$, the SVM dual objective function $L_1(\alpha)$ can be represented as:

$$L_1(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\alpha\mathbf{G}\alpha^T$$

Similarly, $L_3(\breve{\alpha})$ can be represented using $\tilde{\mathbf{G}}$ as:

$$L_3(\breve{\alpha}) = \sum_{i=1}^{N}\breve{\alpha}_i - \frac{1}{2}\breve{\alpha}\tilde{\mathbf{G}}\breve{\alpha}^T$$

Applying $\mathbf{u}_i = \sum_{t=1}^{M}\gamma_t^i\mathbf{z}_t$, $\forall\mathbf{z}_t \in \mathbf{Z}^*$ to the definition of $\tilde{\mathbf{G}}$, we get:

$$\tilde{\mathbf{G}} = \Gamma\mathbf{A}\Gamma^T$$

Here $\mathbf{A}$ is the MxM matrix comprised of $\mathbf{A}_{ts} = \mathbf{y}_t\mathbf{y}_s\mathbf{z}_t^T\mathbf{z}_s$, $\forall\mathbf{z}_t, \mathbf{z}_s \in \mathbf{Z}^*$ and Γ is the NxM matrix with the elements $\Gamma_{it} = \gamma_t^i$. Hence the rank of $\tilde{\mathbf{G}} = M$ and intermediate dual problem (**??**) is a low rank approximation of the SVM dual problem (**??**).

The Gram matrix approximation error can be quantified using (**??**) and (**??**) as:

$$\begin{aligned} \text{Trace}(\mathbf{G} - \tilde{\mathbf{G}}) &= \sum_{i=1}^{N}\left[\mathbf{z}_i^T\mathbf{z}_i - (\sum_{t=1}^{M}\gamma_t^i\mathbf{z}_t)^T(\sum_{s=1}^{M}\gamma_s^i\mathbf{z}_s)\right] \\ &= \sum_{i=1}^{N}\left[\tau_i^T\tau_i + 2\sum_{t=1}^{M}\gamma_t^i\mathbf{z}_t^T\tau_i\right] \leq N\epsilon + 2\sum_{t=1}^{M}\mathbf{z}_t^T\sum_{i=1}^{N}\gamma_t^i\tau_i \end{aligned}$$

By the principle of duality, we know that $F_3(\mathbf{w}, b) \geq L_3(\breve{\alpha}),\ \forall \mathbf{w} \in \mathbb{H}$ and $b \in \mathbb{R}$, where $\breve{\alpha}$ is any feasible point of (**??**). Using Theorem 1 on the inequality above, we get

$$F_2(\mathbf{w}, b) \geq L_3(\breve{\alpha}),\ \forall \mathbf{w} \in \mathbb{H}, b \in \mathbb{R} \text{ and feasible } \breve{\alpha}$$

Thus the AESVM problem minimizes an upper bound ($F_2(\mathbf{w}, b)$) of a rank M Gram matrix approximation of $L_1(\alpha)$. ■

Based on the theoretical results in this section, it is reasonable to suggest that for small values of ϵ, the solution of AESVM is close to the solution of SVM.

4. Computation of the representative set

In this section, we present algorithms to compute the representative set. *The AESVM formulation can be solved with any standard SVM solver such as SMO and hence we do not discuss methods to solve it.* As described in Section **??**, we require an algorithm to compute approximate extreme points in kernel space. **?** proposed an algorithm to derive extreme points of the convex hull of a dataset in kernel space. Their algorithm is computationally intensive, with a time complexity of $O(N\ S(N))$, and is unsuitable for large datasets as $S(N)$ typically has a super-linear dependence on N. The function $S(N)$ denotes the time complexity of a SVM solver (required by their algorithm), to train on a dataset of size N. We next propose two algorithms leveraging the work by **?** to compute the representative set in kernel space $\mathbf{Z}^*$ with much smaller time complexities.

We followed the divide and conquer approach to develop our algorithms. The dataset is first divided into subsets $\mathbf{X}_q, q = 1, 2, .., Q$, where $|\mathbf{X}_q| < P$, $Q \geq \frac{N}{P}$ and $\mathbf{X} = \{\mathbf{X}_1, \mathbf{X}_2, .., \mathbf{X}_Q\}$. The parameter P is a predefined large integer. It is desired that each subset $\mathbf{X}_q$ contains data vectors that are more similar to each other than data vectors in other subsets. Our notion of similarity of data vectors in a subset, is that the distances between data vectors within a subset is less than the distances between data vectors in distinct subsets. This first level of segregation is followed by another level of segregation. We can regard the first level of segregation as coarse segregation and the second as fine segregation. Finally, the approximate extreme points of the subsets obtained after segregation, are computed. The two different algorithms to compute the representative set differ only in the first level of segregation as described in the following sections.

4.1 First level of segregation

We propose the methods, FLS1 and FLS2 given below to perform a first level of segregation. In the following description we use arrays Δ' and Δ'_2 of N elements. Each element of Δ' (Δ'_2), δ_i (δ_i^2) , contains the index in $\mathbf{X}$ of the last data vector of the subset to which $\mathbf{x}_i$ belongs. It is straight forward to replace this N element array with a smaller array of size equal to the number of subsets. We use a N element array for ease of description.

1. FLS1($\mathbf{X}', P$)

For some applications, such as anomaly detection on sequential data, data vectors are found to be homogeneous within intervals. For example, the atmospheric conditions typically do not change within a few minutes and hence weather data is homogeneous for a short span. For such datasets it is enough to segregate the data vectors based on its position in the training dataset. The same method can also be used on very large datasets without any homogeneity, in order to reduce computation time. The complexity of this method is $O(N')$, where $N' = |\mathbf{X}'|$.

2. FLS2($\mathbf{X}', P$)

When the dataset is not homogeneous within intervals or it is not excessively large we use the more sophisticated algorithm, FLS2, of time complexity $O(N'\ \log_2 \frac{N'}{P})$ given below. In step 1 of

$[\mathbf{X}', \Delta'] = \text{FLS1}(\mathbf{X}', P)$

1. For outerIndex = 1 ***to*** ceiling($\frac{|\mathbf{X}'|}{P}$)
2. For innerIndex = (outerIndex - 1)P ***to*** min((outerIndex)P,$|\mathbf{X}'|$)
3. Set $\delta_{innerIndex} = min((outerIndex)P, |\mathbf{X}'|)$

FLS2, the distance d_i in kernel space of all $\mathbf{x}_i \in \mathbf{X}'$ from $\mathbf{x}_j$ is computed as $d_i = \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2 = k(\mathbf{x}_i, \mathbf{x}_i) + k(\mathbf{x}_j, \mathbf{x}_j) - 2k(\mathbf{x}_i, \mathbf{x}_j)$. The algorithm FLS2($\mathbf{X}', P$), in effect builds a binary search tree, with each node containing the data vector $\mathbf{x}_k$ selected in step 2 that partitions a subset of the dataset into two. The size of the subsets successively halve, on downward traversal from the root of the tree to the other nodes. When the size of all the subsets at a level become $\leq P$ the algorithm halts. The complexity of FLS2 can be derived easily when the algorithm is considered as an incomplete binary search tree building method. The last level of such a tree will have $O(\frac{N'}{P})$ nodes and consequently the height of the tree is $O(\log_2 \frac{N'}{P})$. At each level of the tree the calls to the BFPRT algorithm (**?**) and the rearrangement of the data vectors in steps 2 and 3 are of $O(N')$ time complexity. Hence the overall time complexity of FLS2($\mathbf{X}', P$) is $O(N' \log_2 \frac{N'}{P})$.

$[\mathbf{X}', \Delta'] = \text{FLS2}(\mathbf{X}', P)$

1. Compute distance d_i in kernel space of all $\mathbf{x}_i \in \mathbf{X}'$ from the first vector $\mathbf{x}_j$ in $\mathbf{X}'$
2. Select $\mathbf{x}_k$ such that there exists $\frac{|\mathbf{X}'|}{2}$ data vectors $\mathbf{x}_i \in \mathbf{X}'$ with $d_i < d_k$, using the linear time BFPRT algorithm
3. Using $\mathbf{x}_k$, rearrange $\mathbf{X}'$ as $\mathbf{X}' = \{\mathbf{X}^1, \mathbf{X}^2\}$, where $\mathbf{X}^1 = \{\mathbf{x}_i : d_i < d_k, \mathbf{x}_i \in \mathbf{X}'\}$ and $\mathbf{X}^2 = \{\mathbf{x}_i : \mathbf{x}_i \in \mathbf{X}' \text{ and } \mathbf{x}_i \notin \mathbf{X}^1\}$
4. If $\frac{|\mathbf{X}'|}{2} \leq P$

   For i where $\mathbf{x}_i \in \mathbf{X}^1$, set δ_i = index of last data vector in $\mathbf{X}^1$.

   For i where $\mathbf{x}_i \in \mathbf{X}^2$, set δ_i = index of last data vector in $\mathbf{X}^2$.
5. If $\frac{|\mathbf{X}'|}{2} > P$

   Run FLS2($\mathbf{X}^1, P$) and FLS2($\mathbf{X}^2, P$)

4.2 Second level of segregation

After the initial segregation, another method SLS($\mathbf{X}', V, \Delta'$) is used to further segregate each set $\mathbf{X}_q$ into smaller subsets $\mathbf{X}_{q_r}$ of maximum size V, $\mathbf{X}_q = \{\mathbf{X}_{q_1}, \mathbf{X}_{q_2}, ..., \mathbf{X}_{q_R}\}$, where V is predefined ($V < P$) and $R = ceiling(\frac{|\mathbf{X_q}|}{V})$. The algorithm SLS($\mathbf{X}', V, \Delta'$) is given below. In step 2.b, $\mathbf{x}_t$ is the data vector in $\mathbf{X}_q$ that is farthest from the origin in the space of the data vectors. For some kernels, such as the Gaussian kernel, all data vectors are equidistant from the origin in kernel space. If the algorithm chooses $\mathbf{a}^l$ in step 2.b based on distances in such kernel spaces, the choice would be arbitrary and such a situation is avoided here. Each iteration of the For loop in step 2 involves several runs of the BFPRT algorithm, with each run followed by a rearrangement of $\mathbf{X}_q$. Specifically, the BFPRT algorithm is first run on P data vectors, then on $P - V$ data vectors, then on $P - 2V$ data vectors and so on. The time complexity of each iteration of the For loop including the BFPRT algorithm run and the rearrangement of data vectors is: $O(P + (P - V) + (P - 2V) + .. + V) \Rightarrow O(\frac{P^2}{V})$. The

overall complexity of SLS($\mathbf{X}', V, \Delta'$) considering the Q For loop iterations is $O(\frac{N'}{P}\frac{P^2}{V}) \Rightarrow O(\frac{N'P}{V})$, since $Q = O(\frac{N'}{P})$.

$[\mathbf{X}', \Delta_2'] = \text{SLS}(\mathbf{X}', V, \Delta')$

---

1. Initialize $l = 1$
2. For q = 1 ***to*** Q
   (a) Identify subset $\mathbf{X}_q$ of $\mathbf{X}'$ using Δ'
   (b) Set $\mathbf{a}^l = \phi(\mathbf{x}_t)$, where $\mathbf{x}_t \in \underset{i}{\mathbf{argmax}}\ \|\mathbf{x}_i\|^2, \mathbf{x}_i \in \mathbf{X}_q$
   (c) Compute distance d_i in kernel space of all $\mathbf{x}_i \in \mathbf{X}_q$ from $\mathbf{a}^l$
   (d) Select $\mathbf{x}_k$ such that, there exists V data vectors $\mathbf{x}_i \in \mathbf{X}_q$ with $d_i < d_k$, using the BFPRT algorithm
   (e) Using $\mathbf{x}_k$, rearrange $\mathbf{X}_q$ as $\mathbf{X}_q = \{\mathbf{X}^1, \mathbf{X}^2\}$, where $\mathbf{X}^1 = \{\mathbf{x}_i : d_i < d_k, \mathbf{x}_i \in \mathbf{X}_q\}$ and $\mathbf{X}^2 = \{\mathbf{x}_i : \mathbf{x}_i \in \mathbf{X}_q \text{ and } \mathbf{x}_i \notin \mathbf{X}^1\}$
   (f) For i where $\mathbf{x}_i \in \mathbf{X}^1$, set δ_i^2 = index of last data vector in $\mathbf{X}^1$, where δ_i^2 is the i^{th} element of Δ_2'
   (g) Remove $\mathbf{X}^1$ from $\mathbf{X}_q$
   (h) If $|\mathbf{X}^2| > V$
   Set: $l = l + 1$ and $\mathbf{a}^l = \mathbf{x}_k$
   Repeat steps 2.c to 2.h
   (i) If $|\mathbf{X}^2| \leq V$
   For i where $\mathbf{x}_i \in \mathbf{X}^2$, set δ_i^2 = index of last data vector in $\mathbf{X}^2$

4.3 Computation of the approximate extreme points

After computing the subsets $\mathbf{X}_{q_r}$, the algorithm DeriveAE is applied to each $\mathbf{X}_{q_r}$ to compute its approximate extreme points. The algorithm DeriveAE is described below. DeriveAE uses three routines. SphereSet($\mathbf{X}_{q_r}$) returns all $\mathbf{x}_i \in \mathbf{X}_{q_r}$ that lie on the surface of the smallest hypersphere in kernel space that contains $\mathbf{X}_{q_r}$. It computes the hypersphere as a hard margin support vector data descriptor (SVDD) (**?**). SphereSort($\mathbf{X}_{q_r}$) returns data vectors $\mathbf{x}_i \in \mathbf{X}_{q_r}$ sorted in descending order of distance in the kernel space from the center of the SVDD hypersphere. CheckPoint($\mathbf{x}_i, \Psi$) returns TRUE if $\mathbf{x}_i$ is an approximate extreme point of the set Ψ in kernel space. The operator $A \backslash B$ indicates a set operation that returns the set of the members of A excluding $A \cap B$. The matrix $\mathbf{X}_{q_r}^*$ contains the approximate extreme points of $\mathbf{X}_{q_r}$ and $\overline{\beta_{q_r}}$ is a $|\mathbf{X}_{q_r}^*|$ sized vector.

CheckPoint($\mathbf{x}_i, \Psi$) is computed by solving the following quadratic optimization problem:

$$\underset{\mu^i}{\mathbf{min}}\ p(\mathbf{x}_i, \Psi) = \|\phi(\mathbf{x}_i) - \sum_{t=1}^{|\Psi|} \mu_t^i \phi(\mathbf{x}_t)\|^2$$

$$\text{s.t. } \mathbf{x}_t \in \Psi, 0 \leq \mu_t^i \leq 1 \text{ and } \sum_{t=1}^{|\Psi|} \mu_t^i = 1$$

$[\mathbf{X}^*_{q_r}, \overline{\beta_{q_r}}] = \text{DeriveAE}(\mathbf{X}_{q_r})$

1. Initialize: $\mathbf{X}^*_{q_r} = \text{SphereSet}(\mathbf{X}_{q_r})$ and $\Psi = \emptyset$
2. Set $\zeta = \text{SphereSort}(\mathbf{X}_{q_r} \backslash \mathbf{X}^*_{q_r})$
3. For each $\mathbf{x}_i$ taken in order from ζ, call the routine CheckPoint$(\mathbf{x}_i, \mathbf{X}^*_{q_r} \cup \Psi)$
   If it returns $FALSE$, then set $\Psi = \Psi \cup \mathbf{x}_i$
4. For each $\mathbf{x}_i \in \Psi$, execute CheckPoint$(\mathbf{x}_i, \mathbf{X}^*_{q_r} \cup \{\Psi \backslash \mathbf{x}_i\})$
   If it returns $FALSE$, set $\mathbf{X}^*_{q_r} = \mathbf{X}^*_{q_r} \cup \mathbf{x}_i$
5. Initialize a matrix Γ of size $|\mathbf{X}_{q_r}|\text{x}|\mathbf{X}^*_{q_r}|$ with all elements set to 0
   Set $\mu^k_k = 1 \; \forall \mathbf{x}_k \in \mathbf{X}^*_{q_r}$, where μ^i_j is the element in the i^{th} row and j^{th} column of Γ
6. For each $\mathbf{x}_i \in \mathbf{X}_{q_r}$ and $\mathbf{x}_i \notin \mathbf{X}^*_{q_r}$, execute CheckPoint$(\mathbf{x}_i, \mathbf{X}^*_{q_r})$
   Set the i^{th} row of $\Gamma = \overline{\mu^i}$, where $\overline{\mu^i}$ is the result of CheckPoint$(\mathbf{x}_i, \mathbf{X}^*_{q_r})$
7. For j = 1 ***to*** $|\mathbf{X}^*_{q_r}|$
   Set $\beta^j_{q_r} = \sum_{k=1}^{|\mathbf{X}_{q_r}|} \mu^k_j$

where $\|\phi(\mathbf{x}_i) - \sum_{t=1}^{|\Psi|} \mu^i_t \phi(\mathbf{x}_t)\|^2 = K(\mathbf{x}_t, \mathbf{x}_t) + \sum_{t=1}^{|\Psi|} \sum_{s=1}^{|\Psi|} \mu^i_t \mu^i_s K(\mathbf{x}_t, \mathbf{x}_s) - 2 \sum_{t=1}^{|\Psi|} \mu^i_t K(\mathbf{x}_i, \mathbf{x}_t)$. If the optimized value of $p(\mathbf{x}_i, \Psi) \leq \epsilon$, CheckPoint$(\mathbf{x}_i, \Psi)$ returns TRUE and otherwise it returns FALSE. It can be seen that the formulation of $p(\mathbf{x}_i, \Psi)$ is similar to (**??**). The value of $\overline{\mu^i}$ computed by CheckPoint$(\mathbf{z}_i, \Psi_0)$, is used in step 6 of DeriveAE.

Now we compute the time complexity of DeriveAE. We use the fact that the optimization problem in CheckPoint$(\mathbf{x}_i, \Psi)$ is essentially the same as the dual optimization problem of SVM given in (**??**). Since DeriveAE solves several SVM training problems in steps 1,3,4, and 6, it is necessary to know the training time complexity of a SVM. As any SVM solver method can be used, we denote the training time complexity of each step of DeriveAE that solves an SVM problem as $O(S(A_{q_r}))$ [2]. Here A_{q_r} is the largest value of $\mathbf{X}^*_{q_r} \cup \Psi$ during the run of DeriveAE$(\mathbf{X}_{q_r})$. This enables us to derive a generic expression for the complexity of DeriveAE, independent of the SVM solver method used. Hence the time complexity of step 1 is $O(S(A_{q_r}))$. The time complexity of steps 3, 4 and 6 are $O(V\ S(A_{q_r}))$, $O(A_{q_r}\ S(A_{q_r}))$, and $O(A_{q_r}\ S(A_{q_r}))$ respectively. The time complexity of step 2 is $O(V\ |\Psi_1| + V\ \log_2 V)$, where $\Psi_1 = \text{SphereSet}(\mathbf{X}_{q_r})$. Hence the time complexity of DeriveAE is $O(V\ |\Psi| + V\ \log_2 V + V\ S(A_{q_r}) + A_{q_r}\ S(A_{q_r}))$. Since $|\Psi_1|$ is typically very small and $A_{q_r} \leq V$, we denote the time complexity of DeriveAE by $O(V\ \log_2 V + V\ S(A_{q_r}))$.

4.4 Combining all the methods to compute X^*

To derive X^*, it is required to first rearrange $\mathbf{X}$, so that data vectors from each class are grouped together as $\mathbf{X} = \{\mathbf{X}^+, \mathbf{X}^-\}$. Here $\mathbf{X}^+ = \{\mathbf{x}_i : y_i = 1, \mathbf{x}_i \in \mathbf{X}\}$ and $\mathbf{X}^- = \{\mathbf{x}_i : y_i = -1, \mathbf{x}_i \in \mathbf{X}\}$. Then the selected segregation methods are run on $\mathbf{X}^+$ and $\mathbf{X}^-$ separately. The algorithm DeriveRS given below, combines all the algorithms defined earlier in this section with a few additional steps, to compute the representative set of $\mathbf{X}$. The complexity of DeriveRS [3] can easily be computed by summing the complexities of its steps. The complexity of steps 1 and 6 is O(N). The complexity

2. For SMO based implementations, such as the implementation we used for Section **??**, $S(A) = O(A^2)$
3. We present DeriveRS as one algorithm in spite of its two variants that use FLS1 or FLS2, for simplicity and to conserve space.

of step 2 is $O(N)$ if FLS1 is run or $O(N \log_2 \frac{N}{P})$ if FLS2 is run. In step 3, the $O(\frac{NP}{V})$ method SLS is run. In steps 4 and 5, DeriveAE is run on all the subsets $\mathbf{X}_{q_r}$ giving a total complexity of $O(N \log_2 V + V \sum_{q=1}^{Q} \sum_{r=1}^{R} S(A_{q_r}))$. Here we use the fact that the number of subsets $\mathbf{X}_{q_r}$ is $O(\frac{N}{V})$. Thus the complexity of DeriveRS is $O(N(\frac{P}{V} + \log_2 V) + V \sum_{q=1}^{Q} \sum_{r=1}^{R} S(A_{q_r}))$ when FLS1 is used and $O(N(\log_2 \frac{N}{P} + \frac{P}{V} + \log_2 V) + V \sum_{q=1}^{Q} \sum_{r=1}^{R} S(A_{q_r}))$ when FLS2 is used.

$[\mathbf{X}^*, \mathbf{Y}^*, \overline{\beta}]$ = DeriveRS($\mathbf{X}$,$\mathbf{Y}$,P,V)

---

1. Set $\mathbf{X}^+ = \{\mathbf{x}_i : \mathbf{x}_i \in \mathbf{X}, y_i = 1\}$ and $\mathbf{X}^- = \{\mathbf{x}_i : \mathbf{x}_i \in \mathbf{X}, y_i = -1\}$
2. Run $[\mathbf{X}^+, \Delta^+]$ = FLS($\mathbf{X}^+$,P) and $[\mathbf{X}^-, \Delta^-]$ = FLS($\mathbf{X}^-$,P), where FLS is FLS1 or FLS2
3. Run $[\mathbf{X}^+, \Delta_2^+]$ = SLS($\mathbf{X}^+$,V,Δ^+) and $[\mathbf{X}^-, \Delta_2^-]$ = SLS($\mathbf{X}^-$,V,Δ^-)
4. Using Δ_2^+, identify each subset $\mathbf{X}_{q_r}$ of $\mathbf{X}^+$ and run $[\mathbf{X}^*_{q_r}, \overline{\beta_{q_r}}]$ = DeriveAE($\mathbf{X}_{q_r}$)
   Set N^{+*} = sum of number of data vectors in all $\mathbf{X}^*_{q_r}$ derived from $\mathbf{X}^+$
5. Using Δ_2^-, identify each subset $\mathbf{X}_{q_r}$ of $\mathbf{X}^-$ and run $[\mathbf{X}^*_{q_r}, \overline{\beta_{q_r}}]$ = DeriveAE($\mathbf{X}_{q_r}$)
   Set N^{-*} = sum of number of data vectors in all $\mathbf{X}^*_{q_r}$ derived from $\mathbf{X}^-$
6. Combine in the same order, all $\mathbf{X}^*_{q_r}$ to obtain $\mathbf{X}^*$ and all $\overline{\beta_{q_r}}$ to obtain $\overline{\beta}$
   Set $\mathbf{Y}^* = \{y_i : y_i = 1$ for $i = 1, 2, .., N^{+*}$; and $y_i = -1$ for $i = 1 + N^{+*}, 2 + N^{+*}, .., N^{-*} + N^{+*}\}$

---

5. Experiments

We focused our experiments on an SMO (**?**) based implementation of AESVM and DeriveRS. We evaluated the classification performance of AESVM using the nine datasets, described below. Next, we present an evaluation of the algorithm DeriveRS, followed by an evaluation of AESVM.

5.1 Datasets

Nine datasets of varied size, dimensionality and density were used to evaluate DeriveRS and our AESVM implementation. For datasets D2, D3 and D4, we performed five fold cross validation. We did not perform five fold cross-validation on the other datasets, because they have been widely used in their native form with a separate training and testing set.

D1: *KDD'99 intrusion detection dataset*[4]- This dataset is available as a training set of 4898431 data vectors and a testing set of 311027 data vectors, with forty one features ($D = 41$). As described in **?**, a huge portion of this dataset is comprised of repeated data vectors. Experiments were conducted only on the distinct data vectors. The number of distinct training set vectors was $N = 1074974$ and the number of distinct testing set vectors was $N = 77216$. The training set density = 33%.

4. `http://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data`

D2: *Localization data for person activity*[5] - This dataset has been used in a study on agent-based care for independent living (**?**). It has $N = 164860$ data vectors of seven features. It is comprised of continuous recordings from sensors attached to five people and can be used to predict the activity that was performed by each person at the time of data collection. In our experiments we used this dataset to validate a binary problem of classifying the activities 'lying' and 'lying down' from the other activities. Features 3 and 4, that gives the time information, were not used in our experiments. Hence for this dataset $D = 5$. The dataset density = 96%.

D3: *Seizure detection dataset*- This dataset has $N = 982863$ data vectors, three features ($D = 3$) and density = 100%. It is comprised of continuous EEG recordings from rats induced with status epilepticus and is used to evaluate algorithms that classify seizure events from seizure-free EEG. An important characteristic of this dataset is that it is highly unbalanced, the total number of data vectors corresponding to seizures is minuscule compared to the remaining data. Details of the dataset can be found in **?**, where it is used as dataset A.

D4: *Forest cover type dataset*[6]- This dataset has $N = 581012$ data vectors and fifty four features ($D = 54$) and density = 22%. It is used to classify the forest cover of areas of 30mx30m size into one of seven types. We followed the method used in **?**, where a classification of forest cover type 2 from the other cover types was performed.

D5 : *IJCNN1 dataset* [7]- This dataset was used in IJCNN 2001 generalization ability challenge (**?**). The training set and testing set have 49990 ($N = 49990$) and 91701 data vectors respectively. It has 22 features ($D = 22$) and training set density = 59%

D6 : *Adult income dataset* [8]- This dataset derived from the 1994 Census database, was used to classify incomes over $50000 from those below it. The training set has $N = 32561$ with $D = 123$ and density = 11%, while the testing set has 16281 data vectors. The data is pre-processed as described in **?**.

D7 : *Epsilon dataset* [9]- This is a dataset that was used for 2008 Pascal large scale learning challenge and in **?**. It is comprised of 400000 data vectors that are 100% dense with $D = 2000$. Since this is too large for our experiments, we used the first 10% of the training set giving $N = 40000$. The testing set has 100000 data vectors.

D8 : *MNIST character recognition dataset* [10]- The widely used dataset (**?**) of hand written characters has a training set of $N = 60000$, $D = 780$ and density = 19%. We performed the binary classification task of classifying the character '0' from the others. The testing set has 10000 data vectors.

D9 : *w8a dataset* [11]- This artificial dataset used in **?** was randomly generated and has $D = 300$ features. The training set has $N = 49749$ with a density = 4% and the testing set has 14951 data vectors.

5.2 Evaluation of DeriveRS

We began our experiments with an evaluation of the algorithm DeriveRS, described in Section **??**. The performances of the two methods FLS1 and FLS2 were compared first. We ran DeriveRS on D1, D2, D4 and D5 with the parameters $P = 10^4, V = 10^3, \epsilon = 10^{-3}$, and $g = [2^{-4}, 2^{-3}, 2^{-2}, ..., 2^2]$,

5. http://archive.ics.uci.edu/ml/datasets/Localization+Data+for+Person+Activity
6. http://archive.ics.uci.edu/ml/datasets/Covertype
7. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html
8. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html
9. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html
10. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html
11. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html

first with FLS1 and then FLS2. For D2, DeriveRS was run on the entire dataset for this particular experiment, instead of performing five fold cross-validation. This was done because, D2 is a small dataset and the difference between the two first level segregation methods can be better observed when the dataset is as large as possible. The relatively small value of $P = 10^4$ was also chosen considering the small size of D2 and D5. To evaluate the effectiveness of FLS1 and FLS2, we also ran DeriveRS with FLS1 and FLS2 after randomly reordering each dataset. The results are shown in Figure ??.

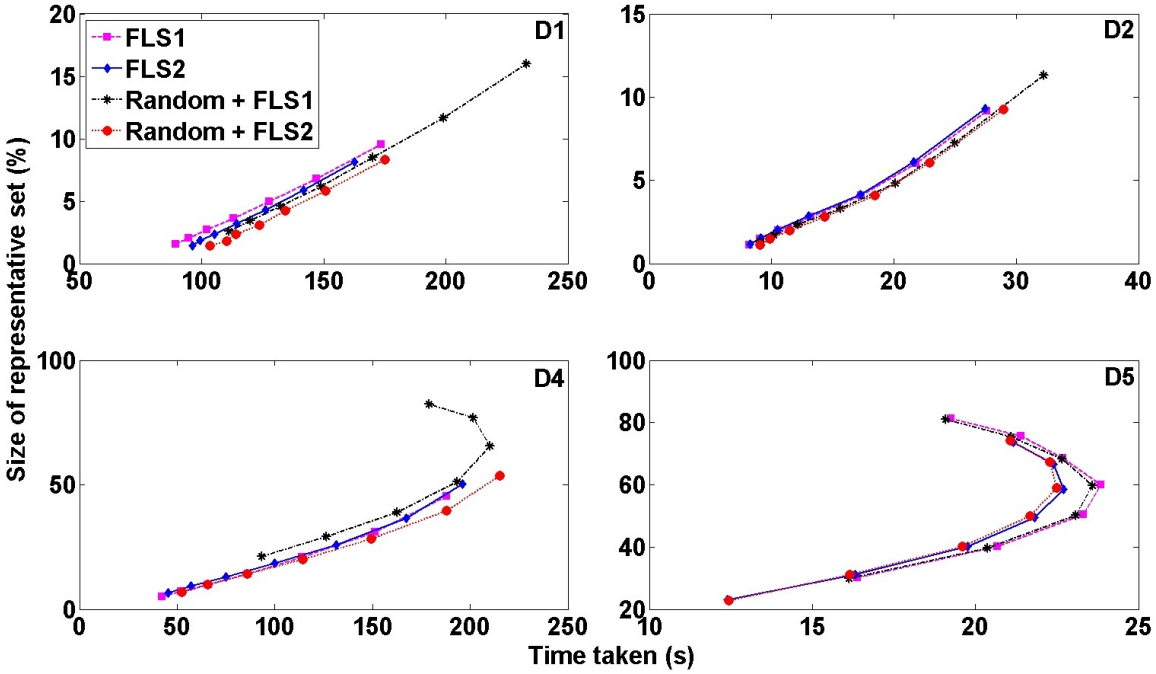

Figure 1: Performance of variants of DeriveRS with $g = [2^{-4}, 2^{-3}, 2^{-2}, ..., 2^2]$, for datasets D1, D2, D4, and D5. The results of DeriveRS with FLS1 and FLS2, after randomly reordering the datasets are shown as Random+FLS1 and Random+FLS2, respectively

For datasets D1 and D5, FLS2 gave smaller representative sets in a shorter duration than FLS1. As expected, for the relatively homogeneous dataset D2, FLS1 and FLS2 gave similar results, with FLS2 giving slightly larger representative sets. Dataset D4 was seen to have much smaller representative sets with FLS1 than with FLS2. The results of DeriveRS obtained after randomly rearranging the datasets, indicate the utility of FLS2. For all the datasets, the results of FLS2 after random reordering was seen to be significantly better than the results of FLS1 after random rearrangement. Hence we can infer that the good results obtained with FLS2 are not caused by any pre-existing order in the datasets. After D2 and D4 were randomly rearranged a sharp increase was observed in representative set sizes and computation times for DeriveRS with FLS1. This indicates the importance of dataset homogeneity to the performance of FLS1. The results indicated for randomized experiments on DeriveRS are the averages of five repetitions.

Next we investigated the impact of changes in the values of the parameters P and V on the performance of DeriveRS. All combinations of $P = \{10^4, 5\text{x}10^4, 10^5, 2\text{x}10^5\}$ and $V = \{10^2, 5\text{x}10^2, 10^3, 2\text{x}10^3, 3\text{x}10^3\}$ were used to compute the representative set of D1. The computations were performed for $\epsilon = 10^{-3}$ and $g = 1$. The method FLS2 was used for the first level segregation in DeriveRS. The results are shown in Table ??. As expected for an algorithm of time complexity $O(N(\log_2 \frac{N}{P} + \frac{P}{V} + \log_2 V) +$

$V\sum_{q=1}^{Q}\sum_{r=1}^{R}S(A_{q_r})$), the computation time was generally observed to increase for an increase in the value of V or P. It should be noted that our implementation of DeriveRS was based on SMO and hence $S(A_{q_r}) = O(A_{q_r}^2)$. In some cases the computation time decreased when P or V increased. This is caused by a decrease in the value of $O(\sum_{q=1}^{Q}\sum_{r=1}^{R}A_{q_r}^2)$, which is inferred from the observed decrease of the size of the representative set M ($M \approx \sum_{q=1}^{Q}\sum_{r=1}^{R}A_{q_r}$). A sharp decrease in M was observed when V was increased. The impact of increasing P on the size of the representative set was found to be less drastic. This observation indicates that DeriveAE selects fewer approximate extreme points when V is larger.

$\frac{M}{N}$x100% (Computation time in seconds)					
P	$V = 10^2$	$V = 5\text{x}10^2$	$V = 10^3$	$V = 2\text{x}10^3$	$V = 3\text{x}10^3$
10^4	10.7(27)	6.1(67)	5.1(131)	4.5(258)	4.3(338)
$5\text{x}10^4$	9.9(78)	5.3(72)	4.4(130)	3.9(249)	3.7(351)
10^5	9.8(142)	5.2(83)	4.3(134)	3.7(242)	3.5(352)
$2\text{x}10^5$	9.8(254)	5.1(104)	4.2(144)	3.7(240)	3.4(355)

Table 1: The impact of varying P and V on the result of DeriveRS

As described in Section **??**, we compared several SVM training algorithms with our implementation of AESVM. We performed a grid search with all combinations of the SVM hyper-parameters $C' = \{2^{-4}, 2^{-3}, ..., 2^6, 2^7\}$ and $g = \{2^{-4}, 2^{-3}, 2^{-2}, ..., 2^1, 2^2\}$. The hyper-parameter C' is related to the hyper-parameter C as $C' = \frac{C}{N}$. We represent the grid in terms of C' as it is used in several SVM solvers such as LIBSVM, LASVM, CVM and BVM. Furthermore, the use of C' enables the application of the same hyper-parameter grid to all datasets. To train AESVM with all the hyper-parameter combinations in the grid, the representative set has to be computed using DeriveRS for all values of kernel hyper-parameter g in the grid. This is because the kernel space varies when the value of g is varied. For all the computations, the input parameters were set as $P = 10^5$ and $V = 10^3$. The first level segregation in DeriveRS was performed using FLS2. Three values of the tolerance parameter ϵ were investigated, $\epsilon = 10^{-3}, 10^{-4}$ or 10^{-5}.

The results of the computation for datasets D1 - D5, are shown in the Table **??**. The percentage of data vectors in the representative set was found to increase with increasing values of g. This is intuitive, as when g increases the distance between the data vectors in kernel space increases. With increased distances, more data vectors $\mathbf{x}_i$ become approximate extreme points. The increase in the number of approximate extreme points with g causes the rising trend of computation time shown in Table **??**. For a decrease in the value of ϵ, M increases. This is because, for smaller ϵ fewer $\mathbf{x}_i$ would satisfy the condition: optimized $p(\mathbf{x}_i, \Psi) \leq \epsilon$ in CheckPoint$(\mathbf{x}_i, \Psi)$. This results in the selection of a larger number of approximate extreme points in DeriveAE.

The results of applying DeriveRS to the high-dimensional datasets D6-D9 are shown in Table **??**. It was observed that $\frac{M}{N}$ was much larger for D6-D9 than for the other datasets. We computed the representative set with $\epsilon = 10^{-3}$ only, as for smaller values of ϵ we expect the representative set to be close to 100% of the training set. The increasing trend of the size of the representative set with increasing g values can be observed in Table **??** also.

5.3 Comparison of AESVM to SVM solvers

To judge the accuracy and efficiency of AESVM, its classification performance was compared with the SMO implementation in LIBSVM, ver. 3.1. We chose LIBSVM because it is a state-of-the-art

$\frac{M}{N}$x100% (Computation time in seconds)								
ϵ	Dataset	g = $\frac{1}{2^4}$	g = $\frac{1}{2^3}$	g = $\frac{1}{2^2}$	g = $\frac{1}{2}$	g = 1	g = 2^1	g = 2^2
10^{-3}	D1	1.5(98)	1.9(104)	2.4(110)	3.2(119)	4.3(132)	5.9(148)	8.1(168)
	D2	1.2(7)	1.5(8)	2(9)	2.8(11)	4.1(15)	6(18)	9.2(23)
	D3	0.6(37)	0.6(37)	0.6(36)	0.6(36)	0.5(37)	0.6(37)	0.6(39)
	D4	4.3(45)	6.4(57)	9.4(74)	13.9(103)	20.7(139)	30.7(178)	44.8(216)
	D5	4.5(7)	8.3(9)	14(11)	21.8(14)	31.8(18)	43.7(21)	54.9(22)
10^{-4}	D1	3(136)	4(159)	5.3(191)	7.2(240)	9.9(297)	13.3(362)	17.4(435)
	D2	2.8(12)	3.8(18)	5(27)	6.8(37)	9.3(44)	13.5(44)	19.9(82)
	D3	0.5(36)	0.6(37)	0.6(38)	0.7(39)	0.8(41)	0.9(43)	1.1(47)
	D4	13.5(135)	18.3(211)	24.9(300)	34.2(400)	47.7(493)	63.5(513)	74.4(445)
	D5	20.1(16)	27.9(22)	37.4(27)	47.6(31)	57.3(34)	66(34)	74(34)
10^{-5}	D1	7(316)	9.3(425)	12.2(552)	15.7(726)	19.6(926)	24.2(1112)	28.9(1235)
	D2	6.2(59)	7.8(87)	9.8(98)	13(109)	18.3(138)	25.6(187)	34.3(235)
	D3	0.7(39)	0.8(42)	0.9(45)	1.1(50)	1.4(59)	1.7(73)	2.2(100)
	D4	30.7(607)	39.5(814)	51.9(1051)	66(1171)	75.1(1044)	77.8(839)	78.4(649)
	D5	43.3(50)	51.8(58)	60.3(62)	67.7(63)	73.8(59)	78.7(52)	81.8(44)

Table 2: The percentage of the data vectors in $\mathbf{X}^*$ (given by $\frac{M}{N}$x100) and its computation time for datasets D1-D5

$\frac{M}{N}$x100% (Computation time in seconds)							
Dataset	g = $\frac{1}{2^4}$	g = $\frac{1}{2^3}$	g = $\frac{1}{2^2}$	g = $\frac{1}{2}$	g = 1	g = 2^1	g = 2^2
D6	69.3(19)	70.4(19)	73.4(19)	80.3(14)	83.9(9)	84(8)	87.9(8)
D7	84.4(1077)	84.6(1089)	84.9(1069)	85.6(1085)	86.9(1079)	89.9(1032)	94.7(818)
D8	90(131)	96.6(94)	98.8(78)	99.5(72)	100(70)	100(71)	100(63)
D9	60.8(34)	62.9(36)	67(30)	70.8(21)	72.7(16)	75.2(14)	76.7(15)

Table 3: The percentage of data vectors in $\mathbf{X}^*$ and its computation time for datasets D6-D9 with $\epsilon = 10^{-3}$

SMO implementation that is routinely used in similar comparison studies. To compare the efficiency of AESVM to other popular approximate SVM solvers we chose CVM, BVM, LASVM, SVM$^{\text{perf}}$, and RfeatSVM. A description of these methods is given in Section **??**. We chose these methods because they are widely cited, their software implementations are freely available and other studies (**?**) have reported fast SVM training using some of these methods. LASVM is also an efficient method for online SVM training. However, since we do not investigate online SVM learning in this paper, we did not test the online SVM training performance of LASVM. We compared AESVM with CVM and BVM even though they are L2-SVM solvers, as they has been reported to be faster alternatives to SVM implementations such as LIBSVM.

The implementation of AESVM and DeriveRS were built upon the LIBSVM implementation. All methods except SVM$^{\text{perf}}$ were allocated a cache of size 600 MB. The parameters for DeriveRS were $P = 10^5$ and $V = 10^3$, and the first level segregation was performed using FLS2. To reflect a typical SVM training scenario, we performed a grid search with all eighty four combinations of the SVM hyper-parameters $C' = \{2^{-4}, 2^{-3}, ..., 2^6, 2^7\}$ and $g = \{2^{-4}, 2^{-3}, 2^{-2}, ..., 2^1, 2^2\}$. As mentioned earlier, for datasets D2, D3 and D4, five fold cross-validation was performed. The results of the

comparison have been split into sub-sections given below, due to the large number of SVM solvers and datasets used.

5.3.1 Comparison to CVM, BVM, LASVM and LIBSVM

First we present the results of the performance comparison for D2 in Figures **??** and **??**. For ease of representation, only the results of grid points corresponding to combinations of $C' = \{2^{-4}, 2^{-2}, 1, 2^2, 2^4, 2^6\}$ and $g = \{2^{-4}, 2^{-2}, 1, 2^2\}$ are shown in Figures **??** and **??**. Figure **??** shows the graph between training time and classification accuracy for the five algorithms. Figure **??** shows the graph between the number of support vectors and classification accuracy. We present classification accuracy as the ratio of the number of correct classifications to the total number of classifications performed. Since the classification time of an SVM algorithm is directly proportional to the number of support vectors, we represent it in terms of the number of support vectors. It can be seen that, AESVM generally gave more accurate results for a fraction of the training time of the other algorithms, and also resulted in less classification time. The training time and classification times of AESVM increased when ϵ was reduced. This is expected given the inverse relation of M to ϵ shown in Tables **??** and **??**. The variation in accuracy with ϵ is not very noticeable.

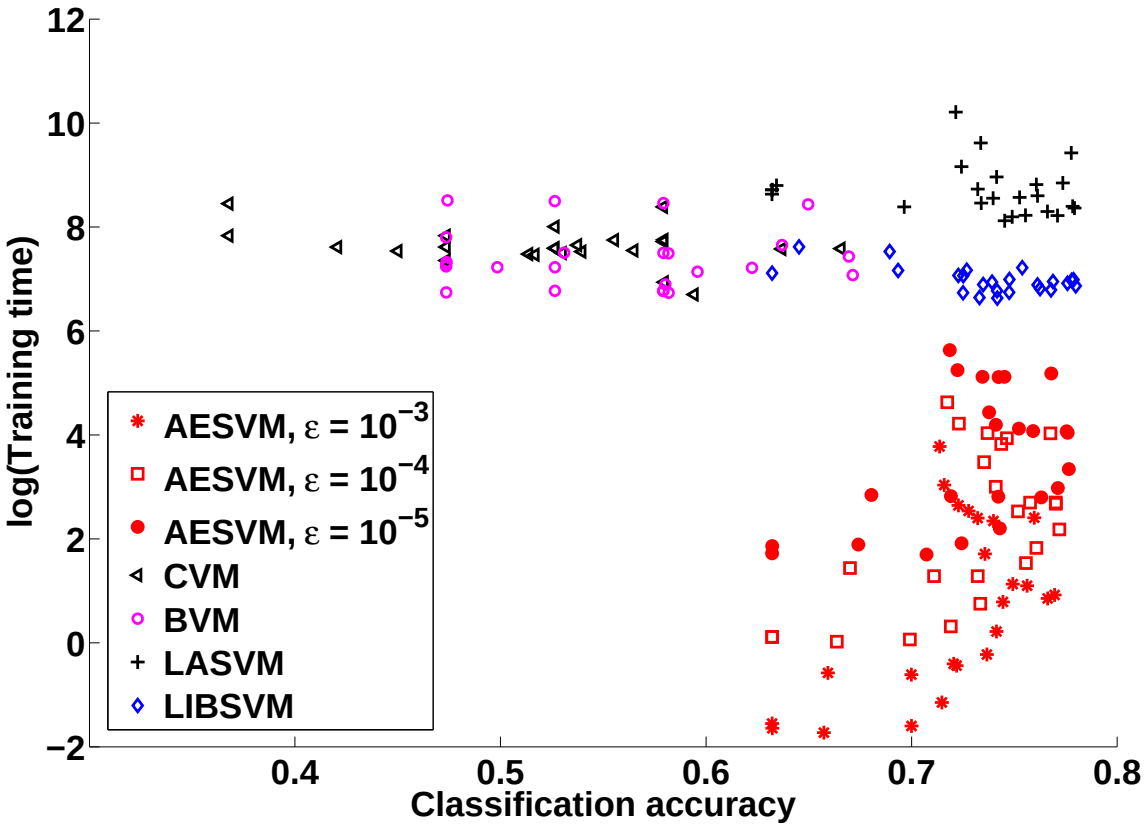

Figure 2: Plot of training time against classification accuracy of the SVM algorithms on D2

Figures **??** and **??** indicate that AESVM gave better results than the other algorithms for SVM training and classification on D2, in terms of standard metrics. To present a more quantitative and easily interpretable comparison of the algorithms, we define the five performance metrics given below. These metrics combine the results of all runs of each algorithm into a single value, for each dataset. For these metrics we take LIBSVM as a baseline of comparison, as it gives the most accurate

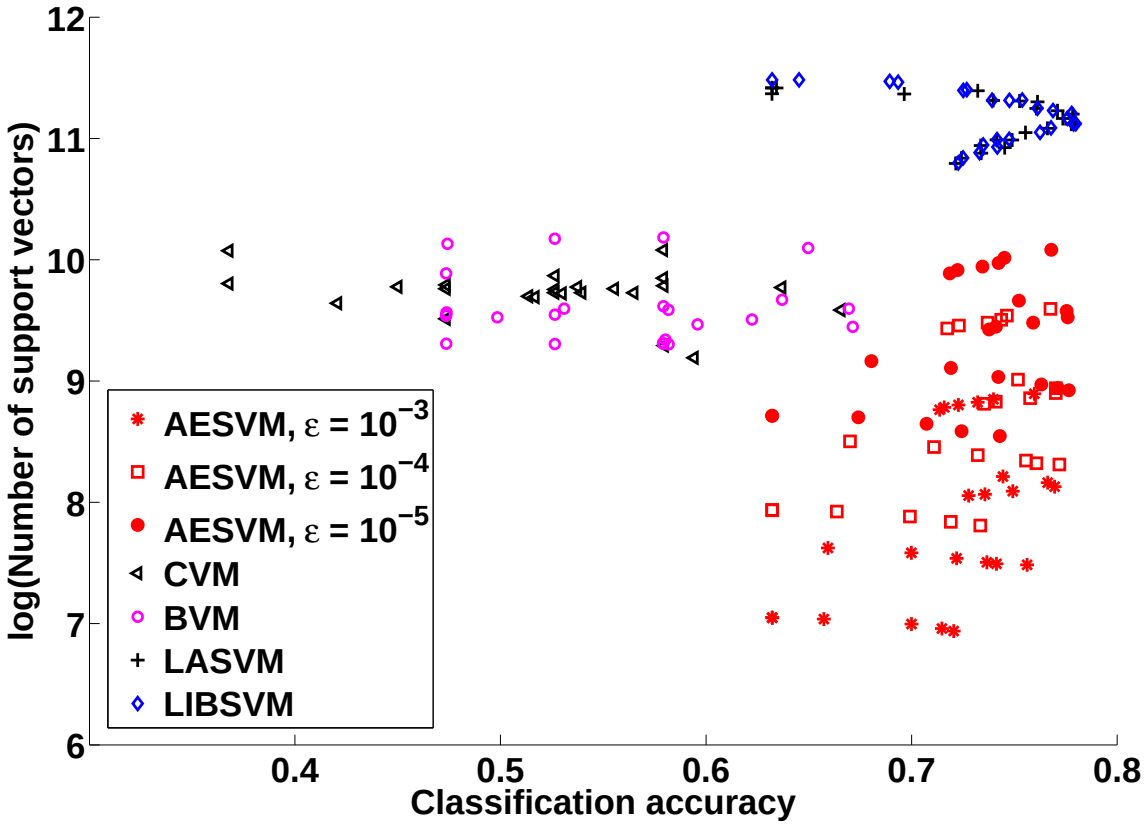

Figure 3: Plot of classification time, represented by the number of support vectors, against classification accuracy of the SVM algorithms on D2

solution among the tested methods. Furthermore, an important objective of these experiments is to show the similarity of the results of AESVM and LIBSVM. In the description given below, $\mathbb{F}$ can refer to any or any approximate SVM algorithm such as AESVM, CVM, LASVM etc.

1. *Root mean squared error of classification accuracy, RMSE*: The similarity of the solution of $\mathbb{F}$ to LIBSVM, in terms of its classification accuracy, is indicated by:

$$RMSE = \left(\frac{1}{RS} \sum_{r=1}^{R} \sum_{s=1}^{S} (CL_s^r - C\mathbb{F}_s^r)^2 \right)^{0.5}$$

   Here CL_s^r and $C\mathbb{F}_s^r$ are the classification accuracy of LIBSVM and $\mathbb{F}$ respectively, in the s^{th} cross-validation fold with the r^{th} set of hyper-parameters of grid search.

2. *Expected training time speedup, ETS*: The expected speedup in training time is indicated by:

$$ETS = \frac{1}{RS} \sum_{r=1}^{R} \sum_{s=1}^{S} \frac{TL_s^r}{T\mathbb{F}_s^r}$$

   Here TL_s^r and $T\mathbb{F}_s^r$ are the training times of LIBSVM and $\mathbb{F}$ respectively.

3. *Overall training time speedup, OTS*: It indicates overall training time speedup for the entire grid search with cross-validation, including the time taken to compute the representative set.

The total time taken by DeriveRS to compute the representative set for all values of g is represented as $\mathbb{TX}^*$. For methods other than AESVM, $\mathbb{TX}^* = 0$.

$$OTS = \frac{\sum_{r=1}^{R}\sum_{s=1}^{S} TL_s^r}{\sum_{r=1}^{R}\sum_{s=1}^{S} T\mathbb{F}_s^r + \mathbb{TX}^*}$$

4. *Expected classification time speedup, ECS*: The expected speedup in classification time is indicated by:
$$ECS = \frac{1}{RS}\sum_{r=1}^{R}\sum_{s=1}^{S}\frac{NL_s^r}{N\mathbb{F}_s^r}$$
Here NL_s^r and $N\mathbb{F}_s^r$ are the number of support vectors in the solution of LIBSVM and $\mathbb{F}$ respectively.
5. *Overall classification time speedup, OCS*: The overall speedup in classification time is indicated by:
$$OCS = \frac{\sum_{r=1}^{R}\sum_{s=1}^{S} NL_s^r}{\sum_{r=1}^{R}\sum_{s=1}^{S} N\mathbb{F}_s^r}$$

The results of the classification performance comparison on datasets D1-D5, are shown in Table **??**. It was observed that for all tested values of ϵ, AESVM resulted in large reductions in training and classification times when compared to LIBSVM for a very small difference in classification accuracy. Most notably, for D3 the expected and overall training time speedups were of the order of 10^4 and 10^3 respectively, which is outstanding. Comparing the results of AESVM for different ϵ values, we see that $RMSE$ generally improves by decreasing when ϵ decreases, while the metrics improve by increasing when ϵ increases. The increase in ETS and OTS is of a larger order than the increase in $RMSE$ when ϵ increases.

Comparing AESVM to CVM, BVM and LASVM, we see that AESVM in general gave the least values of $RMSE$ and the largest values of ETS, OTS, ECS and OCS. In a few cases LASVM gave low $RMSE$ values. However, in all our experiments LASVM took longer to train than the other algorithms including LIBSVM. *We could not complete the evaluation of LASVM for D4 due to its large training time, which was more than 40 hours for some hyper-parameter combinations.* It was also found that LASVM sometimes resulted in a larger classification time than the other algorithms including LIBSVM. CVM and BVM generally gave high vales of $RMSE$.

Table **??** compares the classification accuracy of CVM, BVM, LASVM and AESVM to the exact SVM solution given by LIBSVM. Another method to compare the algorithms is in terms of the maximum classification accuracy, and the mean and standard deviation of the classification accuracies, without using LIBSVM as a reference point. Such a comparison for datasets D1-D5, is given in Table **??**. The five algorithms under comparison were found to give similar maximum classification accuracies except for D2 and D4, where CVM and BVM gave significantly smaller values. Another interesting result is that for D3, the mean and standard deviation of accuracy of LASVM was found to be widely different from the other algorithms. For all the tested values of ϵ the maximum, mean and standard deviation of the classification accuracies of AESVM were found to be similar.

Next we present the results of performance comparison of CVM, BVM, LASVM, AESVM, and LIBSVM on the high-dimensional datasets D6-D9. As described in Section **??**, DeriveRS was run with only $\epsilon = 10^{-3}$ for these datasets. The results of the performance comparison are shown in Tables **??** and **??**. *CVM was found to take longer than 40 hours to train on D6, D7 and D8 with*

Metric	Dataset	AESVM $\epsilon = 10^{-3}$	AESVM $\epsilon = 10^{-4}$	AESVM $\epsilon = 10^{-5}$	CVM	BVM	LASVM
$RMSE$ (x10^2)	D1	0.28	0.16	0.21	0.44	0.6	0.12
	D2	2.56	1.81	1.19	26.59	24.06	2.18
	D3	0.16	0.10	0.05	0.33	0.39	55.2
	D4	1.08	0.82	0.74	9.4	9.44	–
	D5	0.99	0.39	0.23	0.74	0.84	0.13
ETS	D1	451.5	145	41.7	8.9	28.6	0.8
	D2	1614.7	289.6	62.8	0.7	0.8	0.2
	D3	28012.3	14799.3	7573.8	60.4	76.8	0.9
	D4	103.1	13.8	3.4	8	6.6	–
	D5	40.2	5	2	0.3	0.5	0.6
OTS	D1	92.1	34.2	9.5	6.2	21.6	0.8
	D2	148.6	45.5	14.3	0.5	0.5	0.1
	D3	968.5	800.6	514.4	23.9	22.8	0.5
	D4	11.9	4.1	2.2	6.2	4.4	–
	D5	5.2	2.5	1.5	0.2	0.3	0.5
ECS	D1	4.8	3.6	2.8	1.2	2	1.1
	D2	35.9	15.5	7.9	4.7	5	1
	D3	48.7	25.8	13.4	0.4	0.6	0.6
	D4	8.4	3.3	1.8	12.4	12.1	–
	D5	4.3	1.9	1.4	0.8	1	1
OCS	D1	3.8	3.1	2.5	1.1	1.9	1
	D2	23.4	10.9	6.1	4.5	4.4	1
	D3	32.2	16.1	9	0.3	0.5	0.2
	D4	5.4	2.7	1.7	12	10.7	–
	D5	2.8	1.8	1.4	0.8	1	1

Table 4: Performance comparison of AESVM (with $\epsilon = 10^{-3}, 10^{-4}, 10^{-5}$), CVM, BVM, LASVM and LIBSVM on datasets D1-D5

Accuracy	Dataset	AESVM $\epsilon = 10^{-3}$	AESVM $\epsilon = 10^{-4}$	AESVM $\epsilon = 10^{-5}$	CVM	BVM	LASVM	LIBSVM
Maximum (x10^2)	D1	93.4	93.8	93.6	94.1	94.4	94.3	93.9
	D2	77.1	77.2	77.8	70.3	67.1	78.1	78.2
	D3	99.9	99.9	99.9	99.9	99.9	99.9	99.9
	D4	68.3	68.3	68.3	63.7	62.3	–	68.2
	D5	98.7	98.8	98.9	99	99.1	99.2	99
Mean, standard deviation (x10^2)	D1	92.2, 0.7	92.3, 0.8	92.3, 0.8	92.7, 0.8	92.6, 0.9	92.5, 0.8	92.4, 0.8
	D2	72.3, 3.6	73.2, 3.7	73.6, 3.7	52.2, 0.8	54.6, 0.7	73.5, 0.5	74.1, 3.5
	D3	99.8, 0	99.8, 0.1	99.8, 0.1	99.8, 0.2	99.8, 0.2	69.3, 29.9	99.8, 0.1
	D4	61.3, 3.1	61, 3.1	61, 3.1	55.5, 3.1	54.9, 3.4	–	60.6, 3.2
	D5	96, 2.5	96.3, 2.6	96.5, 2.6	96.6, 2.5	97, 2	97, 2	96.6, 2.4

Table 5: Comparison of classification accuracies of AESVM (with $\epsilon = 10^{-3}, 10^{-4}, 10^{-5}$), CVM, BVM, LASVM and LIBSVM on datasets D1-D5

some hyper-parameter values and hence we could not complete its evaluation for those datasets. BVM also took longer than 40 hours to train on D7 and it was also not evaluated for D7. AESVM consistently reported ETS, OTS, ECS and OCS values that are larger than 1 unlike the other algorithms. Similar to the results in Table **??**, LASVM and BVM resulted in very large $RMSE$ values for some datasets. The results in Table **??** are similar to Table **??**, with similar maximum accuracies for all algorithms and significantly lower mean and higher standard deviation of accuracy for BVM and LASVM on some datasets.

Metric	Dataset	AESVM $\epsilon = 10^{-3}$	CVM	BVM	LASVM
$RMSE$ (x10^2)	D6	0.21	-	7.8	0.85
	D7	1.37	-	-	2.37
	D8	0.02	-	17.55	0
	D9	0.15	1	0.89	27.5
ETS	D6	1.8	-	0.6	0.8
	D7	1.4	-	-	0.9
	D8	1.1	-	4.7	1
	D9	1.6	1.4	17.5	0.6
OTS	D6	1.5	-	0.6	0.5
	D7	1.2	-	-	0.7
	D8	1.1	-	2.6	0.9
	D9	1.3	1.2	16.9	0.5
ECS	D6	1.2	-	1.5	1
	D7	1.16	-	-	1
	D8	1	-	3.2	1
	D9	1.2	1.8	4.9	2.3
OCS	D6	1.1	-	1.5	1
	D7	1.1	-	-	1
	D8	1	-	2.6	1
	D9	1.1	1.9	5.2	1.1

Table 6: Performance comparison of AESVM (with $\epsilon = 10^{-3}$), CVM, BVM, LASVM and LIBSVM on datasets D6-D9

Accuracy	Dataset	AESVM $\epsilon = 10^{-3}$	CVM	BVM	LASVM	LIBSVM
Maximum (x10^2)	D6	85.2	-	85.2	85	85.1
	D7	88.3	-	-	88.4	88.6
	D8	99.7	-	99.7	99.7	99.7
	D9	99.3	99.5	99.5	99.5	99.5
Mean, standard deviation (x10^2)	D6	81.3, 2.8	-	80.2, 8.9	81.1, 2.9	81.4, 2.8
	D7	85.3, 5.7	-	-	85.2, 6.2	85.7, 4.8
	D8	92.3, 3.6	-	88.5, 18.1	92.3, 3.6	92.3, 3.6
	D9	98.7, 0.8	98.9, 0.8	98.9, 0.8	85.5, 23.9	98.8, 0.8

Table 7: Comparison of classification accuracies of AESVM (with $\epsilon = 10^{-3}$), CVM, BVM, LASVM and LIBSVM on datasets D6-D9

5.3.2 Comparison to SVM^{perf}

SVM^{perf} differs from the other SVM solvers in its ability to compute a solution close to the SVM solution for a given number of support vectors (k). The algorithm complexity is directly proportional to the parameter k, but with a decrease in k the approximation becomes worse and the difference between the solutions of SVM^{perf} and SVM increases. We used a value of $k = 1000$ for our experiments, as it has been reported to give good performance (**?**). SVM^{perf} was tested on datasets D1, D4, D5, D6, D8 and D9, with the Gaussian kernel [12] and the same hyper-parameter grid as described earlier. The results of the grid search are presented in Table **??**. The results of our experiments on AESVM (with $\epsilon = 10^{-3}$) and LIBSVM are repeated in Table **??** for ease of reference. The maximum, mean and standard deviation of classification accuracies are represented as max. Acc., mean Acc., and std. Acc. respectively.

Dataset	Solver	$RMSE$ (x10^2)	ETS	OTS	ECS	OCS	max. Acc. (x10^2)	mean Acc. (x10^2)	std. Acc. (x10^2)
D1	AESVM	0.28	451.5	92.1	4.8	3.8	93.4	92.2	0.7
	SVM^{perf}	0.74	3.7	0.9	6.8	6.8	94	92.7	0.5
	LIBSVM						93.9	92.4	0.8
D4	AESVM	1.08	103.1	11.9	8.4	5.4	68.3	61.3	3.1
	SVM^{perf}	2.14	3.1	1.2	186.8	186.8	68.1	61.8	2.7
	LIBSVM						68.2	60.6	3.2
D5	AESVM	0.99	40.2	5.2	4.3	2.8	98.7	96	2.5
	SVM^{perf}	0.26	0.2	0.1	5.8	5.8	99	96.7	2.4
	LIBSVM						99	96.6	2.4
D6	AESVM	0.21	1.8	1.5	1.2	1.1	85.2	81.3	2.8
	SVM^{perf}	9.39	1.1	0.9	20	20	85.2	79.6	10.7
	LIBSVM						85.1	81.4	2.8
D8	AESVM	0.02	1.1	1.1	1	1	99.7	92.3	3.6
	SVM^{perf}	54.2	37.6	23.8	49	49	99.9	55.7	42.3
	LIBSVM						99.7	92.3	3.6
D9	AESVM	0.15	1.6	1.3	1.2	1.1	99.3	98.7	0.8
	SVM^{perf}	22.6	1.2	0.9	21.3	21.3	99.2	86.1	18.8
	LIBSVM						99.5	98.8	0.8

Table 8: Performance comparison of SVM^{perf}, AESVM (with $\epsilon = 10^{-3}$), and LIBSVM

SVM^{perf} was found to generally give higher $RMSE$ values than AESVM. In particular, for the high dimensional datasets (D6, D8 and D9), the $RMSE$ values were significantly higher. The training speedup values of SVM^{perf} are much lower than AESVM except for D8. As expected, the classification time speedups of SVM^{perf} are significantly higher than AESVM. The maximum accuracies of all the algorithms were similar. However, the mean and standard deviation of accuracies of SVM^{perf} were very different from AESVM and LIBSVM for the high dimensional datasets D6, D8 and D9.

5.3.3 Comparison to RfeatSVM

? proposed a promising method to approximate non-linear kernel SVM solutions using simpler linear kernel SVMs. This is accomplished by first projecting the training dataset into a randomized feature space and then using any SVM solver with the linear kernel on the projected dataset. We

12. We used the software parameters '-t 2 -w 9 –i 2 –b 0 –k 1000' as suggested in the author's website

concentrated our experiments on investigating the accuracy of the solution of RfeatSVM and its similarity to the SVM solution. LIBSVM with the linear kernel was used to compute the RfeatSVM solution on the projected datasets. We used LIBSVM, in spite of the availability of faster linear SVM implementations, as it is an exact SVM solver. Hence only the performance metrics related to accuracy were used to compare the performance of AESVM, LIBSVM and RfeatSVM. The random Fourier features method, described in Algorithm 1 of **?**, was used to project the datasets D1, D5, D6 and D9 into a randomized feature space of dimension E. The results of the accuracy comparison are given in Table **??**. We used a smaller hyper-parameter grid of all twenty four combinations of $C' = \{2^{-4}, 2^{-2}, 1, 2^2, 2^4, 2^6\}$ and $g = \{2^{-4}, 2^{-2}, 1, 2^2\}$ for our experiments. The results reported in Table **??** for AESVM and LIBSVM were computed for this smaller grid.

Dataset	Solver	$RMSE$ (x10^2)	max. Acc. (x10^2)	mean Acc. (x10^2)	std. Acc. (x10^2)	Original density %	Density after projection %
D1	AESVM	0.24	93.6	92.2	0.9		
	RfeatSVM (E = 100)	56.18	37.8	36.1	1.3	33	100
	LIBSVM		93.6	92.3	0.9		
D5	AESVM	0.9	98.6	95.7	2.8		
	RfeatSVM (E = 100)	5.3	94.7	91.6	1.4	59	100
	LIBSVM		98.9	96.2	2.7		
D6	AESVM	0.16	85.1	81.2	2.9		
	RfeatSVM (E = 1000)	4	81.6	78	2.2	11	100
	LIBSVM		85	81.3	3		
D9	AESVM	0.15	99.3	98.6	0.8		
	RfeatSVM (E = 1000)	0.6	98.7	97.4	0.6	4	95.8
	LIBSVM		99.5	98.8	0.9		

Table 9: Performance comparison of RfeatSVM, AESVM (with $\epsilon = 10^{-3}$), and LIBSVM. The density of the datasets before and after projecting into randomized feature spaces are also shown

We used the same number of dimensions (E) of the randomized feature space for D1 and D6 as in **?**. The $RMSE$ values for RfeatSVM were significantly higher than AESVM for most datasets, especially for D1 and D6. The maximum accuracy for RfeatSVM was found to be much less than AESVM and LIBSVM for all datasets. The time taken to compute the randomized feature space is not reported because it was found to be negligibly small. Another important observation was that the projected datasets were found to be almost 100% dense. The training time of SVM solvers are typically linearly proportional to the density of the dataset and hence a highly dense dataset can take a significant training time even with fast linear SVMs. Dense datasets also have large memory requirements.

5.4 Performance with the polynomial kernel

To validate our proposal of AESVM as a fast alternative to SVM for all non-linear kernels, we performed a few experiments with the polynomial kernel, $k(\mathbf{x}_1, \mathbf{x}_2) = (1 + \mathbf{x}_1^T \mathbf{x}_2)^d$. The hyper-parameter grid composed of all twelve combinations of $C' = \{2^{-4}, 2^{-2}, 1, 2^2\}$ and $d = \{2, 3, 4\}$ was used to compute the solutions of AESVM and LIBSVM on the datasets D1, D4 and D6. The results

of the computation of the representative set using DeriveRS are shown in Table **??**. The parameters for DeriveRS were $P = 10^5$, $V = 10^3$ and $\epsilon = 10^{-3}$, and the first level segregation was performed using FLS2. The performance comparison of AESVM and LIBSVM with the polynomial kernel is shown in Table **??**. Like in the case of the Gaussian kernel, we found that AESVM gave results similar to LIBSVM with the polynomial kernel, while taking shorter training and classification times.

$\frac{M}{N}$x100% (Computation time in seconds)			
Dataset	d = 2	d = 3	d = 4
D1	6.6(410)	14.2(1329)	22.5(3696)
D4	30.3(752)	57.7(1839)	76.5(2246)
D6	69(20)	69.7(21)	70.4(22)

Table 10: Results of DeriveRS for the polynomial kernel

Dataset	Solver	$RMSE$ (x10^2)	ETS	OTS	ECS	OCS	max. Acc. (x10^2)	mean Acc. (x10^2)	std. Acc. (x10^2)
D1	AESVM	0.15	31.2	2	3.1	3.1	94	93.5	0.4
	LIBSVM						94.1	93.5	0.4
D4	AESVM	2.04	3.3	1.5	2	1.9	64.3	60.8	2.5
	LIBSVM						64.5	60.7	2.5
D6	AESVM	0.6	2.7	1.9	1.5	1.5	84.5	80.5	2.5
	LIBSVM						84.6	81	2.3

Table 11: Performance comparison of AESVM (with $\epsilon = 10^{-3}$), and LIBSVM with the polynomial kernel

6. Discussion

AESVM is a new problem formulation that is almost identical to, but less complex than, the SVM primal problem. AESVM optimizes over only a subset of the training dataset called the representative set, and consequently, is expected to give fast convergence with most SVM solvers. In contrast, the other studies mentioned in Section **??** are mostly algorithms that solve the SVM primal or related problems. Methods such as RSVM also use different problem formulations. However, they require special algorithms to solve, unlike AESVM. In fact, AESVM can be solved using many of the methods in Section **??**. As described in Corollary 5, there are some similarities between AESVM and the Gram matrix approximation methods discussed earlier. It would be interesting to see a comparison of AESVM, with the core set based method proposed by **?**. However, due to the lack of availability of a software implementation and of published results on L1-SVM with non-linear kernels using their approach, the authors find such a comparison study beyond the scope of this paper.

The theoretical and experimental results presented in this paper demonstrate that the solutions of AESVM and SVM are similar in terms of the resulting classification accuracy. A summary of the experiments in Section **??**, that compared an SMO based AESVM implementation, CVM, BVM, LASVM, LIBSVM, SVM$^{\text{perf}}$ and RfeatSVM, is presented in Figures **??** to **??**. *It can be seen that AESVM typically gave the lowest approximation error (RMSE), while giving highest overall training time speedup (OTS). AESVM also gave competitively high overall classification time speedup (OCS) in comparison with the other algorithms except SVM^{perf}.* It was found that the maximum

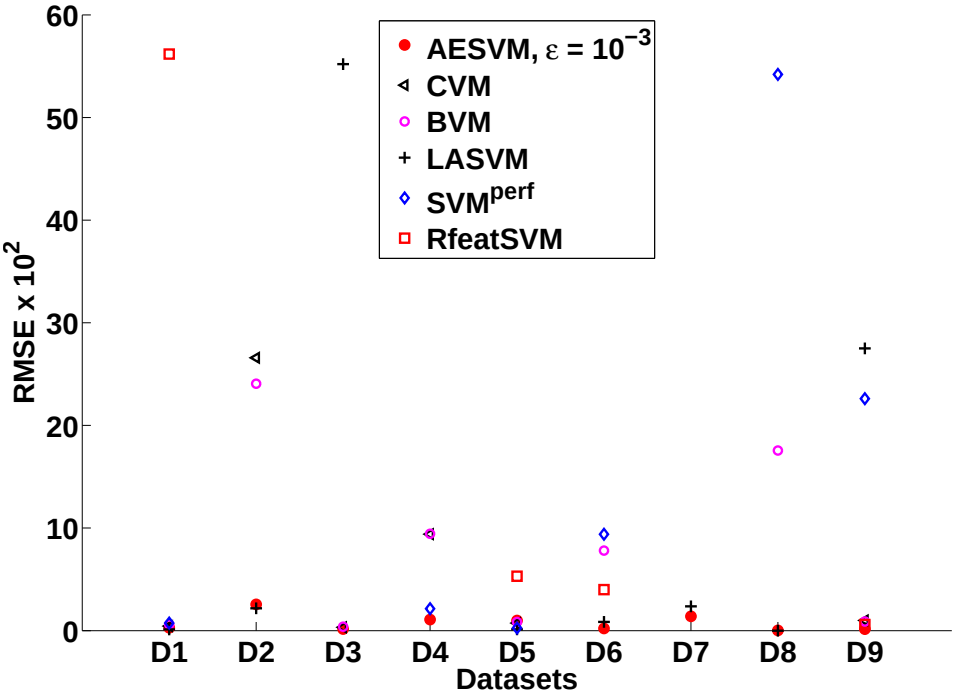

Figure 4: Plot of RMSE values for all SVM solvers

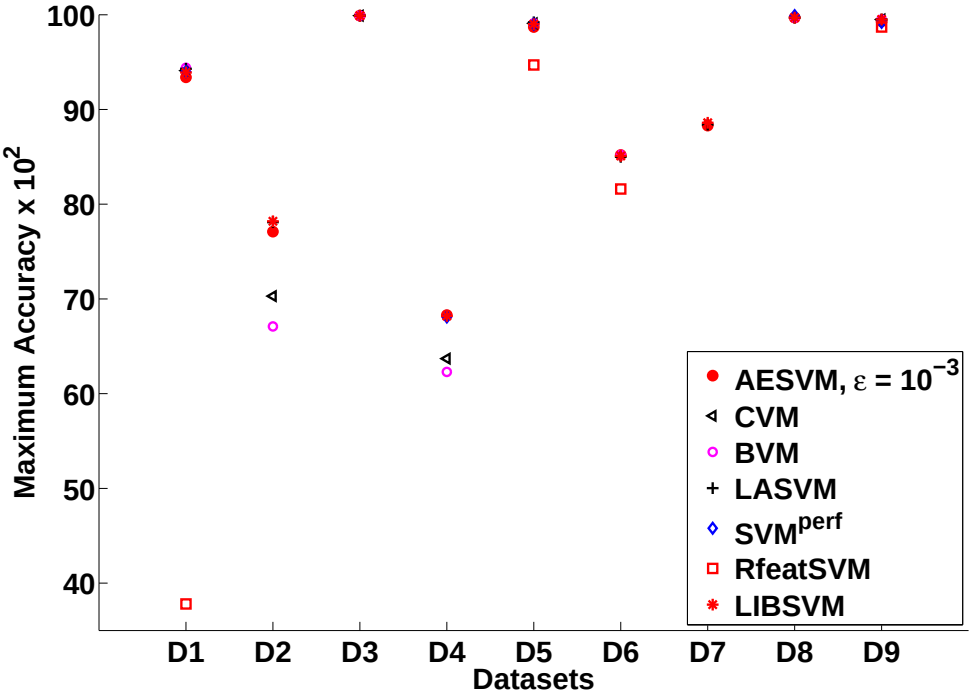

Figure 5: Plot of maximum classification accuracy for all SVM solvers

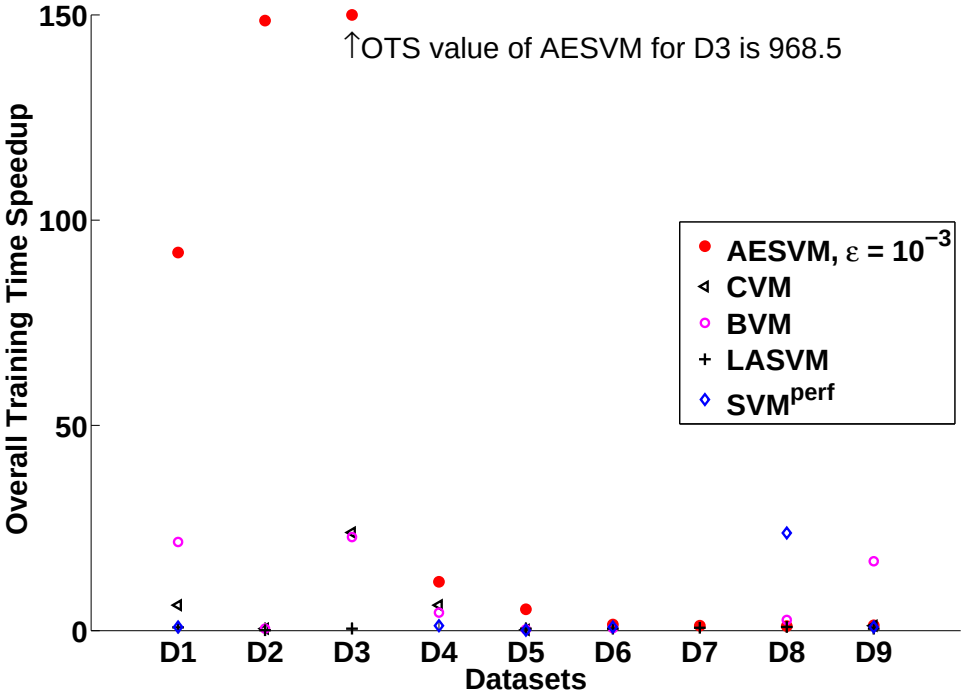

Figure 6: Plot of overall training time speedup (compared to LIBSVM) for all SVM solvers

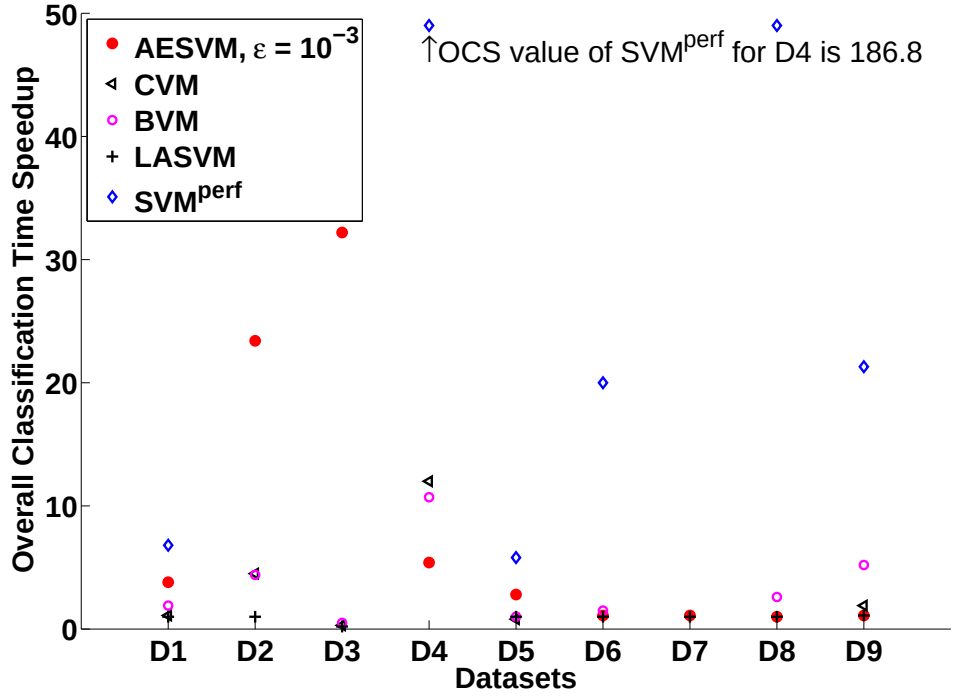

Figure 7: Plot of overall classification time speedup (compared to LIBSVM) for all SVM solvers

classification accuracies of all the algorithms except RfeatSVM were similar. RfeatSVM, and in some cases CVM and BVM, gave lower maximum classification accuracies. Though the results of RfeatSVM illustrated in Figures **??** and **??**, were computed for a smaller hyper-parameter grid (refer Section **??**), we believe it indicates the overall performance of the method. Apart from the excellent experimental results for AESVM with the Gaussian kernel, AESVM also gave good results with the polynomial kernel as described in Section **??**.

The algorithm DeriveRS was generally found to be efficient, especially for the lower dimensional datasets D1-D5. For the high dimensional datasets D6-D9, the representative set was almost the same size as the training dataset, resulting in small gains in training and classification time speedups for AESVM. In particular, for D8 (MNIST dataset) the representative set computed by DeriveRS was almost 100% of the training set. A similar result was reported for this dataset in **?**, where a divide and conquer method was used to speed up nearest neighbor search. Dataset D8 is reported to have resulted in nearly no speedup, compared to a speedup of almost one thousand for other datasets when their method was used. Their analysis found that the data vectors in D8 were very distant from each other in comparison with the other datasets [13]. This observation can explain the performance of DeriveRS on D8, as data vectors that are very distant from each other are expected to have large representative sets. It should be noted that irrespective of the dimensionality of the datasets, AESVM always resulted in excellent performance in terms of classification accuracy. There seems to be no relation between dataset density and the performance of DeriveRS and AESVM.

The authors will provide the software implementation of AESVM and DeriveRS upon request. Based on the presented results, we suggest the parameters $\epsilon = 10^{-3}$, $P = 10^5$ and $V = 10^3$ for DeriveRS. A possible extension of this paper is to apply the idea of the representative set to other SVM variants and to support vector regression (SVR). It is straightforward to see that the theorems in Section **??** can be extended to SVR. It would be interesting to investigate AESVM solvers implemented using methods other than SMO. Modifications to DeriveRS using the methods in Section **??** might improve its performance on high dimensional datasets. The authors will investigate improvements to DeriveRS and the application of AESVM to the linear kernel in their future work.

Acknowledgments

Dr. Khargonekar acknowledges support from the Eckis professor endowment at the University of Florida. Dr. Talathi was partially supported by the Children's Miracle Network, and the Wilder Center of Excellence in Epilepsy Research. The authors acknowledge Mr. Shivakeshavan R. Giridharan, for providing assistance with computational resources.

References

K. P. Bennett and E. J. Bredensteiner. Duality and geometry in SVM classifiers. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 57–64, 2000.

A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 97–104, 2006.

M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, August 1973.

A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, December 2005.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

13. This is indicated by the large expansion constant for D8 illustrated in **?**

J. Cervantes, X. Li, W. Yu, and K. Li. Support vector machine classification for large data sets via minimum enclosing ball clustering. *Neurocomputing*, 71:611–619, January 2008.

C. C. Chang and C. J. Lin. IJCNN 2001 challenge: Generalization ability and text decoding. In *Proceedings of International Joint Conference on Neural Networks*, volume 2, pages 1031 –1036, 2001a.

C. C. Chang and C. J. Lin. LIBSVM: a library for support vector machines. *Software available at* `http://www.csie.ntu.edu.tw/~cjlin/libsvm`, 2001b.

K. L. Clarkson. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. *ACM Transaction on Algorithms*, 6(4), September 2010.

R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computing*, 14(5):1105–1114, 2002.

P. Drineas and M. W. Mahoney. On the Nyström method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175, December 2005.

R. E. Fan, P. H. Chen, and C. J. Lin. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, and C. J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, June 2008.

S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2002.

V. Franc and S. Sonnenburg. Optimized cutting plane algorithm for support vector machines. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 320–327, 2008.

B. Gärtner and M. Jaggi. Coresets for polytope distance. In *Proceedings of the 25th annual symposium on Computational geometry*, pages 33–42, 2009.

J. Guo, N. Takahashi, and T. Nishi. A learning algorithm for improving the classification speed of support vector machines. In *Proceedings of the 2005 European Conference on Circuit Theory and Design*, volume 3, pages 381 – 384, 2005.

C. J. Hsieh, K. W. Chang, C. J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the 25th international conference on Machine learning*, pages 408–415, 2008.

T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006.

T. Joachims. Making large-scale support vector machine learning practical. In *Advances in kernel methods*, pages 169–184. MIT Press, 1999.

T. Joachims and C. N. J. Yu. Sparse kernel SVMs via cutting-plane training. *Machine Learning*, 76:179–193, September 2009.

B. Kaluza, V. Mirchevska, E. Dovgan, M. Lustrek, and M. Gams. An agent-based approach to care in independent living. In *Proceedings of AmI'2010*, 2010.

J. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278 –2324, 1998.

Y. J. Lee and O. L. Mangasarian. RSVM: Reduced support vector machines. In *Proceedings of the SIAM International Conference on Data Mining*, 2001.

M. Nandan, S. S. Talathi, S. Myers, W. L. Ditto, P. P. Khargonekar, and P. R. Carney. Support vector machines for seizure detection in an animal model of chronic epilepsy. *Journal of Neural Engineering*, 7(3), 2010.

E. Osuna and O. Castro. Convex hull in feature space for support vector machines. In *Proceedings of the 8th Ibero-American Conference on AI: Advances in Artificial Intelligence*, IBERAMIA 2002, pages 411–419, 2002.

E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 130 –136, 1997.

D. Pavlov, D. Chudova, and P. Smyth. Towards scalable support vector machines using squashing. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 295–299. ACM, 2000.

J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods*, pages 185–208. MIT Press, 1999.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems*, 2007.

R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1996.

B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12(5):1207–1245, May 2000. ISSN 0899-7667.

S. Shalev-Shwartz and N. Srebro. SVM optimization: Inverse dependence on training set size. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 928–935, 2008.

S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127:3–30, March 2011.

S. S. Talathi, D. U. Hwang, M. L. Spano, J. Simonotto, M. D. Furman, S. M. Myers, J. T. Winters, W. L. Ditto, and P. R. Carney. Non-parametric early seizure detection in an animal model of temporal lobe epilepsy. *Journal of Neural Engineering*, 5:85–98, 2008.

M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *Proceedings of the 2009 IEEE Symposium Computational Intelligence for Security and Defense Applications*, pages 53–58, 2009.

D. Tax and R. Duin. Support vector data description. *Machine Learning*, 54(1):45–66, 2004.

C. H. Teo, S. V. N. Vishwanthan, A. J. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 11:311–365, 2010.

I. W. Tsang, J. T. Kwok, P. Cheung, and N. Cristianini. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

I. W. Tsang, A. Kocsor, and J. T. Kwok. Simpler core vector machines with enclosing balls. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 911–918, 2007.

H. Yu, J. Yang, and J. Han. Classifying large data sets using SVMs with hierarchical clusters. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 306–315, 2003.

G. X. Yuan, C. H. Ho, and C. J. Lin. An improved GLMNET for l1-regularized logistic regression. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 33–41, 2011.

T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, 2004.