FAST SVM TRAINING USING APPROXIMATE EXTREME POINTS

By

MANU NANDAN

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2013
© 2013 Manu Nandan
To Saranya
ACKNOWLEDGMENTS

I thank my advisor, Dr. Pramod Khargonekar for mentoring me over the last five and a half years. He encouraged me to set lofty goals and guided me to achieve them. I thank my co-advisor Dr. Sachin Talathi for training me in disciplined research and publication, and offering advice on a wide range of topics throughout my graduate study. I would also like to thank my committee members Dr. Paul Gader, Dr. Joseph Wilson, Dr. Anand Rangarajan and Dr. Paul Carney. Their insightful comments and course work have greatly assisted me. Finally, I thank Saranya and my parents, for their encouragement and support.
# TABLE OF CONTENTS

| ACKNOWLEDGMENTS                      | 4 |
| LIST OF TABLES                       | 7 |
| LIST OF FIGURES                      | 8 |
| ABSTRACT                             | 10 |

## CHAPTER

### 1 INTRODUCTION

1.1 Support Vector Machines
1.2 Approximate Extreme Point Support Vector Machine
1.3 Outline

### 2 RELATED WORK

2.1 SVM Solvers
2.1.1 Dual Optimization
2.1.1.1 Decomposition methods
2.1.1.2 Approximate methods
2.1.1.3 Coordinate descent
2.1.2 Primal Optimization
2.1.2.1 Stochastic sub-gradient descent
2.1.2.2 Cutting plane algorithm
2.1.2.3 Smoothing of objective function
2.2 Fast SVM Classification

### 3 DESCRIPTION OF AESVM

3.1 Definition of the Representative Set
3.2 Properties of AESVM
3.3 Comparison to Other SVM Solvers

### 4 COMPUTATION OF THE REPRESENTATIVE SET

4.1 First Level of Segregation
4.2 Second Level of Segregation
4.3 Computation of the Approximate Extreme Points
4.4 Combining All the Methods to Compute the Representative Set

### 5 EXPERIMENTS ON AESVM

5.1 Datasets
5.2 Evaluation of DeriveRS
5.3 Comparison of AESVM to SVM Solvers ........................................ 47
  5.3.1 Comparison to CVM, BVM, LASVM and LIBSVM ................. 48
  5.3.2 Comparison to SVMperf ................................................. 53
  5.3.3 Comparison to RfeatSVM ............................................... 56
  5.3.4 Performance with the Polynomial Kernel ............................ 59
5.4 Discussion of Results ....................................................... 59
6 FAST CLASSIFICATION USING APPROXIMATE EXTREME POINTS .... 64
  6.1 Description of DRSpost .................................................... 64
  6.2 Experiments on DRSpost .................................................. 66
7 PARALLEL SVM SOLVER USING APPROXIMATE EXTREME POINTS . 73
  7.1 Description of DRSpl ........................................................ 75
  7.2 Optimizations of DRSpl .................................................... 81
  7.3 Experiments on DRSpl ....................................................... 81
  7.4 Solve AESVM on Data that is Too Large to Fit in Memory with One CPU 94
  7.5 Non-Negative Matrix Factorization using Approximate Extreme Points 96
    7.5.1 Description of AENMF ................................................. 97
    7.5.2 Experiments on AENMF .............................................. 98
8 CONCLUSIONS AND FUTURE WORK ......................................... 105
REFERENCES ........................................................................... 107
BIOGRAPHICAL SKETCH .......................................................... 114
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-1</td>
<td>The impact of varying $P$ and $V$ on the result of DeriveRS</td>
<td>44</td>
</tr>
<tr>
<td>5-2</td>
<td>The percentage of the data vectors in the representative set and its computation time for D1-D5</td>
<td>46</td>
</tr>
<tr>
<td>5-3</td>
<td>The percentage of the data vectors in the representative set and its computation time for D6-D9</td>
<td>46</td>
</tr>
<tr>
<td>5-4</td>
<td>Performance comparison of AESVM, CVM, BVM, LASVM and LIBSVM on datasets D1-D5</td>
<td>52</td>
</tr>
<tr>
<td>5-5</td>
<td>Performance comparison of AESVM, CVM, BVM, LASVM and LIBSVM on datasets D6-D9</td>
<td>54</td>
</tr>
<tr>
<td>5-6</td>
<td>Performance comparison of SVM$^{perf}$, AESVM, and LIBSVM</td>
<td>55</td>
</tr>
<tr>
<td>5-7</td>
<td>Performance comparison of RfeatSVM1, AESVM, and LIBSVM</td>
<td>57</td>
</tr>
<tr>
<td>5-8</td>
<td>Performance comparison of RfeatSVM2, AESVM, and LIBSVM</td>
<td>58</td>
</tr>
<tr>
<td>5-9</td>
<td>Results of DeriveRS for the polynomial kernel</td>
<td>60</td>
</tr>
<tr>
<td>5-10</td>
<td>Performance comparison of AESVM, and LIBSVM with the polynomial kernel</td>
<td>60</td>
</tr>
<tr>
<td>6-1</td>
<td>Comparison of classification performance of DRSpost, SVM$^{perf}$ and LIBSVM</td>
<td>71</td>
</tr>
<tr>
<td>7-1</td>
<td>Performance of DRSpl, BagSVM and LIBLINEAR on D10</td>
<td>86</td>
</tr>
<tr>
<td>7-2</td>
<td>Performance of DRSpl, BagSVM, LIBLINEAR and LIBOCAS on D11</td>
<td>87</td>
</tr>
<tr>
<td>7-3</td>
<td>Performance of DRSpl, BagSVM, LIBLINEAR and LIBOCAS on D12</td>
<td>89</td>
</tr>
<tr>
<td>7-4</td>
<td>Performance of DRSpl, BagSVM, LIBLINEAR and LIBOCAS on D13</td>
<td>90</td>
</tr>
<tr>
<td>7-5</td>
<td>Performance of DRSpl, BagSVM, LIBLINEAR and LIBOCAS on D8</td>
<td>91</td>
</tr>
<tr>
<td>7-6</td>
<td>Convergence of DRSpl with datasets D10, D11, D12 and D13</td>
<td>95</td>
</tr>
<tr>
<td>7-7</td>
<td>Performance of various NMF solvers</td>
<td>101</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Extreme points of a dataset</td>
<td>22</td>
</tr>
<tr>
<td>5-1</td>
<td>Performance of variants of DeriveRS for datasets D1, D2, D4, and D5</td>
<td>43</td>
</tr>
<tr>
<td>5-2</td>
<td>Plot of training time against classification accuracy of the SVM algorithms on D2</td>
<td>49</td>
</tr>
<tr>
<td>5-3</td>
<td>Plot of classification time, represented by the number of support vectors, against classification accuracy of the SVM algorithms on D2</td>
<td>50</td>
</tr>
<tr>
<td>5-4</td>
<td>Plot of mean classification accuracy of all SVM solvers</td>
<td>60</td>
</tr>
<tr>
<td>5-5</td>
<td>Plot of maximum classification accuracy of all SVM solvers</td>
<td>61</td>
</tr>
<tr>
<td>5-6</td>
<td>Plot of overall training time speedup of all SVM solvers</td>
<td>61</td>
</tr>
<tr>
<td>5-7</td>
<td>Plot of classification time speedup for optimal hyper-parameters of all SVM solvers</td>
<td>62</td>
</tr>
<tr>
<td>6-1</td>
<td>Plot of performance of DRSpost on D1</td>
<td>68</td>
</tr>
<tr>
<td>6-2</td>
<td>Plot of performance of DRSpost on D5</td>
<td>69</td>
</tr>
<tr>
<td>6-3</td>
<td>Plot of performance of DRSpost on D6</td>
<td>69</td>
</tr>
<tr>
<td>6-4</td>
<td>Plot of performance of DRSpost on D9</td>
<td>70</td>
</tr>
<tr>
<td>6-5</td>
<td>Illustration of variation of DRSpost with different $C'$ values on D1</td>
<td>70</td>
</tr>
<tr>
<td>6-6</td>
<td>Illustration of variation of DRSpost with different $C'$ values on D5</td>
<td>72</td>
</tr>
<tr>
<td>7-1</td>
<td>Plot of $OTS$ of DRSpl with D10</td>
<td>91</td>
</tr>
<tr>
<td>7-2</td>
<td>Plot of $OTS$ of DRSpl with D11</td>
<td>92</td>
</tr>
<tr>
<td>7-3</td>
<td>Plot of $OTS$ of DRSpl with D12</td>
<td>92</td>
</tr>
<tr>
<td>7-4</td>
<td>Plot of $OTS$ of DRSpl with D13</td>
<td>93</td>
</tr>
<tr>
<td>7-5</td>
<td>Plot of $EOS$ of DRSpl with D10</td>
<td>93</td>
</tr>
<tr>
<td>7-6</td>
<td>Plot of $EOS$ of DRSpl with D11</td>
<td>94</td>
</tr>
<tr>
<td>7-7</td>
<td>Plot of $EOS$ of DRSpl with D12</td>
<td>94</td>
</tr>
<tr>
<td>7-8</td>
<td>Plot of $EOS$ of DRSpl with D13</td>
<td>95</td>
</tr>
<tr>
<td>7-9</td>
<td>Plot of $MER$ of NMF solvers with D14 and D15</td>
<td>102</td>
</tr>
</tbody>
</table>
7-10 Plot of total computation time of NMF solvers with D14 and D15  . . . . . . 103
7-11 Plot of $MNF$ of NMF solvers with D14 and D15 . . . . . . . . . . . 104
Abstract of Dissertation Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

FAST SVM TRAINING USING APPROXIMATE EXTREME POINTS

By
Manu Nandan

December 2013

Chair: Pramod Khargonekar
Cochair: Sachin Talathi
Major: Computer Engineering

Support vectors machines (SVMs) are widely applied machine learning algorithms that have many desirable characteristics. However, with the ever increasing size of datasets, it has become increasingly difficult to use SVMs in modern applications. In this dissertation we address two disadvantages of SVMs. Firstly, non-linear kernel SVM solvers need excessive training times with large datasets. Secondly, linear SVM solvers need to be parallelized to train on web-scale datasets that are large and high dimensional. State-of-the-art linear SVM solvers, though fast, are difficult to parallelize.

We propose a modification, called the approximate extreme points support vector machine (AESVM), that is aimed at overcoming these disadvantages. 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. Linear or log-linear time algorithms based on convex hulls and extreme points are used to compute the representative sets. We also propose an algorithm to post-process the solution of SVMs to enable fast classification. A variant of the algorithm is easy to parallelize and is designed to compute efficiently on frameworks such as MapReduce. Extensive computational experiments on thirteen datasets compared our algorithms to other modern SVM solvers.

We compared our non-linear AESVM solver to LIBSVM [15], CVM [76], BVM [75], LASVM [7], SVM$^{\text{perf}}$ [36], and the random features method [61]. 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 500 times faster than LIBSVM and LASVM and 20 times faster than CVM and BVM. Additionally, AESVM also gave competitively fast classification times. To evaluate our parallel linear AESVM solver, DRSpl, we compared against LIBLINEAR [25], LIBOCAS [29], and a bagging based SVM solver. DRSpl computed accurate results efficiently, in comparison to the other SVM solvers. In particular, for a large dataset that was used in the KDD 2010 challenge, DRSpl was more than 60 times faster than LIBLINEAR. DRSpl is designed to use very less communication bandwidth in a parallel computing cluster and is suitable for frameworks like MapReduce.

We demonstrate that our algorithms can be easily modified to perform non-negative matrix factorization (NMF). We propose approximate extreme points NMF (AENMF) to efficiently compute NMF. Our empirical results indicate that AENMF computed matrix factors several orders of magnitude faster than other NMF solvers. In addition, when combined with another NMF solver, it computed matrix factorizations with very less approximation error.
Support vector machines (SVMs) [21] are some of the most established classification algorithms today. Its popularity can be attributed to a number of qualities. SVMs typically have good generalization ability as a consequence of the principle of statistical risk minimization. Its ability to use non-linear kernels efficiently and to give sparse solutions are highly desired for many applications. However, in spite of all these favorable characteristics, the application of SVMs to problems with large datasets is impeded by its large training time requirements.

Training time complexity for SVMs with non-linear kernels is typically quadratic in the size of the training dataset [69]. 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 [32].

Modern linear SVM solvers are very fast with linear time complexities [25, 68]. In addition, classification with linear SVMs is simpler than with non-linear SVMs and is computationally fast. However, with web-scale datasets that are very large and high dimensional, even linear SVMs are difficult to train. Typically, applications that use these datasets run on parallel programming frameworks such as MapReduce [22], to reduce the computational requirements. However, linear SVM solvers are generally iterative and difficult to parallelize.

With the advent of applications with very large datasets, such as in data mining, there is an imperative need for SVM solvers that overcome the above mentioned disadvantages of SVMs. In this dissertation we have proposed an alternative optimization problem that yields a solution similar to that of the SVM problem, while addressing the
difficulties in using SVMs with large datasets. In the next section, we describe the standard SVM binary classification problem. Next, we introduce our alternative problem formulation and compare it to SVM. Then we briefly outline the other chapters in this dissertation.

1.1 Support Vector Machines

Consider a two class dataset of \( \mathbf{X} = \{ \mathbf{x}_i : \mathbf{x}_i \in \mathbb{R}^D, i = 1, 2, \ldots, N \} \), and the corresponding target labels \( \mathbf{Y} = \{ y_i : y_i \in [-1, 1], i = 1, 2, \ldots, N \} \). The SVM primal problem is given below:

\[
\min_{\mathbf{w}, b, \xi} F_1(\mathbf{w}, b, \xi) = \frac{1}{2} \| \mathbf{w} \|^2 + \frac{C}{N} \sum_{i=1}^{N} \xi_i
\]

subject to \( 1 - y_i(\mathbf{w}^T \phi(x_i) + b) \leq \xi_i, \xi_i \geq 0 \quad \forall x_i \in \mathbf{X} \)

where \( \phi : \mathbb{R}^D \to \mathcal{H}, b \in \mathbb{R}, \) and \( \mathbf{w} \in \mathcal{H} \), a Hilbert space

Note that SVM formulations where the penalty parameter \( C \) is divided by \( N \) have been used extensively \([28, 36, 66]\). These formulations enable improved analysis of the scaling of \( C \) with \( N \) \([35]\). Formulations of the form given above, where \( \sum_{i=1}^{N} \xi_i \) is minimized, are also called L1-SVMs as described in Chapter 2.

In this article, we use the unconstrained problem formulation of (1–1) given below \([68, 74]\):

\[
\min_{\mathbf{w}, b} F_1(\mathbf{w}, b) = \frac{1}{2} \| \mathbf{w} \|^2 + \frac{C}{N} \sum_{i=1}^{N} l(\mathbf{w}, b, \phi(x_i))
\]

where \( l(\mathbf{w}, b, \phi(x_i)) = \max \{ 0, 1 - y_i(\mathbf{w}^T \phi(x_i) + b) \}, \forall x_i \in \mathbf{X} \)

Here \( l(\mathbf{w}, b, \phi(x_i)) \) is the hinge loss of \( x_i \). The SVM formulations in (1–1) and (1–2) are equivalent. We use the form in (1–2) as it leads to simpler proofs for the theorems in Chapter 3.2.

The problem in (1–2) requires optimization over \( N \) variables. In general, the computation time of SVM solvers will reduce if the size of the training dataset is reduced.
This is the motivation behind the alternative optimization problem proposed in the next section.

1.2 Approximate Extreme Point Support Vector Machine

The proposed problem formulation, approximate extreme point support vector machine (AESVM), optimizes over only a subset of the training dataset. The selected subset is called the representative set $\mathbf{Z}^*$ and is comprised of an approximation to the extreme points of the convex hull of $\mathbf{Z}$, where $\mathbf{Z} = \{\mathbf{z}_i : \mathbf{z}_i = \phi(x_i), \forall x_i \in \mathbf{X}\}$. A detailed definition of the approximate extreme points is given in Chapter 3.1. The formulation of AESVM is given below:

$$\min_{w,b} F_2(w, b) = \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{t=1}^{M} \beta_t I(w, b, z_t)$$

(1-3)

where $z_t \in \mathbf{Z}^*$, $w \in \mathbb{H}$, and $b \in \mathbb{R}$

Here $M$ is the number of vectors in the representative set $\mathbf{Z}^*$. The constants $\beta_t$ are defined in (3–7). We will prove in Chapter 3.2 that:

(a) $F_1(w_1^*, b_1^*) - F_2(w_2^*, b_2^*) \leq C \sqrt{C \epsilon}$, where $(w_1^*, b_1^*)$ and $(w_2^*, b_2^*)$ are the solutions of (1–2) and (1–3) respectively, where $\epsilon$ is a parameter that specifies the maximum approximation error.

(b) Under the assumptions given in Corollary 4, $F_1(w_2^*, b_2^*) - F_1(w_1^*, b_1^*) \leq 2 C \sqrt{C \epsilon}$

(c) 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 (1–3) yields a solution close to that of (1–2). 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.
1.3 Outline

This dissertation is organized as follows. In Chapter 2, we describe several widely used SVM solvers and their characteristics. This is followed in Chapter 3, by the definition of the representative set and the theoretical properties of the AESVM problem that indicate its similarity to the SVM problem. Next, we present efficient algorithms to compute the representative set and analyze its computational complexity. In Chapter 5, we present the experimental results that illustrate the feasibility of using AESVM as an alternative for the SVM formulation. In Chapter 6 we present an algorithm to improve classification time complexity of SVMs. In Chapter 7, we describe a linear AESVM solver that is designed for parallel programming frameworks, and discuss the feasibility of using this algorithm to compute non-negative matrix factorization of web-scale datasets. Finally, in Chapter 8, we discuss some conclusions and future work.
A multitude of methods have been proposed to efficiently solve the SVM optimization problem. SVMs require special algorithms, as standard optimization algorithms such as interior point methods [8, 68] have large memory and training time requirements that make it infeasible for large datasets. In the following sections we first discuss the most widely used strategies used to solve the SVM optimization problem. Then we discuss methods that enable fast SVM classification.

2.1 SVM Solvers

SVM solvers can be broadly divided into two categories as described below:

2.1.1 Dual Optimization

The SVM primal problem in (1–1) is a convex optimization problem with strong duality [8]. Hence its solution can be arrived at by solving the dual formulation of (1–1) given below:

\[
\max_{\alpha} L_1(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j)
\]

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

Here \(K(x_i, x_j) = \phi(x_i)^T \phi(x_j)\), is the kernel product [65] of the data vectors \(x_i\) and \(x_j\), and \(\alpha\) is a vector of all variables \(\alpha_i\). Solving the dual problem is computationally simpler, especially for non-linear kernels. Hence, a majority of the algorithms use dual optimization to solve SVMs. Typical dual optimization algorithms are based on the Karush-Kuhn-Tucker (KKT) conditions, that are necessary and sufficient for the optimality of a solution of (2–1). Given below is a discussion of some of the major dual optimization algorithms.
2.1.1.1 Decomposition methods

Decomposition methods [56] have been widely used to solve (2–1). These methods optimize over a subset of the training dataset, called the ‘working set’, at each algorithm iteration. SVM\textsuperscript{light} [34] and SMO [60] are popular examples of decomposition methods. Both these methods have a quadratic time complexity for linear and non-linear SVM kernels [69].

SVM\textsuperscript{light} lets the user decide the working set size. SMO, on the other hand, uses the smallest possible working set size of two, which greatly simplifies the algorithm. It iteratively minimizes $L_1(\alpha)$ by modifying a pair of $\alpha_i$ values [26] in each iteration, under the constraints in (2–1). Both the algorithms terminate when the KKT conditions are met [39]. Heuristics such as shrinking and caching [34] enable fast convergence of decomposition methods and reduce their memory requirements. LIBSVM [15] and SVM\textsuperscript{light} software libraries are heavily cited implementations of SMO and SVM\textsuperscript{light} respectively. A class of related methods optimizes over only the active set [63] at each algorithm iteration. The active set is comprised of the unbounded dual variables $\alpha_i \in (0, \frac{C}{N})$. Active set methods are typically suited for incremental training of SVMs [71].

2.1.1.2 Approximate methods

Approximations of the Gram matrix using methods such as Drineas and Mahoney [23], Fine and Scheinberg [27] have been proposed to increase training speed and reduce memory requirements of SVM solvers. The Gram matrix is the square matrix composed of the $N \times N$ kernel products $K(x_i, x_j), \forall x_i, x_j \in X$. The approximations of the Gram matrix can be used with interior point methods to solve (2–1) faster.

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 [59] to select the subsets. In Yu et al. [81] 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 Cervantes et al. [13] to remove data vectors that are unlikely to contribute to the SVM training.

Bordes et al. [7] proposed the algorithm LASVM \(^1\) that uses *active selection* techniques to train SVMs on a subset of the training dataset. *Random sampling* of training data is another approach followed by such methods. Lee and Mangasarian [45] 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 uses a \(J \times N\) kernel matrix, where \(J\) is the number of vectors in the subset. The L2-SVM differs from L1-SVM of (1–1), as it minimizes the \(l^2\)-norm of \(\xi\) instead of its \(l^1\)-norm as shown below.

\[
\min_{w, b; \xi} F_4(w, b, \xi) = \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{i=1}^{N} \xi_i^2 \\
\text{subject to } 1 - y_i(w^T \phi(x_i) + b) \leq \xi_i, \quad \xi_i \geq 0 \quad \forall x_i \in X
\]

A *core set* can be loosely defined \(^2\) as the subset of \(X\) for which the solution of an optimization problem such as (2–2) has a solution similar to that for the entire dataset \(X\). Fast SVM training using core sets has been vigorously researched in recent years.

Tsang et al. [76] proved that the L2-SVM is a reformulation of the minimum enclosing ball problem. 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 Tsang et al. [75], where only an enclosing ball is computed. Gärtner and Jaggi [31] proposed an algorithm to solve the L1-SVM problem, by computing the shortest distance between two polytopes [3] using core sets. However,

\(^1\) LASVM also supports online SVM training. A discussion on online SVM training is beyond the scope of this proposal

\(^2\) A formal definition can be found in Clarkson [19]
there are no published results on solving L1-SVM with non-linear kernels using their algorithm.

Another method used to approximately solve (2–2) is to map the data vectors into a randomized feature space that is relatively low dimensional compared to the kernel space $\mathbb{H}$ [61]. This effectively reduces the non-linear SVM problem into the simpler linear SVM problem, enabling the use of faster linear SVM solvers. Shi et al. [70] extended this work by using hash functions to perform the mapping of data vectors.

2.1.1.3 Coordinate descent

A dual coordinate descent SVM solver modifies one variable $\alpha_i$ per algorithm iteration, to arrive at the optimal $\alpha$ value. Similarly a coordinate descent primal form solver modifies one component of $w$ per algorithm iteration. These algorithms have been proposed primarily for the linear kernel. Chang et al. [16] solves a primal L2-SVM formulation similar to (2–2) using coordinate descent. Hsieh et al. [33] solves a dual L1-SVM formulation similar to (2–1) and their implementation LIBLINEAR [25] is reported to be among the fastest linear SVMs solvers today.

2.1.2 Primal Optimization

A majority of early SVM solvers were based on dual optimization as the primal SVM problem is non-differentiable compared to the doubly differentiable dual problem. Furthermore, applications that require non-linear kernels can typically be solved only using dual optimizers. Recently, with the advent of applications with high-dimensional datasets such as text classification, linear SVMs are being increasingly applied. This has lead to a surge in publications on efficient primal SVM solvers that are mostly used for linear SVMs. To overcome the difficulties caused by the non-differentiability of the primal problem, one of the methods given below is used.

2.1.2.1 Stochastic sub-gradient descent

Stochastic gradient descent methods iteratively update the optimization variable by computing the gradient at some data vector $x_i$. Similarly, stochastic sub-gradient
descent [85] uses the sub-gradient computed at some data vector $\mathbf{x}$, to iteratively update $\mathbf{w}$. Shalev-Shwartz et al. [68] proposed a stochastic sub-gradient descent SVM solver, Pegasos, that is reported to be among the fastest linear SVM solvers. It has been demonstrated that the run-time of Pegasos has an inverse dependence to the dataset size for linear SVMs [69].

2.1.2.2 Cutting plane algorithm

Cutting plane algorithms [40] solve the primal problem by successively tightening a piecewise linear approximation. It was employed by Joachims [35] to solve linear SVMs with their implementation $\text{SVM}^{\text{perf}}$. This work was generalized [36] to include non-linear SVMs by approximately estimating $\mathbf{w}$ with arbitrary basis vectors using the fix point iteration method (Section 18.2.2 in Schölkopf and Smola [65]). Teo et al. [74] proposed a related method for linear SVMs that fixed some stability issues in the cutting plane methods.

2.1.2.3 Smoothing of objective function

Another widely used method to solve the SVM primal problem is to replace the hinge loss function with a smooth approximation [46]. Zhou et al. [86], and Ouyang and Gray [58] use Nesterov’s smoothing method [52] to solve SVM using gradient descent algorithms. These algorithms have the advantage of a theoretically proven, fast convergence rate. Chapelle [17] approximated the SVM primal problem by using the Huber loss function instead of hinge loss and then solved it using Newton’s method.

2.2 Fast SVM Classification

Many of the methods described in the previous sections of this chapter, achieve fast classification in addition to fast SVM training. The training set selection methods, core set methods and RSVM have fast classification as a result of the decreased training set size, similar to our observations with AESVM. The cutting plane algorithm $\text{SVM}^{\text{perf}}$, enables the user to pick the number of basis vectors used to approximate $\mathbf{w}$. Thus it is capable of producing extremely sparse solutions resulting in fast classification. However,
the algorithm in Joachims and Yu [36] does not bound the approximation error, denoted as $\delta$, that results from this selection of arbitrary basis vectors.

To reduce the classification time, many studies modify the SVM solution to reduce the number of support vectors. Reduced set methods [9, 10, 53, 64] approximate $w$ as a linear combination of other vectors. Wu et al. [79] computes a sparse approximation of $w$ by introducing an additional constraint to the SVM primal problem. Nguyen et al. [54] modified the SVM training algorithm to iteratively combine pairs of data vectors. Osuna and Girosi [57] uses a reformulation of the SVM primal problem to obtain solutions that were observed to be sparser in some cases. Zhan and Shen [84] performs several SVM training runs to obtain a sparse solution. Most of these methods were intended only to produce sparse representations of $w$ and not to reduce the SVM training complexity. In fact, most of these methods reduce classification time without any reduction in training time and sometimes at the expense of additional processing of the SVM solution.

Bakır et al. [2] presented a method to obtain fast SVM solutions that are also sparse. Keerthi et al. [38] presented a greedy strategy for fast L2-SVM training that builds upon the previously mentioned methods to obtain sparse, although approximate, solutions. However, both these methods do not have any theoretical bounds on the approximation error.
As mentioned in the introduction, AESVM is an optimization problem on a subset of the training dataset called the representative set. In this chapter 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 (1–2).

3.1 Definition of the Representative Set

The convex hull of a set \( \mathbf{X} \) is the smallest convex set containing \( \mathbf{X} \) \([62]\) 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 \( EP(\mathbf{X}) \):

\[
\mathbf{x}_i = \sum_{\mathbf{x}_t \in EP(\mathbf{X})} \pi_{i,t} \mathbf{x}_t, \text{ where } 0 \leq \pi_{i,t} \leq 1, \text{ and } \sum_{\mathbf{x}_t \in EP(\mathbf{X})} \pi_{i,t} = 1
\]

For the example illustrated in Figure 3-1, \( \mathbf{x}_4 \) and \( \mathbf{x}_5 \) can be represented as convex combinations of the other vectors.

Figure 3-1. The vectors \( \mathbf{x}_1, \mathbf{x}_2, \) and \( \mathbf{x}_3 \) are the extreme points of the dataset

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 \( \{\pi_{i,t}\} \). 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, \( K(x_i, x_j) = \phi(x_i)^T \phi(x_j) = 1 \). This implies that all the data vectors lie on the surface of the unit ball \(^1\) 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:

\[
Z = \{ z_i : z_i = \phi(x_i), \forall x_i \in X \} \tag{3-1}
\]

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 \( \epsilon \) be a small positive real number. For notational simplicity, we assume \( N \) is divisible by \( V \). Let \( Z_l \) be subsets of \( Z \) for \( l = 1, 2, \ldots, \left( \frac{N}{V} \right) \), such that \( Z = \bigcup Z_l \) and \( Z_l \cap Z_m = \emptyset \) for \( l \neq m \), where \( m = 1, 2, \ldots, \left( \frac{N}{V} \right) \).

We require that the subsets \( Z_l \) satisfy \( |Z_l| = V \), \( \forall l \) and

\[
\forall z_i, z_j \in Z_l, \text{ we have } y_i = y_j \tag{3-2}
\]

where \( |Z_l| \) denotes the cardinality of \( Z_l \). Let \( Z_{l_0} \) be an arbitrary subset of \( Z_l \), \( Z_{l_0} \subseteq Z_l \).

Next, for any \( z_i \in Z_l \) we define:

\[
f(z_i, Z_{l_0}) = \min_{\mu} ||z_i - \sum_{z_t \in Z_{l_0}} \mu_{i,t} z_t||^2 \tag{3-3}
\]

s.t. \( 0 \leq \mu_{i,t} \leq 1 \), and \( \sum_{z_t \in Z_{l_0}} \mu_{i,t} = 1 \)

A subset \( Z_l^* \) is said to be an \( \epsilon \) - approximate extreme points subset of \( Z_l \) if:

\[
\max_{z_i \in Z_l} f(z_i, Z_l^*) \leq \epsilon \tag{3-4}
\]

\(^1\) We define the square of the distance \( x \) from origin in kernel space as \( K(x, x) \).
We will drop the prefix $\epsilon$ for simplicity and refer to $Z^*_i$ as approximate extreme points subset. Note that it is not unique. Intuitively, its cardinality will be related to computational savings obtained using the approach proposed in this paper. We have chosen to not use approximate extreme points subset of smallest cardinality to maintain flexibility.

It can be seen that $\mu_{i,t}$ for $z_t \in Z^*_i$ are analogous to the convex combination weights $\pi_{i,t}$ for $x \in EP(X)$. The representative set $Z^*$ of $Z$ is the union of the sets of approximate extreme points of its subsets $Z_i$.

$$Z^* = \bigcup_{i=1}^{N} Z^*_i$$

The representative set has properties that are similar to $EP(X)$. Given any $z_i \in Z$, we can find $Z_i$ such that $z_i \in Z_i$. Let $\gamma_{i,t} = \{\mu_{i,t} \text{ for } z_t \in Z^*_i \text{ and } z_i \in Z_i, \text{ and } 0 \text{ otherwise}\}$. Now using (3–3), we can write:

$$z_i = \sum_{z_t \in Z^*_i} \gamma_{i,t} z_t + \tau_i \quad (3–5)$$

Here $\tau_i$ is a vector that accounts for the approximation error $f(z_i, Z_{i'})$ in (3–3). From (3–3)-(3–5) we can conclude that:

$$\|\tau_i\|^2 \leq \epsilon \forall z_i \in Z \quad (3–6)$$

Since $\epsilon$ will be set to a very small positive constant, we can infer that $\tau_i$ is a very small vector. The weights $\gamma_{i,t}$ are used to define $\beta_t$ in (1–3) as:

$$\beta_t = \sum_{i=1}^{N} \gamma_{i,t} \quad (3–7)$$

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

24
3.2 Properties of AESVM

Consider the following optimization problem.

\[
\min_{w, b} F_3(w, b) = \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{i=1}^{N} l(w, b, u_i)
\]  \hspace{1cm} (3–8)

where \(u_i = \sum_{t=1}^{M} \gamma_{i, t} z_t, z_t \in Z^*, w \in H\), and \(b \in \mathbb{R}\).

We use the problem in \((3–8)\) as an intermediary between \((1–2)\) and \((1–3)\). The intermediate problem \((3–8)\) 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:

\[
\max(0, A + B) \leq \max(0, A) + \max(0, B)
\]  \hspace{1cm} (3–9)

\[
\max(0, A - B) \geq \max(0, A) - \max(0, B)
\]  \hspace{1cm} (3–10)

\[
\sum_{i=1}^{N} \max(0, c_i A) = \max(0, A) \sum_{i=1}^{N} c_i
\]  \hspace{1cm} (3–11)

for \(A, B, c_i \in \mathbb{R}\) and \(c_i \geq 0\).

**Theorem 1** Let \(F_3(w, b)\) and \(F_2(w, b)\) be as defined in \((3–8)\) and \((1–3)\) respectively.

Then,

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

**Proof.** Let \(\mathcal{L}_2(w, b, X^*) = \frac{C}{N} \sum_{t=1}^{M} l(w, b, z_t) \sum_{i=1}^{N} \gamma_{i, t}\) and \(\mathcal{L}_3(w, b, X^*) = \frac{C}{N} \sum_{i=1}^{N} l(w, b, u_i)\), where \(u_i = \sum_{t=1}^{M} \gamma_{i, t} z_t\). From the properties of \(\gamma_{i, t}\) in \((3–3)\), and from \((3–2)\) we get:

\[
\mathcal{L}_3(w, b, X^*) = \frac{C}{N} \sum_{i=1}^{N} \max \left[ 0, \left\{ 1 - y_i (w^T \sum_{t=1}^{M} \gamma_{i, t} z_t + b) \right\} \right]
\]  \hspace{1cm} (3–12)

\[
= \frac{C}{N} \sum_{i=1}^{N} \max \left[ 0, \sum_{t=1}^{M} \gamma_{i, t} \left\{ 1 - y_t (w^T z_t + b) \right\} \right]
\]
Using properties (3–9) and (3–11) we get:

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

\[
= \frac{C}{N} \sum_{t=1}^{M} \max \left[ 0, 1 - y_t (\mathbf{w}^T \mathbf{z}_t + b) \right] \sum_{i=1}^{N} \gamma_{i,t}
\]

\[
= \mathcal{L}_2(\mathbf{w}, b, \mathbf{X}^*)
\]

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 (1–2) and (3–8) respectively. Then,

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

\[\forall \mathbf{w} \in \mathbb{H} \text{ and } b \in \mathbb{R}, \text{ where } \tau_i \in \mathbb{H} \text{ is the vector defined in (3–5).}\]

**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 (1–2) and \(\mathcal{L}_3(\mathbf{w}, b, \mathbf{X}^*)\) be as defined in Theorem 1. Using (3–5) and (1–2) we get:

\[
\mathcal{L}_1(\mathbf{w}, b, \mathbf{X}) = \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, 1 - y_i (\mathbf{w}^T \mathbf{z}_i + b) \right\}
\]

\[
= \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, 1 - y_i (\mathbf{w}^T \left( \sum_{t=1}^{M} \gamma_{i,t} \mathbf{z}_t + \tau_i \right) + b) \right\}
\]
From the properties of $\gamma_{i,t}$ in (3–3), and from (3–2) we get:

$$L_1(w, b, X) = \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, \sum_{t=1}^{M} \gamma_{i,t} \left(1 - y_i(w^T z_t + b)\right) - y_i w^T \tau_i \right\} \quad (3-13)$$

Using (3–9) on (3–13), we get:

$$L_1(w, b, X) \leq \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, \sum_{t=1}^{M} \gamma_{i,t} \left(1 - y_i(w^T z_t + b)\right) \right\} + \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, -y_i w^T \tau_i \right\}$$

$$= L_3(w, b, X^*) + \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, -y_i w^T \tau_i \right\}$$

Using (3–10) on (3–13), we get:

$$L_1(w, b, X) \geq \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, \sum_{t=1}^{M} \gamma_{i,t} \left(1 - y_i(w^T z_t + b)\right) \right\} - \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, y_i w^T \tau_i \right\}$$

$$= L_3(w, b, X^*) - \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, y_i w^T \tau_i \right\}$$

From the two inequalities above we get,

$$L_3(w, b, X^*) - \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, y_i w^T \tau_i \right\} \leq L_1(w, b, X) \leq L_3(w, b, X^*) + \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, -y_i w^T \tau_i \right\}$$

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

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

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

**Corollary 3** Let $(w_1^*, b_1^*)$ be the solution of (1–2) and $(w_2^*, b_2^*)$ be the solution of (1–3).

Then,

$$F_1(w_1^*, b_1^*) - F_2(w_2^*, b_2^*) \leq C \sqrt{C \epsilon}$$
Proof. It is known that $\|w_1^*\| \leq \sqrt{C}$ (refer Theorem 1 in Shalev-Shwartz et al. [68]). It is straightforward to see that the same result also applies to AESVM, $\|w_2^*\| \leq \sqrt{C}$. Based on (3–6) we know that $\|\tau_i\| \leq \sqrt{C}$. From Theorem 2 we get:

$$F_1(w_2^*, b_2^*) - F_3(w_2^*, b_2^*) \leq \frac{C}{N} \sum_{i=1}^{N} \max \{0, -y_i w_2^T \tau_i\} \leq \frac{C}{N} \sum_{i=1}^{N} \|w_2^*\| \|\tau_i\|$$

$$\leq \frac{C}{N} \sum_{i=1}^{N} \sqrt{C} \epsilon = C \sqrt{C} \epsilon$$

Since $(w_1^*, b_1^*)$ is the solution of (1–2), $F_1(w_1^*, b_1^*) \leq F_1(w_2^*, b_2^*)$. Using this property and Theorem 1 in the inequality above, we get:

$$F_3(w_1^*, b_1^*) - F_2(w_2^*, b_2^*) \leq F_1(w_1^*, b_1^*) - F_3(w_2^*, b_2^*)$$

$$\leq F_1(w_2^*, b_2^*) - F_3(w_2^*, b_2^*) \leq C \sqrt{C} \epsilon \quad (3–14)$$

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:

$$\max_{\alpha} L_2(\hat{\alpha}) = \sum_{t=1}^{M} \hat{\alpha}_t - \frac{1}{2} \sum_{t=1}^{M} \sum_{s=1}^{M} \hat{\alpha}_t \hat{\alpha}_s y_t y_s z_t^T z_s$$

subject to $0 \leq \hat{\alpha}_t \leq \frac{C}{N} \sum_{i=1}^{N} \gamma_{i,t}$ and $\sum_{t=1}^{M} \hat{\alpha}_t y_t = 0$ \quad (3–15)

The dual form of the intermediate problem is given by:

$$\max_{\alpha} L_3(\hat{\alpha}) = \sum_{i=1}^{N} \hat{\alpha}_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \hat{\alpha}_i \hat{\alpha}_j y_i y_j u_i^T u_j$$

subject to $0 \leq \hat{\alpha}_i \leq \frac{C}{N}$ and $\sum_{i=1}^{N} \hat{\alpha}_i y_i = 0$ \quad (3–16)

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

$$h(\hat{\alpha}) = \{\hat{\alpha}_t : \hat{\alpha}_t = \sum_{i=1}^{N} \gamma_{i,t} \hat{\alpha}_i\}$$

(3–17)
It can be seen that the objective functions $L_2(h(\tilde{\alpha}))$ and $L_3(\tilde{\alpha})$ are identical.

\[
L_2(h(\tilde{\alpha})) = \sum_{t=1}^{M} \tilde{\alpha}_t - \frac{1}{2} \sum_{t=1}^{M} \sum_{s=1}^{M} \tilde{\alpha}_t \tilde{\alpha}_s y_t y_s z_t^T z_s
\]

\[
= \sum_{i=1}^{N} \tilde{\alpha}_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \tilde{\alpha}_i \tilde{\alpha}_j y_i y_j u_i^T u_j
\]

\[
= L_3(\tilde{\alpha})
\]

It is also straight forward to see that, for any feasible $\tilde{\alpha}$ of (3–16), $h(\tilde{\alpha})$ is a feasible point of (3–15) as it satisfies the constraints in (3–15). However, the converse is not always true. With that clarification, we present the following corollary.

**Corollary 4** Let $(w_1^*, b_1^*)$ be the solution of (1–2) and $(w_2^*, b_2^*)$ be the solution of (1–3). Let $\hat{\alpha}_2$ be the dual variable corresponding to $(w_2^*, b_2^*)$. Let $h(\hat{\alpha}_2)$ be as defined in (3–17). If there exists an $\tilde{\alpha}_2$ such that $\hat{h}(\tilde{\alpha}_2) = \hat{\alpha}_2$ and $\tilde{\alpha}_2$ is a feasible point of (3–16), then,

\[
F_1(w_2^*, b_2^*) - F_1(w_1^*, b_1^*) \leq 2C\sqrt{C}\epsilon
\]

**Proof.** Let $(w_3^*, b_3^*)$ be the solution of (3–8) and $\tilde{\alpha}_3$ the solution of (3–16). We know that $L_3(\tilde{\alpha}_2) = L_2(\hat{\alpha}_2) = F_2(w_2^*, b_2^*)$ and $L_3(\tilde{\alpha}_3) = F_3(w_3^*, b_3^*)$. Since $L_3(\tilde{\alpha}_3) \geq L_3(\tilde{\alpha}_2)$, we get

\[
F_3(w_3^*, b_3^*) \geq F_2(w_2^*, b_2^*)
\]

But, from Theorem 1 we know $F_3(w_3^*, b_3^*) \leq F_3(w_2^*, b_2^*) \leq F_2(w_2^*, b_2^*)$. Hence

\[
F_3(w_3^*, b_3^*) = F_3(w_2^*, b_2^*)
\]

From the above result we get

\[
F_3(w_2^*, b_2^*) - F_3(w_1^*, b_1^*) \leq 0 \quad (3–18)
\]
From Theorem 2 we have the following inequalities:

\[
- \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, y_i \mathbf{w}_1^T \tau_i \right\} \leq F_1(\mathbf{w}_1^*, b_1^*) - F_3(\mathbf{w}_1^*, b_1^*) \tag{3-19}
\]

\[
F_1(\mathbf{w}_2^*, b_2^*) - F_3(\mathbf{w}_2^*, b_2^*) \leq \frac{C}{N} \sum_{i=1}^{N} \max \left\{ 0, -y_i \mathbf{w}_2^T \tau_i \right\} \tag{3-20}
\]

Adding (3–19) and (3–20) we get:

\[
F_1(\mathbf{w}_2^*, b_2^*) - F_1(\mathbf{w}_1^*, b_1^*) \leq R + \frac{C}{N} \sum_{i=1}^{N} \left[ \max \left\{ 0, -y_i \mathbf{w}_2^T \tau_i \right\} + \max \left\{ 0, y_i \mathbf{w}_1^T \tau_i \right\} \right] \tag{3-21}
\]

where \( R = F_3(\mathbf{w}_2^*, b_2^*) - F_3(\mathbf{w}_1^*, b_1^*) \). Using (3–18) and the properties \( \| \mathbf{w}_2^* \| \leq \sqrt{C} \) and \( \| \mathbf{w}_1^* \| \leq \sqrt{C} \) in (3–21):

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

\[
\leq \frac{C}{N} \sum_{i=1}^{N} \| \mathbf{w}_2^* \| \| \tau_i \| + \| \mathbf{w}_1^* \| \| \tau_i \|
\]

\[
\leq \frac{C}{N} \sum_{i=1}^{N} 2\sqrt{C} \epsilon = 2C \sqrt{C} \epsilon
\]

\[\square\]

Now we prove a relationship between AESVM and the Gram matrix approximation methods mentioned in Chapter 2.

**Corollary 5** Let \( L_1(\alpha) \), \( L_3(\tilde{\alpha}) \), and \( F_2(\mathbf{w}, b) \) be the objective functions of the SVM dual (2–1), intermediate dual (3–16) and AESVM (1–3) respectively. Let \( \mathbf{z}_i, \tau_i, \) and \( \mathbf{u}_i \) be as defined in (3–1), (3–5), and (3–8) respectively. Let \( \mathbf{G} \) and \( \tilde{\mathbf{G}} \) be the \( N \times N \) matrices with \( G_{ij} = y_i y_j \mathbf{z}_i^T \mathbf{z}_j \) and \( \tilde{G}_{ij} = y_i y_j \mathbf{u}_i^T \mathbf{u}_j \) respectively. Then for any feasible \( \tilde{\alpha}, \alpha, \mathbf{w}, \) and \( b \):

1. \hspace{1cm} Rank of \( \tilde{\mathbf{G}} = M \), \( L_1(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \alpha \mathbf{G} \alpha^T \), \( L_3(\tilde{\alpha}) = \sum_{i=1}^{N} \tilde{\alpha}_i - \frac{1}{2} \tilde{\alpha} \tilde{\mathbf{G}} \tilde{\alpha}^T \), and \( \text{Trace}(\mathbf{G} - \tilde{\mathbf{G}}) \leq N \epsilon + 2 \sum_{t=1}^{M} \sum_{i=1}^{N} \gamma_{i,t} \tau_i \).
2. \( F_2(w, b) \geq L_3(\bar{x}) \)

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

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

Applying \( u_i = \sum_{t=1}^{M} \gamma_{i,t} z_t, \forall z_t \in \mathbf{Z}^* \) to the definition of \( \mathbf{\tilde{G}} \), we get:

\[
\mathbf{\tilde{G}} = \Gamma \mathbf{A} \Gamma^T
\]

Here \( \mathbf{A} \) is the \( M \times M \) matrix comprised of \( A_{zs} = y_i y_s z_i^T z_s \), \( \forall z_t, z_s \in \mathbf{Z}^* \) and \( \Gamma \) is the \( N \times M \) matrix with the elements \( \Gamma_{it} = \gamma_{i,t} \). Hence the rank of \( \mathbf{\tilde{G}} = M \) and intermediate dual problem (3–16) is a low rank approximation of the SVM dual problem (2–1).

The Gram matrix approximation error can be quantified using (3–5) and (3–6) as:

\[
\text{Trace}(\mathbf{G} - \mathbf{\tilde{G}}) = \sum_{i=1}^{N} \left[ z_i^T z_i - \left( \sum_{t=1}^{M} \gamma_{i,t} z_t \right) ^T \left( \sum_{s=1}^{M} \gamma_{i,s} z_s \right) \right] = \sum_{i=1}^{N} \left[ \tau_i^T \tau_i + 2 \sum_{t=1}^{M} \gamma_{i,t} z_t^T \tau_i \right] \leq N \epsilon + 2 \sum_{t=1}^{M} z_t^T \sum_{i=1}^{N} \gamma_{i,t} \tau_i
\]

By the principle of duality, we know that \( F_3(w, b) \geq L_3(\bar{x}), \forall w \in \mathbb{H} \) and \( b \in \mathbb{R} \), where \( \bar{x} \) is any feasible point of (3–16). Using Theorem 1 on the inequality above, we get

\[
F_2(w, b) \geq L_3(\bar{x}), \forall w \in \mathbb{H}, b \in \mathbb{R} \text{ and feasible } \bar{x}
\]

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

Based on the theoretical results in this chapter, it is reasonable to suggest that for small values of \( \epsilon \), the solution of AESVM is close to the solution of SVM.
3.3 Comparison to Other SVM Solvers

In this section we compare and contrast AESVM to the SVM solvers described in Chapter 2. As mentioned in the previous section, AESVM has a few properties in common with the methods in Chapter 2.1.1.2. Corollary 3 shows that the difference between the optimal SVM objective value and the optimal AESVM objective value has an upper bound of \( C \sqrt{C \epsilon} \). This claim is of the same theoretical merit as the claim in Fine and Scheinberg [27], where the difference between the optimal SVM objective value and the optimal objective value of the SVM problem on a low-rank approximation of the Gram matrix is shown to be bounded. A similar error bound on an approximation of the Gram matrix was presented in Drineas and Mahoney [23].

The claim in Corollary 4, can be compared to the claim for SVM\textsuperscript{perf} [36], where a similar upper bound was derived for the difference between its solution and the optimal SVM solution. Apart from these methods, the rationale for AESVM is the same as the rationale for training set selection methods such as RSVM and the clustering based methods in Chapter 2.1.1.2. However, unlike those methods we prove several properties of AESVM in Section 3.2 that gives a theoretical justification for our approach. Another important merit of AESVM is that the subset computation has to be performed only once for a given kernel hyper-parameter value. This eliminates repetition of computations while performing grid search. AESVM is a new problem formulation that is almost identical to (but less complex than) the SVM primal problem, and can be solved using most of the existing SVM solvers. In contrast, the other studies are mostly on algorithms that solve the SVM primal or related problems. Methods such as RSVM that also use different problem formulations require special algorithms to solve them unlike AESVM.
CHAPTER 4
COMPUTATION OF THE REPRESENTATIVE SET

In this chapter, 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 Chapter 3.1, we require an algorithm to compute approximate extreme points in kernel space. Osuna and Castro [55] 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 \( O(N \cdot S(N)) \), 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 Osuna and Castro [55] 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 \( \mathbf{X}_q, q = 1, 2, \ldots, Q \), where \( |\mathbf{X}_q| < P, \ Q \geq \frac{N}{P} \) and \( \mathbf{X} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_Q\} \). 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. Since performing such a segregation is computationally expensive, heuristics are used to greatly simplify the process. Instead of computing the distance of all data vectors from each other, only the distance from a few selected data vectors are used to segregate the data in the methods FLS2 and SLS described below.

The 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 below.

### 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 $\Delta'$ and $\Delta_2'$ of $N$ elements. Each element of $\Delta'$ ($\Delta_2'$), $\delta_i$ ($\delta^2_i$), contains the index in $X$ of the last data vector of the subset to which $x_i$ belongs. It is straightforward 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. The set $X'$ denotes any set of data vectors.

1. **FLS1**($X'$, $P$)

   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 $N' = |X'|$.

   

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

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

2. **FLS2**($X'$, $P$)

   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 FLS2, the distance $d_i$ in kernel space of all $x_i \in X'$ from $x_j$ is computed as $d_i = \|\phi(x_i) - \phi(x_j)\|^2 = k(x_i, x_i) + k(x_j, x_j) - 2k(x_i, x_j)$. The algorithm FLS2($X'$, $P$), in effect builds a binary search tree, with each node containing the data.
vector $x_k$ 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 $O\left(\frac{N'}{2}\right)$ nodes and consequently the height of the tree is $O\left(\log_2\frac{N'}{P}\right)$. At each level of the tree the calls to the BFPRT algorithm \[6\] and the rearrangement of the data vectors in steps 2 and 3 are of $O(N')$ time complexity. Hence the overall time complexity of $\text{FLS2}(X', P)$ is $O\left(N' \log_2\frac{N'}{P}\right)$.

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

1. Compute distance $d_i$ in kernel space of all $x_i \in X'$ from the first vector $x_j$ in $X'$
2. Select $x_k$ such that there exists $\frac{|X'|}{2}$ data vectors $x_i \in X'$ with $d_i < d_k$, using the linear time BFPRT algorithm
3. Using $x_k$, rearrange $X'$ as $X' = \{X^1, X^2\}$, where $X^1 = \{x_i : d_i < d_k, x_i \in X'\}$ and $X^2 = \{x_i : x_i \in X' \text{ and } x_i \notin X^1\}$
4. If $\frac{|X'|}{2} \leq P$
   \begin{itemize}
   \item For $i$ where $x_i \in X^1$, set $\delta_i = \text{index of last data vector in } X^1$.
   \item For $i$ where $x_i \in X^2$, set $\delta_i = \text{index of last data vector in } X^2$.
   \end{itemize}
5. If $\frac{|X'|}{2} > P$
   Run FLS2($X^1, P$) and FLS2($X^2, P$)

4.2 Second Level of Segregation

After the initial segregation, another method SLS($X', V, \Delta'$) is used to further segregate each set $X_q$ into smaller subsets $X_{q_i}$ of maximum size $V$, $X_q = \{X_{q_1}, X_{q_2}, \ldots, X_{q_r}\}$, where $V$ is predefined ($V < P$) and $R = \text{ceiling}(\frac{|X_q|}{V})$. The algorithm SLS($X', V, \Delta'$) is given below. In step 2.b, $x_i$ is the data vector in $X_q$ 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 $a_i$ 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 \( X_q \).

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: \( O(P + (P - V) + (P - 2V) + \ldots + V) \Rightarrow O\left(\frac{P^2}{V}\right)\). The overall complexity of \( SLS(X', V, \Delta') \) considering the Q For loop iterations is \( O\left(\frac{N'}{P}\right) \Rightarrow O\left(\frac{N'}{V}\right)\), since \( Q = O\left(\frac{N'}{P}\right)\).

\[
[X',\Delta'_2] = SLS(X', V, \Delta')
\]

1. Initialize \( l = 1 \)
2. For \( q = 1 \) to \( Q \)
   (a) Identify subset \( X_q \) of \( X' \) using \( \Delta' \)
   (b) Set \( a_l = \phi(x_i) \), where \( x_i \in \arg\max_i \|x_i\|^2, x_i \in X_q \)
   (c) Compute distance \( d_i \) in kernel space of all \( x_i \in X_q \) from \( a_l \)
   (d) Select \( x_k \) such that, there exists \( V \) data vectors \( x_i \in X_q \) with \( d_i < d_k \), using the BFPRT algorithm
   (e) Using \( x_k \), rearrange \( X_q \) as \( X_q = \{X^1, X^2\} \), where \( X^1 = \{x_i : d_i < d_k, x_i \in X_q\} \) and \( X^2 = \{x_i : x_i \in X_q \text{ and } x_i \notin X^1\} \)
   (f) For \( i \) where \( x_i \in X^1 \), set \( \delta^2_i = \text{index of last data vector in } X^1 \), where \( \delta^2_i \) is the \( i \)th element of \( \Delta'_2 \)
   (g) Remove \( X^1 \) from \( X_q \)
   (h) If \( |X^2| > V \)
       Set: \( l = l + 1 \) and \( a_l = x_k \)
       Repeat steps 2.c to 2.h
   (i) If \( |X^2| \leq V \)
       For \( i \) where \( x_i \in X^2 \), set \( \delta^2_i = \text{index of last data vector in } X^2 \)

### 4.3 Computation of the Approximate Extreme Points

After computing the subsets \( X_{q_i} \), the algorithm DeriveAE is applied to each \( X_{q_i} \) to compute its approximate extreme points. The algorithm DeriveAE is described below. DeriveAE uses three routines. SphereSet(\( X_{q_i} \)) returns all \( x_i \in X_{q_i} \) that lie on the surface of the smallest hypersphere in kernel space that contains \( X_{q_i} \). It
computes the hypersphere as a hard margin support vector data descriptor (SVDD) [73]. SphereSort($X_q$) returns data vectors $x_i \in X_q$ sorted in descending order of distance in the kernel space from the center of the SVDD hypersphere. CheckPoint($x_i, \psi$) returns TRUE if $x_i$ is an approximate extreme point of the set $\psi$ in kernel space. The operator $A \setminus B$ indicates a set operation that returns the set of the members of $A$ excluding $A \cap B$. The matrix $X_q^*$ contains the approximate extreme points of $X_q$ and $\beta_{q*}$ is a $|X_q^*|$ sized vector.

$$[X_q^*, \beta_{q*}] = \text{DeriveAE}(X_q)$$

1. Initialize: $X_q^* = \text{SphereSet}(X_q)$ and $\psi = \emptyset$
2. Set $\zeta = \text{SphereSort}(X_q \setminus X_q^*)$
3. For each $x_i$, taken in order from $\zeta$, call the routine CheckPoint($x_i, X_q^* \cup \psi$)
   If it returns FALSE, then set $\psi = \psi \cup x_i$
4. Initialize a matrix $\Gamma$ of size $|X_q| \times |X_q^*|$ with all elements set to 0
   Set $\mu_{k,k} = 1 \forall x_k \in X_q^*$, where $\mu_{i,j}$ is the element in the $i^{th}$ row and $j^{th}$ column of $\Gamma$
5. For each $x_i \in X_q$ and $x_i \not\in X_q^*$, execute CheckPoint($x_i, X_q^*$)
   Set the $i^{th}$ row of $\Gamma = \overline{\mu_i}$, where $\overline{\mu_i}$ is the result of CheckPoint($x_i, X_q^*$)
6. For $j = 1$ to $|X_q^*|$
   Set $\beta_{q*} = \sum_{k=1}^{\psi} \mu_{k,j}$

CheckPoint($x_i, \psi$) is computed by solving the following quadratic optimization problem:

$$\min_{\mu} \rho(x_i, \psi) = \|\phi(x_i) - \sum_{t=1}^{\psi} \mu_{i,t} \phi(x_t)\|^2$$

$$\text{s.t. } x_t \in \psi, 0 \leq \mu_{i,t} \leq 1 \text{ and } \sum_{t=1}^{\psi} \mu_{i,t} = 1$$

where $\|\phi(x_i) - \sum_{t=1}^{\psi} \mu_{i,t} \phi(x_t)\|^2 = K(x_i, x_i) + \sum_{t=1}^{\psi} \mu_{i,t} \mu_{i,s} K(x_t, x_s) - 2 \sum_{t=1}^{\psi} \mu_{i,t} K(x_i, x_t)$. If the optimized value of $\rho(x, \psi) \leq \epsilon$, CheckPoint($x, \psi$) returns TRUE and otherwise it returns
FALSE. It can be seen that the formulation of \( \hat{\varphi}(\mathbf{x}, \Psi) \) is similar to (3–3). The value of \( \mu_i \) computed by CheckPoint(\( \mathbf{z}_i, \Psi_0 \)) is used in step 5 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 (2–1). Since DeriveAE solves several SVM training problems in steps 1, 3, and 5, 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 \( O(S(A_\varphi)) \) \(^1\). Here \( A_\varphi \) is the largest value of \( \mathbf{X}_\varphi^* \cup \Psi \) during the run of DeriveAE(\( \mathbf{X}_\varphi \)). 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 \( O(S(A_\varphi)) \). The time complexity of steps 3 and 5 are \( O(V S(A_\varphi)) \) and \( O(A_\varphi S(A_\varphi)) \) respectively. The time complexity of step 2 is \( O(V |\Psi_1| + V \log_2 V) \), where \( \Psi_1 = \text{SphereSet}(\mathbf{X}_\varphi) \). Hence the time complexity of DeriveAE is \( O(V |\Psi_1| + V \log_2 V + V S(A_\varphi)) \). Since \( |\Psi_1| \) is typically very small, we denote the time complexity of DeriveAE by \( O(V \log_2 V + V S(A_\varphi)) \).

4.4 Combining All the Methods to Compute the Representative Set

To derive \( \mathbf{X}^* \), 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 chapter with a few additional steps, to compute the representative set of \( \mathbf{X} \). The complexity of DeriveRS \(^2\) can easily be computed by summing the complexities

---

\(^1\) For SMO based implementations, such as the implementation we used for Chapter 5, typically \( S(A) = O(A^2) \)

\(^2\) We present DeriveRS as one algorithm in spite of its two variants that use FLS1 or FLS2, for simplicity and to conserve space.
of its steps. The complexity of steps 1 and 6 is $O(N)$. The complexity 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 $X_q$ giving a total complexity of $O(N \log_2 V + V \sum_{q=1}^{Q} \sum_{r=1}^{R} S(A_q))$. Here we use the fact that the number of subsets $X_q$ is $O(N)$. Thus the complexity of DeriveRS is $O(N(\frac{P}{V} + \log_2 V) + V \sum_{q=1}^{Q} \sum_{r=1}^{R} S(A_q))$ when FLS1 is used and $O(N(\log_2 \frac{N}{P} + \frac{P}{V} + \log_2 V) + V \sum_{q=1}^{Q} \sum_{r=1}^{R} S(A_q))$ when FLS2 is used.

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

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

We focused our experiments on an SMO [26] 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\(^1\) - 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 Tavallaee et al. [72], 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%.

D2: Localization data for person activity\(^2\) - This dataset has been used in a study on agent-based care for independent living [37]. 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

\(^1\) [http://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data]

\(^2\) [http://archive.ics.uci.edu/ml/datasets/Localization+Data+for+Person+Activity]
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 Nandan et al. [51], where it is used as dataset A.

**D4:** *Forest cover type dataset*\(^3\) - 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 Collobert et al. [20], where a classification of forest cover type 2 from the other cover types was performed.

**D5:** *IJCNN1 dataset*\(^4\) - This dataset was used in IJCNN 2001 generalization ability challenge [14]. 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%

\(^3\) [http://archive.ics.uci.edu/ml/datasets/Covertype](http://archive.ics.uci.edu/ml/datasets/Covertype)

\(^4\) [http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#ijcnn1](http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#ijcnn1)
D6: Adult income dataset\(^5\) - 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 Platt [60].

D7: Epsilon dataset\(^6\) - This is a dataset that was used for 2008 Pascal large scale learning challenge and in Yuan et al. [83]. 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 \(N = 40000\) \(^7\). The testing set has 100000 data vectors.

D8: MNIST character recognition dataset\(^8\) - The widely used dataset [43] 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\(^9\) - This artificial dataset used in Platt [60] 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.

\(^5\) [http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#a9a](http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#a9a)

\(^6\) [http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#epsilon](http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#epsilon)

\(^7\) AESVM and the other SVM solvers are fully capable of training on this dataset. However, the excessive training time makes it impractical to train the solvers on the entire dataset for this paper.

\(^8\) [http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#mnist](http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#mnist)

\(^9\) [http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#w8a](http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#w8a)
5.2 Evaluation of DeriveRS

We began our experiments with an evaluation of the algorithm DeriveRS, described in Chapter 4. The performance of the two methods FLS1 and FLS2 were compared first. DeriveRS was run on D1, D2, D4 and D5 with the parameters $P = 10^4$, $V = 10^3$, $\epsilon = 10^{-2}$, and $g = [2^{-4}, 2^{-3}, 2^{-2}, \ldots, 2^2]$, 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 5-1.

![Figure 5-1](image_url)

Figure 5-1. Performance of variants of DeriveRS with $g = [2^{-4}, 2^{-3}, 2^{-2}, \ldots, 2^2]$, 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 all datasets, FLS2 gave smaller representative sets than FLS1. For D1, DeriveRS with FLS2 was significantly faster and gave much smaller results than FLS1. For D2, D4 and D5, even though the representative sets derived by FLS1 and FLS2 are almost equal in size, FLS1 took noticeably less time. 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. A sharp increase was observed in representative set sizes and computation times for FLS1, when the datasets were randomly rearranged.

Next we investigated the impact of changes in the values of the parameters $P$ and $V$ on the performance of DeriveRS. All combinations of $P = \{10^4, 5 \times 10^4, 10^5, 2 \times 10^5\}$ and $V = \{10^2, 5 \times 10^2, 10^3, 2 \times 10^3, 3 \times 10^3\}$ were used to compute the representative set of D1. The computations were performed for $\epsilon = 10^{-2}$ and $g = 1$. The method FLS2 was used for the first level segregation in DeriveRS. The results are shown in Table 5-1. As expected for an algorithm of time complexity $O(N\log_2 N + \frac{P}{V} + \log_2 V) + V \sum_{q=1r=1}^{Q R} S(A_{q_r})$, 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 $O(\sum_{q=1r=1}^{Q R} A_{q_r}^2)$, which is inferred from the observed decrease of the size of the representative set $M$ ($M \approx \sum_{q=1r=1}^{Q R} A_{q_r}$). 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.

As described in Section 5.3, we compared several SVM training algorithms with our implementation of AESVM. We performed a grid search with all combinations of the SVM hyper-parameters $C' = \{2^{-4}, 2^{-3}, \ldots, 2^6, 2^7\}$ and $g = \{2^{-4}, 2^{-3}, 2^{-2}, \ldots, 2^1, 2^2\}$. The
Table 5-1. The impact of varying $P$ and $V$ on the result of DeriveRS. Each entry gives $\frac{M}{N} \times 100\%$ followed by the computation time in seconds in parenthesis.

<table>
<thead>
<tr>
<th>$P$</th>
<th>$V = 10^2$</th>
<th>$V = 5 \times 10^2$</th>
<th>$V = 10^3$</th>
<th>$V = 2 \times 10^3$</th>
<th>$V = 3 \times 10^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^4$</td>
<td>7.0 (27)</td>
<td>3.0 (51)</td>
<td>2.5 (87)</td>
<td>2.2 (161)</td>
<td>2.1 (233)</td>
</tr>
<tr>
<td>$5 \times 10^4$</td>
<td>6.9 (66)</td>
<td>2.9 (59)</td>
<td>2.4 (92)</td>
<td>2.1 (166)</td>
<td>2.0 (239)</td>
</tr>
<tr>
<td>$10^5$</td>
<td>7.0 (121)</td>
<td>2.9 (69)</td>
<td>2.3 (98)</td>
<td>2.1 (169)</td>
<td>1.9 (248)</td>
</tr>
<tr>
<td>$2 \times 10^5$</td>
<td>6.9 (237)</td>
<td>2.9 (94)</td>
<td>2.3 (110)</td>
<td>2.0 (176)</td>
<td>1.9 (250)</td>
</tr>
</tbody>
</table>

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 $\epsilon$ were investigated, $\epsilon = 10^{-2}, 10^{-3}$ or $10^{-4}$.

The results of the computation for datasets D1 - D5, are shown in Table 5-2. 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 $x_i$ 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 5-2. For a decrease in the value of $\epsilon$, $M$ increases. This is because, for smaller $\epsilon$ fewer $x_i$ would satisfy the condition: optimized $p(x_i, \Psi) \leq \epsilon$ in CheckPoint($x_i, \Psi$). This results in the selection of a larger number of approximate extreme points in DeriveAE.
Table 5-2. The percentage of the data vectors in $X^*$ given by $\frac{M}{N} \times 100$ and its computation time for datasets D1-D5

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>Dataset</th>
<th>$g = \frac{1}{x_{30}}$</th>
<th>$g = \frac{1}{x_{31}}$</th>
<th>$g = \frac{1}{x_{32}}$</th>
<th>$g = \frac{1}{x_{33}}$</th>
<th>$g = 1$</th>
<th>$g = 2^1$</th>
<th>$g = 2^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-2}$</td>
<td>D1</td>
<td>0.9 (139)</td>
<td>1.0 (138)</td>
<td>1.3 (140)</td>
<td>1.7 (147)</td>
<td>2.4 (151)</td>
<td>3.3 (157)</td>
<td>4.6 (163)</td>
</tr>
<tr>
<td></td>
<td>D2</td>
<td>0.6 (12)</td>
<td>0.7 (13)</td>
<td>0.8 (13)</td>
<td>1.2 (13)</td>
<td>1.8 (14)</td>
<td>2.8 (15)</td>
<td>4.7 (17)</td>
</tr>
<tr>
<td></td>
<td>D3</td>
<td>0.6 (79)</td>
<td>0.6 (80)</td>
<td>0.6 (80)</td>
<td>0.6 (79)</td>
<td>0.6 (79)</td>
<td>0.6 (78)</td>
<td>0.6 (78)</td>
</tr>
<tr>
<td></td>
<td>D4</td>
<td>1.3 (55)</td>
<td>1.9 (58)</td>
<td>3.1 (61)</td>
<td>5.1 (68)</td>
<td>8.5 (78)</td>
<td>14.5 (91)</td>
<td>25.2 (111)</td>
</tr>
<tr>
<td></td>
<td>D5</td>
<td>5.6 (7)</td>
<td>10.4 (8)</td>
<td>17.7 (10)</td>
<td>28.1 (12)</td>
<td>42.1 (14)</td>
<td>58.0 (15)</td>
<td>71.0 (15)</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>D1</td>
<td>1.6 (142)</td>
<td>2.2 (149)</td>
<td>3.0 (160)</td>
<td>4.2 (168)</td>
<td>6.0 (188)</td>
<td>8.5 (208)</td>
<td>12.1 (231)</td>
</tr>
<tr>
<td></td>
<td>D2</td>
<td>1.3 (13)</td>
<td>1.8 (14)</td>
<td>2.6 (16)</td>
<td>3.8 (19)</td>
<td>5.7 (23)</td>
<td>8.8 (29)</td>
<td>14.4 (35)</td>
</tr>
<tr>
<td></td>
<td>D3</td>
<td>0.6 (80)</td>
<td>0.6 (79)</td>
<td>0.6 (79)</td>
<td>0.6 (79)</td>
<td>0.5 (80)</td>
<td>0.5 (80)</td>
<td>0.6 (81)</td>
</tr>
<tr>
<td></td>
<td>D4</td>
<td>5.5 (71)</td>
<td>8.6 (86)</td>
<td>13.0 (106)</td>
<td>19.9 (136)</td>
<td>31.1 (172)</td>
<td>48.7 (203)</td>
<td>71.3 (204)</td>
</tr>
<tr>
<td></td>
<td>D5</td>
<td>25.8 (15)</td>
<td>36.4 (19)</td>
<td>49.5 (22)</td>
<td>63.5 (23)</td>
<td>76.2 (22)</td>
<td>86.1 (21)</td>
<td>93.5 (19)</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>D1</td>
<td>3.8 (189)</td>
<td>5.4 (217)</td>
<td>7.7 (253)</td>
<td>10.9 (304)</td>
<td>15.2 (358)</td>
<td>20.4 (418)</td>
<td>26.8 (479)</td>
</tr>
<tr>
<td></td>
<td>D2</td>
<td>3.8 (21)</td>
<td>5.1 (28)</td>
<td>6.9 (40)</td>
<td>9.6 (52)</td>
<td>14.3 (61)</td>
<td>22.8 (79)</td>
<td>35.8 (100)</td>
</tr>
<tr>
<td></td>
<td>D3</td>
<td>0.5 (78)</td>
<td>0.5 (79)</td>
<td>0.5 (80)</td>
<td>0.6 (81)</td>
<td>0.7 (83)</td>
<td>0.9 (86)</td>
<td>1.2 (90)</td>
</tr>
<tr>
<td></td>
<td>D4</td>
<td>19.4 (175)</td>
<td>27.1 (249)</td>
<td>38.1 (333)</td>
<td>54.3 (394)</td>
<td>75.5 (387)</td>
<td>92.6 (310)</td>
<td>98.8 (244)</td>
</tr>
<tr>
<td></td>
<td>D5</td>
<td>56.9 (40)</td>
<td>69.1 (43)</td>
<td>80.1 (41)</td>
<td>88.6 (38)</td>
<td>94.9 (32)</td>
<td>98.3 (26)</td>
<td>99.7 (22)</td>
</tr>
</tbody>
</table>

Table 5-3. The percentage of data vectors in $X^*$ and its computation time for datasets D6-D9 with $\epsilon = 10^{-2}$

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$g = \frac{1}{x_{30}}$</th>
<th>$g = \frac{1}{x_{31}}$</th>
<th>$g = \frac{1}{x_{32}}$</th>
<th>$g = \frac{1}{x_{33}}$</th>
<th>$g = 1$</th>
<th>$g = 2^1$</th>
<th>$g = 2^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>D6</td>
<td>83.1 (12)</td>
<td>83.1 (12)</td>
<td>83.1 (13)</td>
<td>83.1 (12)</td>
<td>83.1 (9)</td>
<td>82.7 (9)</td>
<td>86.0 (9)</td>
</tr>
<tr>
<td>D7</td>
<td>97.2 (317)</td>
<td>99.7 (309)</td>
<td>100.0 (325)</td>
<td>100.0 (332)</td>
<td>100.0 (360)</td>
<td>100.0 (330)</td>
<td>100.0 (280)</td>
</tr>
<tr>
<td>D8</td>
<td>100.0 (97)</td>
<td>100.0 (75)</td>
<td>100.0 (62)</td>
<td>100.0 (63)</td>
<td>100.0 (67)</td>
<td>100.0 (64)</td>
<td>100.0 (64)</td>
</tr>
<tr>
<td>D9</td>
<td>72.2 (21)</td>
<td>72.2 (22)</td>
<td>72.2 (21)</td>
<td>72.7 (17)</td>
<td>72.8 (15)</td>
<td>74.4 (14)</td>
<td>76.1 (15)</td>
</tr>
</tbody>
</table>
The results of applying DeriveRS to the high-dimensional datasets D6-D9 are shown in Table 5-3. It was observed that \( \frac{N}{N} \) was much larger for D6-D9 than for the other datasets. We computed the representative set with \( \epsilon = 10^{-2} \) only, as for smaller values of \( \epsilon \) 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 5-3 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 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\textsuperscript{perf}, and RfeatSVM. A description of these methods is given in Chapter 2. We chose these methods because they are widely cited, their software implementations are freely available and other studies [68] 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\textsuperscript{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 \( C' = \{2^{-4}, 2^{-3}, \ldots, 2^6, 2^7\} \) and \( g = \{2^{-4}, 2^{-3}, 2^{-2}, \ldots, 2^1, 2^2\} \). 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 5-2 and 5-3. 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 5-2 and 5-3. Figure 5-2 shows the graph between training time and classification accuracy for the five algorithms. Figure 5-3 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 \( \epsilon \) was reduced. This is expected given the inverse relation of \( M \) to \( \epsilon \) shown in Tables 5-2 and 5-3. The variation in accuracy with \( \epsilon \) is not very noticeable.

Figures 5-2 and 5-3 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 seven performance metrics given below. These metrics combine the results of all runs of each algorithm into a single value, for each dataset. For the first five metrics, we take LIBSVM as a baseline of comparison, as it gives the most accurate 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, \( \mathcal{F} \) can refer to any SVM algorithm such as AESVM, CVM, LASVM etc.
Figure 5-2. Plot of training time against classification accuracy of the SVM algorithms on D2

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

   \[ ETS = \frac{1}{RS} \sum_{r=1}^{R} \sum_{s=1}^{S} \frac{TL_{s}^{r}}{TF_{s}^{r}} \]

   Here \( TL_{s}^{r} \) and \( TF_{s}^{r} \) are the training times of LIBSVM and \( F \) respectively, in the \( s^{th} \) cross-validation fold with the \( r^{th} \) set of hyper-parameters of grid search.

2. **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 \( TX^* \). For methods other than AESVM and RfeatSVM2 (see Section 5.3.3), \( TX^* = 0 \).

   \[ OTS = \frac{\sum_{r=1}^{R} \sum_{s=1}^{S} TL_{s}^{r}}{\sum_{r=1}^{R} \sum_{s=1}^{S} TF_{s}^{r} + TX^*} \]
Figure 5-3. Plot of classification time, represented by the number of support vectors, against classification accuracy of the SVM algorithms on D2

3.  **Expected classification time speedup, ECS**: The expected speedup in classification time is indicated by:

\[
ECS = \frac{1}{RS} \sum_{r=1}^{R} \sum_{s=1}^{S} \frac{NL_s}{N_{FP}^r}
\]

Here \(NL_s\) and \(N_{FP}^r\) are the number of support vectors in the solution of LIBSVM and \(FP\) respectively.

4.  **Classification time speedup for optimal hyper-parameters, CTS**: The speedup in classification time for the optimal hyper-parameters (hyper-parameters that result in maximum classification accuracy) chosen by grid search is indicated by:

\[
CTS = \frac{\max_{r} \sum_{s=1}^{S} NL_s}{\max_{r} \sum_{s=1}^{S} N_{FP}^r}
\]
5. **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:

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

Here $CL'_{s}$ and $C\mathbb{F}'_{s}$ are the classification accuracy of LIBSVM and $\mathbb{F}$ respectively.

6. **Maximum classification accuracy**: It gives the best classification results of an SVM solver, for the set of SVM hyper-parameters that are tested.

$$\text{max. ac.} = \max_{r} \frac{1}{S} \sum_{s=1}^{S} C\mathbb{F}'_{s}$$

7. **Mean and standard deviation of classification accuracies**: It indicates the classification performance of an SVM solver, that can be expected for arbitrary hyper-parameter values.

$$\text{mean acc.} = \frac{1}{RS} \sum_{r=1}^{R} \sum_{s=1}^{S} C\mathbb{F}'_{s}, \text{ and } \text{std. ac.} = \sqrt{\frac{1}{R} \sum_{r=1}^{R} \left( \frac{1}{S} \sum_{s=1}^{S} C\mathbb{F}'_{s} - \text{mean acc.} \right)^2}$$

The results of the classification performance comparison on datasets D1-D5, are shown in Table 5-4. It was observed that for all tested values of $\epsilon$, 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 41728.8 and 488.5 respectively, which is outstanding. Comparing the results of AESVM for different $\epsilon$ values, we see that $RMSE$ generally improves by decreasing when $\epsilon$ decreases, while the metrics improve by increasing when $\epsilon$ increases. The increase in $ETS$ and $OTS$ is of a larger order than the increase in $RMSE$ when $\epsilon$ 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 $CTS$. 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
Table 5-4. Performance comparison of AESVM, CVM, BVM, LASVM and LIBSVM on datasets D1-D5. AESVM1, AESVM2 and AESVM3 represent the results of AESVM with $\epsilon = 10^{-2}$, $10^{-3}$, and $10^{-4}$ respectively.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Solver</th>
<th>$ETS$</th>
<th>$OTS$</th>
<th>$ECS$</th>
<th>$CTS$</th>
<th>$RMSE$ (x10²)</th>
<th>Max. ac. (x10²)</th>
<th>Mean &amp; std. ac. (x10²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>AESVM1</td>
<td>1188.9</td>
<td>156</td>
<td>5.8</td>
<td>3.3</td>
<td>0.22</td>
<td>94.2</td>
<td>92.4, 0.8</td>
</tr>
<tr>
<td></td>
<td>AESVM2</td>
<td>314.8</td>
<td>50.4</td>
<td>3.8</td>
<td>2.6</td>
<td>0.14</td>
<td>93.6</td>
<td>92.3, 0.7</td>
</tr>
<tr>
<td></td>
<td>AESVM3</td>
<td>72.7</td>
<td>14.7</td>
<td>2.4</td>
<td>1.8</td>
<td>0.06</td>
<td>93.8</td>
<td>92.4, 0.8</td>
</tr>
<tr>
<td></td>
<td>CVM</td>
<td>8.9</td>
<td>6.2</td>
<td>1.2</td>
<td>2.3</td>
<td>0.44</td>
<td>94.1</td>
<td>92.7, 0.8</td>
</tr>
<tr>
<td></td>
<td>BVM</td>
<td>28.6</td>
<td>21.6</td>
<td>2</td>
<td>1.9</td>
<td>0.6</td>
<td>94.4</td>
<td>92.6, 0.9</td>
</tr>
<tr>
<td></td>
<td>LASVM</td>
<td>0.8</td>
<td>0.8</td>
<td>1.1</td>
<td>1</td>
<td>0.12</td>
<td>94.3</td>
<td>92.5, 0.8</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td>0.8</td>
<td>0.8</td>
<td>1.1</td>
<td>1</td>
<td>0.12</td>
<td>94.3</td>
<td>92.5, 0.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Solver</th>
<th>$ETS$</th>
<th>$OTS$</th>
<th>$ECS$</th>
<th>$CTS$</th>
<th>$RMSE$ (x10²)</th>
<th>Max. ac. (x10²)</th>
<th>Mean &amp; std. ac. (x10²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2</td>
<td>AESVM1</td>
<td>6067.6</td>
<td>134.5</td>
<td>77.7</td>
<td>17.8</td>
<td>3.85</td>
<td>76.5</td>
<td>71.1, 3.3</td>
</tr>
<tr>
<td></td>
<td>AESVM2</td>
<td>1202.5</td>
<td>86.1</td>
<td>29</td>
<td>9.4</td>
<td>2.43</td>
<td>76.7</td>
<td>72.4, 3.6</td>
</tr>
<tr>
<td></td>
<td>AESVM3</td>
<td>164.5</td>
<td>21.8</td>
<td>10.9</td>
<td>6.2</td>
<td>1.73</td>
<td>77.4</td>
<td>73.1, 3.6</td>
</tr>
<tr>
<td></td>
<td>CVM</td>
<td>0.7</td>
<td>0.5</td>
<td>4.7</td>
<td>4.3</td>
<td>26.59</td>
<td>70.3</td>
<td>52.2, 0.8</td>
</tr>
<tr>
<td></td>
<td>BVM</td>
<td>0.8</td>
<td>0.5</td>
<td>5</td>
<td>5.6</td>
<td>24.06</td>
<td>67.1</td>
<td>54.6, 0.7</td>
</tr>
<tr>
<td></td>
<td>LASVM</td>
<td>0.2</td>
<td>0.1</td>
<td>1</td>
<td>1</td>
<td>2.18</td>
<td>78.1</td>
<td>73.5, 0.5</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td>78.2</td>
<td>74.1</td>
<td>3</td>
<td>5.5</td>
<td>99.9</td>
<td>99.9</td>
<td>99.8, 0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Solver</th>
<th>$ETS$</th>
<th>$OTS$</th>
<th>$ECS$</th>
<th>$CTS$</th>
<th>$RMSE$ (x10²)</th>
<th>Max. ac. (x10²)</th>
<th>Mean &amp; std. ac. (x10²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3</td>
<td>AESVM1</td>
<td>41728.8</td>
<td>488.5</td>
<td>71.5</td>
<td>64.4</td>
<td>0.2</td>
<td>99.9</td>
<td>99.8, 0.1</td>
</tr>
<tr>
<td></td>
<td>AESVM2</td>
<td>21689.3</td>
<td>468</td>
<td>39.5</td>
<td>51.5</td>
<td>0.1</td>
<td>99.9</td>
<td>99.8, 0.1</td>
</tr>
<tr>
<td></td>
<td>AESVM3</td>
<td>12792</td>
<td>429.9</td>
<td>17.1</td>
<td>36</td>
<td>0.09</td>
<td>99.9</td>
<td>99.8, 0.1</td>
</tr>
<tr>
<td></td>
<td>CVM</td>
<td>60.4</td>
<td>23.9</td>
<td>0.4</td>
<td>0.1</td>
<td>0.33</td>
<td>99.9</td>
<td>99.8, 0.2</td>
</tr>
<tr>
<td></td>
<td>BVM</td>
<td>76.8</td>
<td>22.8</td>
<td>0.6</td>
<td>0.2</td>
<td>0.39</td>
<td>99.9</td>
<td>99.8, 0.2</td>
</tr>
<tr>
<td></td>
<td>LASVM</td>
<td>0.9</td>
<td>0.5</td>
<td>0.6</td>
<td>0.7</td>
<td>55.2</td>
<td>99.9</td>
<td>69.3, 29.9</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td>99.9</td>
<td>99.9</td>
<td>0.1</td>
<td>0.1</td>
<td>99.9</td>
<td>99.9</td>
<td>99.8, 0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Solver</th>
<th>$ETS$</th>
<th>$OTS$</th>
<th>$ECS$</th>
<th>$CTS$</th>
<th>$RMSE$ (x10²)</th>
<th>Max. ac. (x10²)</th>
<th>Mean &amp; std. ac. (x10²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D4</td>
<td>AESVM1</td>
<td>962</td>
<td>34.6</td>
<td>24.5</td>
<td>72.8</td>
<td>1.5</td>
<td>68.3</td>
<td>61.6, 3.1</td>
</tr>
<tr>
<td></td>
<td>AESVM2</td>
<td>68.8</td>
<td>6.1</td>
<td>6.3</td>
<td>17.1</td>
<td>0.7</td>
<td>68.1</td>
<td>61, 3.3</td>
</tr>
<tr>
<td></td>
<td>AESVM3</td>
<td>6.7</td>
<td>2.3</td>
<td>2.3</td>
<td>5</td>
<td>0.3</td>
<td>68.1</td>
<td>60.8, 3.2</td>
</tr>
<tr>
<td></td>
<td>CVM</td>
<td>8</td>
<td>6.2</td>
<td>12.4</td>
<td>28</td>
<td>9.4</td>
<td>63.7</td>
<td>55.5, 3.1</td>
</tr>
<tr>
<td></td>
<td>BVM</td>
<td>6.6</td>
<td>4.4</td>
<td>12.1</td>
<td>8.9</td>
<td>9.44</td>
<td>62.3</td>
<td>54.9, 3.4</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td>68.2</td>
<td>60.6</td>
<td>3.2</td>
<td>3.2</td>
<td>99.9</td>
<td>99.9</td>
<td>99.8, 0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Solver</th>
<th>$ETS$</th>
<th>$OTS$</th>
<th>$ECS$</th>
<th>$CTS$</th>
<th>$RMSE$ (x10²)</th>
<th>Max. ac. (x10²)</th>
<th>Mean &amp; std. ac. (x10²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D5</td>
<td>AESVM1</td>
<td>26.6</td>
<td>4.1</td>
<td>3.3</td>
<td>1.6</td>
<td>0.5</td>
<td>98.8</td>
<td>96.2, 2.6</td>
</tr>
<tr>
<td></td>
<td>AESVM2</td>
<td>3.1</td>
<td>1.8</td>
<td>1.5</td>
<td>0.9</td>
<td>0.39</td>
<td>98.9</td>
<td>96.3, 2.6</td>
</tr>
<tr>
<td></td>
<td>AESVM3</td>
<td>1.3</td>
<td>1.1</td>
<td>1.1</td>
<td>0.9</td>
<td>0.25</td>
<td>99</td>
<td>96.4, 2.6</td>
</tr>
<tr>
<td></td>
<td>CVM</td>
<td>0.3</td>
<td>0.2</td>
<td>0.8</td>
<td>0.6</td>
<td>0.74</td>
<td>99</td>
<td>96.6, 2.5</td>
</tr>
<tr>
<td></td>
<td>BVM</td>
<td>0.5</td>
<td>0.3</td>
<td>1</td>
<td>0.9</td>
<td>0.84</td>
<td>99.1</td>
<td>97, 2</td>
</tr>
<tr>
<td></td>
<td>LASVM</td>
<td>0.6</td>
<td>0.5</td>
<td>1</td>
<td>1.1</td>
<td>0.13</td>
<td>99.2</td>
<td>97, 2</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td>99</td>
<td>96.6</td>
<td>2.4</td>
<td>2.4</td>
<td>99.9</td>
<td>99.9</td>
<td>99.8, 0.1</td>
</tr>
</tbody>
</table>

hours for some hyper-parameter combinations. The five algorithms under comparison were found to give similar maximum classification accuracies for D1, D3 and D5.
For D2 and D4, CVM and BVM gave significantly smaller maximum classification accuracies. Another interesting result is that for D3, the mean and standard deviation of classification accuracy of LASVM was found to be widely different from the other algorithms. For all the tested values of $\epsilon$ 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 earlier, DeriveRS was run with only $\epsilon = 10^{-2}$ for these datasets. The results of the performance comparison are shown in Table 5-5. CVM was found to take longer than 40 hours to train on D6, D7 and D8 with 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 $CTS$ values that are larger than 1 unlike the other algorithms, except for D9 where the $CTS$ value for AESVM was 0.6. However it should be noted that the other methods also had similarly low $CTS$ values for D9. Similar to the results in Table 5-4, LASVM and BVM resulted in very large $RMSE$ values for some datasets. The maximum classification accuracies of all algorithms were similar. On some datasets, BVM and LASVM were observed to give significantly lower mean and higher standard deviation of classification accuracy.

5.3.2 Comparison to SVMperf

SVM$^\text{perf}$ 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 depends on $k$ as $O(k^2)$ per iteration. We first used a value of $k = 1000$ for our experiments, as it has been reported to give good performance [36]. SVM$^\text{perf}$
Table 5-5. Performance comparison of AESVM (with $\epsilon = 10^{-2}$), CVM, BVM, LASVM and LIBSVM on datasets D6-D9

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Solver</th>
<th>$ETS$</th>
<th>$OTS$</th>
<th>$ECS$</th>
<th>$CTS$</th>
<th>$RMSE$  ($x10^2$)</th>
<th>Max. ac.</th>
<th>Mean &amp; std. ac. ($x10^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D6</td>
<td>AESVM</td>
<td>1.5</td>
<td>1.4</td>
<td>1.1</td>
<td>1.2</td>
<td>0</td>
<td>85.1</td>
<td>81.4, 2.8</td>
</tr>
<tr>
<td></td>
<td>BVM</td>
<td>0.6</td>
<td>0.6</td>
<td>1.5</td>
<td>1.2</td>
<td>7.8</td>
<td>85.2</td>
<td>80.2, 8.9</td>
</tr>
<tr>
<td></td>
<td>LASVM</td>
<td>0.8</td>
<td>0.5</td>
<td>1</td>
<td>1.1</td>
<td>0.85</td>
<td>85</td>
<td>81.1, 2.9</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>85.1</td>
<td>81.4, 2.8</td>
</tr>
<tr>
<td>D7</td>
<td>AESVM</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.1</td>
<td>0.01</td>
<td>88.3</td>
<td>85.3, 5.7</td>
</tr>
<tr>
<td></td>
<td>LASVM</td>
<td>0.9</td>
<td>0.7</td>
<td>1</td>
<td>0.9</td>
<td>2.37</td>
<td>88.4</td>
<td>85.2, 6.2</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>88.6</td>
<td>85.7, 4.8</td>
</tr>
<tr>
<td>D8</td>
<td>AESVM</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>99.7</td>
<td>92.3, 3.6</td>
</tr>
<tr>
<td></td>
<td>BVM</td>
<td>4.7</td>
<td>2.6</td>
<td>3.2</td>
<td>3.1</td>
<td>17.55</td>
<td>99.7</td>
<td>88.5, 18.1</td>
</tr>
<tr>
<td></td>
<td>LASVM</td>
<td>1</td>
<td>0.9</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>99.7</td>
<td>92.3, 3.6</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.7</td>
<td>92.3, 3.6</td>
</tr>
<tr>
<td>D9</td>
<td>AESVM</td>
<td>1.4</td>
<td>1.3</td>
<td>1.1</td>
<td>0.6</td>
<td>0</td>
<td>99.5</td>
<td>98.8, 0.8</td>
</tr>
<tr>
<td></td>
<td>CVM</td>
<td>1.4</td>
<td>1.2</td>
<td>1.8</td>
<td>0.3</td>
<td>1</td>
<td>99.5</td>
<td>98.9, 0.8</td>
</tr>
<tr>
<td></td>
<td>BVM</td>
<td>17.5</td>
<td>16.9</td>
<td>4.9</td>
<td>0.6</td>
<td>0.09</td>
<td>99.5</td>
<td>98.9, 0.8</td>
</tr>
<tr>
<td></td>
<td>LASVM</td>
<td>0.6</td>
<td>0.5</td>
<td>2.3</td>
<td>0.1</td>
<td>27.5</td>
<td>99.5</td>
<td>85.5, 23.9</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.5</td>
<td>98.8, 0.8</td>
</tr>
</tbody>
</table>

was tested on datasets D1, D4, D5, D6, D8 and D9, with the Gaussian kernel and the same hyper-parameter grid as described earlier. The results of the grid search are presented in Table 5-6. The results of our experiments on AESVM (with $\epsilon = 10^{-2}$) and LIBSVM are repeated in Table 5-6 for ease of reference. The maximum, mean and standard deviation of classification accuracies are represented as max. ac., mean & std. ac. respectively.

Based on the results obtained for $k = 1000$, other values of $k$ were also tested. For datasets D1, D4 and D5, though $SVM_{\text{perf}}$ gave classification accuracies similar to the that of LIBSVM and AESVM, the training times were similar to or higher than the training times of LIBSVM. To test the ability of $SVM_{\text{perf}}$ to give fast training, we also tested it

---

10 We used the software parameters `-t 2 -w 9 -i 2 -b 0` as suggested in the author’s website
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Solver</th>
<th>$ETS$</th>
<th>$OTS$</th>
<th>$ECS$</th>
<th>$CTS$</th>
<th>$RMSE$ (x10^2)</th>
<th>Max. ac.</th>
<th>Mean &amp; std. ac. (x10^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>AESVM</td>
<td>1188.9</td>
<td>156</td>
<td>5.8</td>
<td>3.3</td>
<td>0.22</td>
<td>94.2</td>
<td>92.4, 0.8</td>
</tr>
<tr>
<td></td>
<td>SVM$^{perf}$ k=400</td>
<td>6.7</td>
<td>1.6</td>
<td>17</td>
<td>6.6</td>
<td>0.89</td>
<td>93.9</td>
<td>92.7, 0.4</td>
</tr>
<tr>
<td></td>
<td>SVM$^{perf}$ k=1000</td>
<td>3.7</td>
<td>0.9</td>
<td>2.6</td>
<td>2.6</td>
<td>0.74</td>
<td>94</td>
<td>92.7, 0.5</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>93.9</td>
<td>92.4, 0.8</td>
</tr>
<tr>
<td>D4</td>
<td>AESVM</td>
<td>962</td>
<td>34.6</td>
<td>24.5</td>
<td>72.8</td>
<td>1.5</td>
<td>68.3</td>
<td>61.6, 3.1</td>
</tr>
<tr>
<td></td>
<td>SVM$^{perf}$ k=400</td>
<td>10.2</td>
<td>3.7</td>
<td>467.1</td>
<td>694.3</td>
<td>3.7</td>
<td>68.4</td>
<td>62.9, 2.2</td>
</tr>
<tr>
<td></td>
<td>SVM$^{perf}$ k=1000</td>
<td>3.1</td>
<td>1.2</td>
<td>186.8</td>
<td>277.7</td>
<td>2.14</td>
<td>68.1</td>
<td>61.8, 2.7</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>68.2</td>
<td>60.6, 3.2</td>
</tr>
<tr>
<td>D5</td>
<td>AESVM</td>
<td>26.6</td>
<td>4.1</td>
<td>3.3</td>
<td>1.6</td>
<td>0.5</td>
<td>98.8</td>
<td>96.2, 2.6</td>
</tr>
<tr>
<td></td>
<td>SVM$^{perf}$ k=400</td>
<td>0.8</td>
<td>0.4</td>
<td>14.6</td>
<td>8.2</td>
<td>2.9</td>
<td>98.8</td>
<td>96.5, 2.4</td>
</tr>
<tr>
<td></td>
<td>SVM$^{perf}$ k=1000</td>
<td>0.2</td>
<td>0.1</td>
<td>5.8</td>
<td>3.3</td>
<td>0.26</td>
<td>99</td>
<td>96.7, 2.4</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99</td>
<td>96.6, 2.4</td>
</tr>
<tr>
<td>D6</td>
<td>AESVM</td>
<td>1.5</td>
<td>1.4</td>
<td>1.1</td>
<td>1.2</td>
<td>0</td>
<td>85.1</td>
<td>81.4, 2.8</td>
</tr>
<tr>
<td></td>
<td>SVM$^{perf}$ k=1000</td>
<td>1.1</td>
<td>0.9</td>
<td>20</td>
<td>12.1</td>
<td>9.39</td>
<td>85.2</td>
<td>79.6, 10.7</td>
</tr>
<tr>
<td></td>
<td>SVM$^{perf}$ k=2000</td>
<td>0.3</td>
<td>0.2</td>
<td>10</td>
<td>6</td>
<td>6.5</td>
<td>85.1</td>
<td>80.1, 7.8</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>85.1</td>
<td>81.4, 2.8</td>
</tr>
<tr>
<td>D8</td>
<td>AESVM</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>99.7</td>
<td>92.3, 3.6</td>
</tr>
<tr>
<td></td>
<td>SVM$^{perf}$ k=1000</td>
<td>37.6</td>
<td>23.8</td>
<td>49</td>
<td>9.9</td>
<td>54.2</td>
<td>99.9</td>
<td>55.7, 42.3</td>
</tr>
<tr>
<td></td>
<td>SVM$^{perf}$ k=3000</td>
<td>3.5</td>
<td>1.2</td>
<td>16.3</td>
<td>3.3</td>
<td>51.4</td>
<td>99.8</td>
<td>59.2, 41.6</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.7</td>
<td>92.3, 3.6</td>
</tr>
<tr>
<td>D9</td>
<td>AESVM</td>
<td>1.4</td>
<td>1.3</td>
<td>1.1</td>
<td>0.6</td>
<td>0</td>
<td>99.5</td>
<td>98.8, 0.8</td>
</tr>
<tr>
<td></td>
<td>SVM$^{perf}$ k=1000</td>
<td>1.2</td>
<td>0.9</td>
<td>21.3</td>
<td>3</td>
<td>22.6</td>
<td>99.2</td>
<td>86.1, 18.8</td>
</tr>
<tr>
<td></td>
<td>SVM$^{perf}$ k=2000</td>
<td>0.4</td>
<td>0.3</td>
<td>10.7</td>
<td>1.5</td>
<td>20.6</td>
<td>99.4</td>
<td>87.3, 17.3</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.5</td>
<td>98.8, 0.8</td>
</tr>
</tbody>
</table>
with $k = 400$ for D1, D4 and D5. For the high dimensional datasets (D6, D8 and D9), the \( RMSE \) values were significantly higher for \( SVM_{perf} \), while the mean classification accuracy was noticeably lower than AESVM. Considering the possibility that the value of $k = 1000$ is insufficient to result in an accurate solution for these datasets, we tested D6 and D9 with $k = 2000$ and D8 with $k = 3000$. Even though the training time increased significantly with an increase in $k$, the values of \( RMSE \) and the mean and standard deviation of accuracies did not improve significantly. The training time speedup values of \( SVM_{perf} \) are much lower than AESVM for all tested $k$ values for all datasets, except for D8. The maximum accuracies of all the algorithms were similar. Due to the ability of \( SVM_{perf} \) to approximate $w$ with a small set of $k$ vectors, the classification time speedups of \( SVM_{perf} \) are significantly higher than AESVM. However, this approximation comes at the cost of increased training time and sometimes results in a loss of accuracy, as illustrated in Table 5-6.

### 5.3.3 Comparison to RfeatSVM

Rahimi and Recht [61] 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 first investigated the classification 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. This combination of RfeatSVM and LIBSVM is denoted as RfeatSVM1. 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 RfeatSVM1. The random Fourier features method, described in Algorithm 1 of Rahimi and Recht [61], was used to project the datasets D1, D5, D6 and D9 into a randomized feature space of dimension $E$. 

56
Table 5-7. Performance comparison of RfeatSVM1 (RfeatSVM solved using LIBSVM), AESVM (with $\epsilon = 10^{-2}$), and LIBSVM

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Solver</th>
<th>$RMSE$ (x10^2)</th>
<th>Max. ac. (x10^2)</th>
<th>Mean &amp; std. ac. (x10^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>AESVM</td>
<td>0.25</td>
<td>93.5</td>
<td>92.2, 0.9</td>
</tr>
<tr>
<td></td>
<td>RfeatSVM1 $E = 100$</td>
<td>56.18</td>
<td>37.8</td>
<td>36.1, 1.3</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td>93.6</td>
<td>92.3, 0.9</td>
</tr>
<tr>
<td>D5</td>
<td>AESVM</td>
<td>0.9</td>
<td>98.6</td>
<td>95.7, 2.8</td>
</tr>
<tr>
<td></td>
<td>RfeatSVM1 $E = 100$</td>
<td>5.3</td>
<td>94.7</td>
<td>91.6, 1.4</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td>98.9</td>
<td>96.2, 2.7</td>
</tr>
<tr>
<td>D6</td>
<td>AESVM</td>
<td>0.16</td>
<td>85.1</td>
<td>81.2, 2.9</td>
</tr>
<tr>
<td></td>
<td>RfeatSVM1 $E = 1000$</td>
<td>4</td>
<td>81.6</td>
<td>78, 2.2</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td>85</td>
<td>81.3, 3</td>
</tr>
<tr>
<td>D9</td>
<td>AESVM</td>
<td>0.15</td>
<td>99.3</td>
<td>98.6, 0.8</td>
</tr>
<tr>
<td></td>
<td>RfeatSVM1 $E = 1000$</td>
<td>0.6</td>
<td>98.7</td>
<td>97.4, 0.6</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td>99.5</td>
<td>98.8, 0.9</td>
</tr>
</tbody>
</table>

The results of the accuracy comparison are given in Table 5-7. 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 5-7 for AESVM and LIBSVM were computed for this smaller grid. We selected the number of dimensions (E) of the randomized feature space for D1 and D6 based on Rahimi and Recht [61]. The maximum accuracy for RfeatSVM1 was found to be much less than AESVM and LIBSVM for all datasets. The $RMSE$ values for RfeatSVM1 were significantly higher than AESVM and mean accuracy noticeably lower for most datasets, especially for D1 and D6.

Next we investigated the training and classification time requirements of RfeatSVM by solving it using the fast linear SVM solver LIBLINEAR [25], referred to as RfeatSVM2 in the remainder of this paper. The entire hyper-parameter grid used in the previous sections were used in this experiment. The results of the performance comparison of RfeatSVM2, AESVM and LIBSVM are presented in Table 5-8. The classification time shown in Table 5-8 is the time taken for classification when the SVM solver was trained.
Table 5-8. Performance comparison of RfeatSVM2 (RfeatSVM solved using LIBLINEAR), AESVM (with $\epsilon = 10^{-2}$), and LIBSVM

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Solver</th>
<th>$ETS$</th>
<th>$OTS$</th>
<th>Classification time (s)</th>
<th>$RMSE$ (x10$^2$)</th>
<th>Max. ac.</th>
<th>Mean &amp; std. ac. (x10$^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>AESVM</td>
<td>1188.9</td>
<td>156</td>
<td>6.1</td>
<td>0.22</td>
<td>94.2</td>
<td>92.4, 0.8</td>
</tr>
<tr>
<td></td>
<td>RfeatSVM2</td>
<td>176.3</td>
<td>56.4</td>
<td>0.9</td>
<td>50.3</td>
<td>63.5</td>
<td>43.7, 12.9</td>
</tr>
<tr>
<td></td>
<td>E = 100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RfeatSVM2</td>
<td>77.5</td>
<td>47.7</td>
<td>4.4</td>
<td>43.4</td>
<td>89.3</td>
<td>56, 24.1</td>
</tr>
<tr>
<td></td>
<td>E = 500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td>15</td>
<td>93.9</td>
<td></td>
<td></td>
<td>92.4</td>
<td>0.8</td>
</tr>
<tr>
<td>D5</td>
<td>AESVM</td>
<td>26.6</td>
<td>4.1</td>
<td>9.7</td>
<td>0.5</td>
<td>98.8</td>
<td>96.2, 2.6</td>
</tr>
<tr>
<td></td>
<td>RfeatSVM2</td>
<td>80.7</td>
<td>9.2</td>
<td>0.9</td>
<td>38.6</td>
<td>90.5</td>
<td>64.4, 20</td>
</tr>
<tr>
<td></td>
<td>E = 100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RfeatSVM2</td>
<td>33.2</td>
<td>6.5</td>
<td>4.5</td>
<td>30.9</td>
<td>90.5</td>
<td>70.8, 15.5</td>
</tr>
<tr>
<td></td>
<td>E = 500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RfeatSVM2</td>
<td>18.4</td>
<td>3.6</td>
<td>13.8</td>
<td>31.5</td>
<td>90.5</td>
<td>70.2, 17.8</td>
</tr>
<tr>
<td></td>
<td>E = 1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RfeatSVM2</td>
<td>3.9</td>
<td>0.85</td>
<td>64.5</td>
<td>33.8</td>
<td>90.5</td>
<td>70.2, 19.8</td>
</tr>
<tr>
<td></td>
<td>E = 5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td>16.8</td>
<td>99</td>
<td></td>
<td></td>
<td>96.6</td>
<td>2.4</td>
</tr>
<tr>
<td>D6</td>
<td>AESVM</td>
<td>1.5</td>
<td>1.4</td>
<td>16</td>
<td>0</td>
<td>85.1</td>
<td>81.4, 2.8</td>
</tr>
<tr>
<td></td>
<td>RfeatSVM2</td>
<td>205.7</td>
<td>43.9</td>
<td>2.1</td>
<td>27.8</td>
<td>75.3</td>
<td>54.9, 9.7</td>
</tr>
<tr>
<td></td>
<td>E = 1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RfeatSVM2</td>
<td>48.8</td>
<td>8.9</td>
<td>10.7</td>
<td>29.1</td>
<td>76.4</td>
<td>53.1, 8.1</td>
</tr>
<tr>
<td></td>
<td>E = 5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RfeatSVM2</td>
<td>24.8</td>
<td>5.1</td>
<td>30.9</td>
<td>28.5</td>
<td>76.4</td>
<td>54, 9.2</td>
</tr>
<tr>
<td></td>
<td>E = 10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td>30.5</td>
<td>85.1</td>
<td></td>
<td></td>
<td>81.4</td>
<td>2.8</td>
</tr>
<tr>
<td>D9</td>
<td>AESVM</td>
<td>1.4</td>
<td>1.3</td>
<td>10.5</td>
<td>0</td>
<td>99.5</td>
<td>98.8, 0.8</td>
</tr>
<tr>
<td></td>
<td>RfeatSVM2</td>
<td>245.1</td>
<td>50</td>
<td>2.9</td>
<td>36.9</td>
<td>92.8</td>
<td>63.3, 9.9</td>
</tr>
<tr>
<td></td>
<td>E = 1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RfeatSVM2</td>
<td>57.4</td>
<td>12</td>
<td>15.3</td>
<td>39</td>
<td>95.1</td>
<td>61.5, 11.2</td>
</tr>
<tr>
<td></td>
<td>E = 5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RfeatSVM2</td>
<td>28.9</td>
<td>6.5</td>
<td>45.5</td>
<td>37.4</td>
<td>96.3</td>
<td>63.8, 12.9</td>
</tr>
<tr>
<td></td>
<td>E = 10000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td>5.1</td>
<td>99.5</td>
<td></td>
<td></td>
<td>98.8</td>
<td>0.8</td>
</tr>
</tbody>
</table>

with its optimal hyper-parameters. For RfeatSVM2 the classification time includes the time taken to derive the random Fourier features of the test vectors.
The classification time for RfeatSVM2 was generally less than AESVM, for small values of E. Moreover, it was found that RfeatSVM2 has significantly higher training time speed-ups than AESVM for small values of E, except for D1 where AESVM was much faster. However, with increasing E the classification time and training time increased to more than AESVM for most datasets. For all datasets, the RMSE, and maximum, mean and standard deviation of accuracy of RfeatSVM2 were significantly worse than AESVM. Increasing the number of dimensions E, resulted in only a slight improvement in the classification performance of RfeatSVM2. An important observation was that the projected datasets were found to be almost 100% dense, which results in large memory requirements for RfeatSVM1 and RfeatSVM2. Even though, technically the value of E can be increased arbitrarily, its value is practically limited by the memory requirements of RfeatSVM.

### 5.3.4 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, \( k(x_1, x_2) = (1 + x_1^T x_2)^d \). 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 5-9. The parameters for DeriveRS were \( P = 10^5 \), \( V = 10^3 \) and \( \epsilon = 10^{-2} \), and the first level segregation was performed using FLS2. The performance comparison of AESVM and LIBSVM with the polynomial kernel is shown in Table 5-10. 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.

### 5.4 Discussion of Results

*The experimental results presented in this chapter demonstrate that the solutions of AESVM and SVM are similar in terms of the resulting classification accuracy.* A
Table 5-9. Results of DeriveRS for the polynomial kernel. Each entry gives $\frac{x}{\gamma} \times 100\%$ followed by the computation time in seconds in parenthesis.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$d = 2$</th>
<th>$d = 3$</th>
<th>$d = 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>8.0 (109)</td>
<td>13.2 (199)</td>
<td>26.0 (638)</td>
</tr>
<tr>
<td>D4</td>
<td>20.1 (67)</td>
<td>48.0 (260)</td>
<td>81.3 (1166)</td>
</tr>
<tr>
<td>D6</td>
<td>87.8 (11)</td>
<td>84.0 (13)</td>
<td>91.0 (14)</td>
</tr>
</tbody>
</table>

Table 5-10. Performance comparison of AESVM (with $\epsilon = 10^{-2}$), and LIBSVM with the polynomial kernel.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Solver</th>
<th>ETS</th>
<th>OTS</th>
<th>ECS</th>
<th>CTS</th>
<th>RMSE (x10^2)</th>
<th>Max. ac. (x10^2)</th>
<th>Mean &amp; std. ac. (x10^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>AESVM</td>
<td>21.1</td>
<td>6.4</td>
<td>2.7</td>
<td>2.6</td>
<td>0.13</td>
<td>93.9</td>
<td>93.4, 0.4</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>94.1</td>
<td>93.5, 0.4</td>
</tr>
<tr>
<td>D4</td>
<td>AESVM</td>
<td>7</td>
<td>1.6</td>
<td>2.6</td>
<td>1.9</td>
<td>0.8</td>
<td>64.9</td>
<td>61.2, 2.7</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>64.5</td>
<td>60.7, 2.5</td>
</tr>
<tr>
<td>D6</td>
<td>AESVM</td>
<td>3.8</td>
<td>5.3</td>
<td>1.1</td>
<td>1.1</td>
<td>0.04</td>
<td>84.6</td>
<td>81.0, 2.4</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>84.6</td>
<td>81.0, 2.3</td>
</tr>
</tbody>
</table>

Figure 5-4. Plot of mean classification accuracy of all SVM solvers.
Figure 5-5. Plot of maximum classification accuracy of all SVM solvers

↑OTS value of AESVM for D3 is 488.5

Figure 5-6. Plot of overall training time speedup (compared to LIBSVM) of all SVM solvers
summary of the experimental results in this chapter, that compared an SMO based AESVM implementation, CVM, BVM, LASVM, LIBSVM, SVM\textsuperscript{perf} (with \(k = 1000\)) and RfeatSVM1, is presented in Figures 5-4 to 5-7. The results of RfeatSVM2 are omitted from Figures 5-4 to 5-7, for ease of representation. *It can be seen that AESVM typically gave classification performance similar to LIBSVM, while giving highest overall training time speedup (OTS).* Even though RfeatSVM2 gave higher OTS values in some cases, the degradation in classification accuracy was worse than in RfeatSVM1 as shown in Tables 5-7 and 5-8. **AESVM also gave competitively high classification time speedup for the optimal hyper-parameters (CTS) in comparison with the other algorithms except SVM\textsuperscript{perf} and RfeatSVM2.** It was found that the maximum classification accuracies of all the algorithms except RfeatSVM1 and RfeatSVM2 were similar. RfeatSVM1 and RfeatSVM2, and in some cases CVM and BVM, gave lower maximum classification accuracies. 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 5.3.4.

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 D7 and D8 the representative set computed by DeriveRS was almost 100% of the training set. A similar result was reported for this dataset in Beygelzimer et al. [5], 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. 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. Based on the presented results, we suggest the parameters $\epsilon = 10^{-2}$, $P = 10^5$ and $V = 10^3$ for DeriveRS.

\[\text{11 This is indicated by the large expansion constant for D8 illustrated in Beygelzimer et al. [5]}\]
CHAPTER 6
FAST CLASSIFICATION USING APPROXIMATE EXTREME POINTS

The theoretical properties of AESVM presented in Chapter 3 were derived for the hinge loss function. Now, we briefly discuss the application of approximate extreme points to other optimization problems of the following form.

$$\min_{w,b} \quad G(w) = \frac{1}{C}g(w) + \sum_{i=1}^{N} L(z_i, y_i, w)$$

(6–1)

where $g(w)$ represents a regularization function of the variable $w$, and $L(x_i, y_i, w)$ represents the loss associated with the data vector $x_i$ and target $y_i$. When using other loss functions such as the exponential loss $^1$ or the squared hinge loss of L2-SVM given in (2–2), the theorems in Chapter 3 cannot be applied. Even though approximate extreme points might enable fast computation of the solution of 6–1 with such loss functions, it does not seem to have any theoretical guarantees. For such cases, the algorithm DRSpost can be used to post process the solution computed by other standard algorithms to achieve fast classification.

6.1 Description of DRSpost

The SVM discriminating function is specified by $w$ and $b$, as

$$f(x) = w^T \phi(x) + b$$

(6–2)

For non-linear kernels, $w$ is typically represented as a weighted sum

$$w = \sum_{r=1}^{R} \alpha_r y_r \phi(x_r)$$

(6–3)

Here $x_r$ are the support vectors (SVs), $R$ is the number of SVs, and $\alpha_r$ are non-negative weights resulting from Lagrangian optimization. We propose a post-processing algorithm DRSpost, to reduce the number of support vectors in the solution computed by

---

$^1$ The exponential loss function is of the form $L(z_i, y_i, w) = \exp(w^T z_i - y_i)$
SVM. DRSpost can be used with any algorithm that uses discriminating functions of the form $(6–2)$.

DRSpost is a simple extension of DeriveRS. In DRSpost, the set of transformed data vectors $V = \{v_j : v_j = \alpha_j \phi(x_j), \forall x_j \in SVs\}$ is used. This leads to the use of a modified kernel $k'(x_i, x_j)$, during algorithm execution, given by:

$$k'(x_i, x_j) = \alpha_i \phi(x_i^T) \alpha_j \phi(x_j) = \alpha_i \alpha_j k(x_i, x_j)$$

where $k(x_i, x_j)$ is the kernel used in SVM training. We define the post-processing representative set $V^*$, as given below:

$$V^* \subseteq V$$

$$\|v_r - \sum_{v_s \in V^*, y_s = y_r} \gamma_s^r v_s\|^2 \leq \rho$$

$$\text{s.t. } 0 \leq \gamma_s^r \leq 1, \text{ and } \sum_{v_s \in V^*, y_s = y_r} \gamma_s^r = 1$$

The parameter $\rho$ is the maximum approximation error, analogous to $\epsilon$ in $(3–4)$. Hence the $S$ vectors $v_s$ approximately represent the $R$ vectors $v_r$. DRSpost is identical to DeriveRS, except for the use of the modified kernel $k'(x_i, x_j)$. The result of DRSpost is an approximation to $w$, defined as

$$w' = \sum_{s=1}^{S} \delta_s y_s \phi(x_s), \text{ where } \delta_s = \sum_{r=1}^{R} \alpha_s \gamma_s^r$$

(6–6)

where $x_s$ are vectors in the post-processing representative set and $S$ is the size of $V^*$. The error in approximation of $w$ by $w'$, is bounded as described in the following theorem.

**Theorem 6** Let $w'$ be as defined in (6–6) and $w$, the solution of (1–2), be as defined in (6–3). Then,

$$\|w - w'\| \leq R \sqrt{\rho}$$

**Proof.** From (6–6), we know that $w' = \sum_{r=1}^{R} y_r \alpha_r \sum_{s=1}^{S} \gamma_s^r \phi(x_s)$. Using (6–3) and (6–5), we get:
\[ ||w - w'||^2 = \left\| \sum_{r=1}^{R} y_r \left[ \alpha_r \phi(x_r) - \sum_{s=1}^{S} \gamma'_s \alpha_s \phi(x_s) \right] \right\|^2 \]

\[ = \left\| \sum_{r=1}^{R} y_r \left[ v_r - \sum_{s=1}^{S} \gamma'_s v_s \right] \right\|^2 \]

\[ = \sum_{r=1}^{R} \sum_{p=1}^{R} y_r y_p \left[ v_r - \sum_{s=1}^{S} \gamma'_s v_s \right]^T \left[ v_p - \sum_{s=1}^{S} \gamma'_s v_s \right] \]

\[ \leq \sum_{r=1}^{R} \sum_{p=1}^{R} y_r y_p \rho \]

\[ \leq R^2 \rho \]

The last inequality is derived from the fact that for \( y_r, y_p \in [-1, 1] \), \( y_r y_p \leq 1 \).

For small values of \( \rho \), the classification performance resulting from the use of \( w \) and \( w' \) can be expected to be similar. There are other similar methods for post processing the SVM solution to obtain fast classification. The advantages and disadvantages of these methods were discussed in Chapter 2.2. In comparison to those methods, DRSpost has the following advantages.

1. Time complexity that is at most log-linear in the number of support vectors
2. Theoretical error bound given in Theorem 6

### 6.2 Experiments on DRSpost

We modified the DeriveRS implementation evaluated in Chapter 5, to implement DRSpost. The results of LIBSVM training on Datasets D1, D5, D6 and D9 presented in Chapter 5.3, were post processed using DRSpost, with \( \rho = [10^{-2}, 10^{-3}, 10^{-4}] \). We compared the performance of the solutions of DRSpost to SVM\textsuperscript{perf}. We evaluated DRSpost with the hinge loss function as it enables comparisons with the results in the previous chapters.

To evaluate the performance of DRSpost we used the following metrics:

1. *Expected classification time speedup*, ECS
2. **Classification time speedup for optimal hyper-parameters, CTS**

3. **Root mean squared error of classification accuracy, RMSE**

4. **Overall training time speedup, OTS**: It indicates the expected computation time of post processing the LIBSVM solution with DRSpost in comparison to the training time of LIBSVM.

   \[
   OTS = \frac{\sum_{r=1}^{R} \sum_{s=1}^{S} TL'_s}{\sum_{r=1}^{R} \sum_{s=1}^{S} (TL'_s + T'_{r,s})}
   \]

   Here \( T'_{r,s} \) is the computation time of DRSpost and \( TL'_s \) is the training time of LIBSVM, in the \( s^{th} \) cross-validation fold with the \( r^{th} \) set of hyper-parameters of grid search. For SVM\(^{perf} \), \( OTS \) is computed as defined in Chapter 5.3

5. **Maximum classification accuracy**

6. **Mean and standard deviation of classification accuracy**

The definitions of \( ECS, CTS \) and \( RMSE \) are given in Chapter 5.3. Apart from these metrics, we also present plots of classification accuracies against the number of support vectors for each dataset, similar to Figure 5-3.

Figures 6-1 to 6-4, show the graph between the number of support vectors and classification accuracy of DRSpost, SVM\(^{perf} \) and LIBSVM on datasets D1, D5, D6 and D9 respectively. Like in Figure 5-3, only the results of grid points corresponding to combinations of \( C' = \{2^{-4}, 2^{-2}, 1, 2^{4}, 2^{6}\} \) and \( g = \{2^{-4}, 2^{-2}, 1, 2^{2}\} \) are illustrated. It can be seen that with \( \rho = 0.01 \), for some hyper-parameters DRSpost results in low classification accuracy compared to LIBSVM. This is because when the hyper-parameter \( C' \) is small, the values of \( \alpha \) in (6–6) will be very small. As a result of very small \( \alpha \) values, the constraint (6–5) on \( \gamma \) is easy to satisfy for a given \( \rho \). The deterioration in performance of DRSpost for small \( C' \) is illustrated in Figures 6-5 and 6-6. In Figures 6-5 and 6-6, the classification accuracies for all the 84 combinations of \( C' = \{2^{-4}, 2^{-3}, \ldots, 2^{6}, 2^{7}\} \) and \( g = \{2^{-4}, 2^{-3}, 2^{-2}, \ldots, 2^{1}, 2^{2}\} \) are illustrated. It is clear that in many cases where \( C' \) is small, DRSpost gives erroneous results with \( \rho = 10^{-2} \), and sometimes with \( \rho = 10^{-3} \).
The results of the evaluation of DRSpost are presented in Table 6-1. It can be seen that the $OTS$ values of DRSpost are generally very close to 1. This indicates that the time taken to compute DRSpost is not significant compared to the training time of LIBSVM. Moreover, total time taken by DRSpost and LIBSVM is comparable to or lower than the training time of $SVM_{perf}$ in most cases. For D1, $SVM_{perf}$ seems to give better results overall when compared to DRSpost. However, when the other datasets are considered DRSpost seems to generally perform better than $SVM_{perf}$. For D5, D6 and D9, DRSpost with $\rho = 0.001$ and $\rho = 0.0001$, generally has a larger $OTS$ than $SVM_{perf}$, while having lower $RMSE$. For all datasets, DRSpost with $\rho = 0.01$ seems to give large $RMSE$ values. Based on the results in this section, a value of $\rho = 10^{-4}$ is suggested for use with DRSpost.
Figure 6-2. Plot of performance of DRSpost on D5

Figure 6-3. Plot of performance of DRSpost on D6
Figure 6-4. Plot of performance of DRSpost on D9

Figure 6-5. Illustration of variation of DRSpost with different $C'$ values on D1
Table 6-1. Comparison of classification performance of DRSpost, $\text{SVM}^{\text{perf}}$ and LIBSVM. DRSpost1, DRSpost2, DRSpost3 denote DRSpost computed with $\rho = 0.01$, $\rho = 0.001$ and $\rho = 0.0001$ respectively.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Solver</th>
<th>$OTS$</th>
<th>$ECS$</th>
<th>$CTS$</th>
<th>$\text{RMSE}$ (x10^2)</th>
<th>Max. ac.</th>
<th>Mean &amp; std. ac. (x10^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>DRSpost1</td>
<td>1</td>
<td>17.6</td>
<td>1</td>
<td>23.1</td>
<td>93.8</td>
<td>80.0, 20</td>
</tr>
<tr>
<td></td>
<td>DRSpost2</td>
<td>1</td>
<td>5.8</td>
<td>1.1</td>
<td>14.7</td>
<td>93.9</td>
<td>86.8, 13.9</td>
</tr>
<tr>
<td></td>
<td>DRSpost3</td>
<td>1</td>
<td>2.8</td>
<td>1.1</td>
<td>0.8</td>
<td>93.9</td>
<td>92.4, 0.9</td>
</tr>
<tr>
<td></td>
<td>$\text{SVM}^{\text{perf}}$ k=400</td>
<td>1.6</td>
<td>17</td>
<td>6.6</td>
<td>0.89</td>
<td>93.9</td>
<td>92.7, 0.4</td>
</tr>
<tr>
<td></td>
<td>$\text{SVM}^{\text{perf}}$ k=1000</td>
<td>0.9</td>
<td>2.6</td>
<td>2.6</td>
<td>0.74</td>
<td>94</td>
<td>92.7, 0.5</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>93.9</td>
<td>92.4, 0.8</td>
</tr>
<tr>
<td>D5</td>
<td>DRSpost1</td>
<td>0.96</td>
<td>18.7</td>
<td>3</td>
<td>11.1</td>
<td>99</td>
<td>91.1, 11.2</td>
</tr>
<tr>
<td></td>
<td>DRSpost2</td>
<td>0.94</td>
<td>5.3</td>
<td>3</td>
<td>1.7</td>
<td>99</td>
<td>96.1, 3.1</td>
</tr>
<tr>
<td></td>
<td>DRSpost3</td>
<td>0.93</td>
<td>1.6</td>
<td>3</td>
<td>1.4</td>
<td>99</td>
<td>96.5, 2.5</td>
</tr>
<tr>
<td></td>
<td>$\text{SVM}^{\text{perf}}$ k=400</td>
<td>0.4</td>
<td>14.6</td>
<td>8.2</td>
<td>2.9</td>
<td>98.8</td>
<td>96.5, 2.4</td>
</tr>
<tr>
<td></td>
<td>$\text{SVM}^{\text{perf}}$ k=1000</td>
<td>0.1</td>
<td>5.8</td>
<td>3.3</td>
<td>0.26</td>
<td>99</td>
<td>96.7, 2.4</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99</td>
<td>96.6, 2.4</td>
</tr>
<tr>
<td>D6</td>
<td>DRSpost1</td>
<td>0.93</td>
<td>11.5</td>
<td>1.1</td>
<td>4.03</td>
<td>84.5</td>
<td>79.9, 3.6</td>
</tr>
<tr>
<td></td>
<td>DRSpost2</td>
<td>0.87</td>
<td>1.2</td>
<td>1.2</td>
<td>2.94</td>
<td>85.1</td>
<td>81.4, 2.8</td>
</tr>
<tr>
<td></td>
<td>DRSpost3</td>
<td>0.9</td>
<td>20</td>
<td>12.1</td>
<td>9.39</td>
<td>85.2</td>
<td>79.6, 10.7</td>
</tr>
<tr>
<td></td>
<td>$\text{SVM}^{\text{perf}}$ k=1000</td>
<td>0.2</td>
<td>10</td>
<td>6</td>
<td>6.5</td>
<td>85.1</td>
<td>80.1, 7.8</td>
</tr>
<tr>
<td></td>
<td>$\text{SVM}^{\text{perf}}$ k=2000</td>
<td>0.2</td>
<td>10</td>
<td>6</td>
<td>6.5</td>
<td>85.1</td>
<td>81.4, 2.8</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>85.1</td>
<td>81.4, 2.8</td>
</tr>
<tr>
<td>D9</td>
<td>DRSpost1</td>
<td>0.99</td>
<td>27</td>
<td>2.4</td>
<td>10.3</td>
<td>99.5</td>
<td>96.1, 10.1</td>
</tr>
<tr>
<td></td>
<td>DRSpost2</td>
<td>0.98</td>
<td>7</td>
<td>0.9</td>
<td>0.89</td>
<td>99.5</td>
<td>98.6, 1.3</td>
</tr>
<tr>
<td></td>
<td>DRSpost3</td>
<td>0.95</td>
<td>4.9</td>
<td>0.7</td>
<td>0.17</td>
<td>99.5</td>
<td>98.8, 0.9</td>
</tr>
<tr>
<td></td>
<td>$\text{SVM}^{\text{perf}}$ k=1000</td>
<td>0.9</td>
<td>21.3</td>
<td>3</td>
<td>22.6</td>
<td>99.2</td>
<td>86.1, 18.8</td>
</tr>
<tr>
<td></td>
<td>$\text{SVM}^{\text{perf}}$ k=2000</td>
<td>0.3</td>
<td>10.7</td>
<td>1.5</td>
<td>20.6</td>
<td>99.4</td>
<td>87.3, 17.3</td>
</tr>
<tr>
<td></td>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.5</td>
<td>98.8, 0.8</td>
</tr>
</tbody>
</table>
Figure 6-6. Illustration of variation of DRSpost with different $C'$ values on D5
CHAPTER 7
PARALLEL SVM Solver USING APPROXIMATE EXTREME POINTS

With the advent of high dimensional, web-scale datasets, linear SVMs are being increasingly applied. This is because linear SVMs of the form in (7–1), are computationally easier to solve than non-linear SVMs. The bias term $b$ in (1–2) is typically not used in modern linear SVM solvers such as LIBLINEAR [25], to greatly simplify the optimization problem. By removing $b$ from the optimization problem, fast convex optimization techniques such as dual coordinate descent can be used to solve SVMs with linear time complexity.

$$\min_w F_1(w) = \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{i=1}^{N} l(w, x_i)$$

where $l(w, x_i) = \max\{0, 1 - y_i(w^T x_i)\}, \forall x_i \in X$

The linear AESVM problem is:

$$\min_w F_2(w) = \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{t=1}^{M} \beta_t l(w, x_t)$$

where $\beta_t$ is defined in (3–7)

Typically, SVM solvers are iterative in nature and are difficult to implement on parallel programming frameworks. Although a number of approaches have been proposed to solve SVMs in parallel [11, 18, 78, 87, 88], most of them are not suitable to solve linear SVMs on frameworks such as MapReduce [22] with high dimensional sparse datasets. In particular, the need for excessive inter-node communications and a large number of iterations makes it difficult to implement these methods on MapReduce clusters. For example, if the method in Chu et al. [18] is used to solve linear SVMs with high dimensional datasets (with say $D > 10^5$), each iteration would require transmitting a $D$-dimensional vector from each computing node. Such a requirement makes it infeasible to use these methods to solve linear SVMs in parallel on MapReduce.
Frameworks such as message passing interface (MPI) and general-purpose computing on graphics processing units (GPGPU) might support these methods to a greater extent. However, MapReduce has become increasingly popular, due to its scalability, fault tolerance and ease of use.

Recently, there has been great interest in solving machine learning algorithms on MapReduce [24]. The bagging approach in Zinkevich et al. [89] to solve optimization problems is highly suited for MapReduce. However, the theoretical properties of their method do not extend to SVMs [82]. The method in Catak and Balaban [12], divides the training dataset into local datasets for each cluster node. In each iteration, at each node its local dataset is combined with all the support vectors computed at the previous iteration. Then it solves the SVM problem locally at each node. This method has the advantage that it has a convergence proof. However, it is only suitable in cases where the number of support vectors is small.

We propose DRSpl, an extension of DeriveRS, to solve AESVM on frameworks such as MapReduce. Although DRSpl is designed for parallel programming frameworks such as MapReduce, it can be used to efficiently solve linear AESVM in any parallel programming environment. The algorithm is optimized for large and high dimensional datasets and requires very few iterations. DRSpl is distinct from DeriveRS as follows:

1. DRSpl is specifically designed for the linear kernel. This does not impact the applicability of DRSpl, as typically only the linear kernel is used with high dimensional, large datasets.

2. The size of the representative set, $X^*$, can be specified in DRSpl. This feature addresses the shortcomings of DeriveRS with high dimensional datasets that were discussed in Chapter 5. Since the size of $X^*$ is specified, the maximum approximation error is variable in DRSpl.

3. Iterations of DRSpl use an AESVM solver to progressively derive a good approximation of the SVM solution. Hence DRSpl is itself an approximate linear SVM solver.
7.1 Description of DRSpl

The algorithm DRSpl uses two routines, DRSplMaster and DRSplWorker, to be executed at the master and worker nodes of the computing cluster respectively. Since DRSpl is proposed for use with all parallel programming frameworks, it is presented as a general algorithm and not in the form of map and reduce tasks. As explained later in this section, converting the algorithm into forms suitable for different frameworks is straightforward. The routine DRSplMaster is passed the arguments: the SVM hyper-parameter C, number of worker nodes \( WO \), input files \( IF_1, IF_2, \ldots, IF_{WO} \), desired number of vectors \( M \) in the representative set, and the maximum number of algorithm iterations \( K \) as input. It is assumed that the input data is already available as files \( IF_1, IF_2, \ldots, IF_{WO} \). If this is not the case, such files can be easily generated in linear time complexity. There are no assumptions regarding the nature of division of input data into the files \( IF_i \). Even though DRSpl utilizes files to transfer data between the master and worker nodes, it is not necessary that files are used. For example, for a multi-threaded implementation of DRSpl all the data transfer can be performed in memory without using files on the hard disk. In step 2.b of DRSplMaster, the output files \( OF_k \) of all the executed DRSplWorker routines are combined to get the representative set \( \{X^*, Y^*\} \) and its weights \( \beta \).

DRSplMaster outputs the AESVM solution vector \( w \) that is derived after either its convergence criteria is met or the maximum number of algorithm iterations has been reached. DRSplMaster uses the convergence criteria \( \text{DIFF} w(w_k, w_{k-1}) \leq \theta \) in step 2.d, where \( \theta \) is a parameter set to a small positive number. \( \text{DIFF} w(w_k, w_{k-1}) \) indicates the similarity of the solutions \( w_k \) and \( w_{k-1} \) of iterations \( k \) and \( k - 1 \).

\[
\text{DIFF} w(w_k, w_{k-1}) = \frac{||w_k - w_{k-1}||_2^2}{||w_k - w_{k-1}||_2^2} \tag{7–2}
\]

DRSplMaster trains a linear AESVM on a representative set of size \( M \), in each of its iterations. It uses dual coordinate descent to solve linear AESVM, and hence has a time complexity of \( O(MK) \). From the MapReduce perspective, the \( WO \) DRSplWorker calls
would be the map tasks, and the AESVM training in step 2.d of DRSplMaster, would be the only reduce task.

\[ [w] = \text{DRSplMaster}(C, W_0, IF_1, IF_2, \ldots, IF_{W_0}, M, K) \]

1. Set \( w_0 \leftarrow \emptyset \).
2. For \( k = 1 \) to \( K \)
   (a) Execute the routine DRSplWorker(\( IF_a, \frac{|IF_a| \cdot M}{N}, k, w_{k-1} \)) on all the worker nodes, where \( N \) is the total number of vectors in the input files and \( |IF_a| \) is the number of vectors in \( IF_a \).
   (b) Read all the outputs \( OF^k \) from above, to get \( X^*, Y^* \) and its corresponding \( \beta \).
   (c) Train AESVM on \( \{X^*, Y^*\} \) and \( \beta \), to obtain output \( w_k \) with hyper-parameter \( C \).
   (d) IF \( k > 1 \) and \( \text{DIFF}_{WW}(w_k, w_{k-1}) \leq \theta \), then go to step 3.
3. Set \( w \leftarrow w_k \).

The routine DRSplWorker runs on each worker node and derives the representative set \( \{X^*_a, Y^*_a\} \) and \( \beta_a \) of the vectors in the input file \( IF_a \). It is passed the arguments: input file \( IF_a \), desired size of representative set \( M_a \), algorithm iteration \( k \), and the estimated \( w_{k-1} \) vector. In step 2 of DRSplWorker, the function \( f(x_i) = y_i w^T x_i \) gives a value that indicates distance along \( w \) of vectors \( x_i \) from the separating hyperplane of SVM, multiplied by the label. Vectors that have \( f(x_i) \) within the range \([L_{lim}, U_{lim}]\) are added to the set \( \{X_a, Y_a\} \). This heuristic enables the elimination of vectors that are distant from the separating hyperplane. Since only the vectors close to the hyperplane are likely to be significant while solving AESVM, it is expected that this heuristic does not significantly impact the quality of the solution of DRSpl. The values of \( L_{lim} \) and \( U_{lim} \) are predefined. The IF conditions in steps 2 and 3 are necessary because the first iteration of DRSpl has \( w_{k-1} = w_0 \), which is an empty vector.

The steps 4-9 of DRSplWorker are similar to steps 1-6 of DeriveRS described in Chapter 4. In steps 4-6 of DRSplWorker, all vectors in \( X_a \) are segregated using \( FLS \) and \( SLS \) of DeriveRS. The parameters \( P \) and \( V \) of \( FLS \) and \( SLS \) respectively, are computed.
as follows. For a dataset of size \( N' \), with desired size of representative set \( M' \), \( V = \frac{N'}{M'} \kappa \) and \( P = 50V \), where \( \kappa \) is a predefined positive integer. As described in Chapter 4, \( FLS \) and \( SLS \) can be described as coarse and fine methods of data segregation, where \( SLS \) segregates each \( P \) sized subset computed by \( FLS \) into \( V \) sized subsets. The value of \( P = 50V \) was designed to be large enough to allow efficient segregation of \( P \) sized subsets by \( SLS \), without greatly increasing the computational complexity of the data segregation process. \( DRSplWorker \) creates the file \( OF_a^k \) that contains the representative set \( \{X_a^*, Y_a^*\} \) and \( \beta_a \). A major difference between \( DRSpl \) and \( DeriveRS \) is that \( DRSpl \) uses the routine \( DAEpl \) instead of \( DeriveAE \) to compute the approximate extreme points. Unlike \( DeriveAE \), \( DAEpl \) computes a fixed number \( \kappa \) of extreme points of each \( V \) sized subset computed by \( SLS \).

\[
[OF_a^k] = DRSplWorker(IF_a, M_a, k, w_{k-1})
\]

1. Create empty file \( OF_a^k \) and set \( X_a \leftarrow \emptyset \) and \( Y_a \leftarrow \emptyset \)
2. IF \( k \) is greater than 1, then
   (a) For each \( x_i, y_i \) in \( IF_a \) compute \( f(x_i) = y_iw_{k-1}^T x_i \).
   (b) IF \( Ulim \geq f(x_i) \geq Llim \), then add \( x_i, y_i \) to \( \{X_a, Y_a\} \)
3. IF \( k \) is equal to 1, then add all \( x_i, y_i \) in \( IF_a \) to \( \{X_a, Y_a\} \)
4. Set \( X^+ \leftarrow \{x_i : x_i \in X_1, y_i = 1\} \) and \( X^- \leftarrow \{x_i : x_i \in X_1, y_i = -1\} \)
5. Run \([X^+, \Delta^+] = FLS(X^+, P)\) and \([X^-, \Delta^-] = FLS(X^-, P)\), where \( FLS \) is \( FLS1 \) or \( FLS2 \)
6. Run \([X^+, \Delta'_2] = SLS(X^+, V, \Delta^+)\) and \([X^-, \Delta'_2] = SLS(X^-, V, \Delta^-)\)
7. Using \( \Delta'_2^+ \), identify each subset \( X_{q_r} \) of \( X^+ \) and run \([X^*_r, \beta_{q_r}] = DAEpl(X_{q_r})\)
   Set \( N^{++} \leftarrow \sum \) sum of number of data vectors in all \( X_{q_r} \) derived from \( X^+ \)
8. Using \( \Delta'_2^- \), identify each subset \( X_{q_r} \) of \( X^- \) and run \([X^*_r, \beta_{q_r}] = DAEpl(X_{q_r})\)
   Set \( N^{-} \leftarrow \sum \) sum of number of data vectors in all \( X_{q_r} \) derived from \( X^- \)
9. Combine in the same order, all \( X^*_r \) to obtain \( X^*_r \) and all \( \beta_{q_r} \) to obtain \( \beta_a \)
   Set \( Y^*_a \leftarrow \{y_i : y_i = 1 \text{ for } i = 1, 2, .., N^{++}; \text{ and } y_i = -1 \text{ for } i = 1 + N^{++}, 2 + N^{++}, .., N^{--} + N^{++}\} \)
10. Write \( \{X^*_a, Y^*_a\} \) and \( \beta_a \) to \( OF_a^k \)
The routine DAEpl($X_q$), selects $\kappa$ vectors from $X_q$, using a greedy approach as described below. In step 2, it identifies the vector $x_a \in X_q$, with the largest norm. In step 3, the vector $x_b \in X_q$ that is most distant from $x_a$ is identified. Both $x_a$ and $x_b$ are added to $X_q^*$. In step 4, the remaining extreme points are identified using getError($x_i, X_q^*$). The routine getError($x_i, X_q^*$) returns the approximation error in representing $x_i$ as a convex combination of the vectors in set $X_q^*$. It is computed by solving the following quadratic optimization problem:

$$\min_{\mu} \rho(x_i, \Psi) = \|x_i - \sum_{t=1}^{||\Psi||} \mu_{i,t} x_t \|^2$$

s.t. $x_t \in \Psi, 0 \leq \mu_{i,t} \leq 1$ and $\sum_{t=1}^{||\Psi||} \mu_{i,t} = 1$  \hfill (7–3)

It can be seen that getError($x_i, X_q^*$) is similar to CheckPoint($x_i, X_q^*$) in Chapter 4.3. The value of $\mu_i$ computed by getError($x_i, X_q^*$), is used in step 6 of DAEpl. The matrix $X_q^*$ contains the $\kappa$ selected extreme points of $X_q$ and $\beta_q$ is a $\kappa$ sized vector.

Now we compute the time complexity of DAEpl. Steps 2 and 3 are of linear time complexity, $O(\|X_q\|) = O(V)$. To compute complexity of step 4, we use the fact that the optimization problem in getError($x_i, X_q^*$) is essentially the same as the dual optimization problem of SVM given in (2–1). For SMO based implementations, such as the implementation used in this dissertation, the time complexity is quadratic. Hence the time complexity of step 4 is $O(V \kappa |X_q^*|^2) = O(V \kappa^3)$. In step 6, for each $x_i$ in $X_q$, the problem in (7–3) is solved, giving a complexity of $O(V \kappa^2)$. The time complexity of steps 5 and 7 is $O(V \kappa)$. Hence time complexity of routine DAEpl is $O(V \kappa^3)$. The overall time complexity of DRSplWorker is $O(N_s(\log_2 \frac{M_2}{\kappa} + \kappa^2))$. Due to the cubic dependence of the time complexity of DRSplWorker on $\kappa$, it is set to a small positive value.

Even though DAEpl($X_q$) is computationally simple, it has a desirable theoretical property. The following theorems prove that vectors computed by DAEpl($X_q$) are extreme
\[
[X_q^*, \beta_{q,i}] = DAEp(X_q)
\]

1. Set \(X_q^* \leftarrow \emptyset\)
2. Find \(x_a = \arg \max_{x \in X_q} ||x||^2\)
   Set \(X_q^* \leftarrow X_q^* \cup x_a\)
3. Find \(x_b = \arg \max_{x \in X_q} ||x - x_a||^2\)
   Set \(X_q^* \leftarrow X_q^* \cup x_b\)
4. For \(i = 3\) to \(\kappa\)
   (a) Find \(x_i = \arg \max_{x \in X_q} \text{getError}(x, X_q^*)\)
   (b) Set \(X_q^* \leftarrow X_q^* \cup x_i\)
5. Initialize a matrix \(\Gamma\) of size \(|X_q^*| \times \kappa\) with all elements set to 0
   Set \(\mu_{i,j} \leftarrow 1 \forall x_i \in X_q^*, \) where \(\mu_{i,j}\) is the element in the \(i^{th}\) row and \(j^{th}\) column of \(\Gamma\)
6. For each \(x_i \in X_q^*\) and \(x_j \notin X_q^*\), execute \(\text{getError}(x_i, X_q^*)\)
   Set the \(i^{th}\) row of \(\Gamma \leftarrow \mu_{i,j}\), where \(\mu_{i,j}\) is the result of \(\text{getError}(x, X_q^*)\)
7. For \(j = 1\) to \(\kappa\)
   Set \(\beta_{q,j} \leftarrow \sum_{k=1}^{\kappa} \mu_{k,j}\)

points of \(X_q\). Recall that the extreme points of \(X_q, EP(X_q)\), are defined as the vertices of the convex polygon formed by \(\text{Conv}(X_q)\), the convex hull of \(X_q\).

**Theorem 7** Let \(S\) be a finite convex set with \(EP(S)\) as its extreme points. Let \(T\) be a convex set with extreme points \(EP(T) \subset EP(S)\). Let \(x\) be defined as follows,

\[
x = \arg \max_{x \in S} p(x_i)
\]

where \(p(x_i) = \min_{\Pi} \left\| x_i - \sum_{x_j \in EP(T)} \mu_{i,j} x_j \right\|^2\)

Then, the vector \(x \in EP(S)\).

**Proof.** Let \(P\) be a bounded polyhedron whose vertices are \(EP(S)\). Let \(t\) be the vector in \(T\) that is closest to \(x\). Then
where $\mu$ is the solution of $p(x)$.

Assume $x$ is not an extreme point, $x \not\in EP(S)$. Furthermore, assume $x$ is not on any face of $P$. Then, there exists $x'$ on the intersection of a face $F$ of $P$ and the line joining $x$ and $t$ that is more distant from $t$ than $x$. Additionally, there exists $x_i \in EP(S)$ that is a vertex on $F$, that is more distant from $T$ than $x'$. Hence $x_i$ is more distant from $T$ than $x$. But by definition $x$ is the most distant vector in $S$ from $T$. Therefore our assumptions are invalid, and $x \in EP(S)$. $\square$

**Theorem 8** Let $\chi$ be a set of vectors, such that $|EP(\chi)| \geq \kappa$. Let $\chi^*$ be the $\kappa$ vectors that are computed by the algorithm DAEpl($\chi$). Then,

$$\chi^* \subseteq EP(\chi)$$

**Proof.** First we prove that the vector $x_a = \arg \max_{x_i \in \chi} ||x_i||^2$ that is added to $\chi^*$ in step 2 of DAEpl is an extreme point. Suppose $x_a$ is not an extreme point. Then it is possible to represent $x_a$ as a convex combination of $x_i \in EP(\chi)$, $x_a = \sum x_i \in EP(\chi) \lambda_{a,i} x_i$. Then we get,

$$||x_a||^2 = \sum_{x_i \in EP(\chi)} \lambda_{a,i} \sum_{x_j \in EP(\chi)} \lambda_{a,j} x_i^T x_j, \text{ where } 0 \leq \lambda_{a,i} \text{ and } \sum_{x_i \in EP(\chi)} \lambda_{a,i} = 1$$

$$\leq \max_{x_i \in EP(\chi)} ||x_i||^2$$

But this violates the definition of $x_a$. Hence $x_a \in EP(\chi)$.

To prove that the vector $x_b = \arg \max_{x_i \in \chi} ||x_a - x_i||^2$ that is added to $\chi^*$ in step 3 of DAEpl is an extreme point, we can use Theorem 7. In this case, $x_b \in \chi$ is the most distant vector from convex set $T = x_a \subset EP(\chi)$. Hence $x_b \in EP(\chi)$.

Similarly, any vector $x_i \in \chi$ that is added to $\chi^*$ in step 4 of DAEpl is in $EP(\chi)$, by virtue of Theorem 7. $\square$
7.2 Optimizations of DRSpl

DRSpl is an iterative algorithm, with each worker node producing an output file at each iteration. The output files contain the vectors of the representative set computed by each worker node. This strategy can lead to high communication overheads if the output files are large. However, this problem can be easily solved by storing only the vectors that have not already been written to an output file in an earlier iteration by the same node. This can be efficiently accomplished by assigning an identifier to each vector, such as the index of the vector in the input file of the worker. Then before writing a vector \( \mathbf{x}_i \) to an output file, the identifier of the vector is compared to the identifiers of the vectors already written to output files. If the identifier of \( \mathbf{x}_i \) is not present in the earlier output files, then it is written to the output file. Otherwise, only the identifier is written to the output. At the master node, the representative set is reconstructed based on the output files from all previous iterations. This approach greatly reduces the inter-node communication requirements of DRSpl.

Another optimization that is simple and effective, results from the fact that the output of DRSplWorker for the case of \( k = 1 \) is always the same, irrespective of hyper-parameter \( C \). Hence it need be computed only the first time DRSpl is executed. For subsequent runs of DRSpl, the steps 2.a and 2.b of DRSplMaster can be skipped for \( k = 1 \). The \( \{X^*, Y^*\} \) and \( \beta \) computed in the first iteration of the first run of DRSpl can be reused.

7.3 Experiments on DRSpl

To implement DRSpl, we modified the DeriveRS implementation that was evaluated in Chapter 5. To solve AESVM in step 2.c of DRSplMaster, the LIBLINEAR linear SVM solver was modified. The datasets given below were used to evaluate DRSpl. These datasets are large and high dimensional, and indicate the performance of DRSpl on web-scale datasets.
D10:  *KDD cup 2010 educational data mining challenge, bridge to algebra*\(^1\) - This dataset is available as a training set of 19264097 data vectors and a testing set of 748401 data vectors, with \( D = 29890095 \). The training set is highly sparse with density = 0.0001%. The file size of the training set is 4.8 GB.

D11:  *Malicious URL dataset*\(^2\) - This dataset of URLs of malicious websites has \( D = 3231961 \) features and 2396130 vectors. In our experiments, we used the first 2000000 vectors as the training set and the remaining 396130 vectors as the testing set. The dataset has a density = 0.004%. The file size of the training set is 1.7 GB.

D12:  *Webspam dataset*\(^3\) - This dataset of spam web pages [77] has \( D = 16609143 \) features and 350000 vectors. In our experiments, we used the first 300000 vectors as the training set and the remaining 50000 vectors as the testing set. The dataset has a density = 0.022%. The file size of the training set is 20 GB.

D13:  *Epsilon dataset*\(^4\) - We used the entire epsilon dataset, 10% of which was used as D7 in Chapter 5. It is comprised of 400000 data vectors that are 100% dense with \( D = 2000 \). The testing set has 100000 data vectors. The file size of the training set is 11.3 GB.

Apart from the above datasets, we used D8 (MNIST dataset), for a comparison of DRSpl and with the results presented in Chapter 5. To compare our DRSpl implementation


\(^2\) [http://sysnet.ucsd.edu/projects/url/#datasets](http://sysnet.ucsd.edu/projects/url/#datasets)

\(^3\) [http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#webspam](http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#webspam)

\(^4\) [http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#epsilon](http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#epsilon)
with other linear SVM solvers, we used LIBLINEAR and LIBOCAS \ref{http://cmp.felk.cvut.cz/~xfrancv/ocas/html}. LIBLINEAR gives an accurate linear SVM solution while LIBOCAS implements an optimized cutting-plane algorithm \cite{29} that has been reported to be among the fastest linear SVM solvers\cite{74}. Additionally, LIBOCAS supports parallel execution of its solver on a multi-core processor.

We also compared our implementation with a bagging based approach, where after randomly distributing the data to each worker node, the SVM problem is locally solved at the worker. The results of the local SVMs are averaged to get the estimate of the global solution. We refer to this approach as BagSVM. The algorithm BagSVM(C, WO, IF) is described below. The input IF is a file containing all the data vectors.

\[ [w] = \text{BagSVM}(C, WO, IF) \]

1. Randomly distribute the vectors in IF among the files IF\(_1\), IF\(_2\), ..., IF\(_{WO}\).
2. Solve SVM on each worker nodes, with input file IF\(_t\) and hyper-parameter C, to get output \(w_t\).
3. Set

\[ w \leftarrow \sum_{t=1}^{WO} \frac{w_t}{WO} \]

The various SVM solvers were trained on the selected datasets with parameters \(C' = [0.01, 0.1, 1, 10, 100]\). The experiments on DRSpl and BagSVM were performed with \(WO = [16, 24, 32, 48]\). LIBOCAS was trained with \(WO = 16\), since 16 is the maximum number of threads supported on our multi-core computers. The maximum number of iterations \(K\) in DRSplMaster was set to 10. The convergence parameter \(\theta\) in step 2.d of DRSplMaster was set as \(\theta = 0.1\). In all our experiments, we used \(\kappa = 10\). This value of \(\kappa\) was found to give good results, while keeping the complexity of DAEP small. The distance range parameters used in step 2.b of DRSplWorker was set as \(U_{lim} = 1.1\) and \(L_{lim} = -0.1\).

\[ \text{http://cmp.felk.cvut.cz/~xfrancv/ocas/html} \]
To evaluate the performance of DRSpl we used the following metrics.

1. **Total CPU clock time** $TCT$: The total CPU clock time used by the DRSplMaster process and all the DRSplWorker processes indicate the total computation time of DRSpl.

$$TCT = \sum_{r=1}^{R} (CPU_{master}^r + \sum_{t=1}^{WO} CPU_{slave,t}^r)$$

where $CPU_{master}^r$ and $CPU_{slave,t}^r$ are the CPU times taken by DRSplMaster and the $t^{th}$ DRSplWorker, with the $r^{th}$ hyper-parameter value respectively.

2. **Average CPU clock time per worker** $ACT$: The average CPU clock time of the DRSplWorker processes indicate the expected computation time of each worker process.

$$ACT = \frac{\sum_{r=1}^{R} \sum_{t=1}^{WO} CPU_{slave,t}^r}{R \times WO}$$

3. **Wall clock time** $WCT$: The total time taken from start to end of DRSplMaster, which is the total time taken by DRSpl.

$$WCT = \sum_{r=1}^{R} WALL_{master}^r$$

where $WALL_{master}^r$ is the wall clock time taken by DRSplMaster with the $r^{th}$ hyper-parameter value.

4. **Mean and maximum classification accuracy**: The average and maximum classification accuracies over all the hyper-parameter values.

5. **Root mean squared error of classification accuracy, RMSE**: For a given approximate linear SVM solver $F$, the RMSE of its classification accuracy, is indicated by:

$$RMSE = \left( \frac{1}{R} \sum_{r=1}^{R} (CL^r - C_{F}^r)^2 \right)^{0.5}$$

Here $CL^r$ and $C_{F}^r$ are the classification accuracy of LIBLINEAR and $F$ respectively.

6. **Expected output size EOS**: This metric is the total size of output files of all worker nodes over all iterations $k$ of DRSplMaster as a percentage of the size of the input file. $EOS$ indicates the total communication bandwidth required from the worker.
nodes to the master node in comparison to the size of the input file.

\[
EOS = \sum_{r=1}^{R} \sum_{k=1}^{K} \sum_{l=1}^{WO} |OF_{t,r}^k| / (R * |F|) * 100
\]

where \( OF_{t,r}^k \) is the output file of the \( t^{th} \) worker node, in the \( k^{th} \) DRSpl iteration with the \( r^{th} \) hyper-parameter value and \(|F|\) is the size of the input file.

The results of our evaluation of DR Spl are presented in Tables 7-1 to 7-5. For D10, DR Spl was generally the fastest to compute the solution, taking the least total CPU clock time, \( TC_T \) and wall clock time, \( WC_T \) in most cases, as seen in Table 7-1. The value of \( EOS \) was observed to be roughly equal to the ratio \( N/M \). It is not exactly equal to \( N/M \) because the vectors are of different levels of sparsity and because DR Spl can have multiple iterations. The performance of LIB OCAS on dataset D10 could not be evaluated due to its large memory requirement. Compared to BagSVM, DR Spl had lower \( RMSE \), although the maximum accuracy and mean accuracy of BagSVM was higher. The \( ACT \) values are found to be inversely dependent on the number of worker nodes, \( WO \) and the size of the representative set indicated by \( N/M \).

For D11, BagSVM was the faster than DR Spl, taking the least wall clock time, \( WC_T \). However, DR Spl was found to be much faster than LIBLINEAR and LIB OCAS in terms of \( WC_T \) and \( TC_T \) as seen in Table 7-2. For \( N/M = 50 \) it was found to result in lower \( RMSE \) values than BagSVM in all cases. The performance of LIB OCAS is on par with LIBLINEAR in terms of \( RMSE \), maximum and mean accuracy, while it is noticeably more time consuming as observed from the larger \( TC_T \) and \( WC_T \). This seems to be the general case as observed in Tables 7-2 to 7-5.

Table 7-3 illustrates the performance of DR Spl on dataset D12. It can be seen that DR Spl resulted in significantly better \( RMSE \), maximum and mean accuracy than BagSVM in all cases except for \( N/M = 300 \). In terms of \( WC_T \), DR Spl was faster than LIBLINEAR and LIB OCAS, while it was slower than BagSVM. An interesting observation is the increasing trend of \( WC_T \) with increase in \( WO \), which is not the case in Tables 7-2.
Table 7-1. Performance of DRSpl, BagSVM and LIBLINEAR on D10

<table>
<thead>
<tr>
<th>Solver</th>
<th>ACT (s)</th>
<th>ICT (s)</th>
<th>WCT (s)</th>
<th>EOS (%)</th>
<th>RMSE (x10^2)</th>
<th>Max. ac. (x10^2)</th>
<th>Mean ac. (x10^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>25 801</td>
<td>25 801</td>
<td>0</td>
<td>89.9</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>BagSVM</td>
<td>247.9</td>
<td>21 373.7</td>
<td>1006.3</td>
<td>0.9</td>
<td>89.8</td>
<td>89.6</td>
<td>89.6</td>
</tr>
<tr>
<td>BagSVM</td>
<td>172.4</td>
<td>22 418</td>
<td>852.7</td>
<td>0.9</td>
<td>89.8</td>
<td>89.6</td>
<td>89.6</td>
</tr>
<tr>
<td>BagSVM</td>
<td>131.7</td>
<td>22 938.6</td>
<td>783.2</td>
<td>1</td>
<td>89.7</td>
<td>89.5</td>
<td>89.5</td>
</tr>
<tr>
<td>BagSVM</td>
<td>118.9</td>
<td>25 751.6</td>
<td>779.1</td>
<td>1</td>
<td>89.7</td>
<td>89.5</td>
<td>89.5</td>
</tr>
<tr>
<td>DRSpl</td>
<td>76.5</td>
<td>8327.6</td>
<td>2340.3</td>
<td>7.3</td>
<td>89.2</td>
<td>89.1</td>
<td>89.1</td>
</tr>
<tr>
<td>DRSpl</td>
<td>56.8</td>
<td>4877.8</td>
<td>444.5</td>
<td>1.9</td>
<td>89.0</td>
<td>88.7</td>
<td>88.7</td>
</tr>
<tr>
<td>DRSpl</td>
<td>59.4</td>
<td>4846.5</td>
<td>390.2</td>
<td>0.5</td>
<td>88.8</td>
<td>88.7</td>
<td>88.7</td>
</tr>
<tr>
<td>DRSpl</td>
<td>56.8</td>
<td>9161.9</td>
<td>2384.1</td>
<td>7.3</td>
<td>89.2</td>
<td>89.2</td>
<td>89.2</td>
</tr>
<tr>
<td>DRSpl</td>
<td>42.2</td>
<td>5429.2</td>
<td>399.5</td>
<td>1.9</td>
<td>89.1</td>
<td>88.7</td>
<td>88.7</td>
</tr>
<tr>
<td>DRSpl</td>
<td>38.7</td>
<td>4736.1</td>
<td>284.5</td>
<td>0.5</td>
<td>88.7</td>
<td>88.6</td>
<td>88.6</td>
</tr>
<tr>
<td>DRSpl</td>
<td>42.9</td>
<td>9078.6</td>
<td>2286.4</td>
<td>7.3</td>
<td>89.2</td>
<td>89.1</td>
<td>89.1</td>
</tr>
<tr>
<td>DRSpl</td>
<td>30.6</td>
<td>5242.3</td>
<td>406.4</td>
<td>1.9</td>
<td>89.0</td>
<td>88.7</td>
<td>88.7</td>
</tr>
<tr>
<td>DRSpl</td>
<td>31.2</td>
<td>5083.2</td>
<td>306.9</td>
<td>0.5</td>
<td>88.8</td>
<td>88.6</td>
<td>88.6</td>
</tr>
<tr>
<td>DRSpl</td>
<td>36.9</td>
<td>9632.7</td>
<td>2593.0</td>
<td>7.3</td>
<td>89.2</td>
<td>89.1</td>
<td>89.1</td>
</tr>
<tr>
<td>DRSpl</td>
<td>25.8</td>
<td>5510.7</td>
<td>559.0</td>
<td>1.9</td>
<td>89.0</td>
<td>88.8</td>
<td>88.8</td>
</tr>
<tr>
<td>DRSpl</td>
<td>26.4</td>
<td>5366.8</td>
<td>402.5</td>
<td>0.5</td>
<td>88.7</td>
<td>88.7</td>
<td>88.7</td>
</tr>
</tbody>
</table>

and 7-1. A possible reason for this is the large file access time required for D12, which is the largest dataset we experimented on. It should be noted that our experiments were conducted on a centralized file system and that in a distributed file systems the impact of file size on \( WC_T \) might be less. In Table 7-4, the performance of DRSpl on dataset
Table 7-2. Performance of DRSpl, BagSVM, LIBLINEAR and LIBOCAS on D11

<table>
<thead>
<tr>
<th>Solver</th>
<th>ACT (s)</th>
<th>TCT (s)</th>
<th>WCT (s)</th>
<th>EOS (%)</th>
<th>RMSE (x10^2)</th>
<th>Max. ac. (x10^2)</th>
<th>Mean ac. (x10^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>2111</td>
<td>2111</td>
<td>0</td>
<td>99.4</td>
<td>99.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIBOCAS</td>
<td>19482</td>
<td>8355</td>
<td>0</td>
<td>99.4</td>
<td>99.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BagSVM</td>
<td>15.4</td>
<td>1319.7</td>
<td>66.3</td>
<td>0.7</td>
<td>98.6</td>
<td>98.4</td>
<td></td>
</tr>
<tr>
<td>WO = 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BagSVM</td>
<td>10.2</td>
<td>1317.2</td>
<td>59.8</td>
<td>0.8</td>
<td>98.5</td>
<td>98.3</td>
<td></td>
</tr>
<tr>
<td>WO = 24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BagSVM</td>
<td>7.2</td>
<td>1251.3</td>
<td>57.2</td>
<td>0.9</td>
<td>98.4</td>
<td>98.2</td>
<td></td>
</tr>
<tr>
<td>WO = 32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BagSVM</td>
<td>5.4</td>
<td>1182.5</td>
<td>51.6</td>
<td>1</td>
<td>98.3</td>
<td>98.1</td>
<td></td>
</tr>
<tr>
<td>WO = 40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 16, N = 50</td>
<td>13.7</td>
<td>1303.9</td>
<td>274.4</td>
<td>5.4</td>
<td>98.6</td>
<td>98.5</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 16, N = 100</td>
<td>18.4</td>
<td>1596.1</td>
<td>228.2</td>
<td>4.0</td>
<td>98.5</td>
<td>98.3</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 16, N = 300</td>
<td>23.4</td>
<td>1898.7</td>
<td>178.2</td>
<td>1.9</td>
<td>98.0</td>
<td>97.8</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 24, N = 50</td>
<td>9.6</td>
<td>1333.5</td>
<td>232.7</td>
<td>5.5</td>
<td>98.6</td>
<td>98.5</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 24, N = 100</td>
<td>12.6</td>
<td>1643.8</td>
<td>202.7</td>
<td>4.1</td>
<td>98.4</td>
<td>98.2</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 24, N = 300</td>
<td>15.9</td>
<td>1948.2</td>
<td>144.0</td>
<td>2.0</td>
<td>98.0</td>
<td>97.8</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 32, N = 50</td>
<td>7.7</td>
<td>1441.0</td>
<td>251.1</td>
<td>5.7</td>
<td>98.5</td>
<td>98.4</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 32, N = 100</td>
<td>10.2</td>
<td>1738.4</td>
<td>169.8</td>
<td>4.1</td>
<td>98.4</td>
<td>98.1</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 32, N = 300</td>
<td>11.9</td>
<td>1940.7</td>
<td>129.4</td>
<td>2.0</td>
<td>97.9</td>
<td>97.8</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 40, N = 50</td>
<td>6.8</td>
<td>1592.3</td>
<td>258.8</td>
<td>5.6</td>
<td>98.5</td>
<td>98.4</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 40, N = 100</td>
<td>9.3</td>
<td>1985.0</td>
<td>179.4</td>
<td>4.3</td>
<td>98.4</td>
<td>98.2</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 40, N = 300</td>
<td>10.9</td>
<td>2222.4</td>
<td>145.9</td>
<td>2.0</td>
<td>97.9</td>
<td>97.7</td>
<td></td>
</tr>
</tbody>
</table>

D13 is illustrated. The values of WO tested were [20, 50, 100] and not [50, 100, 300], considering that the dataset has relatively few data vectors all of which are 100% dense. As before, DRSpl resulted in significantly better RMSE than BagSVM in all cases. In
terms of WCT, DRSpl was faster than LIBLINEAR and LIBOCAS, while it was slower than BagSVM.

We can infer the dependence of RMSE with the number of vectors in the training set, based on 7-2 to 7-5. It can be seen that the increase in RMSE for D10 and D11 with increase in \( \frac{N}{M} \) is much less significant than in datasets D12 and D13. The datasets D12 and D13 have relatively few data vectors. For the example of D12, when \( WO \) is 40, the number of vectors in \( IF_s \) of each worker node is 8750. In this case, when \( \frac{N}{M} = 300 \) each worker node computes a maximum of 30 vectors in \( OF_s \). Such a small number of vectors can explain the large RMSE values seen for the case of large \( WO \) and \( \frac{N}{M} \). Hence the parameter \( M \) should be decided based on the value of \( N \) and \( WO \).

Table 7-5 illustrates the performance of DRSpl on dataset D8. The purpose of this illustration to compare DRSpl with the results in Table 5-5. As discussed in Chapter 5, when a non-linear AESVM was trained on the representative set computed by DeriveRS, there was no noticeable reduction in training time when compared to LIBSVM. However, by using DRSpl a significant reduction in WCT can be observed, while the performance in terms of classification accuracy is similar. Table 7-5 also demonstrates that DRSpl can be expected to give good results with small datasets such as D8. DRSpl was run with only \( WO = 4 \) and \( WO = 8 \) on D8, considering the small size of the dataset.

Tables 7-1 to 7-5, demonstrate the feasibility of using DRSpl as an approximate solver for linear SVM. Now we consider the training time speedup obtained by using a parallel programming framework to solve DRSpl. We use the metric overall training time speedup (OTS) defined as follows:

\[
OTS = \frac{WCT \text{ of LIBLINEAR}}{WCT \text{ of } \mathbb{F}}
\]

where \( \mathbb{F} \) is any approximate linear SVM solver such as DRSpl or BagSVM. Figures 7-1 to 7-4 plot the OTS values of DRSpl observed for different values of \( WO \) on datasets D10 to D13. It can be seen that OTS generally increases when \( \frac{N}{M} \) is increased. This is
Table 7-3. Performance of DRSpl, BagSVM, LIBLINEAR and LIBOCAS on D12

<table>
<thead>
<tr>
<th>Solver</th>
<th>ACT (s)</th>
<th>ICT (s)</th>
<th>WCT (s)</th>
<th>EOS (%)</th>
<th>RMSE (x10^2)</th>
<th>Max. ac. (x10^2)</th>
<th>Mean ac. (x10^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>4022</td>
<td>4022</td>
<td>0</td>
<td>99.6</td>
<td>98.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIBOCAS</td>
<td>14429</td>
<td>5058</td>
<td>0</td>
<td>99.6</td>
<td>98.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BagSVM WO = 16</td>
<td>41.7</td>
<td>3667.2</td>
<td>477.3</td>
<td>2.3</td>
<td>99.0</td>
<td>96.5</td>
<td></td>
</tr>
<tr>
<td>BagSVM WO = 24</td>
<td>32.7</td>
<td>4288.0</td>
<td>484.9</td>
<td>2.9</td>
<td>98.8</td>
<td>96.1</td>
<td></td>
</tr>
<tr>
<td>BagSVM WO = 32</td>
<td>29.4</td>
<td>5093.8</td>
<td>476.3</td>
<td>3.5</td>
<td>98.6</td>
<td>95.6</td>
<td></td>
</tr>
<tr>
<td>BagSVM WO = 40</td>
<td>26.0</td>
<td>5649.4</td>
<td>475.6</td>
<td>4.6</td>
<td>98.5</td>
<td>94.9</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 16,</td>
<td>93.4</td>
<td>7930.1</td>
<td>1573.3</td>
<td>4.2</td>
<td>0.7</td>
<td>99.2</td>
<td>97.9</td>
</tr>
<tr>
<td>N = 50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 16,</td>
<td>101.8</td>
<td>8467.6</td>
<td>1450.2</td>
<td>2.3</td>
<td>0.9</td>
<td>98.8</td>
<td>97.7</td>
</tr>
<tr>
<td>N = 100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 16,</td>
<td>106.0</td>
<td>8587.6</td>
<td>1586.5</td>
<td>0.7</td>
<td>4.4</td>
<td>97.2</td>
<td>94.2</td>
</tr>
<tr>
<td>N = 300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 24,</td>
<td>62.4</td>
<td>7919.4</td>
<td>1722.0</td>
<td>4.1</td>
<td>0.4</td>
<td>99.2</td>
<td>98.2</td>
</tr>
<tr>
<td>N = 50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 24,</td>
<td>69.2</td>
<td>8623.8</td>
<td>1754.9</td>
<td>2.2</td>
<td>1.3</td>
<td>98.9</td>
<td>97.3</td>
</tr>
<tr>
<td>N = 100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 24,</td>
<td>72.8</td>
<td>8852.9</td>
<td>1767.6</td>
<td>0.8</td>
<td>4.7</td>
<td>95.7</td>
<td>93.8</td>
</tr>
<tr>
<td>N = 300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 32,</td>
<td>47.2</td>
<td>8042.0</td>
<td>1979.7</td>
<td>4.1</td>
<td>0.3</td>
<td>99.2</td>
<td>98.2</td>
</tr>
<tr>
<td>N = 50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 32,</td>
<td>52.2</td>
<td>8658.4</td>
<td>1789.5</td>
<td>2.3</td>
<td>1.1</td>
<td>98.8</td>
<td>97.4</td>
</tr>
<tr>
<td>N = 100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 32,</td>
<td>48.6</td>
<td>7899.9</td>
<td>1527.5</td>
<td>0.7</td>
<td>5.0</td>
<td>95.6</td>
<td>93.4</td>
</tr>
<tr>
<td>N = 300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 40,</td>
<td>38.6</td>
<td>8193.3</td>
<td>2111.1</td>
<td>4.0</td>
<td>0.4</td>
<td>99.1</td>
<td>98.2</td>
</tr>
<tr>
<td>N = 50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 40,</td>
<td>42.2</td>
<td>8716.9</td>
<td>1570.8</td>
<td>2.3</td>
<td>1.3</td>
<td>98.6</td>
<td>97.2</td>
</tr>
<tr>
<td>N = 100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 40,</td>
<td>42.3</td>
<td>8540.3</td>
<td>1776.3</td>
<td>0.8</td>
<td>7.9</td>
<td>93.5</td>
<td>90.7</td>
</tr>
<tr>
<td>N = 300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

expected, as the time complexity of DRSplMaster is directly proportional to $M$. However, for D12 and D13 the plot is not as uniform as for D10 and D11, as seen in Figures 7-3 and 7-4. As discussed earlier, this is possibly the result of several worker nodes accessing large files concurrently on a centralized file system. The dependence of
Table 7-4. Performance of DRSpl, BagSVM, LIBLINEAR and LIBOCAS on D13

<table>
<thead>
<tr>
<th>Solver</th>
<th>ACT (s)</th>
<th>ICT (s)</th>
<th>WC (s)</th>
<th>EOS (%)</th>
<th>RMSE (x10^2)</th>
<th>Max. ac. (%)</th>
<th>Mean ac. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>2450</td>
<td>2450</td>
<td>0</td>
<td>89.8</td>
<td>89.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIBOCAS</td>
<td>2432</td>
<td>2310</td>
<td>0.1</td>
<td>89.8</td>
<td>89.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BagSVM</td>
<td>23.6</td>
<td>2014.6</td>
<td>193.4</td>
<td>7</td>
<td>89.8</td>
<td>85.7</td>
<td></td>
</tr>
<tr>
<td>WO = 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BagSVM</td>
<td>17.0</td>
<td>2171.5</td>
<td>179.5</td>
<td>8.4</td>
<td>89.8</td>
<td>84.9</td>
<td></td>
</tr>
<tr>
<td>WO = 24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BagSVM</td>
<td>13.4</td>
<td>2262.6</td>
<td>174.5</td>
<td>9.2</td>
<td>89.7</td>
<td>84.3</td>
<td></td>
</tr>
<tr>
<td>WO = 32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BagSVM</td>
<td>11.3</td>
<td>2379.6</td>
<td>175.1</td>
<td>9.3</td>
<td>89.7</td>
<td>84.1</td>
<td></td>
</tr>
<tr>
<td>WO = 40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 16, N/M = 20</td>
<td>49.5</td>
<td>4374.2</td>
<td>694.0</td>
<td>11.3</td>
<td>0.8</td>
<td>89.1</td>
<td>88.7</td>
</tr>
<tr>
<td>DRSpl WO = 16, N/M = 50</td>
<td>56.9</td>
<td>4797.6</td>
<td>550.6</td>
<td>5.9</td>
<td>1.6</td>
<td>88.5</td>
<td>88.0</td>
</tr>
<tr>
<td>DRSpl WO = 16, N/M = 100</td>
<td>66.3</td>
<td>5450.3</td>
<td>610.3</td>
<td>3.7</td>
<td>2.2</td>
<td>87.6</td>
<td>87.2</td>
</tr>
<tr>
<td>DRSpl WO = 24, N/M = 20</td>
<td>35.1</td>
<td>4640.0</td>
<td>648.1</td>
<td>11.2</td>
<td>0.9</td>
<td>89.0</td>
<td>88.6</td>
</tr>
<tr>
<td>DRSpl WO = 24, N/M = 50</td>
<td>36.8</td>
<td>4666.0</td>
<td>441.2</td>
<td>5.6</td>
<td>1.7</td>
<td>88.4</td>
<td>87.8</td>
</tr>
<tr>
<td>DRSpl WO = 24, N/M = 100</td>
<td>45.6</td>
<td>5627.7</td>
<td>505.8</td>
<td>3.8</td>
<td>2.1</td>
<td>87.6</td>
<td>87.3</td>
</tr>
<tr>
<td>DRSpl WO = 32, N/M = 20</td>
<td>26.1</td>
<td>4592.8</td>
<td>570.6</td>
<td>11.2</td>
<td>0.9</td>
<td>89.0</td>
<td>88.6</td>
</tr>
<tr>
<td>DRSpl WO = 32, N/M = 50</td>
<td>29.2</td>
<td>4930.5</td>
<td>407.2</td>
<td>6.0</td>
<td>1.7</td>
<td>88.5</td>
<td>87.9</td>
</tr>
<tr>
<td>DRSpl WO = 32, N/M = 100</td>
<td>34.2</td>
<td>5650.0</td>
<td>464.1</td>
<td>3.9</td>
<td>2.0</td>
<td>87.8</td>
<td>87.4</td>
</tr>
<tr>
<td>DRSpl WO = 40, N/M = 20</td>
<td>23.9</td>
<td>5226.9</td>
<td>567.6</td>
<td>11.2</td>
<td>0.9</td>
<td>89.0</td>
<td>88.6</td>
</tr>
<tr>
<td>DRSpl WO = 40, N/M = 50</td>
<td>25.5</td>
<td>5339.7</td>
<td>373.2</td>
<td>5.7</td>
<td>1.7</td>
<td>88.5</td>
<td>87.8</td>
</tr>
<tr>
<td>DRSpl WO = 40, N/M = 100</td>
<td>31.4</td>
<td>6416.2</td>
<td>433.7</td>
<td>3.8</td>
<td>2.1</td>
<td>87.6</td>
<td>87.3</td>
</tr>
</tbody>
</table>

EOS on WO is illustrated in Figures 7-5 to 7-8. It can be seen that EOS is inversely proportional to \( \frac{N}{M} \). EOS did not vary significantly with changes in WO, in any of our experiments.
Table 7-5. Performance of DRSpl, BagSVM, LIBLINEAR and LIBOCAS on D8

<table>
<thead>
<tr>
<th>Solver</th>
<th>ACT (s)</th>
<th>ICT (s)</th>
<th>WCT (s)</th>
<th>EOS (%)</th>
<th>RMSE (x10^2)</th>
<th>Max. ac.</th>
<th>Mean ac. (x10^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>41.5</td>
<td>41.5</td>
<td>0</td>
<td>99.5</td>
<td>99.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIBOCAS</td>
<td>118.4</td>
<td>47</td>
<td>0.1</td>
<td>99.4</td>
<td>99.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BagSVM WO = 4</td>
<td>0.4</td>
<td>19.5</td>
<td>2.8</td>
<td>0.1</td>
<td>99.3</td>
<td>99.2</td>
<td></td>
</tr>
<tr>
<td>BagSVM WO = 8</td>
<td>0.9</td>
<td>20.3</td>
<td>2.6</td>
<td>0.1</td>
<td>99.4</td>
<td>99.2</td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 4,</td>
<td>1.4</td>
<td>38.3</td>
<td>15.8</td>
<td>0.1</td>
<td>99.4</td>
<td>99.2</td>
<td></td>
</tr>
<tr>
<td>N/M = 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 4,</td>
<td>1.6</td>
<td>37.4</td>
<td>16.5</td>
<td>9.9</td>
<td>0.4</td>
<td>99.3</td>
<td>98.9</td>
</tr>
<tr>
<td>N/M = 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 4,</td>
<td>1.8</td>
<td>41.3</td>
<td>19.1</td>
<td>9.6</td>
<td>0.8</td>
<td>99.3</td>
<td>98.7</td>
</tr>
<tr>
<td>N/M = 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 4,</td>
<td>1.9</td>
<td>41.8</td>
<td>19.1</td>
<td>9.1</td>
<td>1.1</td>
<td>99.3</td>
<td>98.5</td>
</tr>
<tr>
<td>N/M = 20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 8,</td>
<td>0.8</td>
<td>41.1</td>
<td>12.5</td>
<td>11.4</td>
<td>0.1</td>
<td>99.3</td>
<td>99.2</td>
</tr>
<tr>
<td>N/M = 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 8,</td>
<td>0.8</td>
<td>38.9</td>
<td>12.9</td>
<td>10.3</td>
<td>0.4</td>
<td>99.3</td>
<td>99.0</td>
</tr>
<tr>
<td>N/M = 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 8,</td>
<td>0.9</td>
<td>42.8</td>
<td>14.8</td>
<td>9.8</td>
<td>0.8</td>
<td>99.4</td>
<td>98.7</td>
</tr>
<tr>
<td>N/M = 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRSpl WO = 8,</td>
<td>1.0</td>
<td>46.5</td>
<td>16.4</td>
<td>9.8</td>
<td>1.1</td>
<td>99.3</td>
<td>98.5</td>
</tr>
<tr>
<td>N/M = 20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7-1. Plot of OTS of DRSpl with D10
Finally, we discuss the convergence of DRSpl. It was found that in all our experiments DRSpl converged before \((K = 10)\) was met. The maximum and mean number of iterations of DRSpl observed for various input parameter values are shown in Table 7-6.
Based on the results in Table 7-6, we conclude that the convergence criteria of DRSpl is effective.
7.4 Solve AESVM on Data that is Too Large to Fit in Memory with One CPU

When datasets are larger than the memory (RAM) capacity of computers, applying SVM solvers can be problematic. This is because standard algorithms will require
Figure 7-8. Plot of $EOS$ of DRSpl with D13

Table 7-6. Convergence of DRSpl with datasets D10, D11, D12 and D13. The mean and maximum (mean, max.) of number of iterations required to converge are shown.

<table>
<thead>
<tr>
<th>$WO$</th>
<th>$\frac{N}{M}$</th>
<th>D10</th>
<th>D11</th>
<th>D12</th>
<th>D13</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>20</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>16</td>
<td>50</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>16</td>
<td>100</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>16</td>
<td>300</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>24</td>
<td>20</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>24</td>
<td>50</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>24</td>
<td>100</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>24</td>
<td>300</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>32</td>
<td>20</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>32</td>
<td>50</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>32</td>
<td>100</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>32</td>
<td>300</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>40</td>
<td>50</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>40</td>
<td>100</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
<tr>
<td>40</td>
<td>300</td>
<td>2.6, 5</td>
<td>4.5, 5</td>
<td>3.1, 5</td>
<td>4.0, 5</td>
</tr>
</tbody>
</table>

multiple read and write operations to the secondary memory (such as hard disks), which are usually very time consuming. DRSpl can be modified for use as an approximate
linear AESVM solver that needs to read the training dataset from secondary memory only a limited number of times. The data is read in blocks and for each block the representative set is computed. The representative set of each block is written to secondary memory. When all the blocks are read, the solution of the AESVM problem on the representative set is computed. The algorithm is very similar to DRSpl, with the only difference being that each DRSplWorker process is executed serially on the same CPU as DRSplMaster. Due to the similarity of the proposed algorithm with DRSpl, its performance is expected to be similar to DRSpl.

7.5 Non-Negative Matrix Factorization using Approximate Extreme Points

Non-negative matrix factorization (NMF) has a variety of applications in machine learning such as document clustering [80], image recognition [44], and topic modeling [1]. In NMF, the non-negative matrix factors of a given non-negative matrix are computed [67]. For a $D \times N$ non-negative matrix $Z$, it finds a $D \times R$ non-negative matrix $W$ and a $R \times N$ non-negative matrix $H$ such that it minimizes $\| Z - WH \|_F^2$. The matrix product $WH$ is a low rank approximation (rank $R$) of $Z$. NMF has been widely used recently, due to applications that use large non-negative matrices such as in text mining [49]. Alternative factorization techniques such as singular value decomposition have the disadvantage that the factorized matrices are not easy to interpret, unlike the case of NMF. Several algorithms have been proposed to solve NMF efficiently [42, 48, 49, 67]. For the specific case of sparse datasets, techniques such as alternating least squares have been proposed [41]. However, this method can also be slow for large datasets and have several parameters to tune.

We propose Approximate Extreme Points NMF (AENMF) as a technique to be used for large, high dimensional datasets. AENMF breaks down $Z$ as $Z \approx WH$, where $W$ is a $D \times R$ non-negative matrix of $R$ approximate extreme points arranged column-wise and $H$ is a $R \times N$ non-negative matrix of weights. This can be accomplished efficiently, by using a modified version of DRSpl described below. Our approach differs from other
NMF algorithms, where the matrix $W$ is not limited to be selected from the columns of $Z$. Typically, $W$ and $H$ are computed to approximate $Z$ as best as possible. Not limiting $W$ to be a subset of the columns of $Z$ can lead to several scalability issues when dealing with large, highly sparse matrices. To compute a sparse $W$, special constraints can be added to NMF. But adding constraints to the optimization problem typically leads to larger computational complexity.

7.5.1 Description of AENMF

DRSpl can be applied to this problem with a few modifications. The modified algorithm AENMF, is given below.

$$[W, H] = \text{AENMF}(IF, R)$$

1. Add all $N$ vectors $z_i \in IF$ column-wise to $\{Z\}$
2. Run $[Z, \Delta] = \text{FLS2}(Z, P)$
3. Run $[Z, \Delta_2] = \text{SLS}(Z, V, \Delta)$
4. Using $\Delta_2$, identify each subset $Z_{q_r}$ of $Z$ and run $[W_{q_r}] = \text{DAEnmf}(Z_{q_r})$
5. Combine in the same order, all $W_{q_r}$ to obtain $W$
6. Initialize a matrix $H$ of size $R \times N$ with all elements set to 0

Set $h_{i,t} \leftarrow 1 \ \forall z_i \in W$, where $h_{i,j}$ is the element in the $i^{th}$ column and $j^{th}$ row of $H$

7. For each $z_i \in Z$ and $z_i \notin W$, execute getErrorNMF($z_i, W$)

Set the $i^{th}$ column of $H$ to $\overline{h}_i$, where $\overline{h}_i$ is the result of getErrorNMF($z_i, W$)

The values of $P$ and $V$ were set to the same values as in DRSpl. The function getErrorNMF($z_i, W$) is given below.

$$\min_{\overline{h}_i} \rho(z_i, W) = \|z_i - \sum_{t=1}^{\|W\|} h_{i,t} z_t \|^2$$

s.t. $z_t \in W, h_{i,t} \geq 0$

It can be seen that getErrorNMF($z_i, W$) is similar to getError($x_i, \Psi$) of DAEpl. The functions have different constraints, which are simplified for NMF. It can be seen that $z_i$
is approximated by a conical combination of $z_t \in W$ and not a convex combination as in DAEpl and DeriveAE. Even though $z_t \in W$, do not satisfy the definition of approximate extreme points we continue to use that terminology for ease of notation.

\[
[W_q] = DAEnmf(Z_{q_r})
\]

1. Set $W_{q_r} \leftarrow \emptyset$
2. Find $z_3 = \arg \max_{z_t \in Z_{q_r}} ||z_t||^2$
   Set $W_q \leftarrow W_q \cup z_3$
3. Find $z_b = \arg \max_{z_t \in Z_{q_r}} ||z_t - z_3||^2$
   Set $W_q \leftarrow W_q \cup z_b$
4. For $i = 3$ to $\kappa$
   (a) Find $z_i = \arg \max_{z_t \in Z_{q_r}} \text{getErrorNMF}(z_t, W_{q_r})$
   (b) Set $W_{q_r} \leftarrow W_{q_r} \cup z_i$

Even though AENMF can be easily parallelized using the strategy in DRSpl, we present it as a serial algorithm to evaluate the feasibility of using approximate extreme points for NMF. A major difference between AENMF and DRSpl is that the matrix $H$ is computed using the entire matrix $W$, rather than using just a portion of it. If the method in DRSpl or DeriveRS had been followed, $H$ would consist of sub-matrices $H_{q_r}$ of size $\kappa \times |Z_{q_r}|$. This strategy leads to a better approximation of $Z$ by $WH$. The disadvantage is that $H$ can have up to $RN$ non-zero elements, whereas with the method in DRSpl and DeriveRS, there would have been a maximum of only $\kappa N$ non-zero elements. However, since $R$ is typically set to a small value in NMF, this does not pose a serious problem. The time complexity of AENMF is $O(N(R^2 + \kappa^3))$, where the dependence on $R^2$ is due to step 7 of AENMF.

### 7.5.2 Experiments on AENMF

We modified DRSpl to implement AENMF. There are several recent studies on solving NMF with large datasets [4, 30, 42, 48–50]. A thorough comparison
with all these methods is beyond the scope of this work, especially because not all these methods have publically available implementations. We selected three methods \([4, 42, 48]\) to evaluate AENMF. The MATLAB Statistics toolbox function \texttt{nnmf} implements the alternate least squares method described in Berry et al. \([4]\). The method in Lin \([48]\), \texttt{alspgrad}, solves NMF using alternative non-negative least squares with projected gradients. Kim and Park \([42]\) described a active set based algorithm \texttt{asanls}, for non negativity constrained least squares to solve NMF that has good convergence properties.

To evaluate AENMF, we used the following datasets.

**D14:** \textit{Real-sim}\(^6\) - This dataset consists of 72309 data vectors with \(D = 20958\). The dataset has a density = 0.24%.

**D15:** \textit{RCV1}\(^7\) - This dataset \([47]\) has 20242 data vectors of 47236 features. The dataset has a density = 0.16%.

**D6, D8, D9:** These datasets were described in Chapter 5

The datasets given above were selected as they are large enough to consider for factorization, while being small enough to satisfy the large memory requirements of \texttt{nnmf}, \texttt{alspgrad} and \texttt{asanls}. In our experiments, we used the parameters \(\kappa = 10\) and \(R = 20, 40, 60, 80, 100\).

The following metrics were used to evaluate AENMF.

1. \textit{Mean Error residual, MER}: The mean error residual indicates the quality of approximation in NMF, and is computed as

\[
MER = \frac{1}{5} \sum_{t=1}^{5} \| Z - W_t H_t \|_F
\]

\(^6\) http://vikas.sindhwani.org/datasets/lskm/svml/real-sim.svml.gz

\(^7\) http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#rcv1.binary
where \( W_t \) and \( H_t \) denote the factors computed with the \( t^{th} \) value of \( R \).

2. **Total computation time of NMF**: The total time taken in seconds to solve NMF for the selected \( R \) values is useful in judging the computational requirements of an NMF solver.

3. **Mean non-zero elements in factors, MNF**: It indicates the percentage sparsity of the computed solution \( W \) and \( H \) in comparison to \( Z \).

\[
MNF = \frac{1}{5} \sum_{t=1}^{5} \frac{nnz(W_t) + nnz(H_t)}{nnz(Z)} \times 100
\]

where \( nnz(A) \) is the number of non-zero elements in \( A \).

The results of our performance evaluation of AENMF are presented in Table 7-7. AENMF required very low computation time in comparison to the other NMF solvers. The asanls solver, was the most time consuming among all the methods, taking orders of magnitude more time than AENMF. The projected gradient solver, alspgrad was also relatively fast, though it was generally slower than AENMF. When comparing \( MNF \), AENMF was observed to be comparable to the other algorithms. Among the algorithms nmmf, alspgrad, asanls and AENMF, asanls seems to give the least residual error and the sparsest solutions. On observing the low computation time required for AENMF and alspgrad, we combined the two methods by using the AENMF output \( W_t \) and \( H_t \) as initial values of alspgrad. The results of this combination are illustrated as AENMF + alspgrad. As seen in Table 7-7, this combination computed solutions orders of magnitude faster than asanls, while having comparable \( MER \) and \( MNF \) values.

The performance of the NMF solvers on the relatively large datasets D14 and D15, is illustrated in more detail in Figures 7-9 to 7-11. It can be seen that \( MER \) generally decreased with an increase in column size \( R \) of \( W \). The \( MER \) values of asanls and AENMF + alspgrad, are found to be similar in most cases. In Figure 7-10, the total computation time of AENMF can be seen to be much lower than the other algorithms, for all values of \( R \). The total computational time is observed to generally increase with \( R \). This is expected as the time complexity of all the algorithms varies in proportion with
The values of MNF in Figure 7-11 illustrate the impact of the modifications to DRSpl incorporated into AENMF. The results of AENMF were found to be denser than most of the other methods, due to a dense $H$ matrix. However, it should be noted that datasets D14 and D15 are relatively small compared to D10 - D13, and were selected due to the large computational requirements of the other algorithms. For sparse datasets with millions of features like D10, AENMF can be expected to give sparse solutions due to the sparsity of the computed $W$.

Considering the results in this Chapter, AENMF is suggested for use with large datasets such as D10 - D13, due to its ability to compute the solutions very fast. Moreover, AENMF does not have large memory requirements. The other algorithms not only require more computation time, but also more memory space due to the matrix operations that they use. For medium sized datasets such as D14 and D15, AENMF can be used if fast computation is desired. If a smaller MER or MNF is desired, the combination AENMF + alsgrad gives good results, while taking relatively less time.

Table 7-7. Performance of various NMF solvers

<table>
<thead>
<tr>
<th>Metric</th>
<th>Dataset</th>
<th>nnmf</th>
<th>alsgrad</th>
<th>asanls</th>
<th>AENMF</th>
<th>AENMF + alsgrad</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D6</td>
<td>1074.7</td>
<td>555.1</td>
<td>147.3</td>
<td>374.6</td>
<td>154.0</td>
</tr>
<tr>
<td>MER</td>
<td>D8</td>
<td>1234.6</td>
<td>1916.1</td>
<td>895.1</td>
<td>1437.0</td>
<td>976.1</td>
</tr>
<tr>
<td></td>
<td>D9</td>
<td>411.9</td>
<td>757.4</td>
<td>406.3</td>
<td>620.6</td>
<td>415.2</td>
</tr>
<tr>
<td></td>
<td>D14</td>
<td>279.2</td>
<td>268.7</td>
<td>255.6</td>
<td>268.0</td>
<td>257.7</td>
</tr>
<tr>
<td></td>
<td>D15</td>
<td>149.0</td>
<td>142.3</td>
<td>132.5</td>
<td>140.7</td>
<td>133.9</td>
</tr>
<tr>
<td>Total</td>
<td>D6</td>
<td>196.1</td>
<td>17.4</td>
<td>39266.1</td>
<td>3.1</td>
<td>3592.1</td>
</tr>
<tr>
<td>computation</td>
<td>D8</td>
<td>4712.6</td>
<td>44.5</td>
<td>31762.1</td>
<td>50.2</td>
<td>3772.9</td>
</tr>
<tr>
<td>time (s)</td>
<td>D9</td>
<td>965.9</td>
<td>25.2</td>
<td>32378.7</td>
<td>8.8</td>
<td>1827.9</td>
</tr>
<tr>
<td></td>
<td>D14</td>
<td>5647.9</td>
<td>262.7</td>
<td>65464.5</td>
<td>69.8</td>
<td>1335.5</td>
</tr>
<tr>
<td></td>
<td>D15</td>
<td>3108.8</td>
<td>198.6</td>
<td>45218.2</td>
<td>20.0</td>
<td>846.3</td>
</tr>
<tr>
<td>MNF %</td>
<td>D6</td>
<td>207.4</td>
<td>433.7</td>
<td>136.5</td>
<td>73.1</td>
<td>160.8</td>
</tr>
<tr>
<td></td>
<td>D8</td>
<td>20.3</td>
<td>40.3</td>
<td>16.8</td>
<td>8.2</td>
<td>13.4</td>
</tr>
<tr>
<td></td>
<td>D9</td>
<td>154.6</td>
<td>515.2</td>
<td>120.6</td>
<td>124.6</td>
<td>121.8</td>
</tr>
<tr>
<td></td>
<td>D14</td>
<td>71.2</td>
<td>117.0</td>
<td>46.6</td>
<td>110.7</td>
<td>53.9</td>
</tr>
<tr>
<td></td>
<td>D15</td>
<td>106.0</td>
<td>81.0</td>
<td>51.3</td>
<td>73.1</td>
<td>50.6</td>
</tr>
</tbody>
</table>
Figure 7-9. Plot of MER of NMF solvers with D14 and D15
Figure 7-10. Plot of total computation time of NMF solvers with D14 and D15
Figure 7-11. Plot of $MNF$ of NMF solvers with D14 and D15
CHAPTER 8
CONCLUSIONS AND FUTURE WORK

The algorithms described in this dissertation enable fast and efficient computation of the SVM solution by approximating it. The newly introduced optimization problem, AESVM, is almost identical to, but less complex than, the SVM problem. AESVM optimizes over only a subset of the training dataset and consequently is expected to give fast convergence with most SVM solvers. This subset is derived using the novel concept of approximate extreme points, that are approximations of the extreme points of the convex hull of a dataset. As described in Chapter 3 there are several theoretical results that indicate the similarity of the AESVM and SVM solutions when the error bound $\epsilon$ is small. In Chapter 4, an algorithm is proposed to efficiently compute all the approximate extreme points of a dataset for non-linear SVM kernels. The proposed algorithm, DeriveRS, is a divide and conquer method with a time complexity that is at most log-linear in the size of the training dataset. The extensive experimental results in Chapter 5, demonstrate the efficacy of using DeriveRS and AESVM to solve non-linear SVMs, especially when the training datasets are not high dimensional.

In Chapter 6 we have proposed DRSPost, an algorithm similar to DeriveRS but designed to post-process the solution of non-linear SVMs. By post-processing the solution, fast classification is achieved by replacing the support vectors with a set of its approximate extreme points. This algorithm is particularly useful when a loss function other than the hinge loss is used, as AESVM might not produce good results with such loss functions.

Finally, in Chapter 7 we introduced a fast, parallel-processing linear AESVM solver, DRSPl, that can be used in any parallel programming framework including MapReduce. DRSPl has the ability to compute a representative set of a specified size. This enables it to greatly reduce the communication bandwidth required to compute it on a cluster. Our experiments on large, high dimensional datasets demonstrate the
similarity of the solutions of DRSpl and linear SVM. A slight modification to this algorithm (AENMF) is also proposed to compute non-negative matrix factorization (NMF). Our experiments indicate the feasibility of using AENMF on large datasets due to its small time complexity. Moreover, when combined with another method it gave solutions with very low approximation error.

In the future, a thorough exploration of the state of the art NMF solvers and its comparison with AENMF will be performed. Modifications to AENMF, that will enable the addition of arbitrary basis vectors to the factor matrix \( W \) might result in lower approximation errors. This can be accomplished using the optimization techniques used in other NMF solvers, such as alternating least squares.

It is possible that the modifications in AENMF, if used in the routine DRSplWorker might result in better classification performance of DRSpl. For example, \( \beta_a \) can be computed with the entire set \( X^+ \) in DRSplWorker, rather than on just portions of it as it is done now. The scalability of DRSpl can be improved by having several DRSpl processes work on portions of the training dataset. The computed extreme points can be combined to form a new training set for a linear AESVM solver or it can be used as the input dataset for another DRSpl solver. Other possible extensions of this work are the application of approximate extreme points to other algorithms such as support vector clustering, and support for loss functions other than the hinge loss.
REFERENCES


data sets via minimum enclosing ball clustering. *Neurocomputing*, 71:611–619,

decoding. In *Proceedings of International Joint Conference on Neural Networks*,


l2-loss linear support vector machines. *Journal of Machine Learning Research*, 9:


Map-Reduce for machine learning on multicore. *Advances in Neural Information


[22] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large

[23] P. Drineas and M. W. Mahoney. On the Nyström method for approximating a gram

analyses. In *Proceedings of the Fourth IEEE International Conference on

library for large linear classification. *Journal of Machine Learning Research*, 9:


BIOGRAPHICAL SKETCH

Manu Nandan was born in Trichur, India. After finishing tenth and twelfth grade from Hari Sri Vidhya Nidhi and St. Thomas College respectively, he was admitted to the National Institute of Technology, Calicut (NITC) for his undergraduate studies in Electronics and Communication Engineering. On graduating from NITC, he worked as a software developer at Delphi automotive systems, Bangalore, India for four years. He pursued his desire to conduct research on machine learning by joining the graduate program in the Computer and Information Science and Engineering Department at the University of Florida in 2007. His primary area of research is machine learning and its application to seizure detection and text classification. He graduated with a Ph.D. in computer engineering from the University of Florida in December 2013.