2016

Direct L2 Support Vector Machine

Ljiljana Zigic
zigicl@vcu.edu

DIRECT L2 SUPPORT VECTOR MACHINE

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

LJILJANA ZIGIC

M.Sc. Eng, University of Novi Sad (Serbia), 2012

Director: Vojislav Kecman,

Professor, Department of Computer Science

Virginia Commonwealth University

Richmond, Virginia

April, 2016

Abstract

This dissertation introduces a novel model for solving the L2 support vector machine dubbed Direct L2 Support Vector Machine (DL2 SVM). DL2 SVM represents a new classification model that transforms the SVM's underlying quadratic programming problem into a system of linear equations with nonnegativity constraints. The devised system of linear equations has a symmetric positive definite matrix and a solution vector has to be nonnegative.

Furthermore, this dissertation introduces a novel algorithm dubbed Non-Negative Iterative Single Data Algorithm (NN ISDA) which solves the underlying DL2 SVM's constrained system of equations. This solver shows significant speedup compared to several other state-of-the-art algorithms. The training time improvement is achieved at no cost, in other words, the accuracy is kept at the same level. All the experiments that support this claim were conducted on various datasets within the strict double cross-validation scheme. DL2 SVM solved with NN ISDA has faster training time on both medium and large datasets.

In addition to a comprehensive DL2 SVM model we introduce and derive its three variants. Three different solvers for the DL2's system of linear equations with nonnegativity constraints were implemented, presented and compared in this dissertation.

Acknowledgements

Immeasurable appreciation and deepest gratitude for the help and support is extended to the following persons who have, in one way or another, contributed to the success of my graduate studies.

I wish to thank Dr. Vojislav Kecman, my advisor and a lifetime researcher, for his support, guidance, and valuable comments that benefited this dissertation greatly.

I am exceptionally thankful to Dr. Robert Strack for his work which has notably advanced this study. I can honestly say I would not have made such a quick progress were I not able to rely on his software platform GSVM.

I am thankful to members of my dissertation committee, their effort in editing this study and sharing valuable ideas during both my proposal and my defense is greatly appreciated.

Lastly, I wish to express appreciation to my family for their love and support along the way. I am equally indebted to all of my dear friends who have, even from far away, made this work more enjoyable.

Contents

List of Algorithms

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

Support vector machines (SVM) are powerful supervised learning algorithms widely used for classification and regression analysis. They represent a major development in machine learning algorithms. The main advantages of SVMs are high generalization ability, adaptability to various problems by changing kernel functions and nice theoretical property of having a unique global optimum. However, in the case of large scale problems (say, hundreds of thousands samples and more) SVM's computation and storage requirements increase rapidly, making many problems infeasible. In practice, SVMs are reduced to a quadratic programming (QP) problem which identifies support vectors from non-support vectors. The training time required to obtain a solution of a QP solver is very much dependent on the topology of training data, that is, the surface of the cost function and constraints. For an extremely hard QP (with a badly conditioned matrix) QP solvers may not even converge to a solution. It is said that the training time complexity of a QP problem is $O(n^3)$ (where n is the number of training patterns) and its space complexity is at least quadratic [1]. Hence, a major stumbling block is in scaling up QP problems to large data sets.

As has become apparent in the last few years, the sizes of datasets are growing steadily and will continue to do so. These datasets provide huge statistical samples which have the power to enhance the analytical results. The general rule is that the larger the data sample, the more accurate the analysis. Data analysis will be inherently diminished if we make a compromise of sampling when we have the potential to explore the entire dataset. This expedites the development of training algorithms that

scale well with the number of samples. The challenges of developing such algorithms involves two aspects. One of them is a thorough theoretical analysis of a problem and the second is a proper implementation that will achieve required computational complexity.

There have been several different approaches to large scale learning problems. First attempts at speeding up the training time and decreasing the memory requirements were aimed at decomposing the underlying SVM's quadratic programming (QP) problem. Vapnik et al. in [2] suggested first such method which became known as *chunking*, where subsets of the training data are optimized iteratively, until the global optimum is reached. Another such strategy was introduced by Osuna et al. [3]. Sequential minimal optimization (SMO) by Platt [4] where the chunk size is reduced to 2 is another approach to decomposing a QP problem. Several of the most popular software packages out there such as SVMlight [5] and LIBSVM [6] are based on these algorithms. Efficient SVM implementations incorporate various heuristics to speedup the training time even more, one such strategy is called *shrinking* and is used for detecting the non-support vectors early on in the computation. Another strategy that leads to a significant savings in computation is caching of the kernel data.

Recent approaches to large scale problems involve parallelization of the learning task. This idea has been exploited in several different ways and on several different hardware architectures ([7, 8, 9, 10]). Improving speed through parallelization is a difficult task due to the high interdependencies of the steps. CPU and GPU parallel frameworks offer enormous computation power, and have the potential to provide huge speedups. However, algorithms must be carefully decomposed and the problem must be approached in a fundamentally different way in order to effectively utilize a specific parallel architecture. Therefore, parallelization of SVMs requires significant effort in fine-tuning the level of parallelization and the overhead induced, and this

fine-tuning must be done for a particular distributed architecture.

Another recent advancement in handling large scale problems is based on geometric interpretation of the SVM model in the kernel induced feature space. These *geometric approaches* include solving the SVMs learning problem by convex hull [11] and/or minimum enclosing ball [1]. The novel algorithms by Strack et al. [12] known as the Sphere Support Vector Machine (SphereSVM) and Minimal Norm Support Vector Machine (MNSVM) combine the two techniques (convex hull and enclosing ball) and demonstrate high capability for handling large datasets.

All of these algorithms have their advantages and disadvantages. Nonetheless, a careful study of the problems posed by large scale data is not fully investigated which is the main motivation for this dissertation. Additional motivation is the fact that these algorithms are readily applicable to small and medium sized datasets, which is still of practical interest.

This dissertation tackles both challenges mentioned previously when dealing with large data. First, it gives a new theoretical insight into L2 SVM learning task. Second, it introduces a new algorithm for solving a system of linear equations with nonnegativity constraints.

Direct L2 Support Vector Machine introduced here can be viewed as a comprehensive model from which several other L2 SVM approaches can be derived with very few adaptations. Namely, DL2 SVM has almost the same problem setting as the geometrical approaches introduced in [1] and [12]. Furthermore, it is similar to Least-Squares Support Vector Machine proposed by Suykens et al. [13] and Proximal Support Vector Machine (PSVM) by Mangasarian et al.[14] in a sense that it leads to a set of linear equations. NN ISDA which is a novel algorithm for solving the DL2's underlying problem shows significant speedup compared to several other state-of-the-art algorithms.

1.1 Contributions of the dissertation

The major contributions of the dissertation are:

- the introduction of a Direct L2 Support Vector Machine
- the introduction of three variants of Direct L2 SVM's
- the introduction and implementation of a novel algorithm Non-Negative Iterative Single Data Algorithm
- implementation of the over-relaxation technique for NN ISDA
- implementation of the probabilistic speed-up technique for NN ISDA

CHAPTER 2

BACKGROUND

The Generalized Portrait algorithm introduced by Vapnik [15] in mid 1960's gave a foundation to modern maximum margin classifiers. Support vector machines, in simple terms, involve finding a hyperplane that best separates two classes of points with the maximum margin. Essentially, it is a constrained optimization problem where the goal is to maximize the margin subject to the constraint that it perfectly classifies the data. For a long time SVMs went largely unnoticed until 1990's when (based on the work of Aizerman et al. [16]) Boser, Guyon and Vapnik [2] generalized the original linear problem, the so called *Hard-Margin Support Vector Machine*, to a nonlinear case. Finally, Cortes and Vapnik [17] introduced the *Soft-Margin Support Vector Machine* which allowed classification in the case of non-separable (overlapped) data.

In training an SVM classifier the goal is to maximize the classification performance on the training data, while keeping the generalization error (i.e., the classification error on unseen data) low. In other words, if a classifier is fitting the training data too well, its generalization ability might be degraded.

This chapter gives theoretical background of SVMs. First, the hard-margin support vector machines are discussed where the training data are linearly separable in input space. Afterwards, we extend the hard-margin classifier to the case of linearly non-separable training data, and give a derivation of soft-margin support vector machine.

2.1 Hard-margin support vector machine

Let $\chi = \{\boldsymbol{x}_i\}$ be a training dataset containing m input vectors $\boldsymbol{x}_i$ from d-dimensional input space i.e., $\boldsymbol{x}_i \in \mathbb{R}^d$, and let y_i represent their associated class labels. In the simplest classification problem, y_i has two possible values. Without the loss of generality we can denote these two values with 1 and -1. This type of classification is called *binary classification*. From now on we assume the input training dataset χ is given as a matrix $\boldsymbol{X}$ of size (m, d) consisting of m input vectors $\boldsymbol{x}_i$ given as rows, and the output label vector $\boldsymbol{y}$ is given as a column vector of length m.

The function $d(\boldsymbol{x})$ is a linear decision function defined as:

$$d(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b, \tag{2.1}$$

where $\boldsymbol{w}$ is a d-dimensional weight vector, and b is a *bias* term. To ensure correct classification, both $\boldsymbol{w}$ and bias b must satisfy the following condition:

$$y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) \geq 1 \quad i = 1, ..., m. \tag{2.2}$$

If the data are linearly separable, there exist $\boldsymbol{w}$ and b such that 2.2 is always satisfied. The goal of the classification is to assign the output value 1 or -1 to the given input vector $\boldsymbol{x}$, therefore, the classification function is given as:

$$\text{sign}(d(\boldsymbol{x})) := \begin{cases} \text{class } +1 & \text{if } d(\boldsymbol{x}) > 0 \\ \text{class -1} & \text{if } d(\boldsymbol{x}) < 0 \end{cases} \tag{2.3}$$

The hyperplane defined as $d(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b = 0$ forms the *separating hyperplane*. There is an infinite number of separating hyperplanes that satisfy the condition given by 2.2 but the location of this hyperplane has a direct influence on the classifier's generalization ability. Namely, SVMs try to maximize the margin (see Fig. 1) between

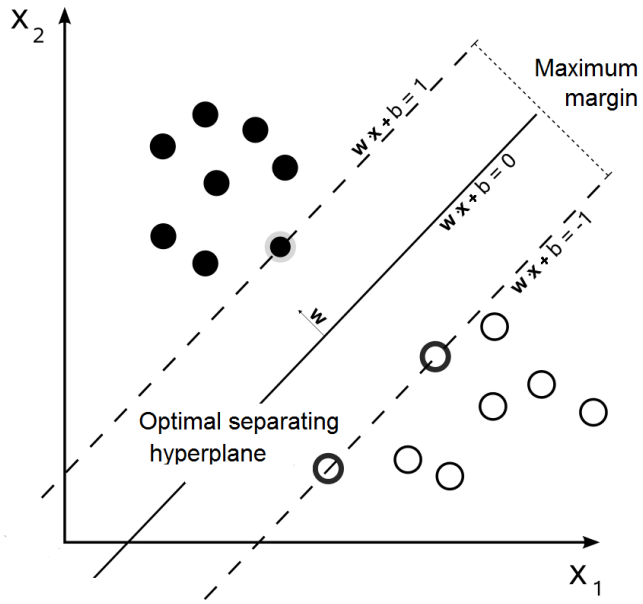


Fig. 1.: Maximum margin hyperplane in a two-dimensional space

the separating hyperplane and the samples from the opposite classes. Intuitively, large distance between the decision boundary and the samples makes the probability of misclassification lower. Mathematically, the width of the margin is always equal to the $2/\|\boldsymbol{w}\|$. The hyperplane with the maximum margin is called the *optimal separating hyperplane*. The training data nearest to the separating hyperplane (circled points in Fig. 1) whose decision function equals exactly 1 or -1 are called the *support vectors*.

In order to maximize the margin we need to minimize the norm of the weight vector $\boldsymbol{w}$. Therefore, the optimal separating hyperplane for the hard-margin SVM problem is obtained by solving the following optimization problem for $\boldsymbol{w}$ and b:

$$\text{minimize} \quad Q(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2, \tag{2.4}$$

$$\text{subject to} \quad y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) \geq 1, \qquad i = 1, ..., m, \tag{2.5}$$

where $\boldsymbol{x}_i$ is the input vector, y_i is the class label, b represents the bias term and m

stands for the size of the dataset (the number of samples). The assumption of linear separability means there exist $\boldsymbol{w}$ and b that satisfy 2.5, but in the case of overlapping classes there is no feasible solution to this optimization problem. To overcome this problem Cortes and Vapnik introduced the soft-margin SVM.

2.2 Soft-margin support vector machine

In the case of soft-margin SVMs, constraints $y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) \geq 1$ are relaxed to allow some errors during the classification. This is done by introducing the nonnegative slack variables ξ_i:

$$y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) \geq 1 - \xi_i \qquad i = 1, ..., m. \tag{2.6}$$

By introducing the slack variables ξ_i the solution to overlapped data always exists. Fig. 2 shows the situation where we have inseparable data in 2-dimensional input space. For the training point $\boldsymbol{x}_3$ the value of slack variable is between 0 and 1, and the data does not lie on the margin but it is still correctly classified. In the case of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ the values of their corresponding slack variables ξ_1 and ξ_2 are greater than 1 and so the points are misclassified. Point $\boldsymbol{x}_4$ lies on the margin and is on the correct side of the separating boundary so its slack value is 0.

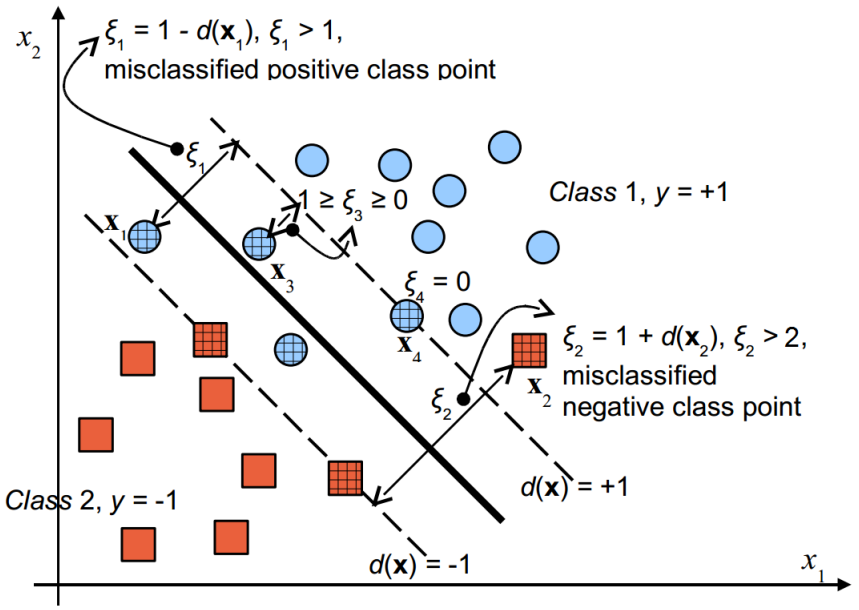


Fig. 2.: The case of inseparable data in a two-dimensional space

By including the slack variables, the SVM's optimization problem has changed. Now, to obtain the optimal separating hyperplane, the following minimization problem needs to be solved:

$$\text{minimize} \quad Q(\boldsymbol{w}, \boldsymbol{\xi}) = \frac{1}{2} \|\boldsymbol{w}\|^2 + \frac{C}{p} \sum_{i=1}^{m} \xi_i^p, \tag{2.7}$$

$$\text{subject to} \quad y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) \geq 1 - \xi_i, \qquad i = 1, ..., m. \tag{2.8}$$

$$\text{and} \quad \xi_i \geq 0, \qquad i = 1, ..., m. \tag{2.9}$$

where C is the penalty parameter controlling the trade-off between the maximization of margin and minimization of the classification error. If we set parameter C to some small value we make the cost of misclassification low thus allowing more errors for the sake of larger margin (wider cushion), in other words we are more flexible about the margin. As we increase C we punish errors more (misclassifications are considered bad) and thus we are forcing the SVM algorithm to *explain* the data stricter.

The obtained separating hyperplane is called the *soft-margin hyperplane*. The value of p in 2.7 is selected to be either 1 or 2 [18]. When the value for p is chosen to be 1 the corresponding support vector machine is called *L1 SVM* since it is minimizing the L1 distance of the points on the wrong side of the boundary to their corresponding margin (in the input space). When $p = 2$ the support vector machine is called *L2 SVM*. Both L1 and L2 SVMs will be explained in details in the following subsections.

Linear soft-margin SVM explained here is capable of dealing with inseparable data because it is allowing some classification errors to be made by including the slack variables, however it is still not fully equipped to handle highly complex input topological structures. This problem is solved by mapping the input space to a high-dimensional feature space in order to enhance linear separability in that feature space. This process will be further explained in 2.2.1.1.

2.2.1 L1 support vector machine

L1 SVM is a special type of soft-margin support vector machine that has p value set to 1 (see 2.7). The optimization problem is given as follows:

$$\text{minimize} \quad Q(\boldsymbol{w}, \boldsymbol{\xi}) = \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{m} \xi_i, \tag{2.10}$$

$$\text{subject to} \quad y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) \geq 1 - \xi_i, \qquad i = 1, ..., m. \tag{2.11}$$

$$\text{and} \quad \xi_i \geq 0, \qquad i = 1, ..., m. \tag{2.12}$$

The given optimization problem with constraints can be solved by converting it into the equivalent dual problem by using the idea of Lagrangian. This has a consequence of decreasing the number of unknown variables. Particularly, the number of unknown variables becomes equal to the number of training data. To perform this conversion,

first the primal Lagrangian is formed:

$$L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i (y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i) - \sum_{i=1}^{m} \beta_i \xi_i, \quad (2.13)$$

where

$$\alpha_i \geq 0 \qquad i = 1, ..., m, \quad (2.14)$$

$$\beta_i \geq 0 \qquad i = 1, ..., m, \quad (2.15)$$

are the nonnegative *Lagrange multipliers*.

For the optimal solution, the following Karush-Kuhn-Tucker (KKT) conditions must be satisfied:

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \boldsymbol{w}} = \mathbf{0}, \quad (2.16)$$

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial b} = 0, \quad (2.17)$$

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \xi_i} = 0, \quad (2.18)$$

$$\alpha_i (y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i) = 0 \qquad i = 1, ..., m, \quad (2.19)$$

$$\beta_i \xi_i = 0 \qquad i = 1, ..., m, \quad (2.20)$$

$$\alpha_i \geq 0, \beta_i \geq 0, \xi_i \geq 0 \qquad i = 1, ..., m. \quad (2.21)$$

The equations 2.16 to 2.18 can be reduced to

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{x}_i, \quad (2.22)$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \qquad i = 1, ..., m, \quad (2.23)$$

$$\alpha_i + \beta_i = C \qquad i = 1, ..., m. \quad (2.24)$$

Now, substituting 2.22 to 2.24 into the formula for primal Lagrangian 2.13 we obtain the following dual problem for L1 SVM:

$$\text{minimize} \quad L_d(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j) - \sum_{i=1}^{m} \alpha_i, \tag{2.25}$$

$$\text{subject to} \quad \sum_{i=1}^{m} \alpha_i y_i = 0, \qquad i = 1, ..., m, \tag{2.26}$$

$$\text{and} \quad 0 \leq \alpha_i \leq C \qquad i = 1, ..., m. \tag{2.27}$$

As can be seen in the expression above, the optimization problem depends on the dot product of samples $(\boldsymbol{x}_i \cdot \boldsymbol{x}_j)$. This idea will be used in the next section, as it brings a very useful feature. The constraints given by 2.27 are called the *box constraints*. For quadratic programming problems, the values of the primal and dual objective functions coincide at the optimal solution, if it exists. Therefore, once we find $\boldsymbol{\alpha}$ we have found the solution to the L1 SVM problem. Dual form of a hard-margin SVM is very similar to L1 SVM's, the only difference is that the Lagrange multipliers α_i do not have an upper bound.

In matrix notation dual Lagrangian is equivalent to:

$$L_d(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T(\boldsymbol{y}\boldsymbol{y}^T \circ XX^T)\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha}, \tag{2.28}$$

where the operator $\circ$ represents element-wise matrix multiplication (Hadamard product).

From the KKT conditions given by 2.19, 2.20 and 2.24 there are three possible cases for the values of α_i:

1. $\alpha_i = 0$ then $\xi_i = 0$. A training point $\boldsymbol{x}_i$ is correctly classified.

2. $0 < \alpha_i < C$ then $y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i = 0$ and $\xi_i = 0$, so $y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) = 1$ and thus $\boldsymbol{x}_i$ is a support vector. These support vectors are called *unbounded (free)*

support vectors and they lie on the margin.

3. $\alpha_i = C$ then $y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i = 0$ and $\xi_i \geq 0$, and so $\boldsymbol{x}_i$ is a support vector. These support vectors are called *bounded support vectors*. In this case if $0 \leq \xi_i < 1$, $\boldsymbol{x}_i$ is correctly classified, but if $\xi_i \geq 1$, the input data $\boldsymbol{x}_i$ is misclassified.

Finally, the decision function for L1 SVM classifier is calculated as:

$$d(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i y_i (\boldsymbol{x}_i \cdot \boldsymbol{x}) + b = \sum_{sv=1}^{\#SV} \alpha_{sv} y_{sv} (\boldsymbol{x}_{sv} \cdot \boldsymbol{x}) + b. \tag{2.29}$$

Note that the summation is performed over the support vectors only because only for support vectors the value of α_i is positive ($0 < \alpha_i \leq C$), otherwise α_i is zero.

From the KKT condition 2.19 one can derive that the bias term satisfies:

$$b = \frac{1}{|U|} \sum_{i:\boldsymbol{x}_i \in U} \left(y_i - \sum_{j:\boldsymbol{x}_j \in S} \boldsymbol{x}_j \cdot \boldsymbol{w}) \right), \tag{2.30}$$

where S represents the set of all support vectors, and U represents the set of all unbounded support vectors (for which $0 < \alpha_i < C$ and $\xi_i = 0$).

2.2.1.1 Kernel L1 SVM

Even though the L1 SVM hyperplane is determined optimally, if the training data are not linearly separable, the classifier may not have good generalization ability. In order to enhance linear separability, the original input space is mapped into a high-dimensional feature space. When a linear SVM is applied in that feature space (that is, applied to those transformed points) the solution corresponds to a non-linear function in the original space.

An easy way to interpret a kernel is to think of it as a "distance function", it takes in two points from the input space (scalars, vectors) and gives their distance.

According to Mercer's theorem if a kernel is positive definite then it has an associated transformation of input points into some high dimensional "feature space" usually denoted as $\varphi(\boldsymbol{x}_i)$. In practice, kernels calculate the scalar product of those transformed feature vectors $\varphi(\boldsymbol{x}_i)$ and $\varphi(\boldsymbol{x}_j)$ which turns out to be a quite simple operation on input space vectors:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \varphi(\boldsymbol{x}_i) \cdot \varphi(\boldsymbol{x}_j). \tag{2.31}$$

The advantage of using a kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ during training and classification, as opposed to just using $\varphi(\boldsymbol{x}_i)$, is that we do not need to deal with high-dimensional feature space explicitly. This technique is called the *kernel trick*. Using this trick the L1 SVM dual problem becomes:

$$\text{minimize} \quad L_d(\boldsymbol{\alpha}) = \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{m} \alpha_i, \tag{2.32}$$

$$\text{subject to} \quad \sum_{i=1}^{m} y_i \alpha_i = 0 \qquad i = 1, ..., m, \tag{2.33}$$

$$\text{and} \quad 0 \leq \alpha_i \leq C \qquad i = 1, ..., m. \tag{2.34}$$

Note that the scalar product $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ in 2.25 was simply replaced by a kernel function. We say that $\boldsymbol{K}$ is a *kernel matrix* or simply *kernel* if $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i \cdot \boldsymbol{x}_j$. If a kernel matrix satisfies a Mercer's condition, it will be *symmetric positive semidefinite*, and thus the optimization problem is a quadratic programming problem which brings up the important consequence - the objective function always has a global minimum, [18]. This is one of the important advantages of SVMs over neural networks which have numerous local minima.

In matrix notation the optimization function for L1 SVM is now equivalent to

$$L_d(\boldsymbol{\alpha}) = \frac{1}{2} \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha} - \mathbf{1}^T \boldsymbol{\alpha}, \tag{2.35}$$

where $\boldsymbol{H} = [y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j}$ is called the *Hessian matrix*, which is, same as kernel matrix $\boldsymbol{K}$, positive semidefinite.

Finally, we can calculate the decision function for L1 support vector machine as

$$d(\boldsymbol{x}) = \sum_{j=1}^{m} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b. \tag{2.36}$$

For the unbounded support vector $\boldsymbol{x}_i$ the bias term satisfies

$$b = y_i - \sum_{j:\boldsymbol{x}_j \in S} \alpha_j y_j k(\boldsymbol{x}_j, \boldsymbol{x}_i), \tag{2.37}$$

where S is the set of all support vectors. To ensure higher precision of the bias we take the average:

$$b = \frac{1}{|U|} \sum_{i:\boldsymbol{x}_i \in U} \left(y_i - \sum_{j:\boldsymbol{x}_j \in S} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}_j) \right), \tag{2.38}$$

where U is a set of unbounded support vectors.

Table 1.: Most frequently used kernel functions

Kernel function	**Name**	**Properties**
$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\boldsymbol{x}_i \boldsymbol{x}_j^T\right)$	Linear	CPD[1]
$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\boldsymbol{x}_i \boldsymbol{x}_j^T + 1\right)^d$	Polynomial of degree d	PD[2]
$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}$	Gaussian RBF	PD[2]
$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = tanh\left(\boldsymbol{x}_i \boldsymbol{x}_j^T + b\right)$	Multilayer Perceptron	CPD[1]

If data are linearly separable in the input space then the linear kernel is used ($k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i \cdot \boldsymbol{x}_j$), but if the mapping function $\varphi(\boldsymbol{x})$ is nonlinear, then the obtained

[1]Conditionally Positive Definite

[2]Positive Definite

decision function will be nonlinear as well. Table 1 shows the most popular kernel functions used in support vector machines. In addition, many other kernels were developed to suite specific applications, such as image processing, text classification, speech recognition, etc. More on classification of kernels, from a statistical point of view, can be found in [19].

2.2.2 L2 support vector machine

L2 support vector machine is a soft-margin SVM which optimizes the sum of squared errors. The L2 SVM's optimization function is given as follows:

$$\text{minimize} \quad Q(\boldsymbol{w}, \boldsymbol{\xi}) = \frac{1}{2} \|\boldsymbol{w}\|^2 + \frac{C}{2} \sum_{i=1}^{m} \xi_i^2 \tag{2.39}$$

$$\text{subject to} \quad y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) \geq 1 - \xi_i \qquad i = 1, ..., m. \tag{2.40}$$

where $\boldsymbol{w}$ is a d-dimensional weight vector, b is the bias term, ξ_i are the slack variables, and C is the penalty parameter. Just as in L1 SVM, the first term in the objective function minimizes the margin, the second term controls the number of misclassifications and C represents the trade-off between the two. The only difference between L1 and L2 SVM is that L2 SVM uses the sum of squared slack variables. This difference leads to some interesting properties in dual space.

To solve this QP problem we will start by introducing the Lagrange multipliers α_i (≥ 0) and forming the primal Lagrangian:

$$L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \frac{1}{2} \|\boldsymbol{w}\|^2 + \frac{C}{2} \sum_{i=1}^{m} \xi_i^2 - \sum_{i=1}^{m} \alpha_i (y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i), \tag{2.41}$$

To get the optimal solution the following Karush-Kuhn-Tucker conditions must be

satisfied:

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{m} \alpha_i \beta_i \boldsymbol{x}_i = \mathbf{0}, \tag{2.42}$$

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha})}{\partial b} = \sum_{i=1}^{m} y_i \alpha_i = 0, \tag{2.43}$$

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha})}{\partial \xi_i} = C\xi_i - \alpha_i = 0, \tag{2.44}$$

$$\alpha_i \left(y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i\right) = 0 \qquad i = 1, ..., m. \tag{2.45}$$

Substituting 2.42 to 2.44 into the primal L2 SVM Lagrangian (2.41) we can obtain its dual form:

$$\text{minimize} \quad L_d(\boldsymbol{\alpha}) = \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j + \frac{\delta_{ij}}{C}) - \sum_{i=1}^{m} \alpha_i, \tag{2.46}$$

$$\text{subject to} \quad \sum_{i=1}^{m} y_i \alpha_i = 0, \tag{2.47}$$

$$\text{and} \quad \alpha_i \geq 0 \qquad i = 1, ..., m, \tag{2.48}$$

where δ_{ij} is Kronecker's delta function in which $\delta_{ij} = 1$ for $i = j$ and $\delta_{ij} = 0$ otherwise. There are two differences with respect to L1 support vector machine worth pointing out here. One of them is the addition of δ_{ij}/C term in 2.46 and the second is the removal of the upper bound C on α_i (see 2.27).

Matrix notation for the L2 SVM dual form 2.46 is:

$$L_d(\boldsymbol{\alpha}) = \frac{1}{2} \boldsymbol{\alpha}^T (\boldsymbol{y}\boldsymbol{y}^T \circ XX^T + \frac{1}{C} I) \boldsymbol{\alpha} - \mathbf{1}^T \boldsymbol{\alpha}, \tag{2.49}$$

where I is an identity matrix and the operator $\circ$ represents element-wise matrix multiplication.

The decision function for L2 SVM is the same as for L1 SVM given in 2.29, but

the bias term b is calculated in a slightly different way:

$$b = \frac{1}{|S|} \sum_{i:\boldsymbol{x}_i \in S} \left(y_i - \sum_{j:\boldsymbol{x}_j \in S} \alpha_j y_j ((\boldsymbol{x}_i \cdot \boldsymbol{x}_j) + \frac{\delta_{ij}}{C}) \right), \tag{2.50}$$

Note the addition of a $\frac{\delta_{ij}}{C}$ term.

2.2.2.1 Kernel L2 SVM

It is possible to apply the kernel trick for L2 SVMs as well by simply substituting the scalar product $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ in 2.46 and 2.50 with a kernel function:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \varphi(\boldsymbol{x}_i) \cdot \varphi(\boldsymbol{x}_j). \tag{2.51}$$

Now, the dual objective function for L2 SVM becomes:

$$\text{minimize} \quad L_d(\boldsymbol{\alpha}) = \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{\delta_{ij}}{C}) - \sum_{i=1}^{m} \alpha_i, \tag{2.52}$$

$$\text{subject to} \quad \sum_{i=1}^{m} y_i \alpha_i = 0, \tag{2.53}$$

$$\text{and} \quad \alpha_i \geq 0 \qquad i = 1, ..., m. \tag{2.54}$$

Note the difference in respect to L1 support vector machines, namely, if we simply replace $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ with $k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{\delta_{ij}}{C}$ and remove the upper bound on α_i we can obtain the L2 support vector machine. Kernel matrix $\boldsymbol{K}$ is now expressed as

$$\boldsymbol{K} = \left[k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{\delta_{ij}}{C} \right]_{ij}. \tag{2.55}$$

The addition of $\frac{1}{C}$ element on a diagonal of the kernel has an important consequence in that now, the kernel is a symmetric positive definite matrix and thus the optimization problem is more computationally stable than that of L1 SVM. Also, by changing the value of parameter C we are directly changing the condition number of a kernel 2.55.

In matrix notation dual L2 SVM problem can be rewritten as

$$L_d(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T(\boldsymbol{H} + \frac{1}{C}\boldsymbol{I})\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha}, \tag{2.56}$$

where $\boldsymbol{H} = [y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j}$ is the Hessian matrix. The Hessian matrix is positive definite leading to a unique solution $\boldsymbol{\alpha}$.

Finally, the decision function is the same as for L1 SVM and is given as

$$d(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b, \tag{2.57}$$

where the bias term b is given by

$$b = \frac{1}{|S|} \sum_{i:\boldsymbol{x}_i \in S} \left(y_i - \sum_{j:\boldsymbol{x}_j \in S} \alpha_j y_j (k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{\delta_{ij}}{C}) \right). \tag{2.58}$$

Following subsections will discuss some variants of L2 SVM which relate to the ideas presented in this dissertation.

2.2.2.2 Least-Squares SVM

Besides the widely used classic L2 SVM presented above, few other variants have been developed. One of them is Least-Squares Support Vector Machine (LS SVM) proposed by Suykens et al. [13].

Least-Squares SVM is a support vector machine in which the training is performed by solving a set of linear equations as opposed to solving a quadratic programming problem. The LS SVM is formulated as follows,

$$\text{minimize} \quad Q(\boldsymbol{w}, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{2}\sum_{i=1}^{m} \xi_i^2 \tag{2.59}$$

$$\text{subject to} \quad y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) = 1 - \xi_i \qquad i = 1, ..., m. \tag{2.60}$$

$\boldsymbol{x}_i$ and y_i $(i = 1, ..., m)$ are m training input/output pairs, ξ_i are slack variables and C

is the penalty parameter, same as in the soft-margin SVMs presented previously. In the case of LS SVM slack variables ξ_i can be negative, so, when $\xi_i \geq 1$, input vector $\boldsymbol{x}_i$ is misclassified, otherwise it is correctly classified. Note that the only difference between L2 and LS SVM is that inequality constraints in L2 soft-margin support vector machines (see 2.40) are converted into equality constraints here. Decision function and all the parameters in this problem are the same as for L2 support vector machine.

Multiplying both sides of the equality constraints 2.60 by y_i we can get:

$$\boldsymbol{x}_i \cdot \boldsymbol{w} + b = y_i - y_i\xi_i \qquad i = 1, ..., m, \tag{2.61}$$

because ξ_i can take either positive or negative value and $|y_i| = 1$ the equation above can be rewritten as:

$$\boldsymbol{x}_i \cdot \boldsymbol{w} + b = y_i - \xi_i \qquad i = 1, ..., m. \tag{2.62}$$

Introducing Lagrange multipliers we obtain the unconstrained objective function for Least-Squares support vector machine,

$$L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{2}\sum_{i=1}^{m}\xi_i^2 - \sum_{i=1}^{m}\alpha_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b - y_i + \xi_i), \tag{2.63}$$

note that, unlike in L1 and L2 SVMs, Lagrange multipliers can be negative. KKT conditions for this problem are given as,

$$\boldsymbol{w} = \sum_{i=1}^{m}\alpha_i\boldsymbol{x}_i, \tag{2.64}$$

$$\sum_{i=1}^{m}\alpha_i = 0, \tag{2.65}$$

$$\alpha_i = C\xi_i \qquad i = 1, ..., m. \tag{2.66}$$

$$\boldsymbol{x}_i \cdot \boldsymbol{w} + b - y_i + \xi_i = 0 \qquad i = 1, ..., m. \tag{2.67}$$

Substituting equations 2.64 and 2.66 into equality constraint 2.67 and expressing it together with 2.65 in a matrix form leads to

$$\begin{pmatrix} \boldsymbol{\Omega} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ b \end{pmatrix} = \begin{pmatrix} \boldsymbol{y} \\ 0 \end{pmatrix} \tag{2.68}$$

where

$$\Omega_{ij} = \boldsymbol{x}_i \boldsymbol{x}_j + \frac{\delta_{ij}}{C}, \tag{2.69}$$

and δ_{ij} represents Kronecker's delta function, $\boldsymbol{y}$ is a vector of y values, and $\boldsymbol{\alpha}$ a vector of α values. Therefore, the original QP problem is transformed into a system of linear equations 2.68 for $\boldsymbol{\alpha}$ and b. The size of this system is $(m+1, m+1)$. The linear LS support vector machine can be extended to a nonlinear case by replacing the scalar product in Ω_{ij} with a kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ which leads to a nonlinear decision function.

To summarize, by changing the inequality to equality constrains the training results in solving a set of linear equations. The downside of this approach is that it produces a non-sparse, i.e. dense solutions. In terms of SVM, this means that all the data points will be support vectors. Note that matrix $\boldsymbol{\Omega}$ scales with the number of training samples so 2.68 shows that LS SVM cannot be used for large and ultra-large datasets because solving this system of linear equations is not feasible when m goes over a dozen of thousands of data points (of course, depending on the platform and condition number this number can be higher).

2.2.2.3 Proximal SVM

Another variant of L2 support vector machine is Proximal Support Vector Machine (PSVM) by Mangasarian et al. [14]. Proximal SVM has a different geometrical

interpretation than other L2 SVMs. Instead of assigning a training point $\boldsymbol{x}_i$ to one of the disjoint half-spaces (in the case of binary classification), PSVM assigns it based on the proximity to closer of the two parallel planes.

The Proximal SVM is formulated as follows:

$$\text{minimize} \quad Q(\boldsymbol{w}, b, \boldsymbol{\xi}) = \frac{1}{2} \|\boldsymbol{w}\|^2 + \frac{C}{2} \sum_{i=1}^{m} \xi_i^2 + \frac{1}{2} b^2 \tag{2.70}$$

$$\text{subject to} \quad y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) = 1 - \xi_i \qquad i = 1, ..., m. \tag{2.71}$$

Note that, the cost function has additional term related to bias b here. The inequality constraints seen in L2 SVM are replaced by equality constraints here (same as in LS SVM). This modification changes the nature of optimization problem in a similar way as LS SVM does. In other words, PSVM avoids solving the quadratic programming problem, instead, it reduces a set of linear equations which will be derived here.

The primal Lagrangian for PSVM is given as:

$$L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \frac{1}{2} \|\boldsymbol{w}\|^2 + \frac{C}{2} \sum_{i=1}^{m} \xi_i^2 + \frac{1}{2} b^2 - \sum_{i=1}^{m} \alpha_i (\boldsymbol{x}_i \cdot \boldsymbol{w} + b - y_i + \xi_i), \tag{2.72}$$

KKT conditions ensure the optimal solution, so setting the gradients of $L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha})$ with respect to $\boldsymbol{w}, b, \xi_i$ and α_i to zero gives the following:

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i \boldsymbol{x}_i, \tag{2.73}$$

$$b = \sum_{i=1}^{m} \alpha_i, \tag{2.74}$$

$$\alpha_i = C\xi_i \qquad i = 1, ..., m. \tag{2.75}$$

$$\boldsymbol{x}_i \cdot \boldsymbol{w} + b - y_i + \xi_i = 0 \qquad i = 1, ..., m. \tag{2.76}$$

Substituting the values for $\boldsymbol{w}$, b and ξ obtained from the first three equations above,

into the 2.76 gives:

$$\boldsymbol{\Omega\alpha} = \boldsymbol{y}, \tag{2.77}$$

where $\boldsymbol{y} = (y_1, ..., y_m)^T$, $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_m)^T$ and elements of a matrix $\boldsymbol{\Omega}$ are given as,

$$\Omega_{ij} = \boldsymbol{x}_i \boldsymbol{x}_j + 1 + \frac{\delta_{ij}}{C}, \tag{2.78}$$

where δ_{ij} represents a Kronecker's delta function. Hence, the original problem is simply solved by solving a system of linear equations 2.77 for $\boldsymbol{\alpha}$. The size of this system is (m, m).

This has a similar consequence as commented above for LS SVM, which is - this problem setting does not have a sparse solution. In other words, it gives a model in which all the data are support vectors. Needless to say, such model would be not applicable for massive datasets.

2.2.2.4 Geometric L2 SVM

Geometric L2 support vector machines represent a relatively new and different approach to SVM learning problem. They utilize existing geometric algorithms to solve not only separable, but also non-separable classification problems, accurately and efficiently. These algorithms utilize some geometric properties of the maximum margin classifiers in the feature space. Basically, the idea is that SVMs can be equivalently formulated as certain problems from computational geometry. Some of these geometric SVMs use convex hull [11] and minimum enclosing ball approach [1]. The most recent and advanced methods by Strack et al. [12] known as the Sphere Support Vector Machine (SphereSVM) and Minimal Norm Support Vector Machine (MNSVM) combine the two techniques (convex hull and enclosing ball). This novel approach has displayed significant speedup with respect to all the other state-of-the-

art algorithms for handling large datasets.

In order to approach finding a solution to L2 SVM in a geometric way Tsang et al. [1] modified the original L2 SVM optimization function as follows:

$$\text{minimize} \quad Q(\boldsymbol{w}, b, \boldsymbol{\xi}, \rho) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{2}\sum_{i=1}^{m}\xi_i^2 + \frac{1}{2}b^2 - \rho, \tag{2.79}$$

$$\text{subject to} \quad y_i(\varphi(\boldsymbol{x}_i)\cdot\boldsymbol{w} + b) \geq \rho - \xi_i \qquad i = 1, ..., m. \tag{2.80}$$

The optimization function for geometric L2 SVM given here is extended with two terms: the term relating to the bias $\frac{1}{2}b^2$ and the term regarding the parameter ρ. Note that the ρ is also found on the right-hand side of the constraints. Using the Lagrange multipliers method this constrained optimization problem can be transformed into its primal form:

$$L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \rho) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{2}\sum_{i=1}^{m}\xi_i^2 + \frac{1}{2}b^2 - \rho - \sum_{i=1}^{m}\alpha_i\left(y_i(\varphi(\boldsymbol{x}_i)\cdot\boldsymbol{w} + b) - \rho + \xi_i\right), \tag{2.81}$$

where α_i represent the nonnegative Lagrange multipliers. For optimal solution the following KKT conditions must be satisfied,

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \rho)}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{m}\alpha_i y_i \varphi(\boldsymbol{x}_i) = \boldsymbol{0}, \tag{2.82}$$

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \rho)}{\partial b} = b - \sum_{i=1}^{m} y_i\alpha_i = 0, \tag{2.83}$$

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \rho)}{\partial \xi_i} = C\xi_i - \alpha_i = 0, \tag{2.84}$$

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \rho)}{\partial \rho} = 1 - \sum_{i=1}^{m}\alpha_i = 0, \tag{2.85}$$

$$\alpha_i\left(y_i(\varphi(\boldsymbol{x}_i)\cdot\boldsymbol{w} + b) - \rho + \xi_i\right) = 0 \qquad i = 1, ..., m. \tag{2.86}$$

Plugging in the first four equations above into the last one, the parameter ρ can be defined as a function of a dual space variable $\boldsymbol{\alpha}$. More importantly, using equations

for $\boldsymbol{w}$ and b from 2.82 and 2.83 respectively, and the fact that $\sum_{i=1}^{m} \alpha_i = 1$ the primal Lagrangian can be transformed into its dual form,

$$\text{minimize} \quad L_d(\boldsymbol{\alpha}) = \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \left(k(\boldsymbol{x}_i, \boldsymbol{x}_j) + 1 + \frac{\delta_{ij}}{C} \right), \tag{2.87}$$

$$\text{subject to} \quad \sum_{i=1}^{m} \alpha_i = 1, \qquad i = 1, ..., m, \tag{2.88}$$

$$\text{and} \quad \alpha_i \geq 0, \qquad i = 1, ..., m. \tag{2.89}$$

In matrix form this optimization problem can be expressed as follows,

$$\text{minimize} \quad L_d(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha}, \tag{2.90}$$

$$\text{subject to} \quad |\boldsymbol{\alpha}|_1 = 1, \tag{2.91}$$

$$\text{and} \quad \boldsymbol{\alpha} \geq \mathbf{0}, \tag{2.92}$$

where elements of a Hessian matrix $\boldsymbol{H}$ are given as $\boldsymbol{H} = [y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) + 1 + \delta_{ij}/C]_{i,j}$. The importance of the optimization problem defined above is that it is possible to treat it as a geometrical problem of finding a minimal norm (finding a point belonging to a convex hull that is closest to the origin, see Fig. 3) or a minimal enclosing ball (finding a center of a smallest sphere that encloses all the points, see Fig. 4). Relation of these computational geometry problems to SVMs and algorithms for solving them can be found in [12].

Finally, the decision function for geometric SVM is the same as for L1 and L2 support vector machines and is given as

$$d(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b, \tag{2.93}$$

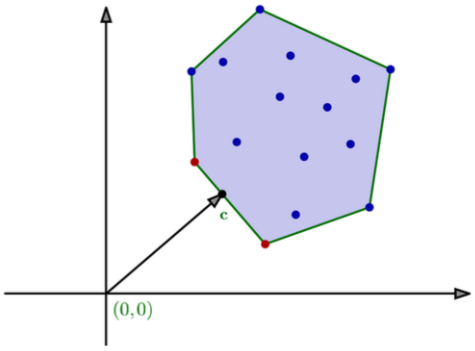


Fig. 3.: Minimal norm problem

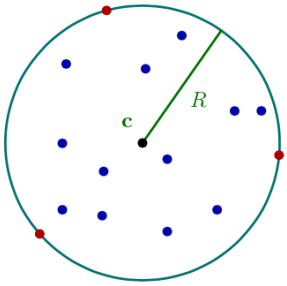


Fig. 4.: Minimal enclosing ball

or, after substituting b from 2.83 it can be rewritten as,

$$d(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + \sum_{i=1}^{m} \alpha_i y_i. \tag{2.94}$$

To summarize, the addition of terms relating to ρ and b in the optimization function enable the transformation of L2 SVM into a corresponding geometric problem and without them this approach would not be possible. This new terms give rise to some important facts. Namely, the constraints in soft-margin L2 SVM $\sum_{i=1}^{m} \alpha_i y_i = 0$ (2.53) have been replaced by $\sum_{i=1}^{m} \alpha_i = 1$ here. Also, the dual form of a geometric SVM has discarded the $-\sum_{i=1}^{m} \alpha_i$ term which can be found in 2.52. Another discrepancy is the addition of element '1' to a kernel matrix in 2.87, which is a consequence of adding a bias term $\frac{1}{2}b^2$ to the optimization function. These differences made this computational geometry approach feasible.

CHAPTER 3

DIRECT L2 SUPPORT VECTOR MACHINE

For the sake of comprehensiveness and in order to have a common model from which all the other L2 SVM approaches mentioned in previous chapter (section 2.2.2) can be derived, we present the most general L2 Support Vector Machine (L2 SVM) learning model as introduced in [20]:

$$\text{minimize} \quad Q(\boldsymbol{w}, b, \boldsymbol{\xi}, \rho) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{2}\sum_{i=1}^{m}\xi_i^2 + \frac{k_b}{2}b^2 - k_\rho\rho, \tag{3.1}$$

$$\text{subject to} \quad y_i(\varphi(\boldsymbol{x}_i)\cdot\boldsymbol{w} + b) \geq \rho - \xi_i \qquad i = 1, ..., m, \tag{3.2}$$

where $\boldsymbol{x}_i$ and y_i are m training input and output pairs, ξ_i are slack variables, $\boldsymbol{w}$ is the weight vector and C is the penalty parameter resolving the trade-off between the maximization of the margin and minimization of the classification error as previously defined. This general DL2 SVM problem setting differs from the classic soft-margin L2 SVM by having two additional terms in the objective function. These two additions are: the term involving the bias b and the term relating to parameter ρ, which are multiplied by scalar values k_b and k_ρ, respectively. Default values for k_b and k_ρ are 1.

DL2 SVM leads to a new approach of finding a solution to L2 support vector machine which enables a variety of algorithms, both iterative and non-iterative, to be used for searching the optimal solution. Detailed steps for to DL2 SVM's underlying problem will be explained in this section. The algorithms for finding a solution to DL2's underlying problem that were discussed in this dissertation will be explained in the succeeding chapter.

L2 SVM models that were mentioned in section *L2 Support Vector Machine* 2.2.2

(all well accepted in practice), can be considered as special cases of DL2 SVM. To be more specific, we give basic similarities and differences between DL2 SVM and all of the previously mentioned L2 SVM models:

- Least-Squares SVM (described in section 2.2.2.2) minimizes the same objective function as DL2 SVM but with both k_b and k_ρ set to zero, meaning, there are no additional terms relating to bias nor ρ in the cost function of LS SVM. Another difference is that LS SVM uses equality constraints (see 2.60) instead of inequality. Furthermore, instead of using the variable parameter ρ on the right-hand side of the constraints, Least-Squares SVM works with a fixed value for it, namely ρ is set to 1. As derivations given in section 2.2.2.2 show, such a setting leads to a system of linear equations. Thus, it is similar to DL2 SVM in a sense that it avoids the QP problem and leads to a set of linear equations, but the major difference is that LS SVM produces dense solutions due to the fact it replaces inequality constraints with equality. In other words, it produces a model where all the data are support vectors. The number of support vectors determines the memory footprint of the learned classifier, as well as the computational cost of using it. In order for SVMs to be practical in large scale applications, it is therefore important to have a small number of support vectors.

- Proximal SVM (described in section 2.2.2.3) poses a similar learning problem to LS and DL2 SVM. PSVM minimizes the same cost function as DL2 but with $k_\rho = 0$ (there is no additional term related to ρ in the cost function). Also, on the right-hand side of the constraints, ρ is always set to 1. Same as LS SVM, PSVM replaces the inequality sign with equality in the equation for constraints. To summarize, PSVM does have an additional term in the cost function relating

to bias, however it uses a fixed value for scalar k_b ($k_b = 1$). This problem setting has similar consequences as in LS SVM, in a sense that one has to solve a system of linear equations which is the main similarity to DL2 SVM. However, same as in LS SVM, this problem setting leads to a dense solution.

- several geometric approaches presented in [1] and [12], dubbed as Core Vector Machine (CVM), Sphere (SphereSVM) and Minimal Norm Support Vector Machines (MNSVM) have almost identical objective function as given in 3.1 subject to 3.2. The only difference is that geometric approaches use fixed values for scalars k_b and k_ρ ($k_b, k_\rho = 1$). These algorithms solve the equivalent geometric problems of finding a minimal enclosing ball or a convex hull in the feature space defined by a kernel function. Because of their iterative nature in obtaining the solution, meaning there is no need to have a whole Hessian matrix stored in a memory at any point, these geometric approaches are well suited for large classification problems. As of now, it seems that Minimal Norm SVM has an edge over the other algorithms in terms of both speed and scalability.

Detailed derivation steps of DL2 SVM model which lead to a new approach of finding the solution to L2 SVM will be discussed bellow. First, we give the summary of a derivation.

DL2 SVM model defined by a QP problem (3.1) and (3.2) is ultimately transformed into the following system of linear equations subject to inequality constraints:

$$\boldsymbol{K}_f \boldsymbol{\beta} = \mathbf{1}, \tag{3.3a}$$

$$\text{subject to} \quad \beta_i \geq 0 \qquad i = 1, ..., m, \tag{3.3b}$$

where $\boldsymbol{\beta}$ represents a vector of nonnegative dual variables β_i (≥ 0) defined as,

$$\beta_i = \frac{\alpha_i}{\rho} \qquad i = 1, ..., m,$$

and $\boldsymbol{\alpha}$ represent the nonnegative Lagrange multipliers. $\boldsymbol{K}_f$ denotes the altered kernel matrix $\boldsymbol{K}$ and is defined as follows:

$$\mathbf{K}_f = \left[\mathbf{K} + \frac{1}{k_b}\mathbf{1}_{m,m} + \text{diag}_{m,m}(\frac{1}{C})\right] \circ \boldsymbol{y}\boldsymbol{y}^{\mathbf{T}}. \tag{3.4}$$

The second term in a summation above denotes a matrix where every element is equal to $1/k_b$. In addition, the original kernel matrix $\boldsymbol{K}$ has an element $1/C$ added to each diagonal term. This element acts as a regularization parameter because adding $1/C$ to a diagonal leads to a better conditioned problem. The size of $\boldsymbol{K}_f$ is the same as for the original kernel matrix (m x m). Another important characteristic of $\mathbf{K}_f$ is that it is a symmetric positive definite (SPD) matrix which makes this problem strictly convex, leading to a unique solution. Elements of the kernel matrix $\boldsymbol{K}_f$ are related to an altered feature space defined by a kernel function:

$$\tilde{k}(\mathbf{x}_i, \mathbf{x}_j) = y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{y_i y_j}{k_b} + \frac{\delta_{ij}}{C} \qquad i, j = 1, ..., m,$$

where δ_{ij} is a Kronecker's delta function in which $\delta_{ij} = 1$ when $i = j$ and $\delta_{ij} = 0$ otherwise while $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ stands for the original kernel function.

Solving the problem given by (3.3a) subject to (3.3b) gives the values for a nonnegative $\boldsymbol{\beta}$ vector which in turn allows the computation of a parameter ρ

$$\rho = \frac{k_\rho}{\sum_{i=1}^{m} \beta_i}. \tag{3.5}$$

After computing the value for ρ, dual variables α_i of the comprehensive DL2 SVM model are calculated as:

$$\alpha_i = \beta_i \cdot \rho \qquad i = 1, ..., m. \tag{3.6}$$

Vector $\boldsymbol{\alpha}$ allows the calculation of the bias b using a formula obtained from Karush-Kuhn-Tucker conditions:

$$b = \frac{1}{k_b} \sum_{i=1}^{m} y_i \alpha_i. \tag{3.7}$$

And so, given some input vector $\mathbf{x}$ the Direct L2 SVM's output $d(\mathbf{x})$ would be calculated as follows,

$$d(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b.$$

Note that, when computing the output, we use the original kernel function $k(\mathbf{x}_i, \mathbf{x})$ as opposed to its altered version $\tilde{k}(\mathbf{x}_i, \mathbf{x})$.

Since any system of linear equations with nonnegativity constraints that has an SPD matrix can be posed as a QP problem, the equations given by (3.3a) and (3.3b) are in fact equivalent to:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} \boldsymbol{\beta}^T \boldsymbol{K}_f \boldsymbol{\beta} - \mathbf{1}^T \boldsymbol{\beta}, \\ \text{subject to} \quad & \beta_i \geq 0 \qquad i = 1, ..., m. \end{aligned}$$

This implies that various existing QP solvers can be used to solve DL2 SVM. However, solving the QP problem is a highly expensive computational task and hence prohibitive for large datasets.

3.1 Detailed derivation of DL2 SVM

What follows is the detailed derivation of a novel Direct L2 SVM model, which leads to a new approach of finding a solution to L2 support vector machine.

DL2 SVM's constrained optimization problem ((3.1) and (3.2)) can be solved using the Lagrange multipliers method. Introducing the nonnegative Lagrange mul-

tipliers (α) we first form the primal Lagrangian:

$$L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \rho, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{C}{2}\sum_{i=1}^{m}\xi_i^2 + \frac{k_b}{2}b^2 - k_\rho\rho \\ - \sum_{i=1}^{m}\alpha_i(y_i(\varphi(\boldsymbol{x}_i)\cdot\boldsymbol{w} + b) - \rho + \xi_i), \quad (3.8)$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_m)^T$ are the nonnegative Lagrange multipliers. The optimal solution to this problem is a saddle point where 3.8 is minimized with respect to $\boldsymbol{w}$, b and $\boldsymbol{\xi}$ and maximized with respect to ρ and $\boldsymbol{\alpha}$.

There are two basic ways one can avoid solving the QP problem of a comprehensive DL2 SVM task posed by (3.1) and (3.2) which we call here: the *classic* and the *less orthodox* approach. Both ways result in the same solution, that is, they lead to the same solution vector $\boldsymbol{\alpha}$. First, we will describe the classic way.

After forming the primal Lagrangian given by (3.8) one can go to a dual problem in a classic way by making derivatives of a primal Lagrangian:

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \rho, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{m}\alpha_i y_i \varphi(\boldsymbol{x}_i) = \boldsymbol{0}, \quad (3.9)$$

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \rho, \boldsymbol{\alpha})}{\partial b} = b - \frac{1}{k_b}\sum_{i=1}^{m} y_i\alpha_i = 0, \quad (3.10)$$

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \rho, \boldsymbol{\alpha})}{\partial \xi_i} = C\xi_i - \alpha_i = 0, \quad (3.11)$$

$$\frac{\partial L_p(\boldsymbol{w}, b, \boldsymbol{\xi}, \rho, \boldsymbol{\alpha})}{\partial \rho} = k_\rho - \sum_{i=1}^{m}\alpha_i = 0, \quad (3.12)$$

(3.9) - (3.12) can be reduced, respectively, to:

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i \varphi(\boldsymbol{x}_i), \tag{3.13}$$

$$b = \frac{1}{k_b} \sum_{i=1}^{m} y_i \alpha_i, \tag{3.14}$$

$$\xi_i = \frac{\alpha_i}{C} \qquad i = 1, ..., m, \tag{3.15}$$

$$\sum_{i=1}^{m} \alpha_i = k_\rho. \tag{3.16}$$

Plugging these expression back into the primal Lagrangian we get the following dual problem:

$$\text{minimize} \quad L_d(\boldsymbol{\alpha}) = \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \left(k(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{1}{k_b} + \frac{\delta_{ij}}{C} \right), \tag{3.17a}$$

$$\text{subject to} \quad \sum_{i=1}^{m} \alpha_i = k_\rho, \qquad i = 1, ..., m, \tag{3.17b}$$

$$\text{and} \quad \alpha_i \geq 0, \qquad i = 1, ..., m. \tag{3.17c}$$

Note that similar derivations which resulted in a less comprehensive dual have been presented in [21], [22] and [12].

Before presenting the faster way to solve (3.17a-c), let's point out a few important characteristics of this QP problem. First, the $(1, m)$ matrix of equality constraints $\mathbf{1}^T$ has a full row rank. Next, the Hessian matrix defined as $\boldsymbol{Z}^T \boldsymbol{K}_f \boldsymbol{Z}$ (where $\boldsymbol{Z}$ is a null-space basis matrix of $\mathbf{1}^T$) is positive definite. Under such properties the unique global solution to (3.17a-c) can be found in a faster way by transforming the QP problem into a system of linear equations [23]. First, we need to define matrix $\boldsymbol{K}_n$:

$$\boldsymbol{K}_n = \begin{bmatrix} \boldsymbol{K}_f & -\mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \tag{3.18}$$

where $\boldsymbol{K}_f$ is given by (3.4). Then, we solve the following system of equations,

$$\boldsymbol{K}_n \boldsymbol{x} = \begin{bmatrix} \mathbf{0} \\ k_\rho \end{bmatrix} \tag{3.19a}$$

$$\text{subject to} \quad x_i \geq 0 \qquad i = 1, ..., m, \tag{3.19b}$$

where $\boldsymbol{x} = [\boldsymbol{\alpha} \quad \lambda]^T$. Note that λ value equals the value of DL2 SVM's parameter ρ. Important thing to mention here is that matrix $\boldsymbol{K}_n$ is not SPD. However, it is nonsingular so the problem posed above in (3.19a) - (3.19b) can, for example, be solved with the Non-Negative Least Squares (NNLS) approach as introduced in [24] or by its various variants. However, NNLS is an active set approach and not suitable for large datasets.

In this dissertation we propose a less orthodox way of transforming the QP problem given by (3.1) and (3.2) into a system of linear equations with a symmetric positive-definite matrix. This, more efficient and suitable, approach proposed here starts with the equation for the primal Lagrangian shown in (3.8). Here, in addition to finding where the primal Lagrangian is stationary (i.e. by taking derivatives of it as given in:(3.9) - (3.12)) we introduce the so-called complementary conditions:

$$\alpha_i \geq 0 \quad \text{and}, \tag{3.20a}$$

$$\alpha_i = 0 \quad \text{if} \quad y_i(\varphi(\boldsymbol{x}_i) \cdot \boldsymbol{w} + b) \geq \rho - \xi_i \qquad i = 1, ..., m. \tag{3.20b}$$

The complementary conditions state that α_i are nonnegative, or equal to zero if the classifier output multiplied by the class label y_i is bigger than $\rho - \xi_i$. Consequently, if $\alpha_i > 0$, the following must hold true: $y_i(\varphi(\boldsymbol{x}_i) \cdot \boldsymbol{w} + b) = \rho - \xi_i$. An alternative formulation, and the one more frequently used in the machine learning community is

this:

$$\alpha_i \left[y_i(\varphi(\boldsymbol{x}_i) \cdot \boldsymbol{w} + b) - \rho + \xi_i \right] = 0 \tag{3.21}$$

$$\alpha_i \geq 0, \qquad i = 1, ..., m, \tag{3.22}$$

Remember that the two inequalities $\alpha_i \geq 0$ and $y_i(\varphi(\boldsymbol{x}_i) \cdot \boldsymbol{w} + b) \geq \rho - \xi_i \quad i = 1, ..., m$ are *complementary*. This means that, for a given data point, at least one constraint is active. In other words, at least one must be an equality. Also, notice that no other variable in the equation for primal Lagrangian (3.8) has any other constraints imposed. The negative values of ξ_i and ρ cannot occur at the optimal solution (see 3.5 and 3.15).

In DL2 SVM, by enforcing the nonnegativity of α_i and by using the equality $y_i(\varphi(\boldsymbol{x}_i) \cdot \boldsymbol{w} + b) = \rho - \xi_i$, the optimal values for α_i will be obtained. This leads to the following system of equations which must be satisfied at the optimal point:

$$y_i \left(\varphi(\boldsymbol{x}_i) \cdot \boldsymbol{w} + b \right) - \rho + \xi_i = 0 \qquad i = 1, ..., m. \tag{3.23}$$

$$\alpha_i \geq 0 \qquad i = 1, ..., m. \tag{3.24}$$

Now, after plugging in the equations (3.13) - (3.16) into (3.23) we get the following system:

$$y_i \left(\varphi(\boldsymbol{x}_i) \cdot \sum_{j=1}^{m} \alpha_j y_j \varphi(\boldsymbol{x}_j) + \frac{1}{k_b} \sum_{j=1}^{m} y_j \alpha_j \right) + \frac{\alpha_i}{C} = \rho \qquad i = 1, ..., m. \tag{3.25}$$

which can be expanded as follows:

$$\begin{aligned} y_i \varphi(\boldsymbol{x}_i)^T \alpha_1 y_1 \varphi(\boldsymbol{x}_1) + y_i \varphi(\boldsymbol{x}_i)^T \alpha_2 y_2 \varphi(\boldsymbol{x}_2) + ... + y_i \varphi(\boldsymbol{x}_i)^T \alpha_m y_m \varphi(\boldsymbol{x}_m) + \\ + y_i \frac{1}{k_b} y_1 \alpha_1 + y_i \frac{1}{k_b} y_2 \alpha_2 + ... + y_i \frac{1}{k_b} y_m \alpha_m + \frac{\alpha_i}{C} = \rho \qquad i = 1, ..., m, \end{aligned}$$

this, in fact, represents a system of linear equations of the form:

$$
\begin{gathered}
y_1y_1k(\boldsymbol{x}_1,\boldsymbol{x}_1)\alpha_1 + y_1y_2k(\boldsymbol{x}_1,,\boldsymbol{x}_2)\alpha_2 + ... + y_1y_mk(\boldsymbol{x}_1,\boldsymbol{x}_m)\alpha_m + y_1y_1\tfrac{1}{k_b}\alpha_1 \\
+y_1y_2\tfrac{1}{k_b}\alpha_2 + ... + y_1y_m\tfrac{1}{k_b}\alpha_m + \tfrac{\alpha_1}{C} = \rho \\
y_2y_1\varphi(\boldsymbol{x}_2,\boldsymbol{x}_1)\alpha_1 + y_2y_2k(\boldsymbol{x}_2,\boldsymbol{x}_2)\alpha_2 + ... + y_2y_mk(\boldsymbol{x}_2,\boldsymbol{x}_m)\alpha_m + y_2y_1\tfrac{1}{k_b}\alpha_1 \\
+y_2y_2\tfrac{1}{k_b}\alpha_2 + ... + y_2y_m\tfrac{1}{k_b}\alpha_m + \tfrac{\alpha_2}{C} = \rho \\
... \\
y_my_1\varphi(\boldsymbol{x}_m,\boldsymbol{x}_1)\alpha_1 + y_my_2k(\boldsymbol{x}_m,\boldsymbol{x}_2)\alpha_2 + ... + y_my_mk(\boldsymbol{x}_m,\boldsymbol{x}_m)\alpha_m + y_my_1\tfrac{1}{k_b}\alpha_1 \\
+y_my_2\tfrac{1}{k_b}\alpha_2 + ... + y_my_m\tfrac{1}{k_b}\alpha_m + \tfrac{\alpha_m}{C} = \rho.
\end{gathered}
$$

Dividing this set of linear equations by a scalar value ρ and introducing new nonnegative dual variables β_i:

$$\beta_i = \frac{\alpha_i}{\rho} \qquad i = 1, ..., m,$$

the original comprehensive DL2 SVM quadratic programming problem is transformed into the following system of equations:

$$\boldsymbol{K}_f\boldsymbol{\beta} = \mathbf{1}, \tag{3.26a}$$

$$\text{subject to} \quad \beta_i \geq 0 \qquad i = 1, ..., m. \tag{3.26b}$$

The matrix $\boldsymbol{K}_f$ is given earlier in 3.4. Computing the values for nonnegative $\boldsymbol{\beta}$ allows the computation of parameter ρ (3.5), and finally, of the dual variables α_i (3.6).

Few remarks are now in order. The steps starting with equation (3.23) and resulting in a model (3.26a) - (3.26b) are equivalent to the procedure from [23] in obtaining (3.19a) - (3.19b) where one transforms all the equality constraints first and the nonnegativities of the variables are accounted for only during solving these two systems of linear equations obtained. Needless to say, they produce the same values for dual variables α_i and parameter ρ, meaning they result in the same SVM model.

Finally, solving both of these systems (3.19a) - (3.19b) and (3.26a) - (3.26b) leads to the sparse SVM model. Our extensive experiments have shown that a slightly simpler model (3.26a) - (3.26b) is faster.

3.2 Variants of DL2 SVM

Basic DL2 SVM defined by 3.1 and 3.2 represents a comprehensive version of this model. Three additional variants of this model can be expressed and derived, all of which are explained in this section.

3.2.1 No parameter ρ in the cost function

As the title suggests one possible variant of the comprehensive DL2 SVM is given as follows:

$$\text{minimize} \quad Q(\boldsymbol{w}, b, \boldsymbol{\xi}, \rho) = \frac{1}{2} \|\boldsymbol{w}\|^2 + \frac{C}{2} \sum_{i=1}^{m} \xi_i^2 + \frac{k_b}{2} b^2, \tag{3.27}$$

$$\text{subject to} \quad y_i(\varphi(\boldsymbol{x}_i) \cdot \boldsymbol{w} + b) \geq 1 - \xi_i \qquad i = 1, ..., m. \tag{3.28}$$

The only difference in respect to the original DL2 SVM model is the absence of parameter ρ in the cost function. This changes the inequality constraints given in 3.2 to the ones given here, where on the right-hand side we use 1 instead of ρ. After similar derivations as in the case of general DL2 SVM, a slightly altered system of linear equations for the model without parameter ρ is obtained:

$$\mathbf{K}_f \boldsymbol{\alpha} = \mathbf{1}, \tag{3.29}$$

$$\text{subject to} \quad \alpha_i \geq 0 \qquad i = 1, ..., m. \tag{3.30}$$

As it can be seen, this model is very similar to the basic DL2 SVM. Namely, by excluding the term ρ in the cost function and setting the value of 1 on the right-hand

side of the constraints, there is no need to introduce dual variables $\boldsymbol{\beta}$ at all. This means that we can directly find the nonnegative Lagrange multipliers $\boldsymbol{\alpha}$. Matrix $\boldsymbol{K}_f$ is the same as in the case of a general DL2 SVM.

It is important to mention the similarity of this model to Proximal SVM which solves quite a similar learning problem with only one difference: the inequality constraints are replaced by the equality in Proximal SVM. Setting this equality has a consequence of finding a dense solution. DL2 SVM model proposed here avoids that problem and leads to a sparse solution.

3.2.2 No bias in the cost function

The second variant of a comprehensive DL2 SVM is the one without the bias term in the cost function:

$$\text{minimize} \quad Q(\boldsymbol{w}, b, \boldsymbol{\xi}, \rho) = \frac{1}{2} \|\boldsymbol{w}\|^2 + \frac{C}{2} \sum_{i=1}^{m} \xi_i^2 - k_\rho \rho, \tag{3.31}$$

$$\text{subject to} \quad y_i(\varphi(\boldsymbol{x}_i) \cdot \boldsymbol{w} + b) \geq 1 - \xi_i \qquad i = 1, ..., m. \tag{3.32}$$

Following the same steps in deriving the solution as before, we arrive at the following problem:

$$\mathbf{K}_{f'} \boldsymbol{\beta} = \mathbf{1}, \tag{3.33}$$

$$\text{subject to} \quad \beta_i \geq 0 \qquad i = 1, ..., m. \tag{3.34}$$

Just as in the original problem setting, this version introduces new dual variables $\boldsymbol{\beta}$ that are used to calculate $\boldsymbol{\alpha}$ using the same formulas as in the original DL2 model, namely 3.5 and 3.6.

However, the matrix $\mathbf{K}_{f'}$ is different from the $\boldsymbol{K}_f$ used in the previous two

variants of the DL2. The matrix $\mathbf{K}_{f'}$ for this variant of DL2 SVM is given as:

$$\mathbf{K}_{f'} = \left[\mathbf{K} + \text{diag}_{m,m}(\frac{1}{C})\right] \circ \boldsymbol{y}\boldsymbol{y}^{\mathbf{T}}. \tag{3.35}$$

The only difference in respect to the original $\mathbf{K}_f$ matrix is the absence of $\frac{1}{k_b}\mathbf{1_{m,m}}$ element in a summation. Just as $\mathbf{K}_f$, matrix $\mathbf{K}_{f'}$ is symmetric positive definite, but due to the absence of the $1/k_b$ term in a summation it is better conditioned than $\mathbf{K}_f$.

3.2.3 No bias b and no ρ in the cost function

Final DL2 SVM problem setting is the one that neither has bias term b nor parameter ρ in the cost function. In fact, this is a very similar problem to the basic L2 SVM as given in section 2.2.2 except that, in this variant of DL2 SVM, we do not use b in the model either. This setting can also be solved as a set of linear equations with the nonnegativity constraints given as:

$$\mathbf{K}_{f'}\boldsymbol{\alpha} = \mathbf{1}, \tag{3.36}$$

$$\text{subject to} \quad \alpha_i \geq 0 \qquad i = 1, ..., m. \tag{3.37}$$

The matrix $\mathbf{K}_{f'}$ for this problem is the same as previously defined in 3.35. Also, similar to the previous variant we can directly solve for α.

Least-Squares SVM mentioned previously has a similar cost function as the the basic L2 SVM (and this variant of DL2 SVM), only difference is that the linear inequality constraints are replaced by the equality constraints in LS SVM. As already mentioned, this small difference in LS SVM leads to a dense solution, which is not the case in any of the DL2 SVM models introduced here.

To generalize, DL2 SVM solves a system of equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ subject to the constraint that the solution vector $\boldsymbol{x}$ has nonnegative elements, $x_i \geq 0, \quad i = 1, ..., m$.

This can be solved in a least squares sense, so that the solution vector $\boldsymbol{x}$ minimizes

$$\|\boldsymbol{A}\boldsymbol{x} - \mathbf{1}\| \tag{3.38}$$

$$\text{subject to} \quad x_i \geq 0 \qquad i = 1, ..., m. \tag{3.39}$$

This represents a well known Non-negative Least Squares (NNLS) problem. In the case of DL2 SVM $\boldsymbol{A}$ is always a symmetric positive-definite matrix. Table 2 gives the summary of all four versions of DL2 SVM models.

Table 2.: Summary of DL2 SVM variants

	DL2 SVM model			
	basic	no ρ in the cost function	no bias in the cost function	no bias and no ρ in the cost function
matrix A=	K_f	K_f	$K_{f'}$	$K_{f'}$
solve for x=	β	α	β	α

Extensive comparison of all four variants of DL2 SVM was given in [25].

3.2.4 Summarization of L2 SVM models

Solution to the comprehensive L2 SVM problem and its variants can be found using three different algorithms (discussed in the next chapter). All three produce the exact, sparse and the same SVM model given as $d(\boldsymbol{x}) = \varphi(\boldsymbol{x}) \cdot \boldsymbol{w} + b$ except for the variant given in section 3.2.3 which doesn't use the bias term b. For this variant, the final model is calculated as $o(\boldsymbol{x}) = \varphi(\boldsymbol{x}) \cdot \boldsymbol{w}$.

Table 3 gives the summary of different representations of the five L2 SVM models. In the second column we give dual QP problems, and in the column 3 we show the transformed duals. The fourth column gives the DL2 SVM's posing of the corre-

sponding problems.

The classic SVM solution approach, and the one typically used in the SVM community, leads to solving a dual QP problem as presented in the second column. Obviously, this is not a convenient training method for large datasets. However, in this setting the $(1,\ m)$ matrix of equality constraints $\mathbf{1}^T$ has a full row rank and the reduced Hessian matrix $\boldsymbol{Z}^T\boldsymbol{K}_f\boldsymbol{Z}$ (where $\boldsymbol{Z}$ is a null-space basis matrix of $\mathbf{1}^T$) is positive definite, meaning a dual QP problem can be transformed into a system of linear equations with nonnegative constraints. The models given in column 3 show the dual QP problems transformed into their corresponding constrained systems of linear equations.

The contribution of this dissertation is the fourth column where the solution can be obtained by directly solving the system of linear equations with an SPD matrix. Column 4 shows the models for all the L2 SVM variants except the classic L2 SVM which can't be transformed into a linear SPD system of equations with nonnegative constraints. DL2 SVM models lend themselves perfectly to iterative solvers, one of which is a highly efficient NN ISDA introduced in the next chapter.

It can be useful to make one historic remark regarding Least Squares and Proximal SVM in solving the classic L2 SVM problem. Inability to transform the classic L2 SVM problem into a system of linear equations with SPD matrix led to proposing LS SVM (by replacing inequality constraints with equality) and introducing Proximal SVM (by adding b^2 term to the cost function and also, by replacing inequality constraints by the equality ones). As shown in the table above, the classic L2 SVM can be transformed into a system of linear equations albeit without SPD matrix. Such system can be solved with the NN LS algorithm introduced in [24].

Hence, as the Table 3 shows, the major original contribution of this dissertation is a replacement of dual QP problem of DL2 SVM and all its variants by the linear sys-

Table 3.: Three different model representations for five L2 SVM variants

Cost Function Variant	QP Dual	Transformed QP Dual Solve:	DL2 SVM Solve:
b^2, ρ	$\min \boldsymbol{L}_d(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{K}_f \boldsymbol{\alpha}$ s.t. $\mathbf{1}^T\boldsymbol{\alpha} = k_\rho$, $\alpha_i \geq 0$	$\begin{bmatrix} \boldsymbol{K}_f & -\mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ k_\rho \end{bmatrix}$ s.t. $\alpha_i \geq 0, \lambda \geq 0$	$\boldsymbol{K}_f\boldsymbol{\beta} = \mathbf{1}$ s.t. $\beta_i \geq 0$
b^2, No ρ	$\min \boldsymbol{L}_d(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{K}_f \boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha}$ s.t. $\alpha_i \geq 0$	$\boldsymbol{K}_f\boldsymbol{\alpha} = \mathbf{1}$ s.t. $\alpha_i \geq 0$	$\boldsymbol{K}_f\boldsymbol{\alpha} = \mathbf{1}$ s.t. $\alpha_i \geq 0$
No b^2, ρ	$\min \boldsymbol{L}_d(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{K}_{f'} \boldsymbol{\alpha}$ s.t. $\mathbf{1}^T\boldsymbol{\alpha} = k_\rho$, $\alpha_i \geq 0$	$\begin{bmatrix} \boldsymbol{K}_{f'} & -\mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ k_\rho \end{bmatrix}$ s.t. $\alpha_i \geq 0, \lambda \geq 0$	$\boldsymbol{K}_{f'}\boldsymbol{\beta} = \mathbf{1}$ s.t. $\beta_i \geq 0$
No b^2, No ρ No b in the model	$\min \boldsymbol{L}_d(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{K}_{f'} \boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha}$ s.t. $\alpha_i \geq 0$	$\boldsymbol{K}_{f'}\boldsymbol{\alpha} = \mathbf{1}$ s.t. $\alpha_i \geq 0$	$\boldsymbol{K}_{f'}\boldsymbol{\alpha} = \mathbf{1}$ s.t. $\alpha_i \geq 0$
No b^2, No ρ L2 SVM, b is in the model	$\min \boldsymbol{L}_d(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{K}_{f'} \boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha}$ s.t. $\boldsymbol{y}^T\boldsymbol{\alpha} = 0$, $\alpha_i \geq 0$	$\begin{bmatrix} \boldsymbol{K}_{f'} & -\boldsymbol{y} \\ \boldsymbol{y}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \gamma \end{bmatrix} = \begin{bmatrix} \mathbf{1} \\ 0 \end{bmatrix}$ s.t. $\alpha_i \geq 0$	No DL2 SVM model

tem of equations with an SPD matrix where the only constraint is the nonnegativity of dual variables α_i (i.e. β_i).

3.3 Geometric insights

Recently, several publications have been pointing out the similarity between computational geometry and the L2 SVM learning task. This research has been driven by a desire to develop learning algorithms that can handle ultra-large datasets and it

succeeded in a sense that geometric approaches are showing better scalability than all the other L2 SVMs as well as the classic L1 SVMs. As mentioned before, geometric approaches related to L2 SVMs are utilizing the fact that finding dual variables is a task equivalent to finding the minimal enclosing/interpolating sphere around the image points in kernel induced feature space.

Before proceeding into the geometric insights of DL2 SVM and other L2 SVM models we will give a brief introduction to kernel induced feature spaces. It is said that every positive semidefinite kernel implicitly defines a feature map $\varphi(\boldsymbol{x}) : \chi \to \Phi$ into some Hilbert space (more on reproducing kernel Hilbert space and positive semidefinite kernels can be found in [26]). SVMs utilize the fact that positive semidefinite kernels have associated feature space for determining the optimal hyperplane in this induced space. In other words, when a linear SVM is applied in that feature space (that is, applied to those transformed points) the solution corresponds to a non-linear function in the original space. In practice, kernels calculate the scalar product of transformed feature vectors $\varphi(\boldsymbol{x}_1)$ and $\varphi(\boldsymbol{x}_2)$ which turns out to be quite a simple operation on input space vectors:

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \varphi(\boldsymbol{x}_1) \cdot \varphi(\boldsymbol{x}_2) \tag{3.40}$$

For example, Gaussian RBF kernel is given as:

$$k\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \varphi(\boldsymbol{x}_i) \cdot \varphi(\boldsymbol{x}_j) = e^{-\gamma \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|^2}. \tag{3.41}$$

It is important to see here that this inner product in feature space is computed as a function k which takes as input two vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in the original space and returns one number. Since the inner product means a projection of $\varphi(\boldsymbol{x}_i)$ on $\varphi(\boldsymbol{x}_j)$, or in simple terms how much overlap those two vectors have, we can think of kernels as similarity functions of two vectors in their feature space. Since we are only

computing the inner product there is no need to perform the feature transformation explicitly. So, SVM uses kernel function which enables it to operate in this high-dimensional (potentially infinite-dimensional) feature space without ever computing the coordinates of the data in that space, instead, they simply compute the inner product between the images of all pairs of data in the feature space. This operation is of course, less computationally expensive than the explicit computation of the coordinates.

The fact that kernels use a scalar product of mapping functions $\varphi(\boldsymbol{x}_i)$ gives rise to some important geometrical properties, see [27] for more details. Namely, when using positive-definite Gaussian kernel all transformed points $\varphi(\boldsymbol{x}_i)$ lie on a hypersphere around the origin with a radius R that can be derived from:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_i) = \varphi(\boldsymbol{x}_i) \cdot \varphi(\boldsymbol{x}_i) = R^2, \tag{3.42}$$

Hence, for a Gaussian kernel it holds that a radius equals 1. Another important property of Gaussian kernels is that the kernel matrix has a full rank, thus all points in a feature space are linearly independent.

The feature space induced by Gaussian kernels is infinite dimensional, however in order to show some geometric properties it is sufficient to look at only m-dimensional subspace which is spanned by m points in feature space. In Fig 5 we show the example of a geometric setup in a feature space for the simple case of three points.

After the mapping, feature points $\varphi(\boldsymbol{x}_1), \varphi(\boldsymbol{x}_2)$ and $\varphi(\boldsymbol{x}_3)$ lie on a 3-dimensional sphere with a center at the origin $\mathbf{0}$ and a radius defined by a kernel, that is to say: $R = \sqrt{k(\boldsymbol{x}_i, \boldsymbol{x}_i)} = 1$.

For the most commonly used coefficient value $k_b = 1$ in our comprehensive DL2 SVM model, the data points will be mapped onto a hypersphere with a radius not 1

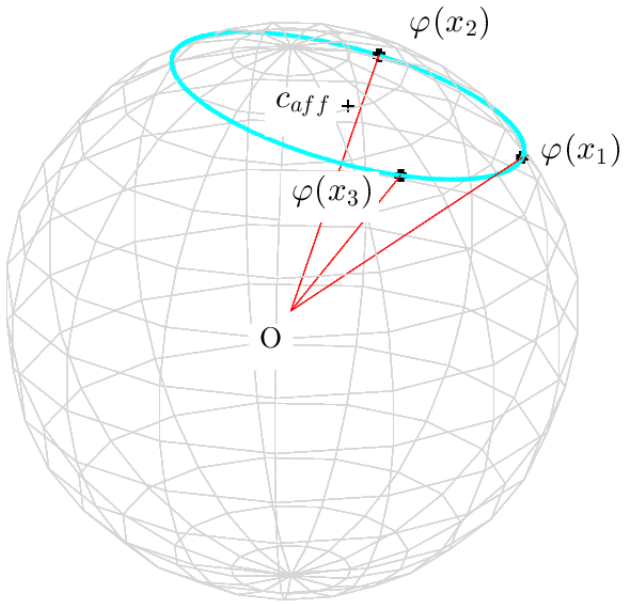


Fig. 5.: Visualization of a feature space with three sample points; c_{aff} represents the center of the (hyper) sphere interpolating $\varphi(\boldsymbol{x}_i)$

but:

$$R = \sqrt{k(\boldsymbol{x}_i, \boldsymbol{x}_i)} = \sqrt{1 + \frac{1}{k_b} + \frac{1}{C}} = \sqrt{2 + \frac{1}{C}}. \tag{3.43}$$

Note another highly important feature of a kernel induced space. Suppose that $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m$ are distinct points in the original space, therefore, their transformed feature vectors are linearly independent, and since any m independent points lie on a $m-1$ dimensional hypersphere, the feature vectors also lie on $m-1$ dimensional hypersphere. In example given in Fig 5, three feature points thus lie on a circle (marked with cyan color) with a center at c_{aff} (subscript aff stands for *affine* which will be explained bellow).

Some basic findings about the inequalities and/or similarities of various approaches for solving L2 SVM problem will be pointed out now.

Proposition 1: Learning algorithms presented in several geometric approaches

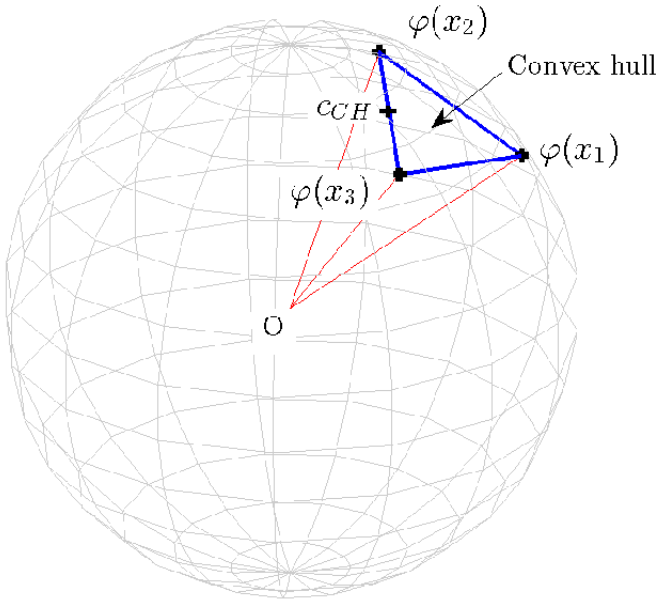


Fig. 6.: Visualization of a convex hull in feature space spanned by $\varphi(\boldsymbol{x}_i)$ points

dubbed as Core (CVM), Sphere (SphereSVM) and Minimal Norm (MN) SVM have the same solution as comprehensive Direct L2 SVM[1].

Proof: The cost function given in 3.1 subject to 3.2 is equal in all four models. Geometric algorithms that solve this problem are iterative by nature and their convergence to the unique solution has been proven. Matrix $\boldsymbol{K}_f$ (see 3.4) of our DL2 SVM model is a positive definite matrix, meaning, it guarantees the same unique solution, the only difference is that a solution is found by solving the NNLS problem. Since the solution to DL2 SVM learning problem, whether it is solved as NNLS or a corresponding geometric problem is the same, we can geometrically represent it as one point in our simple example: point c_{CH} in Fig 6 (CH stands for convex hull).

[1]under the condition that coefficients $k_b, k_\rho = 1$

The point c_{CH} is a linear combination of feature points:

$$c_{CH} = \sum_{i=1}^{3} \alpha_i \varphi(\boldsymbol{x}_i), \tag{3.44}$$

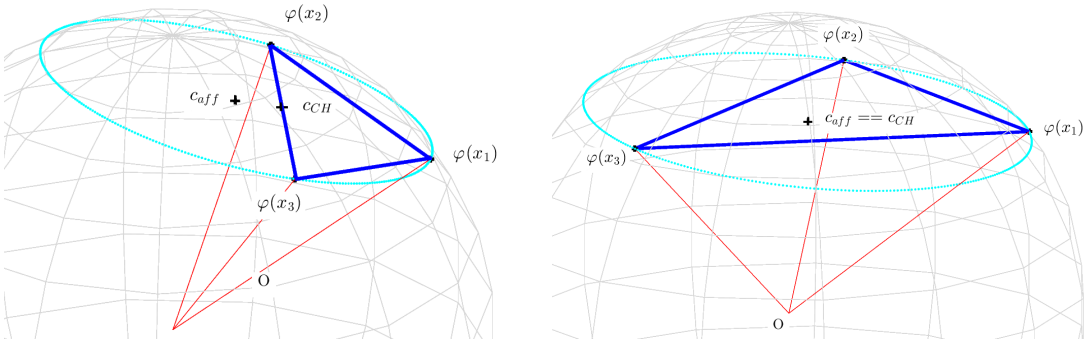
further more, constraints on coefficients α (namely, $\sum_{i=1}^{3} \alpha_i = 1$) restrain this point to belong to a convex hull spanned by $\varphi(\boldsymbol{x}_i)$. Therefore, in the example above, $\alpha_1 = 0, \alpha_2 = 0.5$ and $\alpha_3 = 0.5$. In our comprehensive DL2 SVM model the sum of α values must be equal to k_ρ.

Another interesting geometric property of this problem is that for normalized kernels the problem of finding a point c_{CH} is in fact equivalent to a problem of finding a center of minimal enclosing hypersphere for $\varphi(\boldsymbol{x}_i)$ that is closest to the origin. The center of minimal enclosing hypersphere is thus the same as the point c_{CH}.

Proposition 2: Least-Squares SVM finds the affine solution to the L2 SVM learning task, in other words, it finds the center c_{AFF} of the *interpolating* hypersphere for $\varphi(\boldsymbol{x}_i)$ points.

Proof: LS SVM solves the system of linear equations (see 2.68) which is solved by a regular matrix inversion, always resulting in a dense (here called affine) solution. This solution is thus the same as the center c_{AFF} which interpolates all the $\varphi(\boldsymbol{x}_i)$ points, as depicted in Fig 5.

As already pointed out, Least-Squares SVM replaces the inequality constraints with the equality constraints. Important consequence of that setting is that it allows dual variables $\boldsymbol{\alpha}$ to be negative. Consequently, the linear combination of feature points $\varphi(\boldsymbol{x}_i)$ can fall outside of the convex hull, which is the situation shown in Fig. 7a. So LS SVM arrives at the affine solution even if $\varphi(\boldsymbol{x}_i)$ points do not enclose the affine center (again, see Fig. 7a). In addition, LS SVM has no bias term b in the cost function, as a result, b can not be eliminated from the learning phase of LS SVM.



(a) c_{aff} and c_{CH} do not match (b) c_{aff} and c_{CH} do match

Fig. 7.: Affine and convex hull solutions

To sum up, LS SVM's system of linear equations must be solved by a regular matrix inversion, always resulting in a dense (here called affine) solution for the interpolating hypersphere centered at c_{AFF}. In DL2 SVM problem formulation, the use of bias term b in the cost function and keeping 3.2 as a set of inequality equations enables both, the elimination of b during the learning phase and a formulation of the novel Non-Negative Least-Squares problem that enables finding the sparse solution.

Remark: For very small values of coefficient k_b (say, whenever $k_b < 1e-5$) dual variables $\boldsymbol{\beta}$ in DL2 SVM and $\boldsymbol{\alpha}$ in L2 and LS SVM are numerically equal as long as the feature points $\varphi(\boldsymbol{x}_i)$ surround the center of the affine space c_{AFF}. This is the case when the convex hull solution c_{CH} coincides with the affine solution c_{AFF}, as shown in example in Fig. 7b. Stating it differently, DL2, L2 and LS SVM solutions are equal as long as all the data are support vectors. The only difference is that, in the LS SVM model, dual variables α_i corresponding to negative class have a negative sign. Remember that the decision function $d(\boldsymbol{x})$ for DL2 and L2 SVMs is obtained by multiplying dual variables by their class label y_i, hence, this sign difference is compensated. Also note that, the 'true' dual variables of DL2 SVM $\alpha_{i(DL2)}$ are scaled by parameter ρ (they are calculated as $\alpha_{i(DL2)} = \beta_i \cdot \rho$). In other words, when all

the data are support vectors and the value of k_b is small ($< 1e-5$), the connections between dual variables in L2, LS and DL2 SVMs are related via parameter ρ as follows,

$$\alpha_{i(L2,LS)} = \frac{\alpha_{i(DL2)}}{\rho}. \tag{3.45}$$

When using a default value for $k_b = 1$, $\alpha_{i(DL2)}$ values are no longer related to the solutions $\alpha_{i(L2,LS)}$ in a given way. This is a consequence of the DL2 problem formulation, specifically when k_b is bigger, DL2 SVM 3.1 minimizes bias term b as well, while L2 and LS SVM minimize the norm of the weights $\boldsymbol{w}$ only.

Remark: For small values of Gaussian RBF kernel's variance (i.e., its shape parameter σ) and penalty parameter C, the original points are mapped onto the affine space enclosing the center of interpolating hypersphere c_{AFF}, as depicted in Fig. 7b, and therefore solutions for DL2 SVM ($\beta_i's$), LS and standard L2 SVM ($\alpha_{i(L2,LS)}$) are numerically equal except for the negative sign in α_{iLS} as commented in the previous paragraph. As both, the shape σ and penalty C parameter increase in value, the points converge to one side of the affine space and cease to enclose its center (see Fig. 8). In this scenario, the center c_{AFF} moves closer towards the origin $\boldsymbol{O}$ but it never reaches it. This leads to the sparse solution for both DL2 and L2 SVM, i.e., to the c_{CH} solution and the 3.45 still holds for these two models. However, LS SVM in this case still finds the affine center c_{AFF} as the non-sparse solution for LS SVM system of equations.

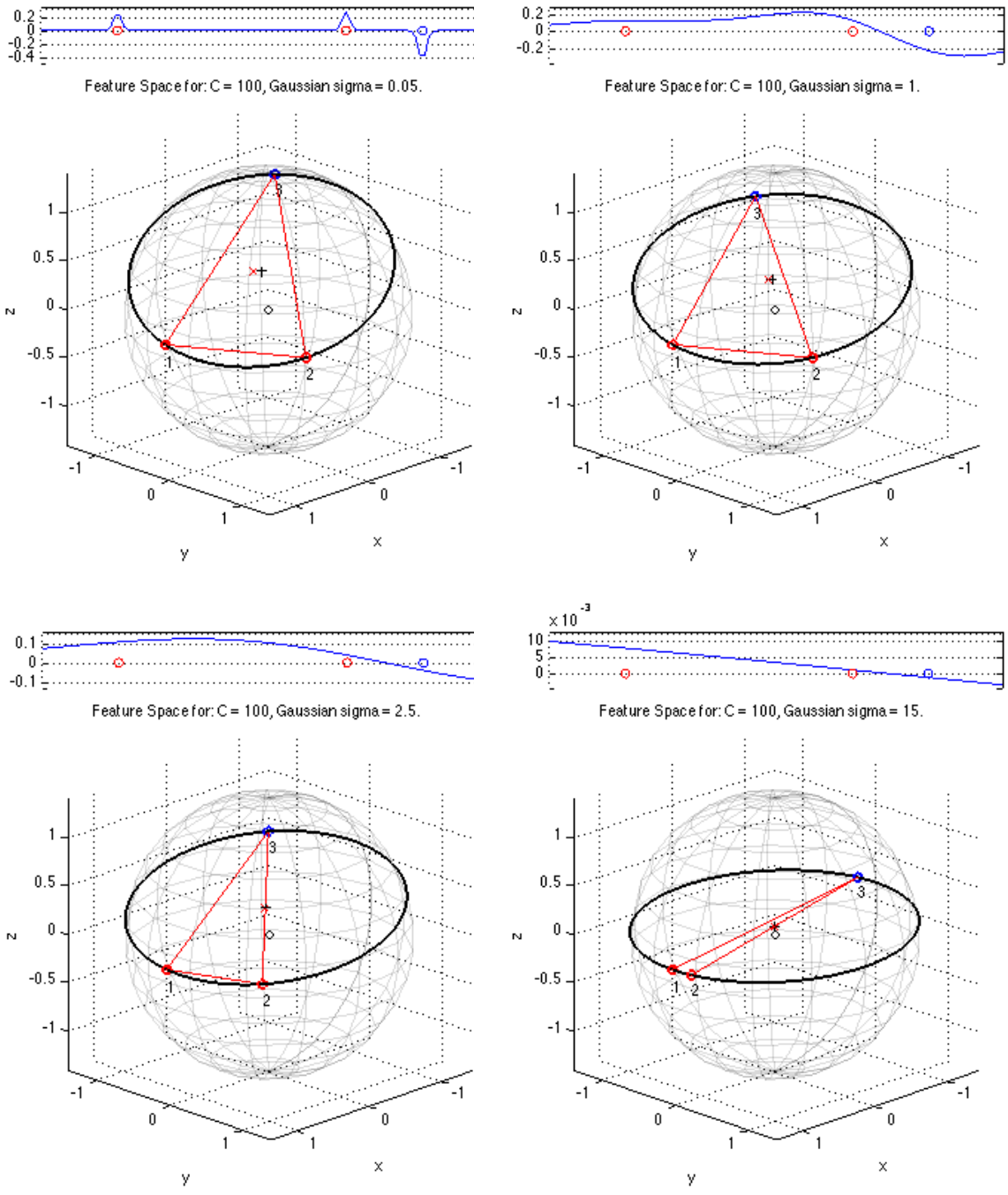


Fig. 8.: Feature space for a fixed value of C and varoius σ

CHAPTER 4

ALGORITHMS FOR SOLVING DIRECT L2 SUPPORT VECTOR MACHINE

In this chapter three different algorithms for finding a solution to DL2 SVM's underlying problem will be explained. Before we start describing the three different solvers, note that all four variants of DL2 SVM essentially solve a similar problem. Namely, both matrices, $\mathbf{K}_f$ and $\mathbf{K}_{f'}$, introduced in chapter 3 are square, symmetric and positive definite, and solution vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ have the same nonnegativity constraints. Therefore, all the algorithms presented bellow are readily applicable to all four DL2 SVM models. Refer to the table 2 to see differences in these four models.

In the following sections in order to generalize, we will refer to finding a solution to the following system of linear equations:

$$\boldsymbol{A}\boldsymbol{x} = \mathbf{1}, \tag{4.1}$$

$$\text{subject to} \quad x_i \geq 0 \qquad i = 1, ..., m, \tag{4.2}$$

Three different algorithms for solving DL2's problem that have been compared in this dissertation are:

- a variant of Lawson and Hanson algorithm [24] which uses a Cholesky decomposition,
- Conjugate Gradient method with nonnegativity constraints by Hestenes [28], and
- a novel algorithm dubbed as Non-Negative Iterative Single Data Algorithm (NN

ISDA) first introduced in [29]

The performances of the first two algorithms were presented in [30]. In [29] we show a comparison of all three algorithms for finding a solution to DL2 SVM.

4.1 Lawson and Hanson solution

The design and implementation of least-squares algorithms has been the subject of considerable work of Lawson and Hanson [24]. Their book contains the first widely used algorithm for solving NNLS problems. The algorithm belongs to a group of so called "active set methods". An active set algorithm is based on the fact that if the true active set is known, the solution to the least squares problem will simply be the unconstrained least squares solution. Or in different words: if the active set is known, the solution to the NNLS problem is obtained by treating the active constraints as equality constraints rather than inequality constraints, [31].

As already pointed out, constrained system of linear equations 4.1-4.2 posed by DL2 SVM can be solved in a least-squares sense, thus the problem becomes of the following form:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad f(\boldsymbol{x}) = \frac{1}{2} \left\| \mathbf{A}\boldsymbol{x} - \mathbf{1} \right\|^2, \tag{4.3}$$

$$\text{subject to} \quad x_i \geq 0 \qquad i = 1, ..., m. \tag{4.4}$$

Lawson-Hanson NNLS algorithm (Chapter 23 in [24]) finds a solution vector $\boldsymbol{x}$ that satisfies the given conditions, in an iterative manner. Meaning, the algorithm updates the current solution with the use of a previously calculated one. This sequence simply repeats itself with the addition of one or more support vectors in each iteration until the solution satisfies a well-defined termination condition. Lawson-Hanson NNLS algorithm is an active-set algorithm that has been proven to be optimal according to

Karush-Kuhn-Tucker (KKT) conditions.

As mentioned before, matrix $\mathbf{A}$ of our problem is a symmetric positive definite, which allowed us to increase the efficiency and numerical stability of Lawson-Hanson algorithm by using a Cholesky factorization [32] with update. Cholesky factorization with update is used to solve a subproblem in each iteration of the algorithm.

The pseudocode for the algorithm is given in Alg. 1.

Algorithm 1: NNLS Algorithm with Cholesky Decomposition

Result: $\boldsymbol{x}$ that minimizes 4.3 subject to 4.4

Initialization: $\boldsymbol{x} \leftarrow \mathbf{0}$, $\boldsymbol{w} \leftarrow \mathbf{1}$, $Z \leftarrow \{1, 2, ..., m\}$, $P \leftarrow NULL$;

while $Z \neq NULL \quad or \quad \exists i \in Z : w_i \geq 0$ **do**

  $u \leftarrow \underset{i \in Z}{\operatorname{argmax}}\ w_i$ % FIND THE MAXIMUM VIOLATOR;

  $Z \leftarrow Z - \{u\}\,, \quad P \leftarrow P + \{u\}$;

  $\boldsymbol{A}_P \boldsymbol{x'} \cong \mathbf{1}$ % USE CHOLESKY DECOMPOSITION TO FIND THE NEW ACTIVE-SET SOLUTION $\boldsymbol{x'}$;

  while $x'_i \not> 0, \forall i \in P$ **do**

    % SOLUTION DOES NOT SATISFY NN CONSTRAINTS;

    $q \leftarrow \underset{j \in P}{argmin} \frac{x_j}{x_j - x'_j}$;

    $\alpha \leftarrow \frac{x_q}{x_q - x'_q}$;

    $\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha(\boldsymbol{x'} - \boldsymbol{x})$ % ADJUST THE SOLUTION VECTOR;

    $Z \leftarrow Z + \{\forall j \in P : x_j = 0\}$;

    $P \leftarrow P - \{\forall j \in P : x_j = 0\}$;

    $\boldsymbol{A}_P \boldsymbol{x'} \cong \mathbf{1}$;

  $\boldsymbol{x} := \boldsymbol{x'}$;

  $\boldsymbol{w} = \mathbf{1} - \boldsymbol{A}\boldsymbol{x}$;

Upon termination, elements of a dual vector $\boldsymbol{w}$, which is introduced in the initialization step, will satisfy:

$$w_i = 0, \quad \forall i \in P \tag{4.5}$$

$$w_i \leq 0, \quad \forall i \in Z, \tag{4.6}$$

where sets P and Z are index sets introduced in the algorithm and are being modified throughout the course of execution. Sets P and Z represent *active* and *non-active* sets, respectively. In SVM terminology set P will hold indexes of all support-vectors and set Z indexes of all non-support-vectors.

Initialization: The algorithm starts with the feasible starting point, i.e., the solution vector $\boldsymbol{x}$ set to $\mathbf{0}$. Index sets Z and P are initialized as well. Variables indexed in set Z have values set to zero, while those indexed in P can take values different from zero, however, if a value indexed in P takes on a non-positive value, the algorithm will either move the variable to a positive range or set it to zero and move its index to set Z.

Stopping Criteria: Step 2 of the algorithm above represents the starting point of the main loop. The condition for exiting the algorithm, i.e. stopping criteria, is met when either set Z is empty or a dual vector $\boldsymbol{w}$ computed as,

$$\boldsymbol{w} = \mathbf{1} - \boldsymbol{A}\boldsymbol{x}, \tag{4.7}$$

has values less than zero for all indexes contained in set Z.

Finding a violator: Step 3 of the algorithm above represents the search for the maximum violator. The maximum violator is the variable indexed in set Z that has maximum value of a vector $\boldsymbol{w}$ (i.e. max $w_i, \forall i \in Z$). This variable is chosen to be introduced into the solution. Furthermore, the index of max violator is moved from

set Z to set P (step 4). In other words, this variable is moved to the *active set*. In support vector machine terminology this step chooses the maximum violator and adds this data to the set of support vectors.

Solving the subproblem: Step 5 represents the most time-consuming part of the algorithm, and thus, to achieve speed up it is necessary to implement this step efficiently. In this particular step, the new 'tentative' solution vector $\boldsymbol{x'}$ is computed to satisfy

$$\boldsymbol{A}_P \boldsymbol{x'} \cong \mathbf{1}. \tag{4.8}$$

In each iteration this step is executed with slightly altered matrix $\boldsymbol{A}_P$ which contains only those components of matrix $\boldsymbol{A}$ that are indexed in set P., i.e., we let $\boldsymbol{A}_P$ denote the matrix defined by

$$\text{column } i \text{ of } \boldsymbol{A}_P := \begin{cases} \text{column } i \text{ of } \boldsymbol{A} & \textbf{if} \quad i \in P \\ 0 & \textbf{if} \quad i \in Z \end{cases} \tag{4.9}$$

Note that in this step only the i^{th} components of the tentative solution vector $\boldsymbol{x'}$ are computed, where $i \in P$. In the end, the solution vector $\boldsymbol{x'}$ is a vector of length m having zeros for all i indexed in set Z.

Adjustment Step: As Step 6 of the algorithm states, if tentative solution vector satisfies the nonnegativity conditions, that is, if $x'_i > 0, \forall i \in P$ then we set $\boldsymbol{x} := \boldsymbol{x'}$ and repeat the entire process (if stopping criteria is not met).

Otherwise, we adjust the subproblem and find a new tentative solution vector $\boldsymbol{x'}$ that does satisfy the nonnegativity constraints. This is done by 'moving' the vector $\boldsymbol{x}$ so that every value that is indexed in set P becomes positive. This move replaces

the $\boldsymbol{x}$ by

$$\boldsymbol{x} + \alpha(\boldsymbol{x'} - \boldsymbol{x}), \quad 0 < \alpha \leq 1, \tag{4.10}$$

where coefficient α is chosen to be as large as possible while keeping the new $\boldsymbol{x}$ nonnegative. In this step indexes of violators are moved from set P to Z and new subproblem with different matrix $\boldsymbol{A}_P$ needs to be solved inside the while loop (step 13). This loop repeats until all variables indexed in P are positive. In many examples the algorithm rarely enters this while loop, usually the sequence of simply adding the new support vectors repeats itself until the termination is eventually satisfied.

Finiteness of two loops given in Alg. 1 have been proven by Lawson and Hanson, [24]. It is also said that for small test cases the main loop typically required $\sim \frac{1}{2}m$ iterations.

Very important feature of a NNLS problem formulated in DL2 SVM, is that the matrix $\boldsymbol{A}$ is square, symmetric and positive definite which allows efficient computation and guarantees a unique solution. Also, as mentioned in step 5 of Alg. 1, in each iteration a subproblem given as $\boldsymbol{A}_P\boldsymbol{x'} \cong \mathbf{1}$ is computed, where matrix $\boldsymbol{A}_P$ gets augmented by one linearly independent vector (or diminished in case nonnegativity conditions are not satisfied). This particularity enables the use of Cholesky decomposition with an update. Namely, by retaining the previously found decomposition we can simply append one new column/row vector to a triangular decomposition matrix. Thus, a very efficient way for finding a new tentative solution vector $\boldsymbol{x'}$ for the newly altered matrix $\boldsymbol{A}_P$ in step 5 was achieved.

4.2 Conjugate gradient solution

The conjugate gradient method is an iterative method which allows us to approximately solve the system of linear equations where the size of a problem is too

large and therefore slow for a direct method implementation, [28]. General Conjugate Gradient (CG) algorithm (without constraints) can be modified to accommodate the case of nonnegative constraints as explained by Hestenes [28]. Here we give a brief explanation of CG method with nonnegative constraints adjusted to DL2 SVM notation. DL2 SVM's problem as given in 4.1 subject to 4.2 is the problem of finding a minimum point $\boldsymbol{x}^{(*)}$ of $f(\boldsymbol{x})$ such that each component of $\boldsymbol{x}^{(*)}$ has a nonnegative value.

Lets first assume that $\boldsymbol{x}^{(0)}$ is the initial guess for solution vector $\boldsymbol{x}^{(*)}$ and that $\boldsymbol{x}^{(0)} = \mathbf{0}$. Starting with this initial guess the algorithm searches for the solution with the help of a certain metric that tells if the guess is closer to the solution or not. The metric used here is the residual vector $\mathbf{r}$ which becomes smaller as the algorithm gets closer to the unique solution vector $\boldsymbol{x}^{(*)}$. Fig. 9 shows the comparison of a steepest (gradient) descent and CG descent. As one can see, conjugate gradient method converges faster, it only takes at most n steps (assuming no round-off errors) where n is the size of the matrix (here $n = 2$). However, every iteration of gradient descent method is cheaper than that of conjugate gradient's.

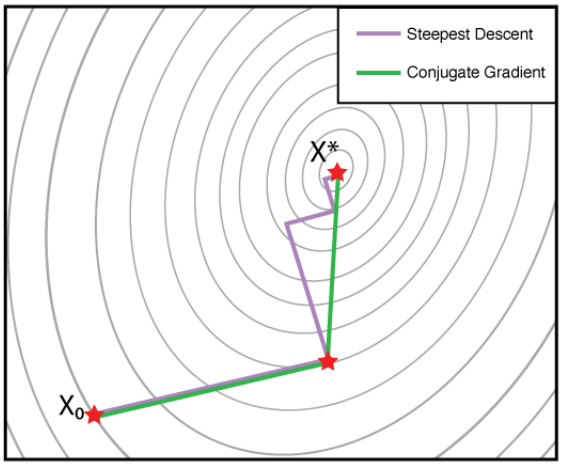


Fig. 9.: Conjugate Gradient vs Steepest Descent method

Before we present the algorithm it is important to introduce the set I and the residual vector $\mathbf{r}$. Let I be the set of all indexes $i \leq m$ such that $x_i^{(k)} = 0$. Then, at a point $\boldsymbol{x}^{(k)}$ residual vector can be calculated as

$$\mathbf{r}^{(k)} = -f'(\boldsymbol{x}^{(k)}) = \mathbf{1} - \mathbf{A}\boldsymbol{x}^{(k)}. \tag{4.11}$$

As one can see, the residual vector $\boldsymbol{r}$ is in fact the negative gradient of $f(\boldsymbol{x})$ at point $\boldsymbol{x}^{(k)}$. The algorithm can be explained through the following steps:

Step 1: Select an initial point $\boldsymbol{x}^{(*)} = \boldsymbol{x}^{(0)}$ that satisfies the nonnegativity constraints, e.g. $\boldsymbol{x}^{(0)} = 0$. The residual vector $\mathbf{r}^{(*)} = \mathbf{r}^{(0)}$ used as the first search direction is calculated as $\mathbf{r}^{(0)} = \mathbf{1} - \mathbf{A}\boldsymbol{x}^{(0)}$.

Step 2: Let I be the set of all indexes $i \leq m$ such that

$$x_i^{(*)} = 0, \quad r_i^{(*)} \leq 0 \quad \forall i \in I. \tag{4.12}$$

If residual value $r_i^{(*)} = 0, \forall i \notin I$ (or equivalently $\left|r_i^{(*)}\right| < \tau$, where τ is some small stopping criteria) then $\boldsymbol{x}^{(*)}$ is the solution of the optimization problem and the algorithm terminates.

Step 3: Set the conjugate direction vector $\mathbf{p}^{(0)} = \bar{\mathbf{r}}^{(0)}$, where $\bar{\mathbf{r}}^{(0)}$ is the vector having

$$\bar{r}_i^{(0)} = \begin{cases} 0 & : i \in I \\ r_i^{(*)} & : otherwise \end{cases} \tag{4.13}$$

Step 4: Start with $k = 0$ and perform the standard CG step by computing:

$$\mathbf{s}^{(k)} = \mathbf{A} \cdot \mathbf{p}^{(k)}, \tag{4.14}$$

$$a^{(k)} = \frac{\mathbf{p}^{(k)T} \cdot \mathbf{r}^{(k)}}{\mathbf{p}^{(k)T} \cdot \mathbf{s}^{(k)}}. \tag{4.15}$$

where scalar value $a^{(k)}$ is used to adjust the solution and residual vector as follows:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + a^{(k)} \cdot \mathbf{p}^{k}, \tag{4.16}$$

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - a^{(k)} \cdot \mathbf{s}^{(k)}. \tag{4.17}$$

Step 5: If $\boldsymbol{x}^{(k+1)}$ lies outside of the feasible region, i.e., some $x_i^{(k+1)}$ violate the nonnegativity constraint then go to Step 6 of the algorithm.

Otherwise, if residual value $r_i^{(k+1)} = 0, \forall i \notin I$ (or equivalently $\left|r_i^{(k+1)}\right| < \tau$), reset $\boldsymbol{x}^{(*)}$ and $\mathbf{r}^{(*)}$ as follows:

$$\boldsymbol{x}^{(*)} = \boldsymbol{x}^{(k+1)}, \tag{4.18}$$

$$\mathbf{r}^{(*)} = \mathbf{r}^{(k+1)} = \mathbf{1} - \mathbf{A}\boldsymbol{x}^{(k+1)}, \tag{4.19}$$

set $k = k + 1$ and go to Step 2.

Else, set the new residual $\bar{\mathbf{r}}^{(k+1)}$ and update the conjugate direction vector $\mathbf{p}^{(k+1)}$ as described bellow:

$$\bar{r}_i^{(k+1)} = \begin{cases} 0 & : i \in I \\ r_i^{(k+1)} & : otherwise \end{cases} \tag{4.20}$$

$$\mathbf{p}^{(k+1)} = \bar{\mathbf{r}}^{(k+1)} - \frac{\mathbf{s}^{(kT)} \cdot \bar{\mathbf{r}}^{(\mathbf{k+1})}}{\mathbf{p}^{(kT)} \cdot \mathbf{s}^{(k)}} \cdot \mathbf{p}^{(k)}. \tag{4.21}$$

Set $k = k + 1$ and go to Step 4.

Step 6: Let J be the set of indexes j such that $x_j^{(k+1)} < 0$ and define some scalar value $\bar{a}^{(k)}$ to be the smallest of the ratios:

$$\bar{a}^{(k)} = min\left(-\frac{x_j^{(k)}}{p_j^{(k)}}\right), \forall j \in J. \tag{4.22}$$

Once we have the value of $\bar{a}^{(k)}$ we can reset the solution and residual vector as follows:

$$\boldsymbol{x}^{(*)} = \boldsymbol{x}^{(k)} + \bar{a}^{(k)} \cdot \mathbf{p}^{(k)}, \tag{4.23}$$

$$\mathbf{r}^{(*)} = \mathbf{r}^{(k)} - \bar{a}^{(k)} \cdot \mathbf{s}^{(k)}. \tag{4.24}$$

Redefine set I to be the set of all indexes $i \leq m$ such that $x_i^{(*)} = 0$. If residual vector $r_i^{(*)} = 0, \forall i \notin I$ go to Step 2, else go to Step 3.

The algorithm presented above finds the solution vector in a finite number of steps. It starts with a point $\boldsymbol{x}^{(0)}$ that lies in a feasible region (nonnegative). If $\boldsymbol{x}^{(0)}$ is not a minimum point of f in set S which represents a set of points $\boldsymbol{x}$ whose components are nonnegative, the algorithm locates a pseudo-minimum point $\boldsymbol{x}^{(1)}$ such that $f(\boldsymbol{x}^{(1)}) < f(\boldsymbol{x}^{(0)})$. If $\boldsymbol{x}^{(1)} \neq \boldsymbol{x}^{(*)}$, we locate a new pseudo-minimum point $\boldsymbol{x}^{(2)}$ that minimizes the function on feasible area, such that $f(\boldsymbol{x}^{(2)}) < f(\boldsymbol{x}^{(1)})$. If $\boldsymbol{x}^{(2)} \neq \boldsymbol{x}^{(*)}$ the process is repeated again until the minimum point has been found. Since there are only a finite number of pseudo-minimum points of F on S, this procedure will eventually terminate when minimum point $\boldsymbol{x}^{(*)}$ has been found.

Conjugate gradient method, in the absence of round-off errors, produces the exact solution after a finite number of iterations, which is not larger than the size of the matrix. However, the conjugate method is unstable with even small errors, so the exact solution is never really obtained. The error developed in calculating the direction can be detrimental to the convergence. To overcome this drawback, Fletcher and Reeves [33] suggested to revert to the direction of steepest descent after every n or (n + 1) iterations. To complicate things, accumulated roundoff error in the recursive formulation of the residual 4.17 may yield a false zero residual or a value that is within a predefined stopping criterion. This problem could be resolved by restarting with equation 4.11.

To summarize, the CG method monotonically improves the approximation vector until the exact solution is reached within some tolerance. The speed of improvement depends on the condition number of matrix $\mathbf{A}$, larger the number slower the convergence.

4.3 Non-Negative Iterative Single Data Algorithm solution

In this section we introduce a novel Non-Negative Iterative Single Data Algorithm (NN ISDA) [29] that finds a solution to DL2 SVM's underlying problem introduced in chapter 3. NN ISDA is a coordinate-descent algorithm that iteratively improves the tentative solution vector until it converges to the optimal one. It is a variant of Iterative Single Data Algorithm (ISDA) which was originally introduced in [34] as a solver for L1 SVM. Here, by combining ISDA from [35, 34] along with some insights from [36], a new, simple and efficient NN ISDA algorithm will be derived.

In [35] and [34] it has been shown that L1 SVM learning problem, without the box constraints coincides with the Gauss-Seidel method which is known to converge if the matrix $\boldsymbol{A}$ is positive definite. Likewise, without the positivity constraint, NN ISDA coincides with the Gauss-Seidel method, and converges to a unique solution if $\boldsymbol{A}$ is positive definite. In the case of DL2 SVM, matrix $\boldsymbol{A}$ is always symmetric, positive definite, therefore, the convergence of NN ISDA is guaranteed.

NN ISDA and ISDA are iterative algorithms that find a solution vector by iteratively optimizing the cost function with respect to a single point at a time (which corresponds to solving a system of linear equations with Gauss-Seidel method). The use of a *single data* in dubbing the algorithm is aimed at stressing the difference in respect to the popular Sequential Minimal Optimization (SMO) algorithm which optimizes two data points at a time.

DL2 SVM problem does not have box constraints on a solution vector $\boldsymbol{x}$, it only

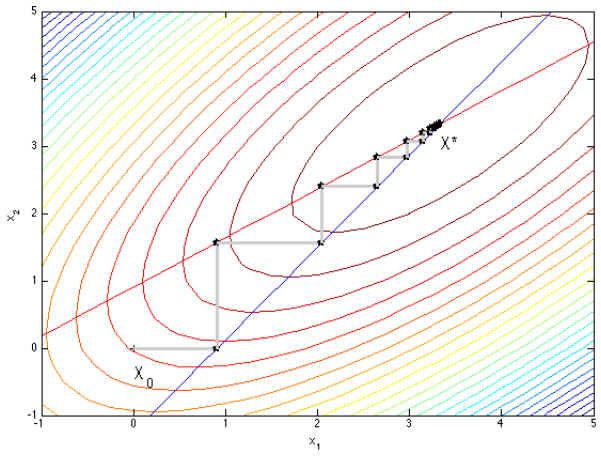


Fig. 10.: The method of NN ISDA. The gray lines represent steps toward convergence for a simple problem defined by 4.25 and 4.26.

has nonnegativity constraints, which makes the modification of ISDA to the case of DL2 SVM fairly easy.

Before we go into the algorithm, to help visualize the solution steps lets take a look at an example of a 2D problem, let's say that matrix $\boldsymbol{A}$ is given as:

$$\begin{bmatrix} 1.1 & -0.8 \\ -0.8 & 1.1 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{4.25}$$

$$\text{subject to} \quad x_1 \geq 0, x_2 \geq 0 \tag{4.26}$$

Fig. 10 shows how the NN ISDA finds a solution to this problem. It starts from a point $(0, 0)$ and searches for a solution by moving in the direction of a steepest descent and thus iteratively updates both gradient and solution vector. In this simple 2D example we can observe alternations in two coordinates being updated. The affine solution to this problem is given as an intersection of two straights lines (blue and red line in Fig. 10) given as $1.1x_1 - 0.8x_2 - 1 = 0$ and $-0.8x_1 + 1.1x_2 - 1 = 0$. This is the solution that Least-Squares SVM finds.

The pseudo-code of the Non-Negative Iterative Single Data Algorithm (NN ISDA) for finding a solution to DL2 SVM's underlying problem is given in Alg. 2.

Algorithm 2: Non-Negative Iterative Single Data Algorithm

Result: $\boldsymbol{\beta}$ that solves $\boldsymbol{K}_f\boldsymbol{\beta} = \mathbf{1}$ subject to $\beta_i \geq 0, \quad i = 1, ..., m$

Initialization: $\boldsymbol{\beta} \leftarrow \mathbf{0}$, $\boldsymbol{g} \leftarrow -\mathbf{1}$, $Z \leftarrow \{1, ..., m\}$;

while $abs(g_i) \geq$ *stopping criterion* **do**

  $i \leftarrow \underset{i \in Z}{\operatorname{argmax}} \frac{|g_i|}{\sqrt{\boldsymbol{K}_f(i,i)}}$ % FIND THE MAXIMUM VIOLATOR;

  if $\beta_i == 0$ *and* $g_i \geq 0$ **then**

    $Z \leftarrow Z - \{i\}$;

  else

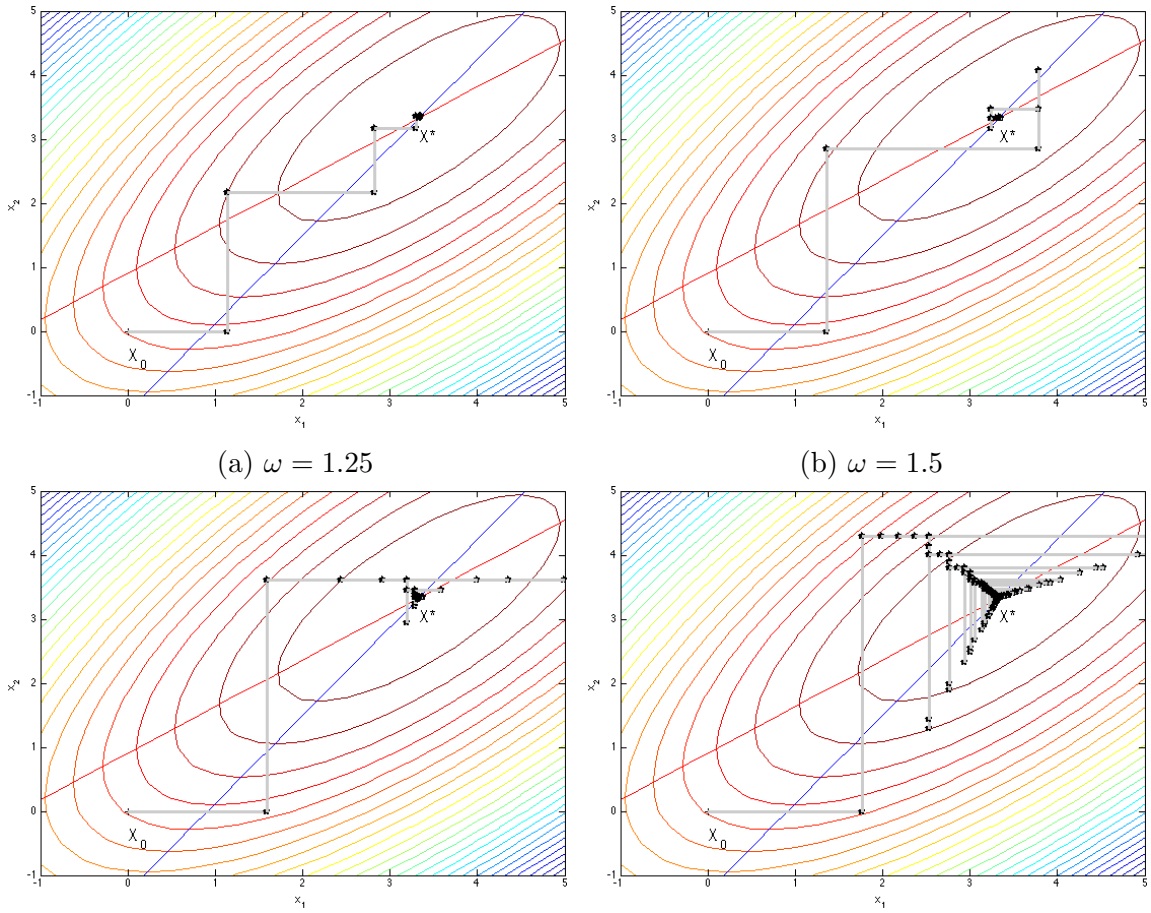
    $\beta_i^{(k+1)} = max\left(\beta_i^{(k)} - \omega\frac{g_i^k}{\boldsymbol{K}_f(i,i)}, 0\right)$ % UPDATE THE SOLUTION VECTOR;

    $\boldsymbol{g}^{k+1} = \boldsymbol{g}^k + \boldsymbol{k}_{fi}\left(\beta_i^{k+1} - \beta_i^k\right)$ % UPDATE THE GRADIENT VECTOR;

Following Huang et al. [34], the iterative update of components β_i of vector $\boldsymbol{\beta}$ should be done in the following way

$$\beta_i^{(k+1)} = max\left(\beta_i^{(k)} - \omega\frac{g_i^k}{K_f(i,i)}, 0\right), \tag{4.27}$$

where $\beta_i^{(k+1)}$ and $\beta_i^{(k)}$ stand for the i-th dual variable β_i at the iteration steps $k+1$ and k, respectively. g_i^k is the gradient in the i-th direction at the step k. $K_f(i,i)$ is a diagonal element of a positive definite matrix $\mathbf{K}_f$, and ω represents the so-called *over-relaxation* parameter. When $\omega = 1$, we can find the optimal β_i value for which a gradient in i-th direction equals 0 by dividing the gradient g_i^k with $K_f(i,i)$. As long as $\omega < 2$ a convergence of the iterative Gauss-Seidel algorithm for positive definite matrix $\mathbf{K}_f$ is guaranteed. The 'proper' value of the over-relaxation parameter ω can speed up the training time up to the order of magnitude (see example in Fig. 11), but



(a) $\omega = 1.25$ (b) $\omega = 1.5$

(c) $\omega = 1.75$ (d) $\omega = 1.95$

Fig. 11.: The convergence rate of NN ISDA with respect to different values of ω. The gray lines represent the steps to the solution given by x^*.

such a 'proper' value of ω is not known in advance. Fortunately, for the DL2 SVM, ω can quite satisfactorily be estimated as the function of penalty parameter C. The basic strategy in estimating a proper value for ω is to increase it with an increase of a condition number of the matrix $\mathbf{K}_f$. The later will happen with the increase of a penalty parameter C.

In order to create an efficient algorithm for solving a linear system by NN ISDA method we followed instructions from Fadeev and Fadeeva [36] which point out two

important things *" ... it is necessary at each step to compute all components of the residual vector, but then to determine the leading index for the following step ... "*. As for the first part, note that the gradient vector $\boldsymbol{g}$ of a (hyper)-quadrics $Q = \frac{1}{2}\boldsymbol{\beta}^T \boldsymbol{K}_f \boldsymbol{\beta} - \mathbf{1}^T \boldsymbol{\beta}$ defined as $\boldsymbol{g} = \boldsymbol{K}_f \boldsymbol{\beta} - \mathbf{1}$ equals the negative of residual $\boldsymbol{r}$ in [36]. Here, instead of $\boldsymbol{r}$ we use $\boldsymbol{g}$, as shown in (4.27).

First, from the expression for a gradient vector $\boldsymbol{g}$ above one can easily show that its iterative update can be calculated as

$$\boldsymbol{g}^{k+1} = \boldsymbol{g}^k + \boldsymbol{K}_f \left(\boldsymbol{\beta}^{k+1} - \boldsymbol{\beta}^k \right).$$

However, this would be a very expensive operation because it involves a matrix vector multiplication in each iteration step. In order to make the gradient vector $\boldsymbol{g}$ update feasible, note that after an update of the i-th component of a dual variables vector $\boldsymbol{g}$ by (4.27), the vector difference $\boldsymbol{\beta}^{k+1} - \boldsymbol{\beta}^k$ is a zero vector except for the i-th components β_i. This fact simplifies, and significantly speeds up, the gradient update by rewriting it as follows

$$\boldsymbol{g}^{k+1} = \boldsymbol{g}^k + \boldsymbol{k}_{fi} \left(\beta_i^{k+1} - \beta_i^k \right),$$

where $\boldsymbol{k}_{fi}$ is the i-th column of the matrix $\boldsymbol{K}_f$. This expression for updating the gradient vector $\boldsymbol{g}$ differs from the update of $\boldsymbol{r}$ given in [36, sec. 38]. The difference is a consequence of a clipping operation, expressed by the max operator, in (4.27).

Next, following the second part of the instruction from [36], the iterations must not go in a cyclic order. The iterative scheme must ensure that the best variable, i.e. component β_i^k, is selected for an update. The obvious choice is to choose the variable for which the gradient vector has the biggest absolute value. (Note that this also means picking up the coordinate with the highest absolute error i.e., residual). Such a choice ensures the fastest convergence to the minimum value of the (hyper)quadrics

Q over the nonnegative (hyper)quadrant.

Finally, note that the clipping defined by max operator in (4.27) suggests that there won't be any update whenever $\beta_i^k = 0$ and $g_i^k \geq 0$. In other words, only an often small fraction of data known as support vectors, for which $\beta_i^k > 0$ (disregarding the value of their gradient g_i^k) will be updated. The nonsupport vectors for which $\beta_i^k = 0$ will also be updated but only if $g_i^k < 0$. Such a checking will also speed up the training phase of DL2 SVM particularly when a fraction of support vectors is small.

For further speed-up, whenever there is no update, i.e. $\beta_i^k = 0$ and $g_i^k \geq 0$ we eliminate the current index from further calculation and shrink the gradient and dual variable vector by one. This elimination shows no impact on accuracy, but does show a significant speed up, particularly when a fraction of support vectors is small.

One more remark regarding the selection of a variable which should be updated is now in order. Note that for Gaussian kernel all the diagonal elements of matrix $\boldsymbol{K}_f$ are equal to $1 + 1/k_b + 1/C$ and in the line 3 of Alg. 2 one can select i based on the value of a maximal absolute gradient only. However, when this is not the case (say, when using a polynomial kernel) according to Householder [37] one should examine the quotients of absolute gradient values divided by the corresponding $\sqrt{\boldsymbol{K}_f(i,i)}$ and select the largest quotient, as given in Alg. 2. (Note that the selection of the worst violator in step 3 is the same, but faster, if used $\frac{|g_i|}{\boldsymbol{K}_f(i,i)}$).

Since ISDA (and thus NN ISDA) is an iterative Gauss-Seidel method the convergence of this algorithm is guaranteed because both $\boldsymbol{K}_f$ and $\boldsymbol{K}'_f$ are symmetric and positive definite.

The basic working principle of NN ISDA is *i)* find the maximum violator from the entire gradient vector, *ii)* recalculate the solution vector, and *iii)* update the gradient vector and repeat the process again. In classical learning setup as presented in Alg. 2, the search for the most informative instance (maximum violator) is performed over

the entire gradient vector. Note that, for large datasets, searching the maximum absolute value of the entire vector is a very time consuming and computationally expensive task. We believe that we do not have to search the entire vector at each iteration.

By using the probabilistic speed-up technique [38], we implemented a selection method, which does not necessitate a full search through the entire gradient vector. Instead, it locates an approximate most informative sample by examining a small constant number of randomly chosen samples. The theory states that by randomly sampling only $\lceil log(.05)/log(.95) \rceil = 59$ instances, regardless of the training set size, there is 95% probability that the max violator in this group will be among the top 5% max violators in the entire gradient vector. This approach scales well since the size of the randomly chosen subset is independent of the training set size. This requires significantly less training time and does not have an adverse effect on the classification performance of the learner. In our experiments, we pick 59 random instances to form the subsample pool at each learning step and pick the maximum violator from this pool.

Although NN ISDA guarantees the convergence and does not require entire kernel matrix to be stored in memory, it does help to have the matrix precomputed so that we don't have to re-compute the kernel vector at each learning step. Note that we do require an i'th column of a kernel matrix $\boldsymbol{K}_f$ in each iteration step in order to update the gradient. Having entire matrix in memory is not feasible for large-datasets, just consider a binary problem with 100,000 training points - if we store each element in an 8-byte representation the kernel matrix would require 80GB of space. Consequently, the storage of the entire matrix becomes the bottleneck in training large-scale SVM problems.

To reduce the total number of kernel evaluations we use the Least-Recently-Used

(LRU) caching technique. In LRU-based caching the least-recently-used kernel row is deleted in order to free up the space for new one if the cache is full. It is based on an assumption that the least-recently-used item may not be used in the future. When the size of the cache is smaller than the size of the problem hit ratio is quite low. In the case of NN ISDA, especially for a well conditioned problem where the number of support vectors is quite low, the non-support-vectors that become support vectors will be kept in memory, while the rest of the non-support-vectors might never even be included in kernel computation.

CHAPTER 5

RESULTS

All the results presented in [12] and [22] serve as a starting point to a performance analysis shown here. Extensive investigations of SphereSVM and MN SVM implemented in open-source framework called *GSVM - Command Line Tool for Geometric SVM Training* showed considerable training time speedups in respect to L1/L2 SVM from the SMO based LIBSVM as well as BVM from LibCVM. Impressive training speedup going over a few orders of magnitude for complex datasets, along with comparative accuracy as the other three algorithms, puts the open-source framework GSVM ahead of the curve. The work done in [22] supports these claims since it gives the comparison of MN SVM with LIBSVM's implementations of L1 and L2 SVM as well as BVM. Tables given there clearly show that MN SVM has the best performance of all in terms of training time, especially when data sizes go into large domains. Besides, this algorithm shows comparative accuracies and it usually generates very sparse model (having small number of support vectors). For your convenience we give the comparisons of MN SVM with L1/L2 SVM from LIBSVM and BVM from LibCVM in Appendix A.

5.1 Datasets and experimental environment

To evaluate proposed models and compare their performances we used some benchmark datasets that are available on LIBSVM[1] or UCI Machine Learning Repos-

[1]available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm/

itory[2] sites. Prokaryotic and eukaryotic datasets can be found in [39]. Checkers is a two-dimensional artificial dataset consisting of two classes that are distributed in form of the checker board having four rows and four columns.

Table 4 lists all the datasets used in this dissertation. For each dataset table lists the number of inputs (instances), the number of features or the dimensionality of data and the number of classes. As can be seen, datasets are "divided" into three groups: *small*, *medium* and *large* based on their size, i.e. number of instances. Note that these are (almost) the same datasets as used in [12] where the aforementioned geometrical algorithms SphereSVM and MNSVM were compared to BVM implementation taken from LibCVM package [1], and L1 and L2 SVM implementation from LIBSVM software package [6].

Comparison of models was obtained using a strict double (nested) cross-validation (CV) procedure. Such experimental environment is computationally expensive but is the only way to fairly assess the model's performance. Double CV procedure is structured as two nested loops where the *inner* cross-validation is used to pick the best hyper-parameters (best model), whereas the *outer* cross-validation estimates the performance of the resulting model. In the outer loop, the dataset is separated into J_1 roughly equal-size parts. In all the experiments performed in this dissertation we used $J_1 = 5$. Each part is held out in turn as the test set, and the remaining four parts are used as the training set. In the inner loop, J_2-fold CV is performed over the training set only, to determine the best values for the hyper-parameters (here, $J_2 = 5$ too). The best model obtained in the inner loop is then applied on the outer test set.

[2]available at https://archive.ics.uci.edu/ml/datasets.html

Table 4.: Dataset information

Dataset	# instances	# features	# classes
Small Datasets			
sonar	208	60	2
dermatology	366	33	6
heart	270	13	2
prokaryotic	997	20	3
eukaryotic	2427	20	4
Medium Datasets			
optdigits	5620	64	10
satimage	6435	36	6
usps	9298	256	10
pendigits	10992	16	10
reuters	11069	8315	2
letter	20000	16	26
Large Datasets			
adult	48842	123	2
w3a	49749	300	2
shuttle	58000	7	7
web	64700	300	2
ijcnn1	141691	22	2
covtype	581012	53	7
checkers	3000000	2	2
intrusion	5209460	127	2

Double CV ensures that the model performance is not optimistically biased as would have happened if we used only one cross-validation. This approach is also consistent with the real-world applications scenario. Obviously, such a rigorous procedure is done in many runs, which slows down the learning phase but if the main goal is to compare different algorithms on the same datasets and under the same conditions, then double cross-validation must be used [12].

The only preprocessing performed on the datasets is a feature normalization, in other words, the range of features was rescaled from 0 to 1. Then, double CV was performed to search for the best hyper-parameters and to objectively assess the obtained model. In the case of Gaussian RBF kernel the hyper-parameters are the Gaussian "bell curve" shape and the penalty parameter C. These parameters were selected among 8x8 possible combinations of the shape parameter γ and penalty parameter C. The same values for parameter C and γ as in [12] were used here and they are:

$$C \in \{4^n\}, \quad n = -2, -1, ..., 5, \tag{5.1}$$

$$\gamma \in \{4^n\}, \quad n = -5, -4, ..., 2. \tag{5.2}$$

Different experiments conducted in this dissertation were run on different computer architectures and they will be commented in further sections. Although the implementation of these algorithms does not support multi-threaded execution, we utilize the multi-core architecture by running independent training and testing processes in parallel. To deal with multi-class classification problems we used one-vs-one (aka pairwise) approach. Pairwise training procedure trains $\frac{c(c-1)}{2}$ binary classifiers (each possible pair of classes) where c is the number of classes. Then, during the prediction, a simple voting scheme is used, namely, all $\frac{c(c-1)}{2}$ models predict the unseen data

sample and the class that got the highest number of "votes" is considered to be the true class for a given sample.

5.2 Comparison of three DL2 SVM solvers

In this section we present comparison of three different solvers for DL2 SVM:

1. NNLS with Cholesky Decomposition (section 4.1),
2. NN Conjugate Gradient algorithm (section 4.2) and
3. NN ISDA (section 4.3)

The main motivation for the following experiments was to examine the training speed achieved by three different DL2 SVM solvers. Figures 12, 13 and 14 show the results obtained through nested CV for the nine selected datasets from the small and medium groups. The experiments were performed on a computer with Intel Core i7-930 processor (4-core, 2.80GHz) and a 6 GB of RAM, the code was written in C++.

Fig. 12 is the most informative here, showing clearly that the NN ISDA is the obvious choice for DL2 SVM learning algorithm. For small datasets NN ISDA is, on the average, 360 times faster than CG NNLS algorithm and 256 times faster than the NNLS based on Cholesky factorization. For datasets from the medium group NN ISDA is on the average 38 times faster than the implementation of a NNLS solver with Cholesky decomposition. And it is 75 times faster than the NN CG algorithm for medium sized datasets. Based on such large differences and clearly visible trends, one can expect that NN ISDA will outperform the other two algorithms as the sizes of datasets increase as well.

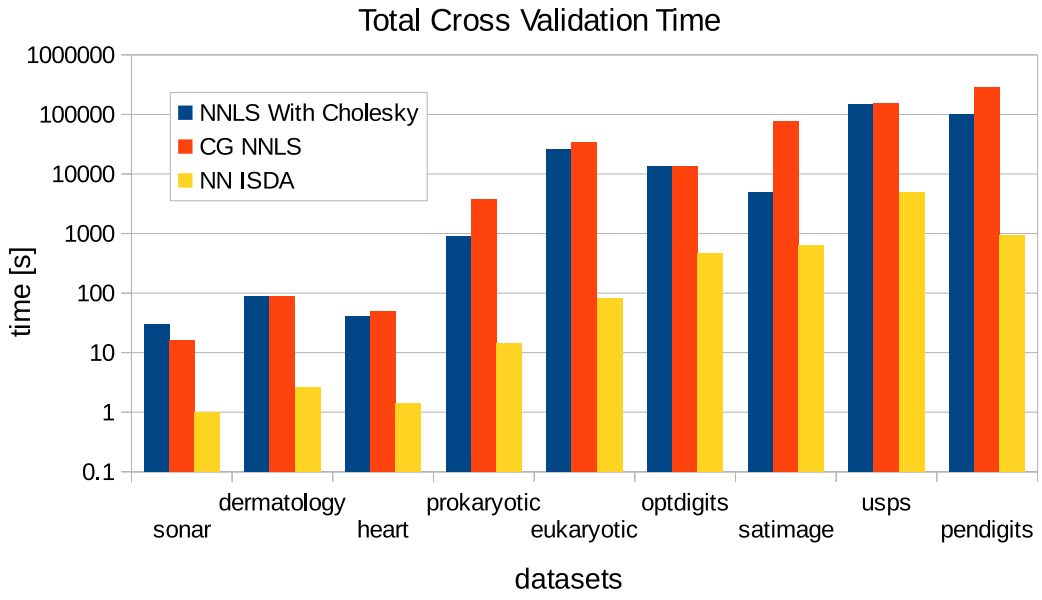


Fig. 12.: Total cross validation time for three different implementation algorithms

Fig. 13 shows the accuracies achieved with three nonnegative algorithms. First remark about the accuracies achieved is that all three learning algorithms are solving the same DL2 SVM problem posed by 3.1 subject to 3.2 and that all three have shown convergence property. The three algorithms show some differences in accuracies, especially with small datasets, namely, NN ISDA and NNLS with Cholesky decomposition seem to have similar accuracies and also slightly higher than CG NNLS. On the average, NN ISDA has a negligible edge in respect to other two DL2 SVM solvers. Differences in accuracies can be attributed to two different factors, one is the random number generator used for splitting the data into folds during the cross-validation, and the other is stopping criteria of these algorithms. All three algorithms perform a successive approximation until some stopping criteria are reached. The path each algorithm takes toward the unique solution is a trait of that algorithm. So, when the stopping criteria are applied, different algorithms can stop in different places near the true solution.

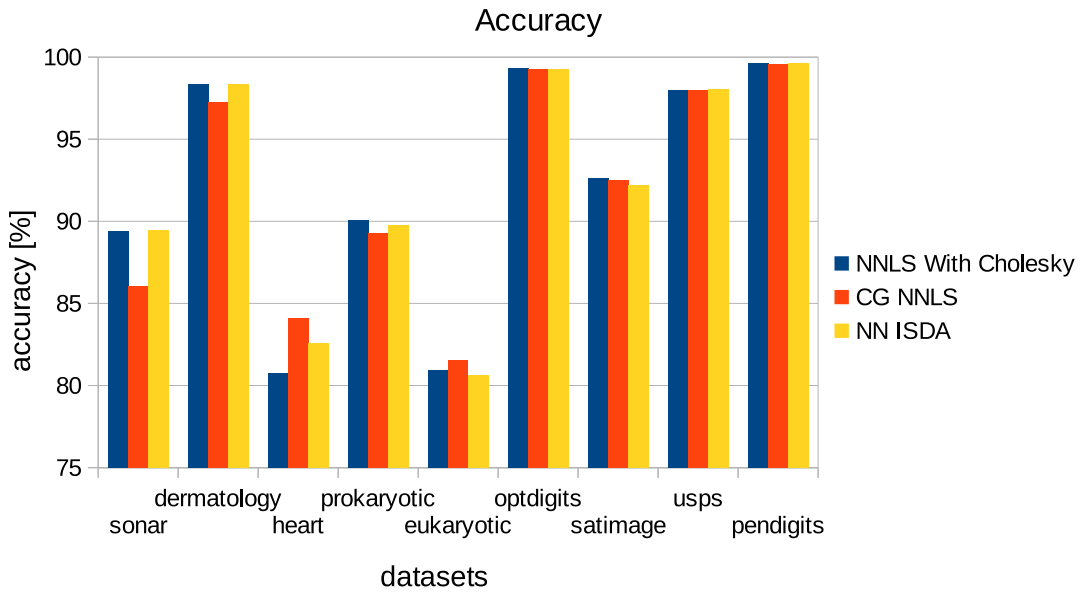


Fig. 13.: Accuracy for three different implementation algorithms

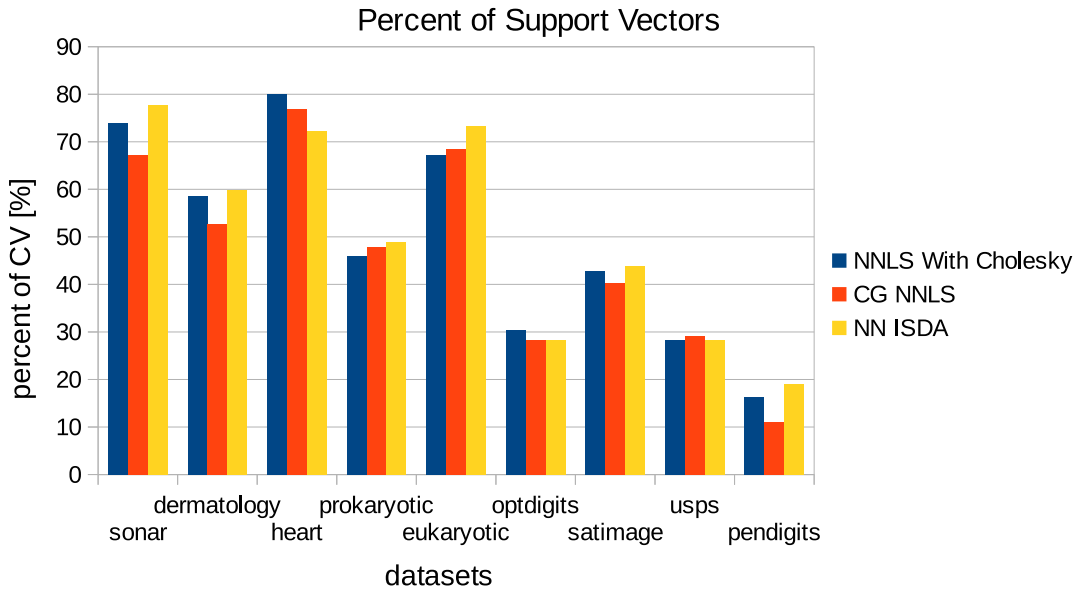


Fig. 14.: Percentage of support vectors for three different implementation algorithms

The average percent of support vectors for three algorithms is shown in the last figure, Fig. 14. On average, NNLS CG has the smallest number of support vectors,

followed by NN with Cholesky, and finally NN ISDA has the largest number of support vectors. However, the differences in number of support vectors are never too extreme and are attributed to the iterative nature of these algorithms.

5.3 Comparison of different variants of Direct L2 SVM

In this section we give the performance observed on medium datasets (the same trend is seen on the small datasets as well) for four different variants of DL2 SVM. In the following three figures where we give total CV time, accuracy and percentage of support vectors you will find the following notation:

- **Model 11**: this is the original DL2 SVM model with both bias term b and a parameter ρ in objective function, as given in 3.1 subject to 3.2.
- **Model 10**: model of DL2 SVM without the parameter ρ as explained in section 3.2.1.
- **Model 01**: model of DL2 SVM without the bias in objective function as introduced in section 3.2.2.
- **Model 00**: finally, this represents a model that has neither bias nor parameter ρ in minimization function (section 3.2.3)

It is important to note what these models have in common. To recap, the first two models have one important thing in common - they work with the same kernel matrix $\mathbf{K}_f$. The difference is that the first model's solution is vector $\boldsymbol{\beta}$ through which we find the real dual variables $\boldsymbol{\alpha}$ and the second model calculates $\boldsymbol{\alpha}$ vector directly so the second model has one additional step towards the solution. The last two models also have one similarity - the same kernel matrix $\mathbf{K}_{f'}$. Difference is the same as between the first two models, one calculates $\boldsymbol{\beta}$ and then the real solution vector $\boldsymbol{\alpha}$

and the other calculates $\boldsymbol{\alpha}$ vector directly.

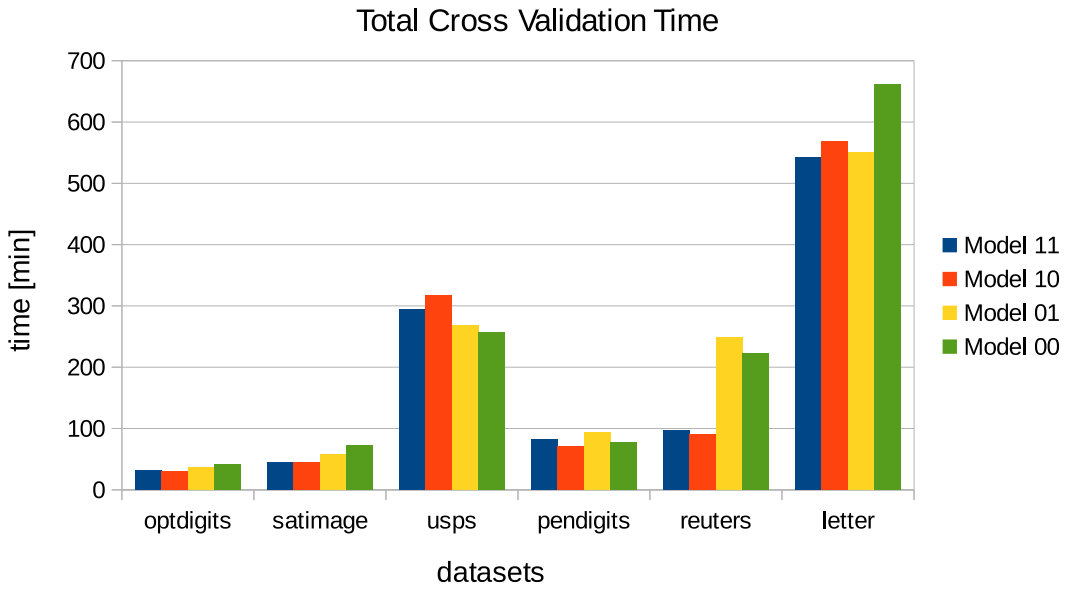


Fig. 15.: Total cross validation time for medium datasets

The hardware used for the comparison of four variants of DL2 SVM was composed of 13 nodes each with four AMD Opteron 6282 SE CPUs with 256 GB RAM.

First figure, Fig. 15 shows the time required to complete the nested cross-validation for medium datasets. Model 11 and model 10 perform better than other two in all cases except for *usps* dataset where model 00 seems to be the winner (the only such case). For the most complex dataset, i.e. *letter* dataset, the original model (model 11) outperforms the others, especially the model without both bias and ρ.

The second figure here, Fig. 16 gives the accuracies obtained during the nested cross validation scheme. Important remark about the accuracies achieved is that all four models have shown convergence property. It can be seen that accuracies for all four models are highly similar. There seems to be no major distinction except for the *satimage* where the model 01 shows a slightly lower accuracy.

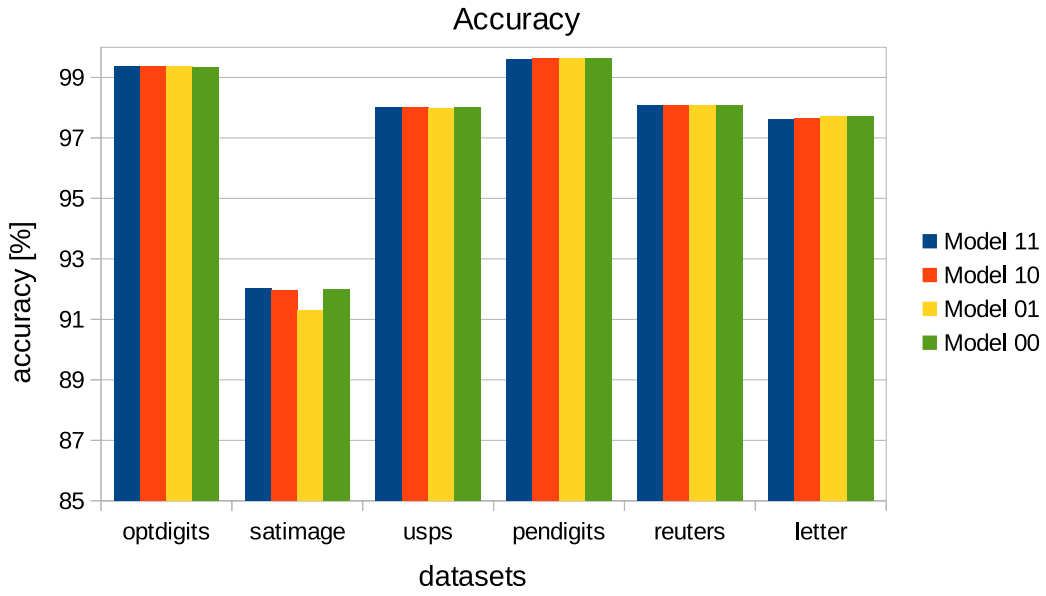


Fig. 16.: Accuracy for medium datasets

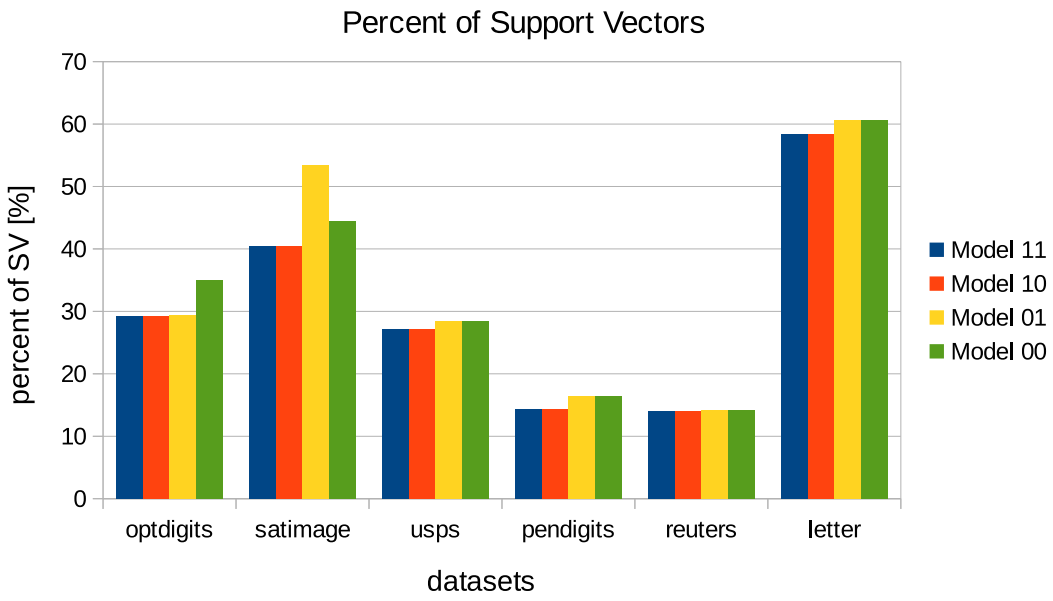


Fig. 17.: Percentage of support vectors for medium datasets

The average number of support vectors (in percentages) is given in Fig. 17. One can observe there is a certain relation between the models with bias term and those

without it - models with the bias term in the objective function give a model with less support vectors in all six datasets.

The conclusion here is that the original model of DL2 SVM and the version without the parameter ρ in objective function are faster than other two models in the majority of cases. The accuracies of all models are practically the same, however there is slight distinction in the number of support vectors - the latter two models have larger numbers of SVs.

5.4 Comparison of Direct L2 and Minimal Norm SVM

Here we present the comparison of DL2 SVM solved with NN ISDA with MN SVM both implemented in GSVM toolkit (written in C++ programming language). As section 5.2 shows, the choice of NN ISDA for DL2 SVM was obvious since it performs better in terms of training speed and accuracy. We used the original DL2 SVM variant with both bias b and parameter ρ in the cost function, and with k_b and k_ρ set to their default value 1. The performance was evaluated based on the average nested CV classification accuracy, CPU time required to complete the cross-validation and the percentage of support vectors or complexity of a model. The following three figures show the results obtained through nested CV for all fourteen datasets from medium and large groups.

These experiments were performed on a computer cluster composed of 6 nodes. Each node was equipped with 2 E5520 Intel Xeon CPUs (4-core, 2.27 GHz) and 24 GB of RAM. As mentioned before, the multi-threaded execution is not available for these algorithms but we did utilize the multi-core environment by decomposing the nested cross-validation procedure into several independent processes.

Figure 18 shows the CPU time needed to complete the nested CV procedure for DL2 SVM (NN ISDA) and MN SVM. One can readily see that the DL2 SVM model

solved by NN ISDA outperforms MN SVM in terms of CPU time in all datasets but one, namely *intrusion*. Note that both algorithms are implemented within the GSVM platform, meaning that the calculation of kernels and double cross-validation procedure are same for both models. Hence, the results presented here are pointing to the very core of numeric and algorithmic characteristics and capacities of the two algorithms.

Based on the results from [12] (which are repeated here in Appendix A for your convenience), this means that the new DL2 SVM algorithm using the NN ISDA is significantly faster than the LIBSVM implementations of L1 and L2 SVMs as well as all the other geometric SVM algorithms represented by the fastest one, BVM. As for the comparisons with the SMO based algorithms note that neither L1 SVM nor L2 SVM algorithm implemented in LIBSVM were able to finish the training in a reasonable time. As per Section 7.1.2.2 in Strack (2013), their simulation runs have

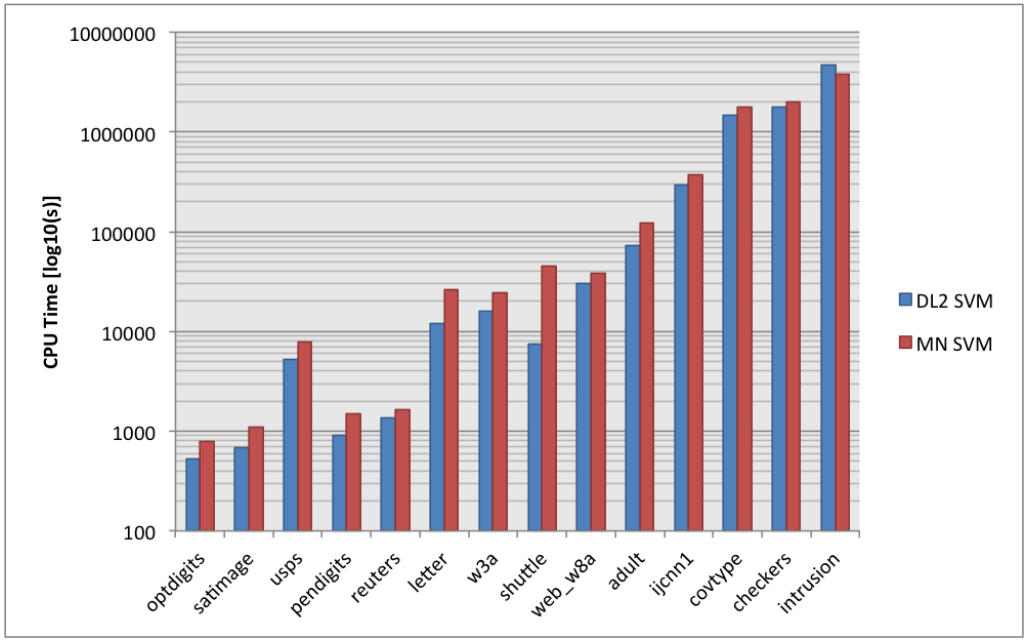


Fig. 18.: Total nested cross-validation time

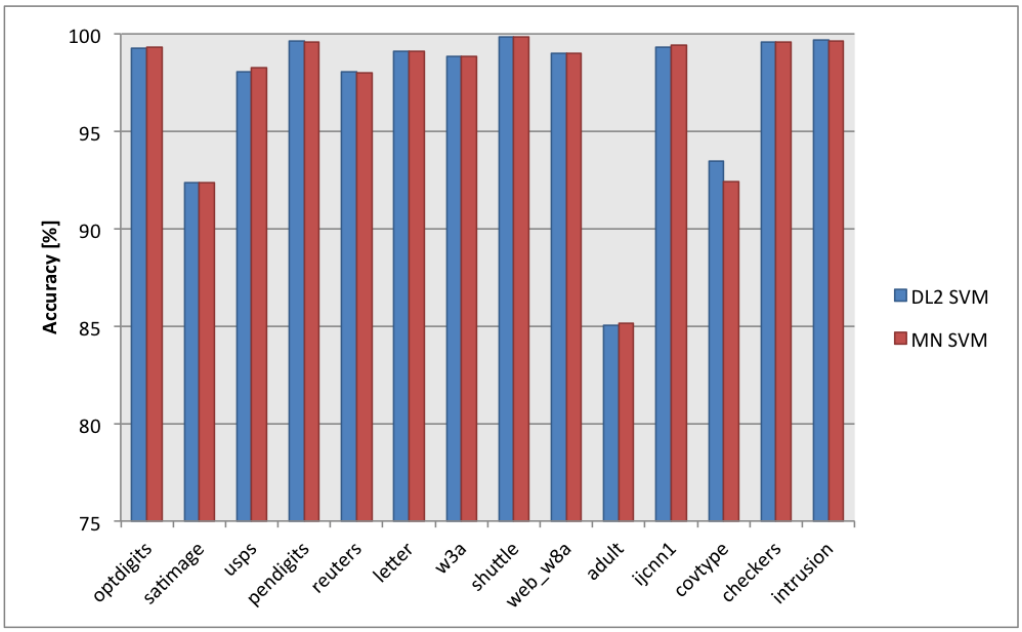


Fig. 19.: Average cross-validation accuracy

been aborted after not being able to finish the training for a single pair of hyper-parameters C and gamma in 60 hours.

Notice that the time axis is in logarithmic scale, that is to say, NN ISDA is in the range of $20 - 80\%$ faster than MN SVM, depending on the dataset.

The average accuracies obtained through nested CV are given in Figure 19. Accuracies for both models are highly similar. Out of fourteen datasets there are only three for which one can notice any difference (*usps, adult* and *covtype*). However, even in those cases the differences are negligible (less than 1%).

Model's size, or the number of support vectors is given in Figure 20 where we can see that MN SVM has lower number of support vectors in majority of datasets (i.e., 11/14). However, this difference is rather small (in most cases within 1%). The only exception is the *shuttle* dataset where DL2 SVM has much less support vectors.

To summarize, Figure 18 is the most informative here, showing clearly that DL2 SVM outperforms MN SVM and thus several other state-of-the-art algorithms in

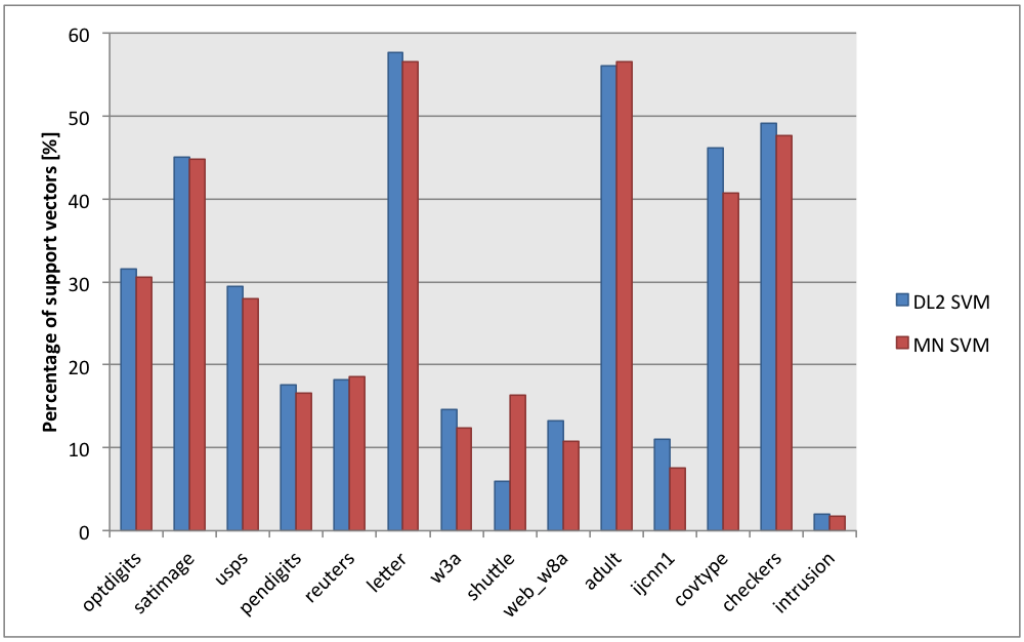
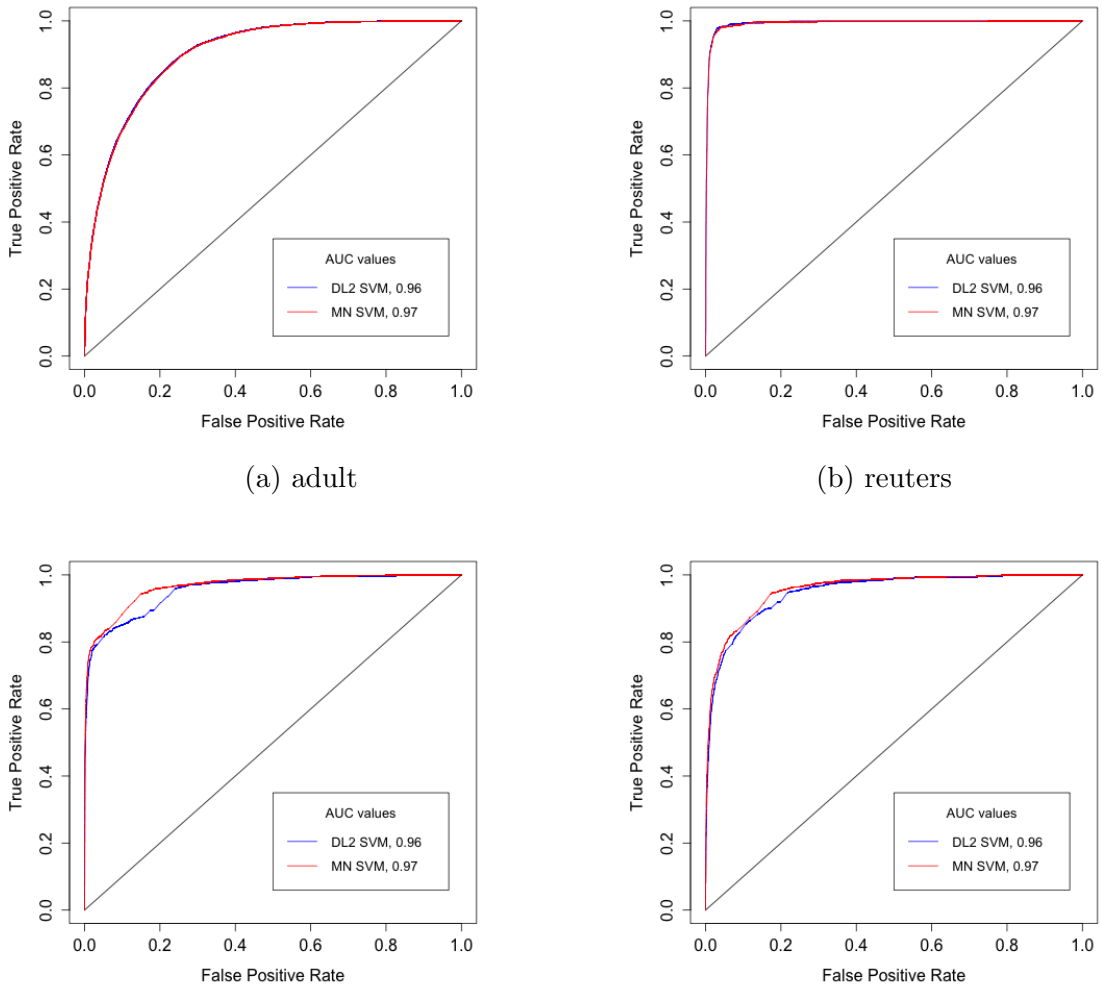


Fig. 20.: Percentage of support vectors obtained in one-vs-one training

terms of training speed. The second figure shows that accuracy is kept at the same level as MN SVM. Therefore, there is no trade-off here between training speed and accuracy which is typical in these scenarios. The size of a DL2 SVM model is only slightly larger than that of MN SVM.

In the following experiment we measured the Receiver Operator Characteristic (ROC) curves, which show how the number of correctly classified positive examples varies with the number of incorrectly classified negative examples. The results are shown on four different datasets: *adult*, *reuters*, *web* and *w3a*, see Figure 21. For this experiment we first computed the best hyper-parameters for both MN SVM and DL2 SVM separately using the 5-fold cross validation and then used the best pair of parameters to evaluate the ROC curve. Area Under Curve (AUC) which is often used as a metric to define how an algorithm performs over the whole ROC space tells us that MN SVM performs only slightly better than DL2 SVM. For this experiment



(a) adult

(b) reuters

(c) web

(d) w3a

Fig. 21.: The ROC analysis for adult, reuters, web and w3a datasets

we used MATLAB's function *perfcurve* to plot the ROC curve and to compute the AUC (*perfcurve* uses trapezoidal approximation to estimate the area).

5.5 Influence of k_b and k_ρ on learning

In this section we describe the influence of parameters k_b and k_ρ on cross-validation accuracy and training time of the original DL2 SVM model with both

bias and parameter ρ in the cost function. All the experiments were run on checkers data, a two-dimensional artificial dataset consisting of two classes that are distributed in a form of checkers board.

Figures 22, 23 and 24 show the accuracy, training time and the average number of SVs obtained through 5-fold cross-validation for 4 different values of Gaussian shape parameter sigma and six different values for parameter k_b which multiplies the bias term b^2 in the cost function of DL2 SVM. To obtain these results we used the NN ISDA solver and a fixed value of 1 for the parameter k_ρ.

What the experiments show is that increasing the value of parameter k_b positively impacts the accuracy of our model. Namely, setting the k_b to a too small value (say 1e-04) directly corresponds to adding a large value to a matrix $\boldsymbol{K}$ (see 3.4) which means that our system matrix $\boldsymbol{K}_f$ becomes too ill-conditioned, which is why we observe such low accuracies for the entire range of sigma values. Slightly larger value of 0.001 performs better, however not as good as what we observe for $k_b \geq 0.1$. For the larger values of k_b the cross-validation accuracies increase (they even overlap for this dataset). For larger values of sigma parameter none of the k_b values give a good solution.

Fig. 23 tells us that larger values of k_b (namely, 1 and above) correspond to a faster training time. The smaller the value for k_b the more ill-conditioned matrix $\boldsymbol{K_f}$ becomes which directly affects the time it takes to find a solution. Fig. 24 shows that for small values of k_b the average number of support vectors is bigger, which reinforces the opinion that larger k_b values should be used.

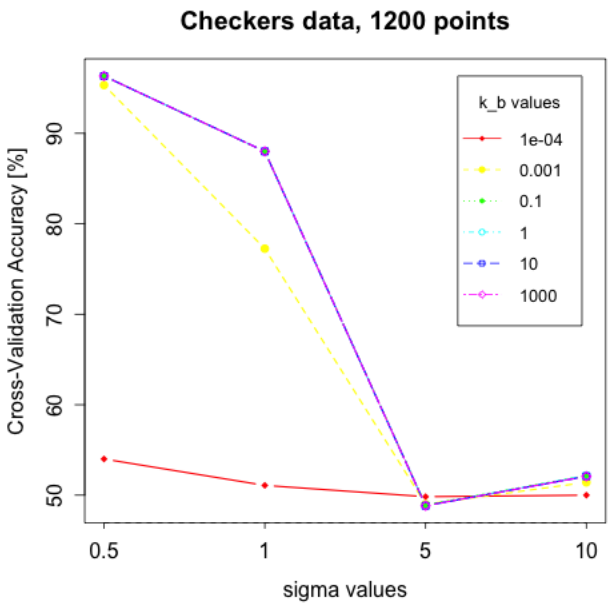


Fig. 22.: Influence of k_b on CV accuracy

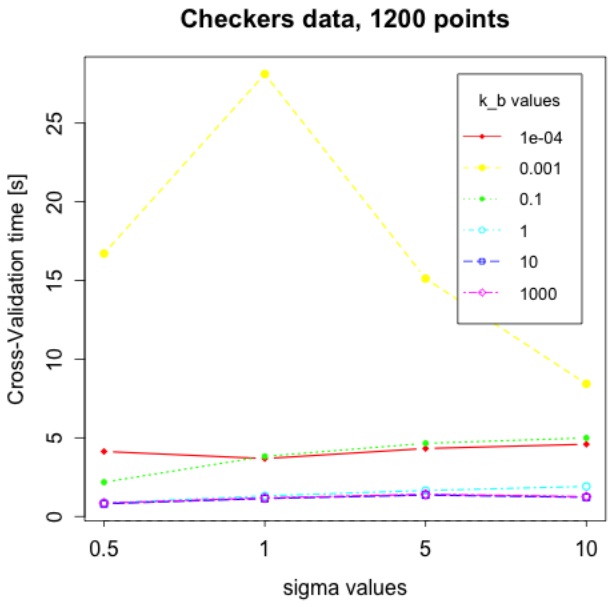


Fig. 23.: Influence of k_b on CV training speed

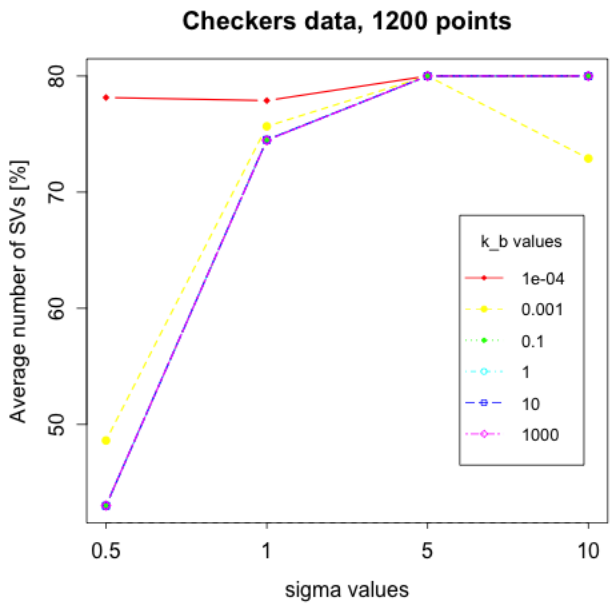


Fig. 24.: Influence of k_b on number of support vectors

Figures 25, 26 and 27 show the accuracy, training time and the average number of SVs obtained through 5-fold cross-validation for 4 different values of Gaussian shape parameter sigma and six different values for parameter k_ρ which multiplies parameter ρ in the cost function of DL2 SVM. To obtain these results we used the NN ISDA solver and a fixed value of 10 for the parameter k_b.

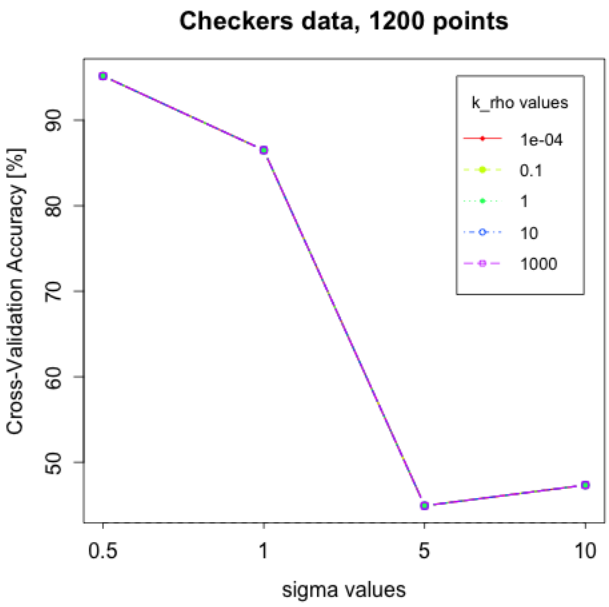


Fig. 25.: Influence of k_ρ on CV accuracy

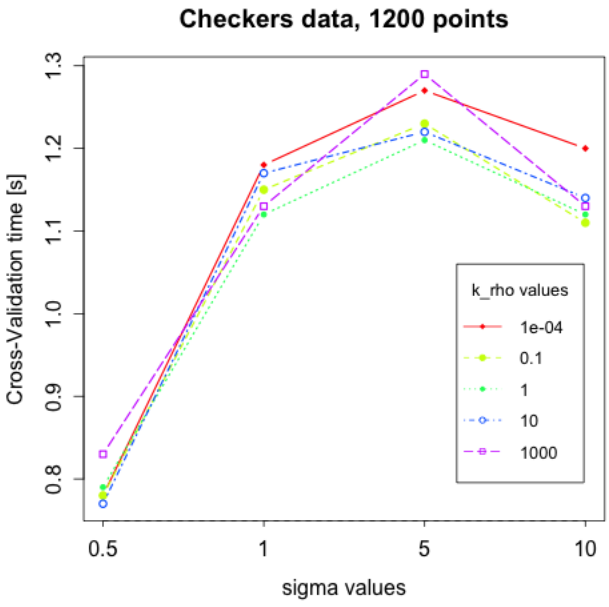


Fig. 26.: Influence of k_ρ on CV training speed

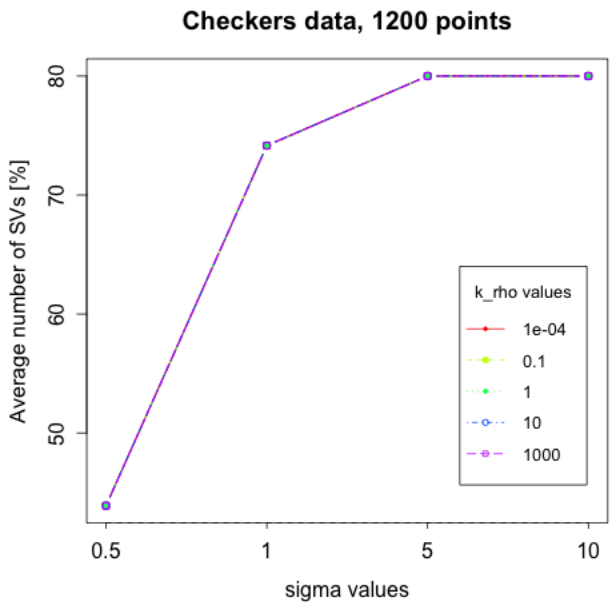


Fig. 27.: Influence of k_ρ on number of support vectors

Parameter k_ρ is not as influential on learning as the parameter k_b explored above. Namely, the parameter k_ρ has no affect on the accuracy nor the number of support vectors of the final model, as the results in figures 25 and 27 show. Theoretically, this is not a surprise since the parameter k_ρ only serves as a scaling factor for ρ and thus for the solution vector $\boldsymbol{\alpha}$ (see 3.6). The larger the value of k_ρ means larger parameterρ and consequently the larger norm of the solution vector $\boldsymbol{\alpha}$.

CHAPTER 6

CONCLUSIONS

The dissertation presents a new formulation of an L2 SVM problem, dubbed Direct L2 SVM (DL2 SVM), in which the minimization of an underlying L2 SVM's QP problem is replaced by a system of linear equations subject to nonnegativity constraints. Starting from the geometric L2 SVM problem setting in primal domain, the novel DL2 SVM model is derived in dual domain. Furthermore, the new Non-Negative Iterative Single Data Algorithm (NN ISDA) is devised and used as a learning algorithm for DL2 SVM.

DL2 SVM represents a comprehensive L2 SVM model that has strong relations to several other popular and well-established models. In this dissertation we have shown its relations to Least-Squares SVM and Proximal SVM. The main difference and an advantage to DL2 SVM, with respect to these two popular and established L2 SVM algorithms, is that the vector of dual variables for DL2 SVM is sparse. This makes DL2 SVM an algorithm of choice when facing large datasets and when a direct solution to a system of equations without constraints (as required for training in both LS and Proximal SVM) is not feasible.

Another strong connection worth pointing out is the one between DL2 SVM and several other geometric SVMs: Core, Ball, Sphere and Minimal Norm. Namely, the cost function in primal domain is the same for all of them when parameters k_b and k_ρ in DL2 SVM are set to 1. Having flexible scalar values for these two parameters allows further optimization in the case of DL2 SVM. Difference between these models is that DL2 SVM solves the problem algebraically, while previously

mentioned geometric SVMs solve the minimization task by utilizing some geometric structure in kernel-induced feature space.

The performance comparisons using the double (nested) cross validation scheme show that for the fourteen datasets DL2 SVM can be up to 80% faster than MN SVM. At the same time DL2 SVM achieves the same accuracy. The choice of MN SVM for comparison is not by chance. Under the entirely same experimental conditions as here, a series of extensive simulations in [12] have shown that MN SVM is faster than both L1 SVM and L2 SVM implementations in LIBSVM. The same experimental runs have also shown that MN SVM is faster than the fastest geometric SVM: BVM implemented in LibCVM platform.

The training time, size of the model and accuracy of DL2 SVM obtained through thorough comparisons with the state-of-the-art SVM algorithms aimed for large datasets make DL2 SVM the strongest candidate for nonlinear SVMs when the number of training data goes to millions.

Appendix A

COMPARISON OF MINIMAL NORM SVM WITH L1, L2 SVM AND BVM

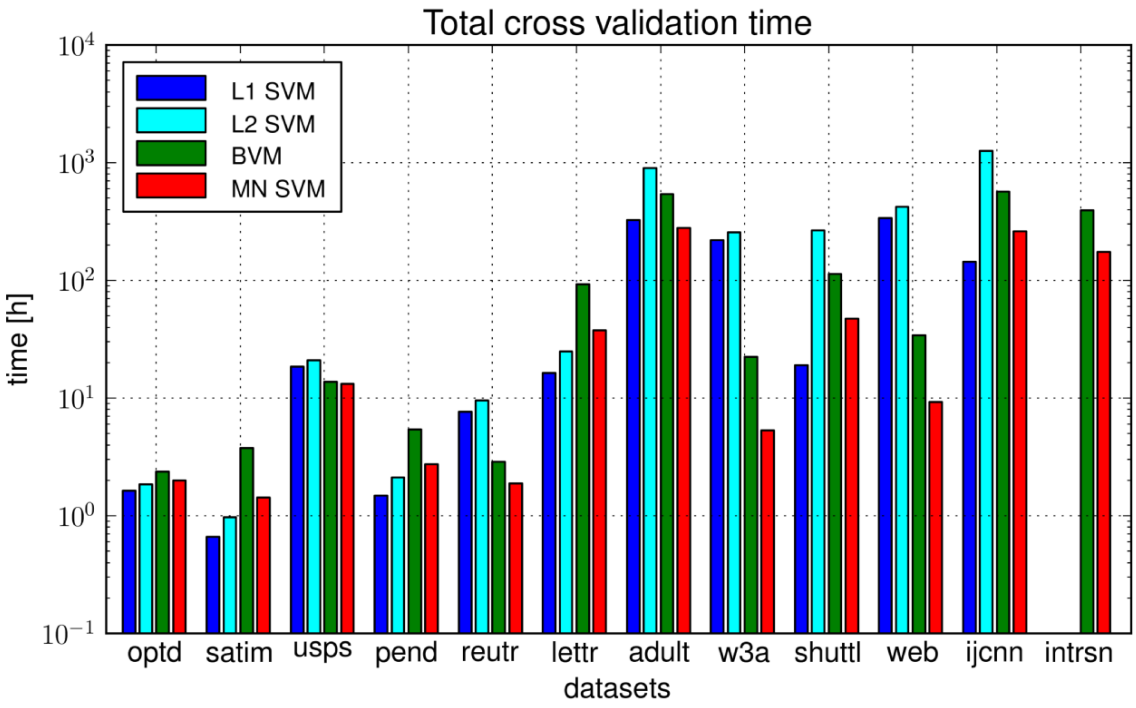


Fig. 28.: Total nested cross validation time. For each dataset the bars represent training time of (bars from left to right): L1 and L2 SVM implementations in LIBSVM, BVM and MNSVM.

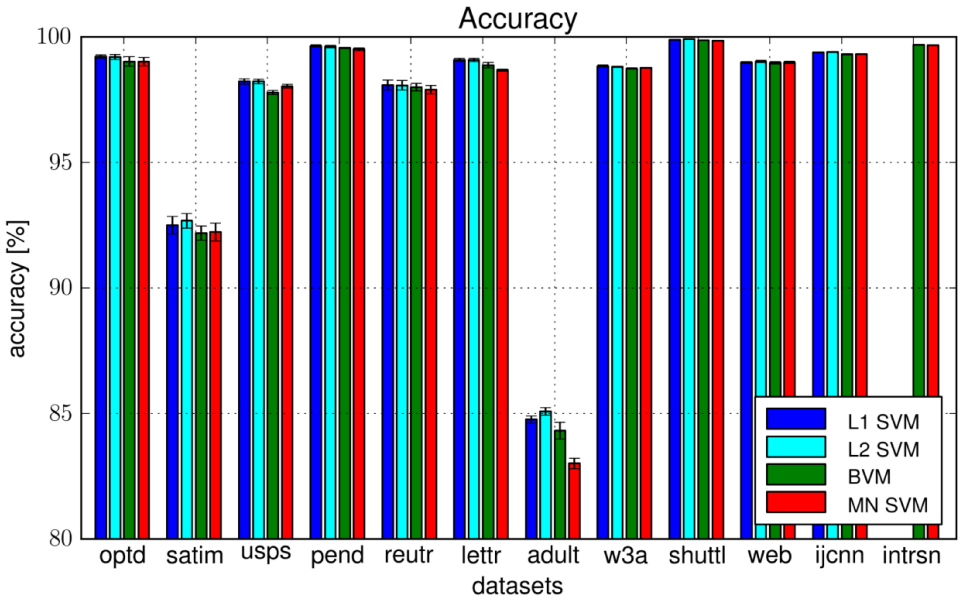


Fig. 29.: Accuracy obtained during nested cross validation. For each dataset the bars represent accuracy of (bars from left to right): L1 and L2 SVM implementations in LIBSVM, BVM and MNSVM.

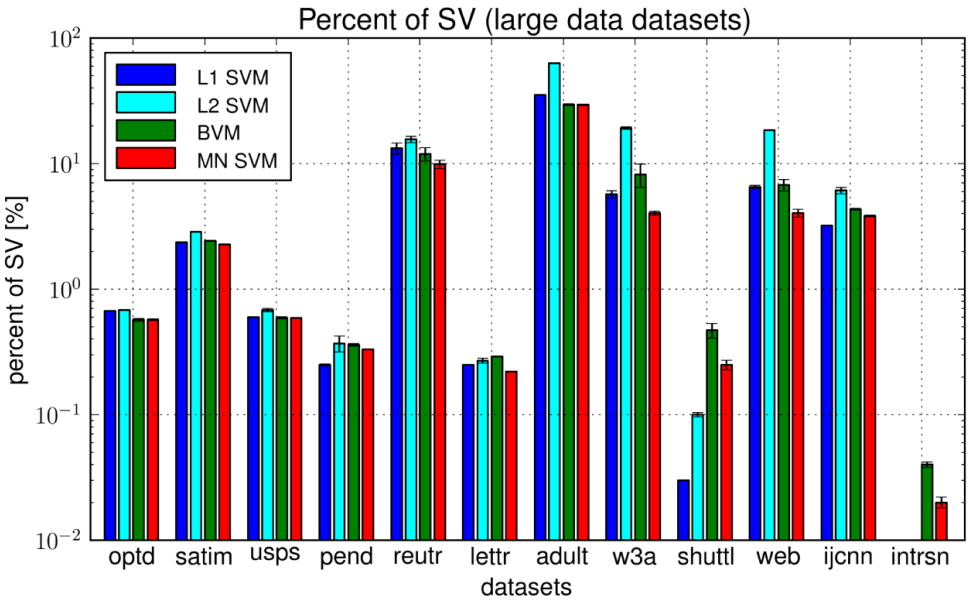


Fig. 30.: Average percent of support vectors obtained in one-vs-one training. For each dataset the bars represent the model size for (bars from left to right): L1 and L2 SVM implementations in LIBSVM, BVM and MNSVM.

REFERENCES

[1] Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. "Core Vector Machines: Fast SVM Training on Very Large Data Sets". In: *Journal of Machine Learning Research* 6 (2005), pp. 363–392.

[2] Boser B. E. Guyon I. M. and Vapnik V.N. "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory* (1992), pp. 144–152.

[3] R. Freund E. Osuna and F. Girosi. "An Improved Training Algorithm for Support Vector Machines". In: *Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop* (1997), pp. 276–285.

[4] John C. Platt. "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines". In: *Microsoft Research, Technical Report MSR-TR-98-14* (1998).

[5] T. Joachims. "Making large-scale support vector machine learning practical". In: *Advances in kernel methodsh* (1999), pp. 169–184.

[6] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: A library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`, 27:1–27:27.

[7] H. Graf E. Cosatto L. Bottou I. Durdanovic V. Vapnik. "Parallel support vector machines: The cascade svm". In: *Advances in Neural Information Processing Systems* (2005), pp. 521–528.

[8] Bengio Y. Collobert R. Bengio S. “A parallel mixture of SVMs for very large scale problems”. In: *Neural Computation* (2002), pp. 1105–1114.

[9] Fox G. Sun Z. “Study on Parallel SVM Based on MapReduce”. In: *International Conference on Parallel and Distributed Processing Techniques and Applications* (2012), pp. 495–561.

[10] Balaban ME Catak FO. “CloudSVM : Training an SVM Classifier in Cloud Computing Systems”. In: *Pervasive Computing and Networked World* (2012), pp. 57–68.

[11] K. P. Bennett and E. J. Bredensteiner. “Duality and Geometry in SVM Classifiers”. In: *Proc. 17th International Conf. on Machine Learning* (2000), pp. 57–64.

[12] Strack R. “Geometric Approach to Support Vector Machines Learning for Large Datasets”. PhD thesis. Virginia Commonwealth University, 2013.

[13] Vandewalle J. Suykens J. “Least Squares Support Vector Machine Classifiers”. In: *Neural Processing Letters* 9 (1999), pp. 293–300.

[14] G. Fung and O. L. Mangasarian. “Proximal Support Vector Machine Classifiers”. In: *Proceedings KDD-2001: Knowledge Discovery and Data Mining, August 26-29, 2001, San Francisco, CA*. Ed. by F. Provost and R. Srikant. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-02.ps. New York: Asscociation for Computing Machinery, 2001, pp. 77–86.

[15] Vapnik and Lerner. “Pattern recognition using generalized portrait method”. In: *Automation and Remote Control* 24 (1963), pp. 774–780.

[16] Braverman E. Aizerman M. and Rozonoer L. "Theoretical foundations of the potential function method in pattern recognition learning". In: *Automation and Remote Control* 25 (1964), pp. 821–837.

[17] Cortes C. and Vapnik V. "Support-vector networks". In: *Machine learning* 297 (1995), pp. 273–297.

[18] Abe Shigeo. *Support Vector Machines for Pattern Classification, 2nd ed.* Springer-Verlag, 2010.

[19] M. G. Genton. "Classes of kernels for machine learning: A statistics perspective". In: *Journal of Machine Learning Research* 2 (2001), pp. 299–312.

[20] Zigic Lj. Strack R. and Kecman V. "L2 Support Vector Machines Revisited - Novel Direct Learning Algorithm And Some Geometric Insights". In: *Proceedings of 19th International Conference on Soft Computing, Brno, Czech Republic* (2013), pp. 334–339.

[21] Leo Doktorski. "Dependence on the regularization parameter". In: *Pattern Recognition and Image Analysis* 21.2 (2011), pp. 254–257.

[22] Robert Strack and Vojislav Kecman. "Minimal Norm Support Vector Machines for Large Classification Tasks". In: *11th International Conference on Machine Learning and Applications*. Boca Raton, FL, 2012, pp. 209–214.

[23] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer series in operations research and financial engineering. Springer, 2006.

[24] Lawson C.L. and Hanson R.J. *Solving Least Squares Problems*. Prentice-Hall, Inc., 1974.

[25] Zigic Lj. and Kecman V. "Variants and Performances of Novel Direct Learning Algorithms for L2 Support Vector Machines". In: *L. Rutkowski et al. (Eds.): ICAISC 2014, Part II, LNAI 8468* (2014), pp. 82–91.

[26] Christensen P. R. Berg C. and Ressel P. *Harmonic Analysis on Semigroups: Theory of Positive Definite and Related Functions*. Springer-Verlag, 1984.

[27] Eigensatz M. "Insights into the Geometry of the Gaussian Kernel and an Application in Geometric Modeling". PhD thesis. Swiss Federal Institute of Technology Zurich, 2006.

[28] Hestenes M.R. *Conjugate Direction Method in Optimization*. Springer-Verlag, 1980.

[29] Kecman V. and Zigic Lj. "Algorithms for Direct L2 Support Vector Machines". In: *Innovations in Intelligent Systems and Applications (INISTA) Proceedings, 2014 IEEE International Symposium, Alberobello, Italy* 8468 (2014), pp. 419–424.

[30] Zigic Lj. and Kecman V. "Direct L2 Support Vector Machine Classifier and Performances of Its Two Implementations". In: *IEEE South-East Conference, Lexington, KY (in print)* (2014).

[31] Rasmus Bro and Sijmen De Jong. "A fast non-negativity-constrained least squares algorithm". In: *Journal of Chemometrics* 11.5 (1997), pp. 393–401.

[32] G.H. Golub and C.F. van Loan. *Matrix Computations, 3rd edition*. The Johns Hopkins University Press, 1996.

[33] R. Fletcher and C. M. Reeves. "Function minimization by conjugate gradients". In: *The Computer Journal* 6 (1964), pp. 149–154.

[34] Kecman V. Huang T. M. and Kopriva I. *Kernel Based Algorithms for Mining Huge Data Sets, Supervised, Semi-supervised, and Unsupervised Learning*. Springer-Verlag, 2006.

[35] V. Kecman, T. -m. Huang, and M. Vogt. "Iterative Single Data Algorithm for Training Kernel Machines from Huge Data Sets: Theory and Performance". In: *Performance, Support Vector Machines: Theory and Applications, Springer-Verlag,.Studies in Fuzziness and Soft Computing*. Springer Verlag, 2005, pp. 255–274.

[36] Fadeev D. K. and Faddeeva V. N. *Computational Methods of Linear Algebra*. W. H. Freeman, Company, San Francisco, and London, 1963.

[37] Alston Householder. *Principles of Numerical Analysis*. McGraw-Hill, 1953.

[38] Smola A. J. and B. Scholkopf. "Sparse Greedy Matrix Approximation for Machine Learning". In: *In Proceedings of International Conference on Machine Learning (ICML)* (2000), pp. 911–918.

[39] Reinhardt A. and Hubbard T. "Using neural networks for prediction of the subcellular location of proteins". In: *Nucleic Acids Research* 26 (1998), pp. 2230–2236.

VITA

Ljiljana Zigic received her MSc degree in Mechatronics, Robotics and Automation from University of Novi Sad, Novi Sad, Serbia, in 2012. She is currently working towards her Ph.D. degree in Computer Science at Virginia Commonwealth University, Richmond, USA. Ljiljana's research is oriented towards Machine Learning, Data Mining and Artificial Intelligence algorithms. Her more specialized field of interest is support vector machine classification.

Papers:

1. **Zigic Lj.**, Kecman V.: *Direct L2 Support Vector Machine.* Under review in: Journal of Neurocomputing, (2016)
2. **Zigic Lj.**, Strack R., Kecman V.: *L2 Support Vector Machines Revisited - Novel Direct Learning Algorithm And Some Geometric Insights.* In: Proceedings of 19th International Conference on Soft Computing, Brno, Czech Republic, pp. 334-339, (2013)
3. **Zigic Lj.** and Kecman V.: *Variants and Performances of Novel Direct Learning Algorithms for L2 Support Vector Machines.* In: L. Rutkowski et al. (Eds.): ICAISC 2014, Part II, LNAI 8468 (2014), pp. 8291, Springer Verlag, (2014)
4. Kecman V. and **Zigic Lj.**: *Algorithms for Direct L2 Support Vector Machines.* In: Innovations in Intelligent Systems and Applications (INISTA) Proceedings, 742014 IEEE International Symposium, Alberobello, Italy 8468, pp. 419-424, (2014)

5. **Zigic Lj.** and Kecman V.: *Direct L2 Support Vector Machine Classifier and Performances of Its Two Implementations.* In: IEEE South-East Conference, 2013 Proceedings of IEEE, Lexington, KY, (2014)

6. Test E., Kecman V., **Zigic Lj.**: *Feature Ranking Using Gini Index, Scatter Ratios, and Nonlinear SVM RFE.* In: IEEE South-East Conference, 2013 Proceedings of IEEE, pp. 1-5, (2013).

Seminars:

1. Kecman V., Strack R., **Zigic Lj.**: *Big Data Mining by L2 SVMs - Geometrical Insights Help.* Seminar at CS Department, Virginia Commonwealth University, VCU, Richmond, VA, April 24, (2013).