2013

Geometric Approach to Support Vector Machines Learning for Large Datasets

Robert Strack
Virginia Commonwealth University

Geometric Approach to Support Vector Machines Learning for Large Datasets

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

by

Robert Strack
M.Sc. Eng., AGH University of Science and Technology (Poland), 2007

Director: VOJISLAV KECMAN
ASSOCIATE PROFESSOR, DEPARTMENT OF COMPUTER SCIENCE

Virginia Commonwealth University
Richmond, Virginia,
May, 2013

Abstract

The dissertation introduces Sphere Support Vector Machines (SphereSVM) and Minimal Norm Support Vector Machines (MNSVM) as the new fast classification algorithms that use geometrical properties of the underlying classification problems to efficiently obtain models describing training data. SphereSVM is based on combining minimal enclosing ball approach, state of the art nearest point problem solvers and probabilistic techniques. The blending of the three speeds up the training phase of SVMs significantly and reaches similar (i.e., practically the same) accuracy as the other classification models over several big and large real data sets within the strict validation frame of a double (nested) cross-validation (CV). MNSVM is further simplification of SphereSVM algorithm. Here, relatively complex classification task was converted into one of the simplest geometrical problems – minimal norm problem. This resulted in additional speedup compared to SphereSVM. The results shown are promoting both SphereSVM and MNSVM as outstanding alternatives for handling large and ultra-large datasets in a reasonable time without switching to various parallelization schemes for SVMs algorithms proposed recently.

The variants of both algorithms, which work without explicit bias term, are also presented. In addition, other techniques aiming to improve the time efficiency are discussed (such as over-relaxation and improved support vector selection scheme). Finally, the accuracy and performance of all these modifications are carefully analyzed and results based on nested cross-validation procedure are shown.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Support Vector Machines are known as one of the best classification tools available today. Many experimental results on variety of classification (and regression) tasks complement the highly appreciated theoretical properties of SVMs. However, there is one property of SVM's learning algorithm which required, and it is still requiring, special attention. This is the fact that the learning phase of SVMs scales poorly with the number of training datapoints. Hence, with an increase of datasets' sizes, the learning can be a quite slow process. The first successful attempts in resolving the issue were the chunking method described by Boser et al. [1] and the decomposition approaches introduced by Osuna et al. [2] which led to several efficient software packages the most popular being SVMlight [3] and LIBSVM [4]. However, the ever increasing size of datasets has driven the SVMs training time beyond the acceptable limits. The two remedy avenues for overcoming the issues of large datasets proposed and used in the last decade were various parallelization attempts (including the newest GPU embedded implementations [5, 6]) and the use of geometric approaches. The later includes solving SVMs' learning problem by both convex hulls and the enclosing ball approach [7, 8]. The most recent and advanced method known as the Ball Vector Machines [9] has demonstrated high capability for handling large datasests.

The Sphere Support Vector Machine (SphereSVM) and Minimal Norm Support Vector Machines (MNSVM) proposed here combine the two techniques (namely, convex hull and enclosing balls approaches). While keeping the level of accuracy they achieve a significant speedup with respect to all three L1 and L2 LIBSVM and BVMs.

Although the most popular SVM solvers (such as Platt's SMO [10]) are very efficient in searching for the solution in the dual space, there is a lot of research conducted towards finding efficient algorithms that work directly in the feature space. These algorithms are mostly based on the geometric interpretation of the maximum margin classifiers.

The geometric properties of the hard margin SVM classifiers have been known for a long time [11]. Recently, Keerthi et al. [12] and Franc [13] proposed algorithms based on geometric interpretation of the SVM algorithm for solving cases with separable classes. Their approach treats the problem of finding the maximum margin between two classes as a problem of finding two closest points belonging to convex polytopes covering the classes. Crisp et al. analyzed geometric properties of ν-SVM algorithm [14] and based on this work, Mavroforakis introduced the Reduced Convex Hulls (RCH) [15] (see Figure 1.1). The idea allowed using geometric approach in solving SVM problems for overlapping classes. Reduced convex hulls enabled a shrink of overlapping convex polytopes covering each class. This reduction creates the margin between two overlapping classes and allows to separate them (which was previously unfeasible with Keerthi's and Franc's algorithms).

Another field of research involves algorithms based on Minimal Enclosing Ball (MEB) problem. Tsang et al. [7, 16] formulated SVM problem as MEB problem and proposed Core Vector Machines (CVM) algorithm as an approach suitable for very large SVM training. Their algorithm is an application of Badoiu and Clarkson's work [17] that investigates the use of coresets in finding approximation of MEB. This work is further generalized in [18] by allowing the use of any kernel function (and not only the normalized ones as it was previously required). Furthermore, Tsang et al. in [9] improve the idea of Core Vector

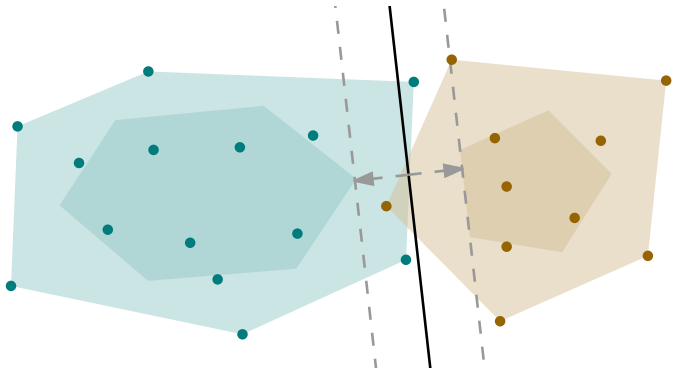

Figure 1.1: Decision boundary for a problem with two overlapping classes that was solved with reduced convex hulls. Dark blue and yellow polygons represent reduced convex hulls obtained for both classes, the black solid line is the decision boundary and the black dashed lines represent the decision margin.

Machines by introducing new algorithm not requiring QP solver – Ball Vector Machines (BVM). Additional speedup was obtained by using "probabilistic speedup" approach proposed by Smola and Schölkopf [19]. Moreover, Asharaf et al. [20] proposed another extension of CVM that is capable of handling multi-class problems.

In this dissertation we propose new algorithms that improve the BVM by applying ideas previously used in SVM learning based on RCH. SVM solver involving finding two closest points on non-overlapping RCH was introduced by Mavroforakis and Theodoridis [15]. It was further improved by López et al. [21] by replacing SK algorithm (Kozinec [22]), that was used in searching for the closest points, with faster MDM algorithm introduced by Michell et al. in [23]. Our work, similarly to López's, introduces an algorithm originated within an MDM solver as the technique for finding minimal enclosing ball. This novel MEB algorithm (SphereSVM) is successfully applied in solving MEB problem corresponding to the L2 SVM learning task. This approach is further simplified by utilizing connection between MEB and minimal norm problems and a new more efficient algorithm is introduced (Minimal Norm SVM).

1.1 Contributions of the Dissertation

The major contributions of the dissertation are:

- introduction of SphereSVM and MNSVM algorithms aiming to solve large classification problems
- proof of convergence and estimation of the computational complexity of the SphereSVM algorithm
- implementation of the over-relaxation technique in solving the Minimal Enclosing Ball and Minimal Norm problems
- study of the role of the bias in SphereSVM and MNSVM training and introduction of the novel version of the algorithms without bias
- application of Improved MDM solver into solving L2 SVM classification tasks based on MNSVM algorithm
- generalization of binary (two-class) SphereSVM and MNSVM algorithms into multi-class SVM solvers
- proposition of a new model selection approach (sparse grid search) that is capable of finding model parameters in approximately linear time preserving the accuracy of the grid search method
- development of an efficient open-source framework called “gsvm” suitable for solving large nonlinear classification tasks

Chapter 2

Background

2.1 Large Margin Classifiers

The Generalized Portrait algorithm introduced by Vapnik [24] in mid 60's gave a foundation to modern maximum margin classifiers. A theoretical basis of a principle of structural risk minimization which is the basis for developing maximal margin classifier is given in [25]. Based on the work of Aizerman [26], Boser, Guyon and Vapnik [1] generalized the original linear algorithm and applied it to a nonlinear case. Finally, the soft margin Support Vector Machines were introduced by Cortes and Vapnik [27]. This modification not only allowed to use maximum margin classifiers for non-separable datasets but also introduced the regularization parameter that can be used to prevent over-fitting the dataset.

The supervised learning is the process of determining an input-output relationship $f(\mathbf{x})$ by using a training dataset $\mathcal{X} = \{\mathbf{x}_i\}$, containing m inputs $\mathbf{x}_i$ from d-dimensional input space $\mathbf{x}_i \in \mathbb{R}^d$, and the labels y_i assigned to each of these input vectors. In case of the simplest classification problem, being called the binary classification, there are two possible output values $y_i \in \{-1, 1\}$. From now on, we assume that the dataset $\mathcal{X}$ is given as a matrix X of size (m, d) consisting of m input vectors $\mathbf{x}_i$ arranged in row order and Y is a column vector

of length m containing vector labels y_i.

The goal of binary SVM classifier is to find a classification function

$$f(\mathbf{x}) = \operatorname{sgn} d(\mathbf{x}), \tag{2.1}$$

that assigns value 1 or -1 to a given vector $\mathbf{x}$ depending what is the predicted class of that vector. The function $d(\mathbf{x})$ is a linear decision function defined as

$$d(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b. \tag{2.2}$$

In the case of hard margin classifiers, that do not permit classification error on the training samples, both $\mathbf{w}$ and bias term b must satisfy the following conditions

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geqslant 1, \quad i = 1, \ldots, m. \tag{2.3}$$

In addition, weight vector $\mathbf{w}$ must be of the minimal norm in order to maximize the margin between two classes. This way we minimize the chance of misclassification performed on previously unseen data samples. The width of the margin is equal to $\frac{2}{\|\mathbf{w}\|}$. Therefore, the optimization criterion for linear hard margin SVM problem can be defined as the following quadratic optimization problem

$$\arg\min_{\mathbf{w}} \|\mathbf{w}\|^2, \tag{2.4}$$

subject to

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geqslant 1, \quad i = 1, \ldots, m. \tag{2.5}$$

Unfortunately, this idea is not applicable to all classification problems because it does not tolerate misclassified training samples and therefore cannot be used for overlapping classes. In order to overcome this problem Cortes and Vapnik introduced soft margin

SVM [27]. The slack variables ζ_i represent the algebraic distance of a training data point $\mathbf{x}_i$ from the margin. This way they made it possible to solve the SVM training problem for overlapping classes which finally allowed to apply SVM to the broader range of classification problems.

The hard margin constrains (2.3) have been relaxed by allowing some missclassification in the following way

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geqslant 1 - \zeta_i, \quad i = 1, \ldots, m, \tag{2.6}$$

and the optimization criterion (2.4) has been changed in such way that it not only maximizes the classification margin but also minimizes the sum of distances ζ_i of overlapped data points from their margins

$$E = \sum_{i=1}^{m} e\left(\zeta_i\right), \tag{2.7}$$

where $e\left(\zeta_i\right)$ is an error function.

Finally, the soft margin SVM can be defined as follows

$$\arg\min_{\mathbf{w},\zeta} \|\mathbf{w}\|^2 + C \sum_{i=1}^{m} e\left(\zeta_i\right), \tag{2.8}$$

subject to

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geqslant 1 - \zeta_i, \quad i = 1, \ldots, m. \tag{2.9}$$

The parameter C is a predefined constant called penalty parameter. One may try to resolve bias-variance dilemma by proper adjustment of the value of C. For instance, a large value of C forces a maximization of the margin and in the case of non-linear classifier may eventually lead to over-fitting and reduction of generalization capabilities of the model.

However soft margin linear SVM defined in (2.8) is still helpless when dealing with data having very complex topological structure. This problem will be addressed in further sections. Namely, nonlinear SVM will be derived by using the so-called “kernel trick”.

2.1.1 L1 Support Vector Machines

L1 SVM is a special type of soft margin SVM that has linear error function $e(\zeta_i) = \zeta_i$. The optimization problem is defined as

$$\arg\min_{\mathbf{w},\zeta} \|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \zeta_i, \tag{2.10}$$

subject to

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geqslant 1 - \zeta_i, \quad i = 1, \ldots, m, \tag{2.11a}$$

$$\zeta_i \geqslant 0, \quad i = 1, \ldots, m. \tag{2.11b}$$

Equation (2.10) minimizes the sum of distances ζ_i i.e., the classification error, and maximizes the margin between the classes. The penalty parameter C is used to find the trade off between these two tasks.

The constrained optimization problem (2.20) can be solved using the Lagrange multipliers method. The primal Lagrangian is

$$L_p(\mathbf{w}, b, \zeta, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \zeta_i - \sum_{i=1}^{m} \alpha_i \left(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \zeta_i\right) - \sum_{i=1}^{m} \beta_i \zeta_i, \tag{2.12}$$

where

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m, \tag{2.13a}$$

$$\beta_i \geqslant 0, \quad i = 1, \ldots, m, \tag{2.13b}$$

are the Lagrange multipliers. The Karush-Kuhn-Tucker condition are as follows

$$\alpha_i \left(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - \rho + \zeta_i\right) = 0, \quad i = 1, \ldots, m, \tag{2.14a}$$

$$\beta_i \zeta_i = (C - \alpha_i)\zeta_i = 0, \quad i = 1, \ldots, m, \tag{2.14b}$$

By equaling derivatives of $L_p(\mathbf{w}, b, \zeta, \boldsymbol{\alpha}, \boldsymbol{\beta})$ with respect of $\mathbf{w}$, b and ζ_i to 0 the following holds

$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = 0, \tag{2.15}$$

$$\frac{\partial L_p}{\partial b} = \sum_{i=1}^{m} \alpha_i y_i = 0, \tag{2.16}$$

$$\frac{\partial L_p}{\partial \zeta_i} = C - \alpha_i - \beta_i = 0. \tag{2.17}$$

Equations (2.15) and (2.17) can be further simplified to

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \tag{2.18}$$

$$\alpha_i + \beta_i = C, \quad i = 1, \ldots, m. \tag{2.19}$$

Finally, the dual form of (2.10) is as follows

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j\right) - \sum_{i=1}^{m} \alpha_i, \tag{2.20}$$

subject to

$$0 \leqslant \alpha_i \leqslant C, \quad i = 1, \ldots, m, \tag{2.21a}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0. \tag{2.21b}$$

In the matrix notation, (2.20) is equivalent to

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2} \boldsymbol{\alpha}^\intercal \left(YY^\intercal \circ XX^\intercal\right) \boldsymbol{\alpha} - \mathbf{1}^\intercal \boldsymbol{\alpha}, \tag{2.22}$$

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

From the KKT condition (2.14a) one can derive that

$$b = \frac{1}{|U|} \sum_{i:\, \mathbf{x}_i \in U} \left(y_i - \sum_{j:\, \mathbf{x}_j \in S} \alpha_j y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j \right) \right), \tag{2.23}$$

where S is the set of all support vectors and $U \subset S$ is the set of unbounded (i.e. free) support vectors ($\mathbf{x}_i$ satisfying $0 < \alpha_i < C$ and $\zeta_i = 0$). Finally, this leads us to the following decision function

$$d(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i \left(\mathbf{x}_i \cdot \mathbf{x} \right) + b. \tag{2.24}$$

By replacing the scalar product from (2.20), (2.23) and (2.24) by a kernel function, being scalar product in a feature space Φ,

$$k(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_i), \tag{2.25}$$

where $\varphi(x) : \mathcal{X} \to \Phi$ represents the mapping of vector $\mathbf{x}_i$ into the feature space Φ, we obtain nonlinear soft margin L1 SVM. Namely, the optimization criterion (2.20) becomes

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{m} \alpha_i, \tag{2.26}$$

subject to

$$0 \leqslant \alpha_i \leqslant C, \quad i = 1, \ldots, m. \tag{2.27}$$

In matrix notation (2.26) can be expressed as

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2} \boldsymbol{\alpha}^{\intercal} H \boldsymbol{\alpha} - \mathbf{1}^{\intercal} \boldsymbol{\alpha}, \tag{2.28}$$

where $H = \left[y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \right]_{ij}$ is the kernel matrix.

Finally, the decision function for the L1 SVM problem becomes

$$d(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b, \tag{2.29}$$

where bias b is equal to

$$b = \frac{1}{|U|} \sum_{i:\, \mathbf{x}_i \in U} \left(y_i - \sum_{j:\, \mathbf{x}_j \in S} \alpha_j y_j k(\mathbf{x}_i, \mathbf{x}_j) \right). \tag{2.30}$$

If the mapping $\varphi(\mathbf{x})$, that defines the kernel function $k(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)$, is nonlinear then the decision function (2.29) is also nonlinear. If $\varphi(\mathbf{x})$ is linear then the decision function (2.29) is equivalent to (2.24).

2.1.2 L2 Support Vector Machines

If we use error function $e(\zeta_i) = \zeta_i^2$ in the definition of soft margin SVM (2.8) we obtain the optimization problem of L2 SVM

$$\arg\min_{\mathbf{w},\zeta} \|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \zeta_i^2, \tag{2.31}$$

subject to

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geqslant 1 - \zeta_i, \quad i = 1, \ldots, m. \tag{2.32}$$

Here, the meaning of the penalty parameter C is the same as in previously defined L1 SVM – it controls the trade off between the size of the margin and sum of square distances of the training data points from their corresponding margin.

The dual form of (2.31) is as follows

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j + \frac{\delta_{ij}}{C} \right) - \sum_{i=1}^{m} \alpha_i, \tag{2.33}$$

subject to

$$\sum_{i=1}^{m} \alpha_i y_i = 0, \tag{2.34a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m, \tag{2.34b}$$

where δ_{ij} is a Kronecker delta. Furthermore, the criterion (2.33) can be expressed in the matrix notation as

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^\intercal \left(YY^\intercal \circ XX^\intercal + \frac{1}{C}I\right)\boldsymbol{\alpha} - \mathbf{1}^\intercal\boldsymbol{\alpha}, \tag{2.35}$$

where I is the identity matrix and $\circ$ is Hadamard product (element-wise matrix product).

The decision function for L2 SVM is given in (2.2). By substituting $\mathbf{w}$ with

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \tag{2.36}$$

we obtain that

$$d(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i \left(\mathbf{x}_i \cdot \mathbf{x}\right) + b, \tag{2.37}$$

where

$$b = \frac{1}{|S|} \sum_{i=1}^{m} \left(y_i - \sum_{j=1}^{m} \alpha_j y_j \left(\left(\mathbf{x}_i \cdot \mathbf{x}_j\right) + \frac{\delta_{ij}}{C} \right) \right), \tag{2.38}$$

and S is the set of all support vectors.

It is possible to replace the scalar products from (2.33), (2.38) and (2.37) by a kernel function representing a scalar product in some feature space Φ

$$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_i), \tag{2.39}$$

where $\varphi(x) : \mathcal{X} \to \Phi$ is the function mapping vector $\mathbf{x}_i$ into feature space Φ. This change

leads to nonlinear L2 SVM where the optimization criterion (2.33) becomes

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_{i=1}^{m}\alpha_i\alpha_j y_i y_j\left(k\left(\mathbf{x}_i,\mathbf{x}_j\right)+\frac{\delta_{ij}}{C}\right)-\sum_{i=1}^{m}\alpha_i, \tag{2.40}$$

subject to

$$\sum_{i=1}^{m}\alpha_i y_i = 0, \tag{2.41a}$$

$$\alpha_i \geqslant 0, \quad i=1,\ldots,m. \tag{2.41b}$$

In matrix notation (2.40) can be written as

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^{\intercal}\left(H+\frac{1}{C}I\right)\boldsymbol{\alpha}-\mathbf{1}^{\intercal}\boldsymbol{\alpha}, \tag{2.42}$$

where $H=\left[y_i y_j k\left(\mathbf{x}_i,\mathbf{x}_j\right)\right]$ is the kernel matrix. The decision function for the problem (2.40) has the same structure as in L1 SVM, namely

$$d(\mathbf{x})=\sum_{i=1}^{m}\alpha_i y_i k(\mathbf{x}_i,\mathbf{x})+b, \tag{2.43}$$

where bias b is again equal to

$$b=\frac{1}{|S|}\sum_{i=1}^{m}\left(y_i-\sum_{j=1}^{m}\alpha_j y_j\left(k\left(\mathbf{x}_i,\mathbf{x}_j\right)+\frac{\delta_{ij}}{C}\right)\right), \tag{2.44}$$

The detailed description and extensive comparison of L1 and L2 SVM was performed by Abe in [28].

2.1.3 Kernel SVM

Two types of nonlinear SVM classifiers are presented in Sections 2.1.1 and 2.1.2. The nonlinearity was introduced by application of the so called "kernel trick". Namely, all

Name	Kernel function	Properties
$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \left(\mathbf{x}_i^\intercal \mathbf{x}_j\right)$	Linear, dot product	CPD[1]
$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \left(\mathbf{x}_i^\intercal \mathbf{x}_j + 1\right)^d$	Complete polynomial of degree d	PD[2]
$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$	Gaussian RBF	PD[2]
$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \tanh\left(\left(\mathbf{x}_i^\intercal \mathbf{x}_j\right) + b\right)$	Multilayer perceptron	CPD[1]
$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \frac{1}{\sqrt{\|\mathbf{x}_i - \mathbf{x}_j\|^2 + \beta}}$	Inverse multiquadric function	PD[2]

Table 2.1: Most frequently used kernel types.

scalar products were replaced by a function representing dot product in a feature space Φ

$$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_i), \tag{2.45}$$

where $\varphi(x) : \mathcal{X} \to \Phi$ is a function mapping data points from space $\mathcal{X}$ into the feature space Φ.

The most popular kernel functions are presented in Table 2.1. Beside these very frequently used kernels there are more sophisticated ones e.g. kernels designed to work with graphs or images. More examples of kernel functions with explanation how to create them can be found in [29].

This dissertation exploits properties of normalized kernels i.e., kernels that fulfill the following condition

$$k\left(\mathbf{x}, \mathbf{x}\right) = \tau. \tag{2.46}$$

The common property of such kernels is that they map all data points onto multidimensional sphere with center in the origin and radius $\sqrt{\tau}$. An example of such kernel is the Gaussian kernel. Even if a kernel is not normal, it can be normalized by the following

[1] conditionally positive definite

[2] positive definite

operation

$$k'\left(\mathbf{x}_i, \mathbf{x}_j\right) = \frac{k\left(\mathbf{x}_i, \mathbf{x}_j\right)}{\sqrt{k\left(\mathbf{x}_i, \mathbf{x}_i\right) k\left(\mathbf{x}_j, \mathbf{x}_j\right)}}. \tag{2.47}$$

It is obvious that kernel $k'\left(\mathbf{x}_i, \mathbf{x}_j\right)$ is normalized since it fulfills condition (2.46)

$$k'\left(\mathbf{x}, \mathbf{x}\right) = 1. \tag{2.48}$$

2.2 Fundamental Problems of the Computational Geometry

There are several Computational Geometry problems that are especially important for the SVM algorithms. The following sections describe details of Minimal Norm Problem (MNP), Nearest Point Problem (NPP) and Minimal Enclosing Ball (MEB) problem. Their relations to Support Vector Machines are shown in Section 2.3.

2.2.1 Minimal Norm Problem

The Minimal Norm Problem is a problem of finding a point $\mathbf{c}$ closest to the origin that belongs to the convex hull $H(\mathcal{X})$ spanned by points $\mathbf{x}_i \in \mathcal{X}$. It can be mathematically described as

$$\arg\min_{\mathbf{c} \in H(\mathcal{X})} \|\mathbf{c}\|^2, \tag{2.49}$$

where $\mathbf{c}$ is a point belonging to the convex hull $H(\mathcal{X})$ (see Figure 2.1).

Definition The convex hull spanned by the points $\mathbf{x}_i \in \mathcal{X}$ is a set of all convex combinations of points $\mathbf{x}_i \in \mathcal{X}$

$$H(\mathcal{X}) = \left\{ \sum_{i=1}^{m} \alpha_i \mathbf{x}_i \right\}, \tag{2.50}$$

such that $\sum_{i=1}^{m} \alpha_i = 1$ and $\alpha_i \geqslant 0$, for all $i = 1, \ldots, m$.

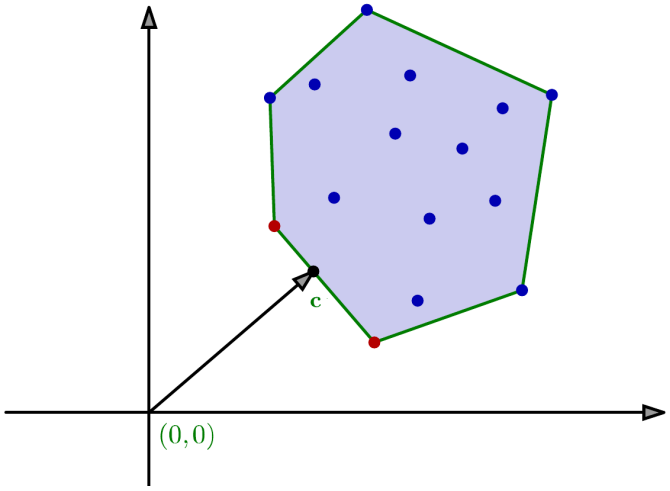

Figure 2.1: Minimal Norm Problem.

In other words, the solution of the problem **c**, which is a point that is closest to the origin and is enclosed by the convex hull $H(\mathcal{X})$, can be expressed as a linear combination of points defining the convex hull

$$\mathbf{c} = \sum_{i=1}^{m} \alpha_i \mathbf{x}_i, \tag{2.51}$$

where α_i are the mixing coefficients that fulfill the following conditions

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m, \tag{2.52}$$

and

$$\sum_{i=1}^{m} \alpha_i = 1. \tag{2.53}$$

This allows us to reformulate the minimization criterion (2.49) as

$$\arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \sum_{i=j}^{m} \alpha_i \alpha_j \left(\mathbf{x}_i \cdot \mathbf{x}_j \right), \tag{2.54}$$

subject to

$$\sum_{i=1}^{m} \alpha_i = 1, \tag{2.55a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m. \tag{2.55b}$$

Therefore, the solution of the MNP problem $\mathbf{c} = \sum_{i=1}^{m} \alpha_i \mathbf{x}_i$ can be obtained by finding mixing coefficients α_i that minimize (2.54) and satisfy (2.52) and (2.53).

The optimization task (2.54) can be expressed in the matrix notation as

$$\arg\min_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^{\intercal} X X^{\intercal} \boldsymbol{\alpha}, \tag{2.56}$$

subject to

$$\mathbf{1}^{\intercal} \boldsymbol{\alpha} = 1, \tag{2.57a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m. \tag{2.57b}$$

2.2.1.1 Generalization to Kernel MNP

The generalization of MNP problem to the kernel-MNP is straightforward. Simply, the scalar product $\mathbf{x}_i \cdot \mathbf{x}_j$ from (2.54) should be replaced by a scalar product in the feature space Φ, defined by mapping $\varphi(x) : \mathcal{X} \to \Phi$. Then, after defining an appropriate kernel function $k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)$, (2.54) can be rewritten as

$$\arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k\left(\mathbf{x}_i, \mathbf{x}_j\right), \tag{2.58}$$

subject to

$$\sum_{i=1}^{m} \alpha_i = 1, \quad i = 1, \ldots, m, \tag{2.59a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m. \tag{2.59b}$$

In a matrix notation (2.58) is as follows

$$\arg\min_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^\intercal \mathbf{K} \boldsymbol{\alpha}, \tag{2.60}$$

subject to

$$\mathbf{1}^\intercal \boldsymbol{\alpha} = 1, \tag{2.61a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m, \tag{2.61b}$$

where $\mathbf{K} = \left[k\left(\mathbf{x}_i, \mathbf{x}_j\right)\right]_{i,j}$ is the kernel matrix.

2.2.2 Nearest Point Problem

The nearest Point Problem can be seen as a generalization of the Minimal Norm Problem. Here, instead of searching for a point of the convex hull closest to the origin, we are searching for two closest points $\mathbf{c}'$ and $\mathbf{c}''$ belonging to two non-overlapping convex hulls $H(\mathcal{X}')$ and $H(\mathcal{X}'')$

$$\arg\min_{\mathbf{c}' \in H(\mathcal{X}'), \mathbf{c}'' \in H(\mathcal{X}'')} \|\mathbf{c}' - \mathbf{c}''\|^2, \tag{2.62}$$

as can be visualized in Figure 2.2. Note that for $\mathcal{X}'' = \{\mathbf{0}\}$, $\mathbf{c}'' = \mathbf{0}$ and (2.62) simplifies to (2.49). Moreover, the NPP problem can be transformed to the MNP problem by using the Minkowski difference, as shown below.

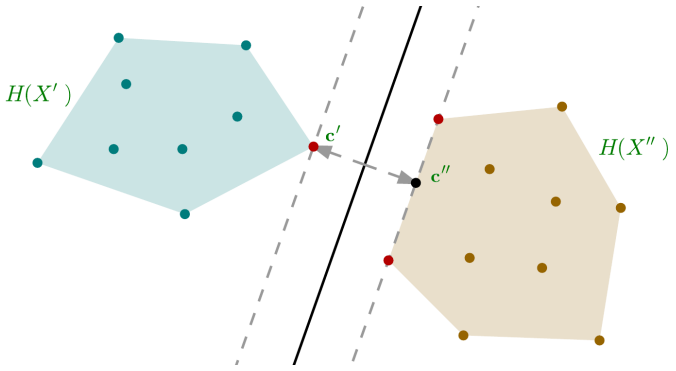

Figure 2.2: Nearest Point Problem.

Definition The Minkowski difference is defined as

$$A \ominus B = \left\{ \mathbf{a}_i - \mathbf{b}_j \mid \mathbf{a} \in A \wedge \mathbf{b} \in B \right\}. \tag{2.63}$$

Finally, (2.62) can be rewritten as

$$\underset{\mathbf{c} \in H(\mathcal{X}') \ominus H(\mathcal{X}'')}{\arg\min} \|\mathbf{c}\|^2, \tag{2.64}$$

where

$$\mathbf{c} = \mathbf{c}' - \mathbf{c}''. \tag{2.65}$$

Since the region $H(\mathcal{X}') \ominus H(\mathcal{X}'')$ is convex, the criterion (2.64) is equivalent to the criterion of the minimal norm problem (2.49). Note that if $H(\mathcal{X}')$ and $H(\mathcal{X}'')$ are overlapping then the NPP problem does not have a unique solution.

2.2.3 Minimal Enclosing Ball Problem

The Minimal Enclosing Ball problem is a problem of finding a smallest sphere with a center $\mathbf{c}$ and a radius R that encloses all points $\mathbf{x}_i \in \mathcal{X}$ (see Figure 2.3). It can be expressed

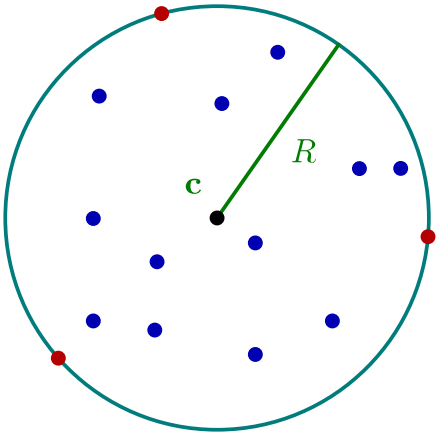

Figure 2.3: Minimal Enclosing Ball.

mathematically as the following minimization problem

$$\arg\min_{R,\mathbf{c}} R^2, \tag{2.66}$$

subject to

$$\|\mathbf{c} - \mathbf{x}_i\|^2 \leqslant R^2, \quad i = 1, \ldots, m. \tag{2.67}$$

The optimization problem defined by (2.66) and (2.67) is a standard quadratic programming problem which can be solved by the Lagrange method. The primal Lagrangian is defined as

$$L_p(\mathbf{c}, R, \boldsymbol{\alpha}) = R^2 - \sum_{i=1}^{m} \alpha_i \left(R^2 - \|\mathbf{c} - \mathbf{x}_i\|^2\right) \tag{2.68}$$

$$= R^2 - \sum_{i=1}^{m} \alpha_i \left(R^2 - \|\mathbf{c}\|^2 + 2\left(\mathbf{x}_i \cdot \mathbf{c}\right) - \|\mathbf{x}_i\|^2\right), \tag{2.69}$$

where α_i are the Lagrange multipliers satisfying the following non-negativity condition:

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m. \tag{2.70}$$

The complementary Karush–Kuhn–Tucker conditions are defined as

$$\alpha_i \left(R^2 - \|\mathbf{c} - \mathbf{x}_i\|^2 \right) = 0, \quad i = 1, \ldots, m. \tag{2.71}$$

By equaling the partial derivatives of L_P with respect to the primal variables $\mathbf{c}$ and R, one can obtain

$$\frac{\partial L_p}{\partial R} = 2R - 2R \sum_{i=1}^{m} \alpha_i = 0, \tag{2.72}$$

and

$$\frac{\partial L_p}{\partial \mathbf{c}} = 2 \sum_{i=1}^{m} \alpha_i \mathbf{c} - 2 \sum_{i=1}^{m} \alpha_i \mathbf{x}_i = 0. \tag{2.73}$$

We can assume that the enclosing ball has a non-zero radius $R > 0$, so we can further simplify (2.72) to

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

and using (2.74) it is possible to write (2.73) as

$$\mathbf{c} = \sum_{i=1}^{m} \alpha_i \mathbf{x}_i. \tag{2.75}$$

Moreover, from (2.71) and (2.74) it follows that the radius of the enclosing ball is equal to

$$R^2 = \sum_{i=1}^{m} \alpha_i \|\mathbf{c} - \mathbf{x}_i\|^2. \tag{2.76}$$

Substituting (2.74) and (2.75) into (2.69) we obtain the dual Lagrangian L_d

$$L_d(\boldsymbol{\alpha}) = -\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \left(\mathbf{x}_i \cdot \mathbf{x}_j \right) + \sum_{i=1}^{m} \alpha_i \|\mathbf{x}_i\|^2. \tag{2.77}$$

Consequently, we can write the MEB problem as

$$\arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \left(\mathbf{x}_i \cdot \mathbf{x}_j \right) - \sum_{i=1}^{m} \alpha_i \|\mathbf{x}_i\|^2, \tag{2.78}$$

subject to

$$\sum_{i=1}^{m} \alpha_i = 1, \tag{2.79a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m. \tag{2.79b}$$

That can be expressed in matrix notation as

$$\arg\min_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^\intercal X X^\intercal \boldsymbol{\alpha} - \boldsymbol{\alpha}^\intercal \operatorname{diag}\left(X X^\intercal\right), \tag{2.80}$$

subject to

$$\mathbf{1}^\intercal \alpha = \mathbf{1}, \tag{2.81a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m, \tag{2.81b}$$

where $\operatorname{diag}\left(X X^\intercal\right) = \left[\|\mathbf{x}_i\|^2\right]_i$ is a vector of Euclidean norms of samples $\mathbf{x}_i$.

2.2.3.1 Generalization to Kernel MEB

In order to generalize the MEB problem into the kernel-MEB problem it is necessary to replace the scalar product $\mathbf{x}_i \cdot \mathbf{x}_j$ by a scalar product in the feature space Φ induced by mapping $\varphi(x) : \mathcal{X} \to \Phi$. Let $k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)$ be the kernel representing the scalar product in the feature space $\Phi(\mathcal{X})$. Now, (2.78) can be written as

$$\arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) - \sum_{i=1}^{m} \alpha_i k\left(\mathbf{x}_i, \mathbf{x}_i\right), \tag{2.82}$$

subject to

$$\sum_{i=1}^{m} \alpha_i = 1, \tag{2.83a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m, \tag{2.83b}$$

and the radius of the enclosing ball can be evaluated as

$$R^2 = -\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) + \sum_{i=1}^{m} \alpha_i k\left(\mathbf{x}_i, \mathbf{x}_i\right). \tag{2.84}$$

In the matrix notation, the criterion (2.82) is given as

$$\arg\min_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^\intercal \mathbf{K} \boldsymbol{\alpha} - \operatorname{diag}(\mathbf{K})^\intercal \boldsymbol{\alpha}, \tag{2.85}$$

where $\mathbf{K} = \left[k\left(\mathbf{x}_i, \mathbf{x}_j\right)\right]_{i,j}$ is the kernel matrix and $\operatorname{diag}(\mathbf{K}) = \left[k\left(\mathbf{x}_i, \mathbf{x}_i\right)\right]_i$ is the diagonal of the matrix $\mathbf{K}$.

Assuming that the kernel k is normalized and $k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \tau$ is constant, the (2.82) can be further simplified to

$$\arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j k\left(\mathbf{x}_i, \mathbf{x}_j\right), \tag{2.86}$$

since the term $\sum_{i=1}^{m} \alpha_i k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \tau$ is also constant and does not affect the optimization criterion. This leads to the conclusion that for normalized kernels the MEB problem is identical to the MNP problem (see the (2.58)).

2.3 Geometric L2 Support Vector Machines

In order to approach the SVM learning problem using geometric methods, Tsang et al. [7] modified the original optimization criterion for L2 SVM classifier (Section 2.3) and defined

it as a minimization of the following cost function

$$\arg\min_{\mathbf{w},b,\zeta,\rho} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{b^2}{2} - \rho + \frac{C}{2}\sum_{i=1}^{m}\zeta_i^2, \tag{2.87}$$

subject to

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geqslant \rho - \zeta_i, \quad i = 1, \ldots, m. \tag{2.88}$$

Compared with the original problem (2.31), the optimized function is extended with the term $\frac{b^2}{2}$ including bias and an additional variable ρ. For the L2 SVM optimization criterion, adding term $\frac{b^2}{2}$ is like adding a feature having all values equal to 1 to the feature space

$$\varphi'(x) = \begin{bmatrix} \varphi(x) \\ 1 \end{bmatrix}. \tag{2.89}$$

Adding ρ to the cost function replaces the constraint $\sum_{i=1}^{m}\alpha_i y_i = 0$ in the original L2 SVM problem (2.33) with $\sum_{i=1}^{m}\alpha_i = 1$. Then, the term $-\sum_{i=1}^{m}\alpha_i$ from the optimization criterion (2.33) can be discarded since it is constant. Very similar modification, which adds a new variable ν into the optimization criterion, was performed by Schölkopf and Smola in their ν-SVM algorithm [30] in order to limit the number of support vectors. Here, an additional variable ρ is used just to facilitate a transformation of L2 SVM into MEB and MNP problems.

Since (2.87) is a constrained optimization problem it can be solved using the Lagrange multipliers method. We form the primal Lagrangian as

$$L_p(\mathbf{w}, b, \rho, \zeta, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2}b^2 - \rho + \frac{C}{2}\sum_{i=1}^{m}\zeta_i^2 - \sum_{i=1}^{m}\alpha_i\left(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - \rho + \zeta_i\right), \tag{2.90}$$

where

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m, \tag{2.91}$$

and the KKT condition are as follows

$$\alpha_i \left(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - \rho + \zeta_i \right) = 0, \quad i = 1, \ldots, m. \tag{2.92}$$

Equating derivatives of $L_p(\mathbf{w}, b, \rho, \zeta)$ with respect of $\mathbf{w}$, b, ρ and ζ_i to 0 we obtain

$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = 0, \tag{2.93}$$

$$\frac{\partial L_p}{\partial b} = b - \sum_{i=1}^{m} \alpha_i y_i = 0, \tag{2.94}$$

$$\frac{\partial L_p}{\partial \rho} = 1 - \sum_{i=1}^{m} \alpha_i = 0, \tag{2.95}$$

and

$$\frac{\partial L_p}{\partial \zeta_i} = C\zeta_i - \alpha_i = 0. \tag{2.96}$$

Equations (2.93), (2.94), (2.95) and (2.96) can be further simplified to

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \tag{2.97}$$

$$b = \sum_{i=1}^{m} \alpha_i y_i, \tag{2.98}$$

$$\sum_{i=1}^{m} \alpha_i = 1, \tag{2.99}$$

and

$$\zeta_i = \frac{\alpha_i}{C}. \tag{2.100}$$

Moreover, from (2.92) we can conclude that:

$$\sum_{i=1}^{m} \alpha_i \left(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - \rho + \zeta_i \right) = 0, \tag{2.101}$$

which using (2.97), (2.98), (2.99) and (2.100) can be further simplified to

$$\rho = \|\mathbf{w}\|^2 + b^2 + C \sum_{i=1}^{m} \zeta_i^2. \tag{2.102}$$

or alternatively, it can be expressed as a function in a dual-space:

$$\rho = \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j \right) + \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j + \frac{1}{C} \sum_{i=1}^{m} \alpha_i^2. \tag{2.103}$$

Using Equations (2.97), (2.98) and (2.99) we rewrite Lagrange criterion (2.90) as

$$L_d(\boldsymbol{\alpha}) = -\frac{1}{2} \left(\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j \right) + \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j + \frac{1}{C} \sum_{i=1}^{m} \alpha_i^2 \right). \tag{2.104}$$

This allows to rewrite the original optimization problem from (2.87) as

$$\arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j \right) + \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j + \frac{1}{C} \sum_{i=1}^{m} \alpha_i^2 \tag{2.105}$$

subject to

$$\sum_{i=1}^{m} \alpha_i = 1, \tag{2.106a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m. \tag{2.106b}$$

Equation (2.105), dubbed here a modified L2 SVM, can be written in a matrix notation as

$$\arg\min_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^\intercal \left(YY^\intercal \circ XX^\intercal \right) \boldsymbol{\alpha} + \boldsymbol{\alpha}^\intercal YY^\intercal \boldsymbol{\alpha} + \frac{1}{C} \boldsymbol{\alpha}^\intercal \boldsymbol{\alpha}. \tag{2.107}$$

The decision function of the modified L2 SVM problem, defined in (2.87), is not changed

$$d(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b. \tag{2.108}$$

After substituting $\mathbf{w}$ from (2.97) and b from (2.98), it can be rewritten as

$$d(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i \left(\mathbf{x}_i \cdot \mathbf{x}\right) + \sum_{i=1}^{m} \alpha_i y_i. \tag{2.109}$$

The decision functions of the original and the modified L2 SVM differ in expression of the bias term b (please compare (2.2) and (2.108)). The vector $\mathbf{w}$ is the same for both methods.

2.3.1 Nonlinear Geometric SVM

The "kernel trick" [26] allows us to replace the scalar product $\mathbf{x}_i \cdot \mathbf{x}_j$ by the kernel $k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)$ representing the scalar product in the feature space Φ defined by mapping $\varphi(x) : \mathcal{X} \to \Phi$. The modified L2 SVM problem (2.105) takes the form

$$\arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) + \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j + \frac{1}{C} \sum_{i=1}^{m} \alpha_i^2 \tag{2.110}$$

subject to

$$\sum_{i=1}^{m} \alpha_i = 1, \tag{2.111a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m. \tag{2.111b}$$

Equation (2.110) in the matrix notation is given as

$$\arg\min_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^{\intercal} \mathbf{H} \boldsymbol{\alpha} + \boldsymbol{\alpha}^{\intercal} Y Y^{\intercal} \boldsymbol{\alpha} + \frac{1}{C} \boldsymbol{\alpha}^{\intercal} \boldsymbol{\alpha}, \tag{2.112}$$

where $\mathbf{H} = \left[y_i y_j k\left(\mathbf{x}_i, \mathbf{x}_j\right)\right]_{i,j}$. The decision function (2.109) becomes

$$d(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i k\left(\mathbf{x}_i, \mathbf{x}\right) + \sum_{i=1}^{m} \alpha_i y_i. \tag{2.113}$$

2.3.2 L2 SVM as a Geometric Problem

First, we want to introduce a new kernel $\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_j\right)$ and to show that this particular kernel is a Mercer kernel. Then by using $\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_j\right)$ we show the equivalence of the modified L2 SVM problem (2.110) and two geometric tasks introduced earlier – a minimal enclosing ball problem and a minimal norm problem. In order to do that let us introduce feature space $\tilde{\Phi}$ and the kernel function $\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_j\right) : \mathcal{X} \to \tilde{\Phi}$ that defines dot product between samples $\mathbf{x}_i$ in this space

$$\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_j\right) = y_i y_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) + y_i y_j + \frac{\delta_{ij}}{C}, \tag{2.114}$$

where $k\left(\mathbf{x}_i, \mathbf{x}_j\right)$ is the original kernel used in SVM problem and δ_{ij} is Kronecker delta. This kernel function seamlessly encodes data labels and the original kernel $k\left(\mathbf{x}_i, \mathbf{x}_j\right)$ in such way that no information about samples $\mathbf{x}_i$ is lost.

If $k\left(\mathbf{x}_i, \mathbf{x}_j\right)$ satisfies Mercer's condition

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) \geqslant 0, \tag{2.115}$$

than since

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) \geqslant 0, \tag{2.116}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \geqslant 0, \tag{2.117}$$

and

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \frac{\delta_{ij}}{C} = \sum_{i=1}^{m} \frac{\alpha_i^2}{C} \geqslant 0, \tag{2.118}$$

it is true that $\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_j\right)$ also satisfies Mercer's condition

$$\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_j\right) \geqslant 0. \tag{2.119}$$

2.3.2.1 L2 SVM and Minimal Norm Problem

Equation (2.114) allows to rewrite the problem (2.110) as

$$\arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_j\right), \tag{2.120}$$

subject to

$$\sum_{i=1}^{m}\alpha_i = 1, \tag{2.121a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m. \tag{2.121b}$$

It is clear that the problem stated by the optimization criterion (2.120) is identical to the MNP problem (2.58). In other words, solutions of (2.120) which is basically a minimal norm problem in the feature $\tilde{\Phi}$ and the extended L2 SVM problem are the same and in order to find the decision function (2.113) of the modified L2 SVM (2.110) one can resolve the optimization problem (2.120).

2.3.2.2 L2 SVM and Minimal Enclosing Ball Problem

Minimization problem (2.105) can be also treated as a MEB problem if $\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_i\right)$ is constant for $i = 1, \ldots, m$. If this condition is fulfilled, then from (2.99) it follows that $\sum_{i=1}^{m}\alpha_i\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_i\right)$ is constant as well. In such case, it is possible to subtract $\sum_{i=1}^{m}\alpha_i\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_i\right)$ from the criterion (2.105) without affecting the solution. The resulting minimization task is the same as in

(2.82) which is the optimization problem for the minimal enclosing ball, namely

$$\arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \tilde{k}\left(\mathbf{x}_i, \mathbf{x}_j\right) - \sum_{i=1}^{m} \alpha_i \tilde{k}\left(\mathbf{x}_i, \mathbf{x}_i\right), \tag{2.122}$$

subject to

$$\sum_{i=1}^{m} \alpha_i = 1, \tag{2.123a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m. \tag{2.123b}$$

It is easy to prove that, when $k\left(\mathbf{x}_i, \mathbf{x}_i\right) = \tau$ is constant for all $i = 1, \ldots, m$, then $\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_i\right)$ is also constant

$$\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_i\right) = \tau + 1 + \frac{1}{C}. \tag{2.124}$$

This means that for all normalized kernels, such as Gaussian kernel, it is possible to treat the L2 SVM problem from (2.87) as a minimal enclosing ball problem. Finally, the solution obtained by solving MEB problem in the feature space $\tilde{\Phi}$ and the solution of the modified L2 SVM problem are equal. This means that the decision function (2.113) of the modified L2 SVM (2.110) can be found by finding the minimal enclosing ball (via solving minimization task (2.122)).

2.3.3 Solving L2 SVM based on Minimal Enclosing Ball Approach

This section shortly introduces two popular algorithms proposed by Tsang et al. and designed to solve L2 SVM as a MEB problem. These algorithms are called Core Vector Machines and Ball Vector Machines.

Figure 2.4 shows the hierarchy of the SVM algorithms presented in the following sections. Geometric algorithms designed by Tsang (CVM and BVM) are marked in red while methods introduced in this dissertation (SphereSVM and MNSVM) are marked in

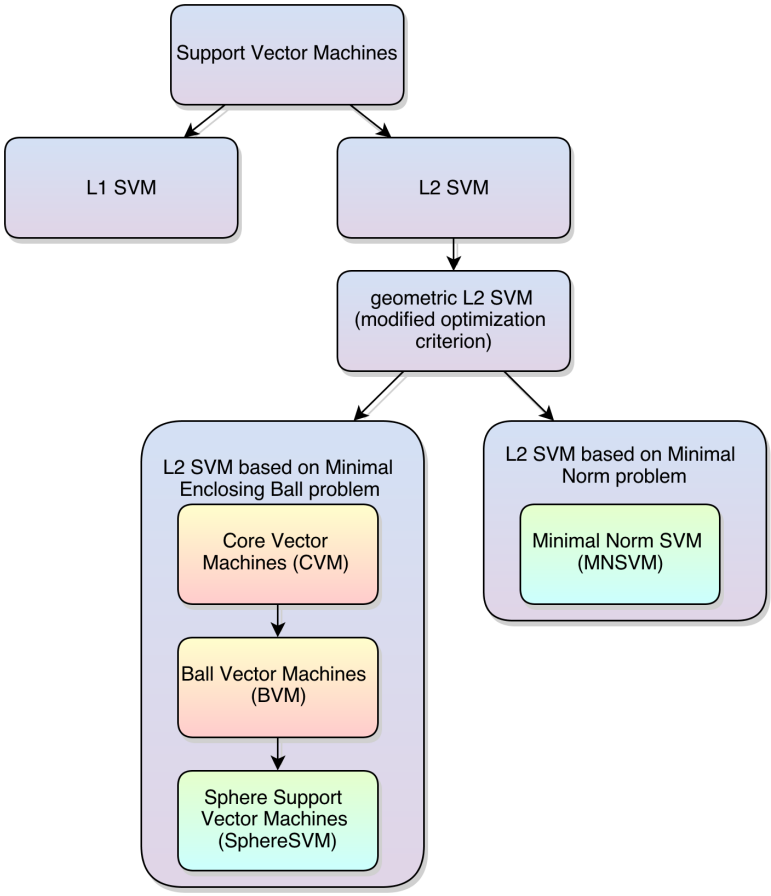

Figure 2.4: Hierarchy of the SVM training algorithms presented in the dissertation.

green.

We use the following notation to represent vectors $\mathbf{x}_i$ in the extended feature space $\tilde{\Phi}$ defined by the kernel $\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_j\right)$

$$\tilde{\mathbf{x}}_i = \tilde{\varphi}(\mathbf{x}_i), \tag{2.125}$$

where $\tilde{\varphi}$ is an unknown mapping function that satisfies

$$\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \tilde{\varphi}(\mathbf{x}_i) \cdot \tilde{\varphi}(\mathbf{x}_j). \tag{2.126}$$

This way we can represent kernel evaluations $\tilde{k}\left(\mathbf{x}_i, \mathbf{x}_j\right)$ as dot products $\tilde{\mathbf{x}}_i \cdot \tilde{\mathbf{x}}_j$ without loosing generality. Moreover, this notation conceals the complexity of the extended feature space $\tilde{\Phi}$ and eventually allows us to introduce CVM and BVM as strictly geometric methods.

2.3.3.1 Core Vector Machines

Tsang et al. [7] introduced an approach to SVM training called Core Vector Machines. There are two key concepts behind this algorithm. First, authors transformed the original L2-SVM problem into Minimal Enclosing Ball Problem described in section 2.2.3. Second, they applied the coreset approach [17, 31, 32, 33] in order to speedup enclosing ball calculation.

The Coreset approach The common feature of the coreset algorithms is avoidance of processing entire dataset $\mathcal{X}$ and focus on a smaller subset of this dataset $S \subset \mathcal{X}$ called the coreset.

Definition The coreset $S(\mathcal{X})$ is a subset of the original dataset $\mathcal{X}$ having the following property – the solution of some problem obtained using a coreset $S \subset \mathcal{X}$ is identical to the solution[1] that would be obtained using the entire dataset $\mathcal{X}$.

To illustrate the definition above, the set of support vectors is a coreset of the original dataset $\mathcal{X}$, since the solution obtained by using this set would be identical to the solution found for any other superset of $\mathcal{X}$.

The main advantage of the coreset approach is its time performance. Since $|S(\mathcal{X})| < |\mathcal{X}|$, the algorithm being executed for the data belonging to the coreset will yield the result faster than the algorithm using entire training dataset. Usually, in the real world applications the coreset is just a small portion of the original dataset $|S(\mathcal{X})| \ll |\mathcal{X}|$ so the speed improvement may be significant, especially when the computational complexity of the algorithm is large. The coreset approach was showed to be applicable for SVM training on large datasets [7, 16].

Algorithm outline The Algorithm 1 contains pseudo code of the CVM algorithm. Note that the CVM algorithm solves the modified L2 SVM problem presented in section 2.3.

[1] or approximates the true solution accurately enough

Algorithm 1 Core Vector Machines Algorithm

Require: $\varepsilon \in [0, 1)$ {the parameter of the stopping criterion}
Ensure: $\mathbf{c} = \sum_{i=1}^{m} \alpha_i \tilde{\mathbf{x}}_i$ {the approximation of the MEB center}
1: $S \leftarrow \{\tilde{\mathbf{x}}_0\}$
2: $c \leftarrow \tilde{\mathbf{x}}_0$
3: $R \leftarrow 0$
4: **while** $\exists i : \|\mathbf{c} - \tilde{\mathbf{x}}_i\| > (1 + \varepsilon)R$ **do**
5: $v \leftarrow \arg\max_i \|\mathbf{c} - \tilde{\mathbf{x}}_i\|$
6: $S \leftarrow S \cup \{\tilde{\mathbf{x}}_v\}$
7: $\mathbf{c}, R \leftarrow \text{MEB}(S)$
8: **end while**

As it was shown earlier, the optimization problem stated in (2.87) can be transformed into corresponding MEB problem. Therefore CVM algorithm finds the L2 SVM model for (2.87) as a solution to the equivalent minimal enclosing ball problem.

The procedure starts from a small coreset S that is extended in each iteration by adding samples violating some predefined conditions. After each coreset modification new enclosing ball is calculated. A third-party QP solver can be used to solve the MEB problem – in the original CVM implementation the SMO algorithm from LIBSVM package [4] was used.

The steps of the Core Vectorm Machine algorithm are visualized on Figure 2.5.

Initialization The coreset may be initialized with a random sample $S = \{\tilde{\mathbf{x}}_0\}$ as in Badoiu and Clarkson work [34]. However, the original algorithm uses more complex initialization procedure introduced by Kumar [31] – first, a random vector $\mathbf{x}_r$ is selected, then two vectors are chosen

$$\tilde{\mathbf{x}}_a = \arg\max_{\tilde{\mathbf{x}}_i} \|\tilde{\mathbf{x}}_r - \tilde{\mathbf{x}}_i\|, \tag{2.127}$$

and

$$\tilde{\mathbf{x}}_b = \arg\max_{\tilde{\mathbf{x}}_i} \|\tilde{\mathbf{x}}_a - \tilde{\mathbf{x}}_i\|. \tag{2.128}$$

Finally, the initial coreset is initialized to $S = \{\tilde{\mathbf{x}}_a, \tilde{\mathbf{x}}_b\}$ and the radius R is set to $R = \frac{\|\tilde{\mathbf{x}}_a - \tilde{\mathbf{x}}_b\|}{2}$.

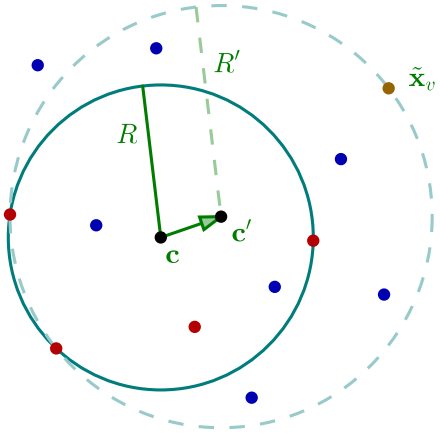

Figure 2.5: A single step of the Core Vector Machines algorithm – red points represent samples $\tilde{\mathbf{x}}_i$ belonging to the coreset S, solid line represents solution (minimal enclosing ball having center $\mathbf{c}$ and radius R) obtained for the current core set, dashed line shows the solution that will be obtained after adding violating vector to the core set (new enclosing ball with center $\mathbf{c}'$ and radius R').

Finding a Violator At the beginning of each iteration a vector $\tilde{\mathbf{x}}_v$ being the worst violator of the stopping criterion is found. Briefly speaking, it is selected by finding a vector that lays farthermost from the current center $\mathbf{c}$

$$\mathbf{x}_v = \arg\max_{\tilde{\mathbf{x}}_i \in \tilde{X}} \|\mathbf{c} - \tilde{\mathbf{x}}_i\|. \tag{2.129}$$

The algorithm continues until there are no vectors being farther than $(1 + \varepsilon)R$ from the center of the minimal enclosing ball surrounding core vectors. ε is a parameter of the algorithm. The smaller ε is, the more accurate the solution is (but the time required for finding MEB increases as well). After the algorithm stops, all data points are within the ball with center $\mathbf{c}$ and radius $(1 + \varepsilon)R$ – this ball is called ε-approximation of the minimal enclosing ball.

Update step In each iteration, after the violator $\tilde{\mathbf{x}}_v$ is found, the entire weight vector $\boldsymbol{\alpha}$ is recalculated and the new center $\mathbf{c}'$ of the minimal enclosing ball, surrounding vectors $\tilde{\mathbf{x}}_i \in S$, is found. An external QP solver is used in order to find the enclosing sphere.

Properties of the Coreset An important property of this algorithm is that the enclosing ball is spanned only by the elements from the coreset and all the vectors belonging to the coreset become the support vectors. Moreover, the following condition holds – if $\alpha_i > 0$ then $\tilde{\mathbf{x}}_i \in S$. It is possible that S contains a vector $\tilde{\mathbf{x}}_i$ corresponding to $\alpha_i = 0$. Such vector could have been inserted into the coreset at some point of the procedure execution but it does not participate in forming of the minimal enclosing ball. This is actually one of the drawbacks of the CVM algorithm because the coreset is not optimal. Namely, it is possible that there exists a smaller set $S' \subset S$ that corresponds to the same solution.

Convergence It was shown in [7] that CVM algorithm converges after at most $\frac{2}{\varepsilon}$ iterations and that its computational complexity to $O\left(\frac{m}{\varepsilon^2} + \frac{1}{\varepsilon^4}\right)$. If the probabilistic speedup technique [34] is used then the time complexity is equal to $O\left(\frac{1}{\varepsilon^8}\right)$ and it is not dependent upon the size of the dataset m.

2.3.3.2 Ball Vector Machines

Tsang et al. [9] improved Core Vector Machines by replacing the complex QP solver, used in minimal enclosing ball calculation, by much simpler iterative algorithm. In each iteration, instead of launching QP solver, only one update to the ball center is performed.

Algorithm 2 Ball Vector Machines Algorithm

Require: $\varepsilon \in [0, 1)$ {the parameter of the stopping criterion}
Ensure: $\mathbf{c} = \sum_{i=1}^{m} \alpha_i \tilde{\mathbf{x}}_i$ {the approximation of the MEB center}

1: $\boldsymbol{\alpha} \leftarrow \mathbf{0}, \alpha_0 \leftarrow 1$
2: $\hat{R} \leftarrow \sqrt{\tau + 1 + \frac{1}{C}}$
3: **while** $\exists i : \|\mathbf{c} - \tilde{\mathbf{x}}_i\| > (1 + \varepsilon)\hat{R}$ **do**
4: $\quad v \leftarrow \arg\max_i \|\mathbf{c} - \tilde{\mathbf{x}}_i\|$
5: $\quad \beta \leftarrow 1 - \frac{R}{\|\mathbf{c} - \mathbf{x}_v\|}$
6: $\quad \boldsymbol{\alpha} \leftarrow (1 - \beta)\boldsymbol{\alpha}$
7: $\quad \alpha_v \leftarrow \alpha_v + \beta$
8: **end while**

Algorithm outline A simplified pseudo code of the algorithm is presented in Algorithm 2 (this listing does not contain probabilistic speedup [34] and multi-scale enclosing ball approximation techniques [9]).

Initialization First, the vector α representing the center of the ball from (2.75) is initialized such that all α_i coefficients are equal to 0 except for a randomly chosen one, whose value is set to 1 (see the initialization procedure for the CVM algorithm). For simplicity, in Algorithm 2, vector $\tilde{\mathbf{x}}_0$ was chosen to initialize the coreset and its weight α_0 was set to 1.

The radius $\hat{R}$ of the enclosing ball is estimated as

$$\hat{R} = \sqrt{\tau + 1 + \frac{1}{C}}, \tag{2.130}$$

where $\tau = k(\mathbf{x}_i, \mathbf{x}_i)$ is the square norm in the original feature space Φ. The estimated radius $\hat{R}$ is in fact an upper bound of the true radius. Fortunately, this estimation is very accurate when either the number of data is large or the feature space is highly dimensional.

Finding a Violator In each iteration, a point $\tilde{\mathbf{x}}_v$, called violating vector and being located outside of the enclosing ball, is selected. The algorithm continues until requirements of the stopping criterion are met (i.e. it is not possible to find a vector located outside of the ball).

Update step The center $\mathbf{c}$ is shifted towards the violator $\tilde{\mathbf{x}}_v$ in such a way that the violating point is laying on the surface of the new ball and the following equation holds

$$\|\mathbf{c}' - \tilde{\mathbf{x}}_v\| = R. \tag{2.131}$$

The update of the center of the ball is performed along the line connecting the center $\mathbf{c}$ and the violator $\tilde{\mathbf{x}}_v$

$$\mathbf{c}' = (1 - \beta)\mathbf{c} + \beta\tilde{\mathbf{x}}_v, \tag{2.132}$$

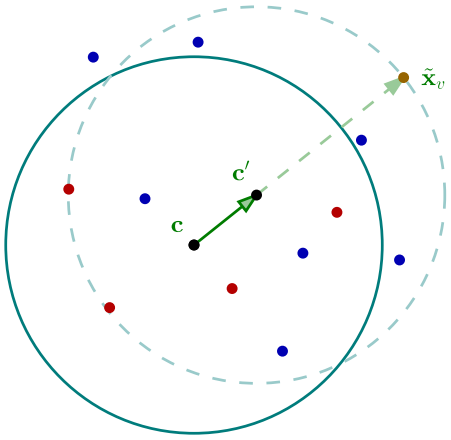

Figure 2.6: A single step of the Ball Vector Machines algorithm – red points represent samples $\tilde{\mathbf{x}}_i$ belonging to the core set S, solid line represents current solution (the minimal enclosing ball), dashed line shows the solution that will be obtained after shifting the current solution towards the violator.

where β is equal to

$$\beta = 1 - \frac{R}{\|\mathbf{c} - \tilde{\mathbf{x}}_v\|}. \tag{2.133}$$

The visualization of the algorithm's steps is shown in Figure 2.6.

Convergence It was proved in [9] that the BVM algorithm terminates in at most $\frac{1}{\varepsilon^2}$ iterations and its computational complexity is $O\left(\frac{1}{\varepsilon^4}\right)$. Similarly as in the case of CVM algorithm, probabilistic speedup technique made the complexity independent of the size of the dataset m.

2.4 Geometric L1 Support Vector Machines

2.4.1 Soft Minimal Enclosing Ball Problem

Let us define the soft-MEB problem as a minimization of

$$\arg\min_{R,c,\mathbf{d}} R^2 + \frac{1}{mv}\sum_{i=1}^{m} d_i^2, \tag{2.134}$$

subject to

$$\|\mathbf{c} - \mathbf{x}_i\|^2 \leqslant R^2 + d_i^2, \tag{2.135}$$

for all $i = 1, \ldots, m$. Namely, we try to minimize the radius of the enclosing ball simultaneously allowing some violators.

It can be proved that the solution of the above problem is equivalent to the solution of the following modified ν-SVM problem

$$\arg\min_{\mathbf{w},b,\zeta,\rho} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{b^2}{2} - \frac{\rho}{m\nu} + \frac{C}{2}\sum_{i=1}^{m}\zeta_i, \tag{2.136}$$

subject to

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geqslant \rho - \zeta_i, \tag{2.137}$$

for all $i = 1, \ldots, m$. Unfortunately, no efficient algorithm capable of solving the soft-MEB problem is known at this point.

Chapter 3

Sphere Support Vector Machines

3.1 Relation to Ball Vector Machines

The SphereSVM algorithm, proposed in this work, is a novel reforumlation of the BVM approach. Therefore, some parts of both algorithms are similar. For instance, the initialization procedure, the way the violating vectors are found and the stopping criterion are the same. However, there are important differences, the main one being the way how the updates of the center are performed. Unlike in BVM, the focus of SphereSVM is directed towards elimination of support vectors being inside of the enclosing ball rather than finding outlying data samples. This approach leads to fulfillment of the KKT conditions and therefore is a correct approach in obtaining correct solution. It applies the ideas introduced in the MDM algorithm, by Michel et al. [23], as a solution to Nearest Point Problem (NPP). Here, we adopted the MDM approach into solving the MEB problem.

The simplified pseudo code of the SphereSVM algorithm is presented in the Algorithm 3. There are two main differences between the pseudo code we present and the actual implementation that was used in our experiments. First, we used kernel cache in order not to repeat unnecessary kernel computations. Second, our implementation contains additional step tuning the values of α vector (in each iteration another update to the

Algorithm 3 SphereSVM Algorithm

Require: $\varepsilon \in [0, 1)$ {the parameter of the stopping criterion}
Ensure: $\mathbf{c} = \sum_{i=1}^{m} \alpha_i \tilde{\mathbf{x}}_i$ {the approximation of the MEB center}
1: $\boldsymbol{\alpha} \leftarrow \mathbf{0}, \alpha_0 \leftarrow 1$
2: $\hat{R} \leftarrow \sqrt{\tau + 1 + \frac{1}{C}}$
3: **while** $\exists i : \|\mathbf{c} - \tilde{\mathbf{x}}_i\| > (1 + \varepsilon)\hat{R}$ **do**
4: $v \leftarrow \arg\max_i \|\mathbf{c} - \tilde{\mathbf{x}}_i\|$
5: $u \leftarrow \arg\min_{i:\alpha_i > 0} \|\mathbf{c} - \tilde{\mathbf{x}}_i\|$
6: $\rho = \frac{(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u) \cdot (\tilde{\mathbf{x}}_v - \mathbf{c})}{\|\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u\|^2}$
7: $\hat{\beta} \leftarrow \rho - \sqrt{\rho^2 - \frac{\|\tilde{\mathbf{x}}_v - \mathbf{c}\|^2 - \hat{R}^2}{\|\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u\|^2}}$
8: $\beta \leftarrow \min\left\{\hat{\beta}, \alpha_u\right\}$
9: $\alpha_v \leftarrow \alpha_v + \beta$
10: $\alpha_u \leftarrow \alpha_u - \beta$
11: **end while**

vector $\boldsymbol{\alpha}$ is performed with such difference that the violator $\tilde{\mathbf{x}}_v$ is searched among support vectors only).

3.2 Steps of the Algorithm

3.2.1 Initialization

During the initialization part of the algorithm (lines 1 to 2) a random support vector is chosen (here, the support vector with index 0) and its weight is initialized to 1. Then, the radius of the enclosing ball is estimated as

$$\hat{R} = \sqrt{\tau + 1 + \frac{1}{C}}, \tag{3.1}$$

where $\tau = k(\mathbf{x}_i, \mathbf{x}_i)$ is the square norm of the vectors $\mathbf{x}_i$ in the feature space Φ inducted by kernel $k\left(\mathbf{x}_i, \mathbf{x}_j\right)$. The algorithm requires that the kernel $k\left(\mathbf{x}_i, \mathbf{x}_j\right)$ is normalized and maps all data points $\mathbf{x}_i$ onto a sphere (having radius $\sqrt{\tau}$).

It was shown in [9] that $\hat{R} \geqslant R$ and that when the size of the dataset and the dimen-

sionality of the feature space are large then the difference between R and $\hat{R}$ is negligible.

3.2.2 Selection of Violating Vectors

In the case of the BVM algorithm all weights α_i corresponding to vectors $\tilde{\mathbf{x}}_i$ belonging to the coreset are modified in each updating step. SphereSVM algorithm, proposed here, updates only two weights α_v and α_u. The first weight α_v corresponds to the vector that is farthermost from the ball center while the other weight α_u belongs to the support vector closest to the center. According to the following KKT conditions of the MEB problem

$$\alpha_i \left(\|\mathbf{c} - \tilde{\mathbf{x}}_i\|^2 - R^2 \right) = 0, \tag{3.2}$$

if the condition $\alpha_i \neq 0$ holds then $\tilde{\mathbf{x}}_i$ lies on the boundary of the minimal enclosing ball. In other words, the vectors inside the ball are not support vectors and do not affect the solution. This observation leads to the conclusion that there are two types of violators – the vectors laying outside of the enclosing ball and the vectors with nonzero weights inside the ball. The SphereSVM algorithm aims at eliminating support vectors from inside the ball.

Similarly as in MDM algorithm, in each iteration two violating vectors are selected. First, a vector $\tilde{\mathbf{x}}_v$, whose distance from the center of the ball $\mathbf{c}$ is greater than $(1 + \varepsilon)\hat{R}$, is chosen. If no outlier satisfying condition $\|\mathbf{c} - \tilde{\mathbf{x}}_i\| > (1 + \varepsilon)\hat{R}$ is found, then the algorithm stops. Finally, after violator $\tilde{\mathbf{x}}_v$ is selected, searching for another violator begins. The algorithm finds a support vector $\tilde{\mathbf{x}}_u$ that violates the KKT conditions (3.2) the most. In other words, the algorithm is searching for a vector $\tilde{\mathbf{x}}_u$ that satisfies $\alpha_u > 0$ and lies closest to the center of the ball.

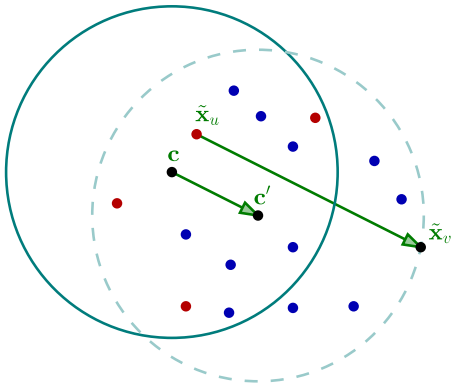

Figure 3.1: One step of the SphereSVM algorithm. The center $\mathbf{c}$ is being shifted parallel to the vector $\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u$ to the new position $\mathbf{c}'$. After that, vector $\tilde{\mathbf{x}}_v$ becomes the support vector. Previously estimated radius $\hat{R}$ of the enclosing ball does not change.

3.2.3 Update Procedure

After the two violating vectors are selected, an update to the center of the ball is performed. Briefly speaking, the center of the ball is shifted parallel to the line connecting the two violating vectors

$$\mathbf{c}' = \mathbf{c} + \beta(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u), \tag{3.3}$$

as shown in Figure 3.1.

The coefficient β is selected in such a way that the new sphere centered at $\mathbf{c}'$ is touching the violator $\tilde{\mathbf{x}}_v$ ($\tilde{\mathbf{x}}_v$ must be laying on the boundary of the new enclosing ball). Specifically, the following condition is to be satisfied

$$\|\mathbf{c}' - \tilde{\mathbf{x}}_v\| = \hat{R}. \tag{3.4}$$

Substituting (3.3) into (3.4) we obtain that

$$\|\mathbf{c} + \beta(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u) - \tilde{\mathbf{x}}_v\|^2 = \hat{R}^2, \tag{3.5}$$

which can be reduced to the following

$$\hat{\beta} = \rho - \sqrt{\rho^2 - \frac{||\tilde{\mathbf{x}}_v - \mathbf{c}||^2 - \hat{R}^2}{||\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u||^2}}, \tag{3.6}$$

where ρ is

$$\rho = \frac{(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u) \cdot (\tilde{\mathbf{x}}_v - \mathbf{c})}{||\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u||^2}. \tag{3.7}$$

In the dual space, (3.3) is equivalent to the increase of α_v by β and the decrease of α_u also by β (lines 9 and 10 of Algorithm 3). It is important to keep all the conditions arising from the Lagrange multiplier method satisfied. In particular, the non-negativity condition of the α_i weights must be fulfilled. Therefore, $\beta \leqslant 1 - \alpha_v$ and $\beta \leqslant \alpha_u$ must hold. The first of these requirements is always fulfilled. However, one must assure non-negativity of all α_i. For this reason β coefficient must be limited from above by the weight α_u

$$\beta = \min\left\{\hat{\beta}, \alpha_u\right\}. \tag{3.8}$$

Having the value β, it is possible to update the center of the ball and resume the algorithm by checking the stopping criterion and by searching for other violators.

3.3 Convergence and Computational Complexity

The goal of SphereSVM is to obtain a ε-approximation of the MEB that satisfies $||\mathbf{c} - \tilde{\mathbf{x}}_i|| \leqslant (1 + \varepsilon)\hat{R}$ for all vectors $\tilde{\mathbf{x}}_i$.

From the KKT conditions (3.2), we know that for $\boldsymbol{\alpha}$, being the solution of the problem stated in (2.66) and (2.67), the following inequality holds

$$\sum_{i=1}^{m} \alpha_i ||\mathbf{c} - \tilde{\mathbf{x}}_i||^2 = R^2 \leqslant \hat{R}^2. \tag{3.9}$$

Moreover, this property holds for all possible vectors $\boldsymbol{\alpha}$ having $\alpha_i \geqslant 0$ and $\sum_{i=1}^{m} \alpha_i = 1$

$$\sum_{i=1}^{m} \alpha_i \|\mathbf{c} - \tilde{\mathbf{x}}_i\|^2 = \hat{R}^2 - \|\mathbf{c}\|^2 \leqslant \hat{R}^2. \tag{3.10}$$

The initialization procedure of the algorithm ensures that $\sum_{i=1}^{m} \alpha_i \|\mathbf{c} - \tilde{\mathbf{x}}_i\|^2 = 0$. In each iteration, the center update expressed in (3.3) changes the value of $\sum_{i=1}^{m} \alpha_i \|\mathbf{c} - \tilde{\mathbf{x}}_i\|^2$ by

$$\sum_{i=1}^{m} \alpha_i' \|\mathbf{c}' - \tilde{\mathbf{x}}_i\|^2 - \sum_{i=1}^{m} \alpha_i \|\mathbf{c} - \tilde{\mathbf{x}}_i\|^2 = -2\beta(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u) \cdot \mathbf{c} - \beta^2 \|\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u\|^2. \tag{3.11}$$

Now, in order to prove the convergence of the algorithm, it is sufficient to show that this change increases the sum $\sum_{i=1}^{m} \alpha_i \|\mathbf{c} - \tilde{\mathbf{x}}_i\|^2$ by a value greater than some positive constant.

Let us assume that the ratio of the number of updates where $\hat{\beta} > \alpha_u$ (clipped updates) to the number of updates having $\beta = \min\left\{\hat{\beta}, \alpha_u\right\} = \hat{\beta}$ (full updates) is limited by a constant. In other words, we postulate that the number of clipped updates (updates limited by the value of weight α_u) is not significantly larger than the number of full updates. This hypothesis is more than feasible. The results presented in figures 7.4 and 7.8 reveal that the numbers of support vectors for BVM algorithm, which is not capable of removing support vectors from the coreset, and SphereSVM method, which can discard support vectors from the coreset (by performing clipped update), are similar. This allows us to conclude that the vectors once selected to become support vectors are very unlikely to be eliminated from the final coreset. Therefore, the number of clipped updates is expected to be much smaller than the number of full updates (the experiments revealed that the number of clipped updates is usually much less than 1% of all updates). For this reason, we can analyze only the updates for which $\beta = \hat{\beta}$ and assume that all other updates do not increase the value of $\sum_{i=1}^{m} \alpha_i \|\mathbf{c} - \tilde{\mathbf{x}}_i\|^2$ at all.

Figure 3.2 visualizes the update step performed by the SphereSVM algorithm. It contains the projection of the feature space $\tilde{\Phi}$ onto the plane determined by the violators $\tilde{\mathbf{x}}_v$, $\tilde{\mathbf{x}}_u$ and the current center of the ball $\mathbf{c}$. The current solution was drawn as the black circle

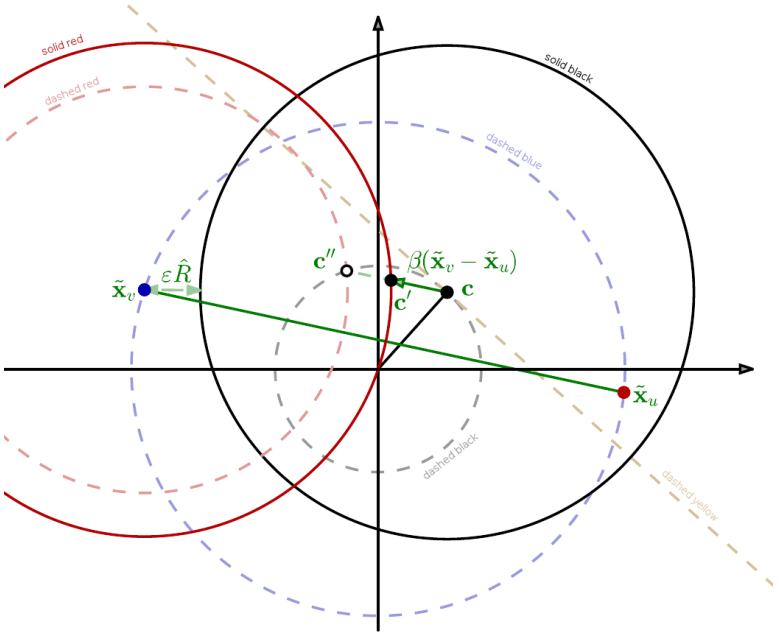

Figure 3.2: Visualization of the update step (projection on the plane determined by points $\tilde{\mathbf{x}}_v$, $\tilde{\mathbf{x}}_u$ and the ball center $\mathbf{c}$) – black circle is the current solution, which is a ball with center $\mathbf{c}$ and radius R; blue (dashed) circle represents the locations of points $\tilde{\mathbf{x}}_i$ in the feature space $\tilde{\Phi}$ (since $\tilde{k}(\mathbf{x}_i, \mathbf{x}_i) = \tilde{\tau}$ is constant, all points are mapped on the sphere).

of radius $\hat{R}$ and center $\mathbf{c}$. The blue dashed circle represents the locations of datapoints $\tilde{\mathbf{x}}_i$. Since we are using normalized kernel and $\tilde{k}(\mathbf{x}_i, \mathbf{x}_i) = \sqrt{\tau + 1 + \frac{1}{C}}$ is constant, all data points are mapped on a sphere with radius $\hat{R} = \sqrt{\tau + 1 + \frac{1}{C}}$ and center located in the origin $\mathbf{0}$.

It is true that $\tilde{\mathbf{x}}_u \cdot \mathbf{c} \geqslant \|\mathbf{c}\|^2$. Otherwise, either $\mathbf{c}$ would not be laying inside the convex hull formed by the support vectors or there would be another support vector closer to the center $\mathbf{c}$. Moreover, $\|\mathbf{c}\| \geqslant \varepsilon\hat{R}$, because otherwise all datapoints would be enclosed within ε-approximation of the enclosing ball, which would form the acceptable solution. Therefore, we can conclude that

$$\|\mathbf{c} - \tilde{\mathbf{x}}_u\| \leqslant \sqrt{\hat{R}^2 - \|\mathbf{c}\|^2} \leqslant \sqrt{1 - \varepsilon^2}\hat{R}. \tag{3.12}$$

From (3.4), (3.12) and the fact that $\|\mathbf{c}'' - \tilde{\mathbf{x}}_v\| = \|\mathbf{c} - \tilde{\mathbf{x}}_u\|$, it follows

$$\|\mathbf{c}' - \mathbf{c}''\| > (1 - \sqrt{1 - \varepsilon^2})\hat{R}. \tag{3.13}$$

Also,

$$\|\beta(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u)\| = \|\mathbf{c} - \mathbf{c}'\| \geqslant \varepsilon\hat{R}, \tag{3.14}$$

because if not, then $\|\mathbf{c}' - \tilde{\mathbf{x}}_v\| > R$ and the new enclosing ball with center $\mathbf{c}'$ and radius R would not include the datapoint $\tilde{\mathbf{x}}_v$ and (3.4) would not be satisfied.

From Figure 3.2 follows that

$$2\,(\mathbf{c} - \mathbf{c}') \cdot \mathbf{c} = (\mathbf{c} - \mathbf{c}'') \cdot (\mathbf{c} - \mathbf{c}')\,, \tag{3.15}$$

which leads to

$$2\,(\mathbf{c} - \mathbf{c}') \cdot \mathbf{c} = (\|\mathbf{c} - \mathbf{c}'\| + \|\mathbf{c}' - \mathbf{c}''\|)\,\|\mathbf{c} - \mathbf{c}'\|. \tag{3.16}$$

Now, the consequence of (3.13), (3.14) and (3.16) is the following lower bound of (3.11)

$$-2\beta(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u) \cdot \mathbf{c} - \|\beta\,(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u)\,\|^2 > \varepsilon\left(1 - \sqrt{1 - \varepsilon^2}\right)\hat{R}^2. \tag{3.17}$$

Hence, in each iteration the value of $\sum_{i=1}^{m} \alpha_i \|\mathbf{c} - \tilde{\mathbf{x}}_i\|^2$ is increased by at least $\eta\hat{R}^2$, where $\eta = \varepsilon\left(1 - \sqrt{1 - \varepsilon^2}\right)$. Now, it is clear that the maximal number of iterations after which $\sum_{i=1}^{m} \alpha_i \|\mathbf{c} - \tilde{\mathbf{x}}_i\|^2 = R^2$ holds true is equal to $\frac{1}{\eta}$. Moreover, $\frac{1}{\eta}$ constitutes the maximal number of support vectors (there cannot be more support vectors than the number of iterations, since in each iteration at most one support vector is added). In each iteration the algorithm must evaluate $\frac{m}{\eta}$ kernels in order to find the violator $\tilde{\mathbf{x}}_v$ and at most $\frac{1}{\eta}$ kernels in order to find the violator $\tilde{\mathbf{x}}_u$. Therefore, the complexity of this algorithm is equal to $O\left(\frac{m}{\eta^2}\right)$. Note however, that in real applications both the bound for number of iterations and number of support vectors will rarely be reached and the training usually finishes in much shorter time.

Chapter 4

Minimal Norm Support Vector Machines

4.1 Relation to MEB-based algorithms

It was proved in the Section 2.3 that the modified L2 SVM criterion (2.87) can be transformed into MNP problem (2.122). That makes it is possible to solve the SVM problem by applying one of the existing MNP solvers. In this section, we propose a novel Minimal Norm Support Vector Machines (MNSVM) algorithm that uses well known MDM approach [23, 35] in finding an SVM model.

Here, instead of searching for an ε-approximation of the minimum enclosing ball, as it was done in BVM and SphereSVM algorithms, we search for the point that is closest to the origin and that belongs to the convex hull spanned by the data points $\tilde{\mathbf{x}}_i$ in the feature space $\tilde{\Phi}$. Because of the fact that BVM, SphereSVM and MNSVM all originate from the same optimization problem (2.87), they yield the same solution. In other words, the center of the minimum enclosing ball found by both MEB-based methods and the point on the convex hull being closest to the origin obtained by MNSVM algorithm are equal.

4.2 Steps of the Algorithm

The Algorithm 4 contains the pseudo code of the MNSVM method. It is divided into

Algorithm 4 Minimum Norm Vector Machines Algorithm

Require: $\varepsilon \in (0, 1)$ {used in stopping criterion}
Ensure: $\mathbf{c} = \sum_{i=1}^{m} \alpha_i \tilde{\mathbf{x}}_i$ {the ε-approximation of the point closest to the origin}
1: $\boldsymbol{\alpha} \leftarrow \mathbf{0}, \alpha_0 \leftarrow 1$
2: **while** $\exists i : \tilde{\mathbf{x}}_i \cdot \mathbf{c} < (1 - \varepsilon)\|\mathbf{c}\|^2$ **do**
3: $v \leftarrow \arg\min_i \tilde{\mathbf{x}}_i \cdot \mathbf{c}$
4: $u \leftarrow \arg\max_{i:\alpha_i > 0} \tilde{\mathbf{x}}_i \cdot \mathbf{c}$
5: $\hat{\beta} \leftarrow \frac{\mathbf{c} \cdot (\mathbf{x}_v - \mathbf{x}_u)}{\|\mathbf{x}_v - \mathbf{x}_u\|^2}$
6: $\beta \leftarrow \min\left\{\hat{\beta}, \alpha_u\right\}$
7: $\alpha_v \leftarrow \alpha_v + \beta$
8: $\alpha_u \leftarrow \alpha_u - \beta$
9: **end while**

several stages. First the vector $\boldsymbol{\alpha}$ is initialized. Then selection of the violating vectors followed by an update to the current solution is performed within a loop. The loop ends when condition of the stopping criterion are satisfied. As a result of the algorithm we obtain the shortest vector $\mathbf{c} = \sum_{i=1}^{m} \alpha_i \tilde{\mathbf{x}}_i$ belonging to the convex hull spanned by the data points $\tilde{\mathbf{x}}_i$.

As we mentioned earlier, there is a large similarity between MNSVM and SphereSVM algorithms. This alikeness is mostly manifested by the way the update to the vector $\mathbf{c}$, constituting the solution of the minimal norm problem, is performed.

4.2.1 Initialization

The initialization of the MNSVM algorithm is almost the same as in BVM and SphereSVM methods. The random vector $\tilde{\mathbf{x}}_i$ is chosen (for example, the one with index 0 as it is shown in line 1 of the Algorithm 4) and its weight is set to 1.

In contrast to MEB-based methods, our approach does not require an initial estimation of the enclosing ball radius. This is very important improvement that eliminates the

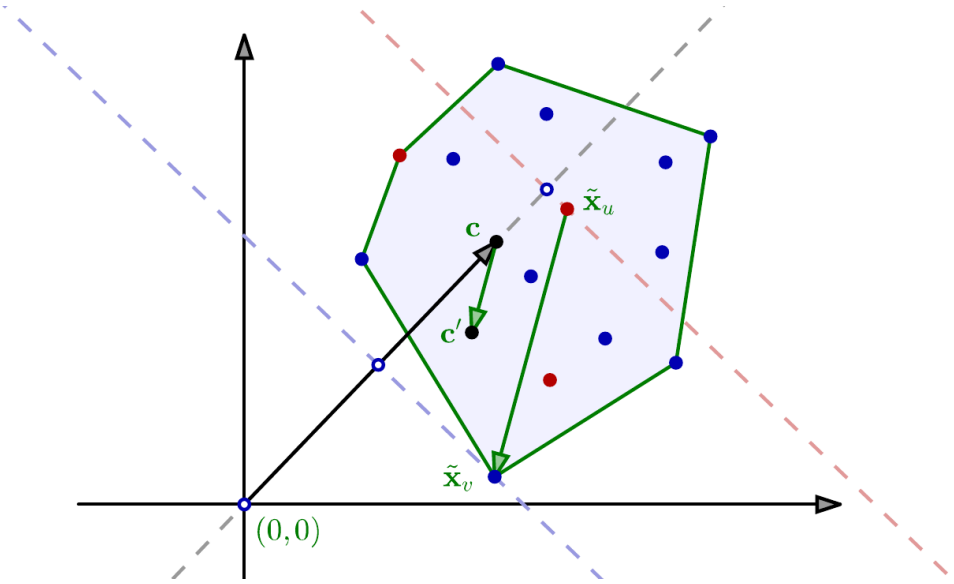

Figure 4.1: Update step of the MNSVM algorithm.

possibility of inaccurate radius estimation, which is very likely to occur when the number of data samples m is small or the dimensionality of the feature space $\tilde{\Phi}$ is low.

4.2.2 Selection of the Violating Vectors

Both SphereSVM and MNSVM algorithms update current solution by shifting it parallel to the vector $\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u$ formed by two violating vectors $\tilde{\mathbf{x}}_v$ and $\tilde{\mathbf{x}}_u$. In the case of SphereSVM the violators correspond to the vector being outside of the enclosing ball (violator $\tilde{\mathbf{x}}_v$) and support vector $\tilde{\mathbf{x}}_u$ being inside of the ball (and therefore not supporting it). Note that in MNSVM algorithm the violators $\tilde{\mathbf{x}}_v$ and $\tilde{\mathbf{x}}_u$ are the vectors laying respectively "in front" and "behind" the current solution $\mathbf{c}$ i.e. $\mathbf{x}_v \cdot \mathbf{c} < \|\mathbf{c}\|^2 < \mathbf{x}_u \cdot \mathbf{c}$ (see Figure 4.1). It is worth mentioning that for the minimal norm problem all support vectors constituting the solution of this problem must fulfill $\tilde{\mathbf{x}}_i \cdot \mathbf{c} = \|\mathbf{c}\|^2$. In other words the support vectors must be lying neither "in front" nor "behind" the solution $\mathbf{c}$.

Although the interpretation of the violating vectors is not the same for both algorithms, it shows out that in these two approaches the vectors $\tilde{\mathbf{x}}_v$ and $\tilde{\mathbf{x}}_u$ are literally the same vectors.

One of the assumptions of the SphereSVM method is that kernel is normalized and all vectors are mapped onto sphere. Therefore, we can write that

$$||\mathbf{c} - \tilde{\mathbf{x}}_i||^2 = ||\mathbf{c}||^2 - 2\tilde{\mathbf{x}}_i \cdot \mathbf{c} + ||\tilde{\mathbf{x}}_i||^2. \tag{4.1}$$

It means that relative distance of the vector $\tilde{\mathbf{x}}_i$ from the center $\mathbf{c}$ is dependent only on the scalar product $\tilde{\mathbf{x}}_i \cdot \mathbf{c}$ since $||\mathbf{c}||^2$ and $||\tilde{\mathbf{x}}_i||^2 = \tilde{k}(\mathbf{x}_i, \mathbf{x}_i)$ are constant for all vectors. Hence, the criteria

$$\tilde{\mathbf{x}}_v = \arg\min_{i:\tilde{\mathbf{x}}_i \in \mathcal{X}_r} \tilde{\mathbf{x}}_i \cdot \mathbf{c}, \tag{4.2}$$

and

$$\tilde{\mathbf{x}}_v = \arg\max_{i:\tilde{\mathbf{x}}_i \in \mathcal{X}_r} ||\mathbf{c} - \tilde{\mathbf{x}}_i||, \tag{4.3}$$

used in selecting violator $\tilde{\mathbf{x}}_v$ in MNSVM and SphereSVM algorithms, are equivalent. Analogously, the criteria

$$\tilde{\mathbf{x}}_u = \arg\max_{i:\alpha_i > 0} \tilde{\mathbf{x}}_i \cdot \mathbf{c}, \tag{4.4}$$

and

$$\tilde{\mathbf{x}}_u = \arg\min_{i:\alpha_i > 0} ||\mathbf{c} - \tilde{\mathbf{x}}_i||, \tag{4.5}$$

used in selecting violator $\tilde{\mathbf{x}}_u$, are also equivalent.

4.2.3 Update Procedure

The crucial difference between the MNSVM and the SphereSVM is the way of the current solution update. In each iteration MNSVM algorithm is trying to minimize the norm $||\mathbf{c}||$ of the current solution. Knowing that the shift of the vector $\mathbf{c}$ is performed along the vector $\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u$

$$\mathbf{c}' = \mathbf{c} + \beta(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u), \tag{4.6}$$

it is possible to find the optimal value of β by finding minimum of

$$\|\mathbf{c}'\|^2 = \|\mathbf{c} + \beta(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u)\|^2. \tag{4.7}$$

Finally, the value of β that minimizes the above norm is equal to

$$\hat{\beta} = \frac{(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u) \cdot \mathbf{c}}{\|\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u\|^2}. \tag{4.8}$$

Similarly, as in the case of SphereSVM algorithm, in order to satisfy the nonnegativity conditions for weights α_i, the value of β must be limited from above by α_u

$$\beta = \min\left\{\hat{\beta}, \alpha_u\right\}. \tag{4.9}$$

4.2.4 Stopping Criterion

In MNSVM, instead of using stopping criterion based on the radius of the enclosing ball, it is possible to apply one of the criteria suggested by López [36]. Our algorithm stops when further progress is not profitable and the current solution $\mathbf{c}$ approximates the true solution well enough – namely, when it is not possible to find a vector $\tilde{\mathbf{x}}_i$ satisfying condition $V(\tilde{\mathbf{x}}_i) > \varepsilon$, where $V(\tilde{\mathbf{x}})$ is defined as

$$V(\tilde{\mathbf{x}}) = \frac{\|\mathbf{c}\|^2 - \tilde{\mathbf{x}} \cdot \mathbf{c}}{\|\mathbf{c}\|^2}, \tag{4.10}$$

and the ε variable is the parameter of the stopping criterion. The function $V(\tilde{\mathbf{x}})$ indicates to what extent a given vector $\tilde{\mathbf{x}}_i$ violates the current solution $\mathbf{c}$. If the solution is optimal, then $V(\tilde{\mathbf{x}}_i) \leqslant 0$ for all $\tilde{\mathbf{x}}_i$. Although the algorithm finds just an approximate solution, one can control the accuracy of this approximation using parameter ε (the smaller the value of ε, the more accurate the solution). Unfortunately, decreasing the value of ε increases the time required to find the solution.

It is also possible to use the same stopping criterion as in BVM algorithm – this would be a valid approach if the two algorithms are to be compared.

Moreover, the fact that the algorithm does not require the estimation of the radius of the enclosing ball $\hat{R}$ (that is used in the stopping criterion of the SphereSVM method) is very important improvement as compared to algorithms based on minimal enclosing ball approach.

4.3 Properties of the Solution and the Feature Space

The points mapped into feature space $\tilde{\Phi}$ are mapped on the sphere with radius

$$R = \|\tilde{\mathbf{x}}_i\| = \sqrt{\tau + 1 + \frac{1}{C}}, \tag{4.11}$$

where $\tau = k(\mathbf{x}_i, \mathbf{x}_i)$. Moreover it is possible to prove that the points occupy only half of this sphere. Namely, since

$$\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j \tilde{k}(\mathbf{x}_i, \mathbf{x}_i) = \|\mathbf{w}\|^2 + b^2 + \frac{1}{C}\sum_{i=1}^{m} \alpha_i^2 \geqslant \frac{1}{mC}, \tag{4.12}$$

the lower bound of $\|\mathbf{c}\|^2$ is equal to $\frac{1}{mC}$ which is greater than 0. If the datapoints in the feature space $\tilde{\Phi}$ were spread across an area that is larger than one hemisphere, then obviously, the vector $\mathbf{c} = \mathbf{0}$ would be the solution of the minimal norm problem and that would be contradictory to the fact that $\|\mathbf{c}\|^2 > 0$.

Let γ be the angle between vectors $\mathbf{c}$ (representing the current solution) and $\mathbf{c}'$ (being the "updated" solution). Using definition of the scalar product we can write that

$$\cos\gamma = \frac{\mathbf{c} \cdot \mathbf{c}'}{\|\mathbf{c}\|\|\mathbf{c}'\|}. \tag{4.13}$$

It is true that the maximal value of γ is less than

$$\gamma \leqslant \arccos \frac{\frac{1}{mC}}{\tau + 1 + \frac{1}{C}} < \arccos \frac{1}{mC\,(\tau + 1)}. \tag{4.14}$$

The inequality (4.14) indicates that the parameter C may affect the stability of the solution. The higher the value C is, the higher the upper bound for the angle γ is. It means that for large values of this parameter one can expect large rotations of the vector $\mathbf{c}$ during consecutive iterations. This conclusion is consistent with our observations regarding MNSVM algorithm. Increasing the "span" of the dataset (upper bound for the angle γ) usually resulted in increase of the computation time. Moreover, we observed instability in accuracy when large values of C were used during training.

We performed some additional tests in order to investigate how $\rho = \|\mathbf{c}\|^2$ depends upon the training parameters C and γ. Two versions of the *checkers* dataset were used – one having 100 samples and one having 1000 samples. The dependency between the value of minimal norm $\|\mathbf{c}\|$ obtained by MNSVM algorithm and the training parameters is presented in Figure 4.2. The value of ρ increases when the penalty parameter C decreases or the "width" of the Gaussian functions decreases (for large values of γ). Moreover, it is interesting how the dataset affects the value of ρ – we observed that 10-fold increase of the training set size resulted in 10-fold decrease of $\|\mathbf{c}\|^2$. This observation is consistent with the conclusions derived from (4.12) – namely, that the minimal value of $\|\mathbf{c}\|^2$ is inversely proportional to the number of samples m. This observation is consistent with results obtained by Tsang. He discovered that the accuracy of BVM algorithm, yielding the same solution as MNSVM algorithm, increases when the size of the data increases. That can be explained by the fact that the precision of the stopping condition of the BVM algorithm is dependent upon the accuracy of the enclosing ball radius estimation which is dependent upon the value of $\|\mathbf{c}\|$

$$\hat{R}^2 = R^2 + \|\mathbf{c}\|^2, \tag{4.15}$$

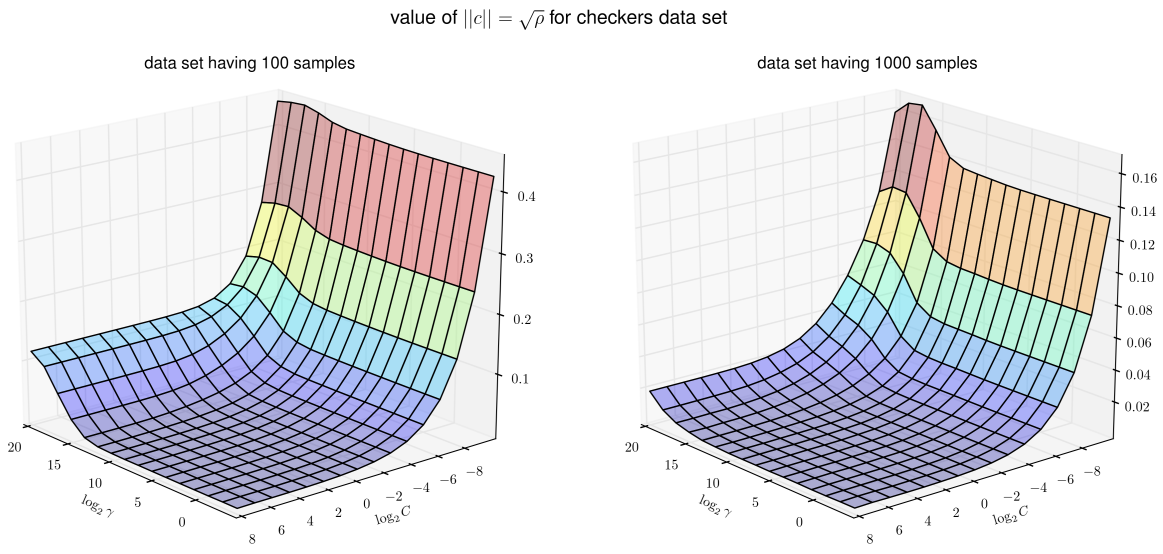

Figure 4.2: Dependency between the minimal norm $\|\mathbf{c}\|$ obtained by MNSVM algorithm and the training parameters C and γ. Plot on the left side depicts results obtained for *checkers* dataset having 100 samples. The right hand side picture shows results obtained for ten times larger training set.

where $\hat{R}$ is the estimated radius and R^2 is the real radius. The smaller the value $\|\mathbf{c}\|$ is the more accurate the radius estimation and the stopping criterion are. As we shown in Figure 4.2, this value decreases when the number of samples increases (which leads to accuracy improvement in BVM and SphereSVM algorithms).

Chapter 5

Implementation Techniques

5.1 Draw Scheme for Geometric SVM Solvers

All of the geometric algorithms presented in this dissertation are iterative. They converge to the final solution by repeating steps consisting of finding a vectors violating the stopping condition and performing an update to the current solution using these violators. A simple "brute force" algorithm for obtaining violators can be implemented as an iteration over all data samples combined with calculation of a mismatch function describing whether a given vector violates the stopping condition. This way, the worst violator can be found. Finally, based on this vector, the algorithm can update the current solution in order to minimize the mismatch of the violating sample.

Since such a naive search for the worst violators may be very expensive, other methods should be used. In the original BVM algorithm the probabilistic speedup heuristic [19] is applied. Instead of finding violating vector among entire dataset, a violator from a random subset of the dataset is picked. The size of the random subset is equal to $N_r = 59$ so the search for a vector that violates KKT condition the most is performed among 59 randomly chosen vectors. This strategy ensures that with probability 95% at least one of these randomly selected vectors will be among 5% vectors that are the worst violators

[7, 37, 19]. In case of LibCVM toolkit, this procedure is repeated up to $N_a = 10$ times. If after examining $N_d = N_a N_r = 590$ vectors no violator is found then the algorithm stops.

Although this method was proved to be quite effective, there seems to be a better approach for choosing the values of the parameters N_r and N_a. During our experiments, it turned out that the algorithm limiting the size of the random set to $N_r = 1$ (but using larger number of repetitions N_a) often outperforms other configurations in terms of speed and accuracy. In other words, selecting the first encountered vector that violates ε-approximation of the final solution results in significantly faster convergence. Moreover, the total number of random draws $N_d = 590$ can be decreased without affecting the accuracy in considerable way. Decreasing the number of random draws N_d can dramatically shorten the time required to find the model (see figures 7.10 and 7.11). This is especially important when the number of model training runs is large (e.g. when using grid search for model selection). That is why we suggest to reduce the size of the random subset to 1.

5.1.1 Impact on the Model Accuracy

In order to understand the impact of the parameter $N_d = N_a N_r$ on the accuracy of the solution estimation, we use Agresti-Coull confidence interval estimator [38] to find the upper bound for the percentage of the vectors violating KKT conditions. This boundary p is expressed by

$$p = \tilde{p} + z_{\alpha/2} \sqrt{\frac{\tilde{p}(1-\tilde{p})}{\tilde{n}}}, \tag{5.1}$$

where $\tilde{n} = N_d + z^2_{\alpha/2}$, $\tilde{p} = \frac{z^2_{\alpha/2}}{2\left(N_d + z^2_{\alpha/2}\right)}$ and $z_{\alpha/2}$ is $1 - \frac{\alpha}{2}$ percentile of a normal distribution. Estimations are shown in Figure 5.1. Moreover, some numeric values for parameters used in our experiments are presented in Table 5.1.

Let us, for example, apply this estimation to SphereSVM-100 algorithm, presented in Section 7.1.1. The property of this method is that it has maximal number of draws $N_d = 100$. According to Table 5.1 one can with confidence of 90% say that there will be

N_d	Upper bound of the percentage of outliers with a confidence level of:		
	90%	95%	99%
100	3.1%	4.4%	7.5%
590	0.55%	0.78%	1.3%

Table 5.1: The upper bound of the percent of datapoints violating KKT conditions after obtaining ε-approximation of the solution.

less than 3.1% datapoints left outside of the enclosing hyper-sphere with radius $(1 + \varepsilon)\hat{R}$ and center $\mathbf{c}$.

These statistics can give us insight into what portion of the data is neglected by the algorithms during the training process. Even if all this ignored samples should become support vectors this would not affect the training error significantly. The reason is that the percent of support vector constitutes the upper bound of the training error (called also the empirical risk). Assuming that the testing error (called the expected risk) is proportional to the training error, one may suppose that the neglected data will not affect the final

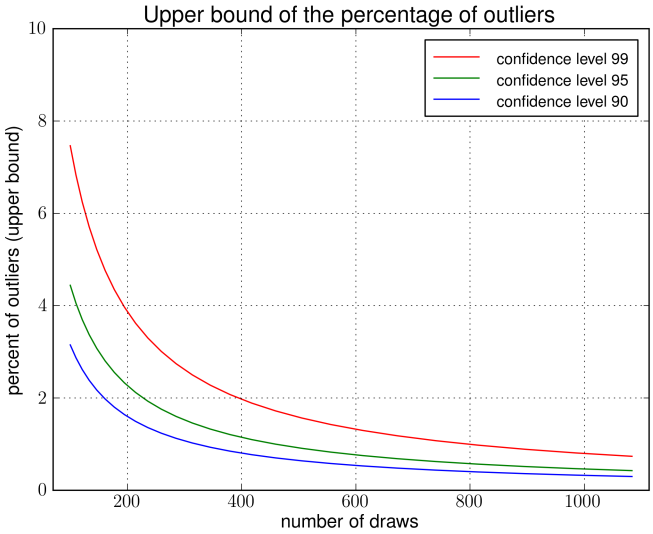

Figure 5.1: The upper bound of the percent of datapoints violating KKT conditions after obtaining ε-approximation of the solution – plots show confidence levels (from top to bottom): 99%, 95% and 90%). Dependency upon the number of draws (for different confidence levels) was calculated with Agresti-Coull estimator.

classification accuracy in significant way.

5.1.2 Impact on the Computational Complexity

In Section 3.3 we proved that the computational complexity of the SphereSVM algorithm is equal to $O\left(\frac{m}{\eta^2}\right)$ where $\eta = \varepsilon\left(1 - \sqrt{1-\varepsilon^2}\right)$ and m is the size of the dataset. Now, let us assume that the probabilistic speedup technique is used and $N_d = N_r N_a < m$. This modification changes the deterministic character of the algorithm into a probabilistic one. Now, the procedure stops when it is unable to find a violator within N_d attempts. In other words the algorithm stops when probability of finding a violator that lies farther than $(1+\varepsilon)\hat{R}$ from the center is estimated to be less than $\frac{1}{N_d}$.

In SphereSVM, the number of support vectors is limited by $\frac{1}{\eta}$. Therefore, the probabilistic SphereSVM requires not more than $\frac{N_d}{\eta}$ kernel computations in each iteration in order to find violator $\tilde{\mathbf{x}}_v$, being a vector located outside of the enclosing ball, and at most $\frac{1}{\eta}$ kernel computations in order to find violator $\tilde{\mathbf{x}}_u$, being a support vector located inside the enclosing ball. Knowing that the method is guaranteed to find ε-approximation of the enclosing ball withing $\frac{1}{\eta}$ iterations we may conclude that the complexity of the probabilistic SphereSVM is $O\left(\frac{N_d}{\eta^2}\right)$. That is equivalent to $O\left(\frac{1}{\eta^2}\right)$ since N_d is a constant independent of the number of the data points. This proves that the probabilistic speedup method makes the complexity of the algorithm independent upon the size of the data.

Similar deduction may be performed for MNSVM algorithm. Here the complexity also becomes independent upon the number of data samples.

5.2 Multi-scale Approximation

All our geometric SVM algorithms use the multi-scale approximation method proposed in [9]. Simply, the final ε-approximation of the solution is obtained in series of regulation steps. First the new variable $\hat{\varepsilon} = \frac{1}{2}$ is introduced and a rough $\hat{\varepsilon}$-approximation

of the solution is obtained. Then iteratively, $\hat{\varepsilon}$ is halved $\hat{\varepsilon} \leftarrow \hat{\varepsilon}/2$ and a more accurate $\hat{\varepsilon}$-approximations are calculated. This procedure is repeated until $\hat{\varepsilon} < \varepsilon$. In other words, whenever the algorithm is not able to make progress we gradually decrease $\hat{\varepsilon}$ towards ε until an accurate enough solution is obtained. The pseudo-code of this method is presented in Algorithm 5

Algorithm 5 Multi-scale Approximation Method

Require: $\varepsilon \in [0, 1)$ {used in stopping criterion}
1: $\hat{\varepsilon} \leftarrow \frac{1}{2}$
2: **repeat**
3: find $\hat{\varepsilon}$-approximation of the SVM model
4: $\hat{\varepsilon} \leftarrow \frac{\hat{\varepsilon}}{2}$
5: **until** $\hat{\varepsilon} < \varepsilon$

The justification for the multi-scale approach is an increase in the accuracy of the obtained models together with small improvement in time performance.

5.3 Over-relaxation

Successive over-relaxation (SOR) technique was originally designed by Young [39] in order to improve Gauss-Seidel method for solving a linear system of equations. The basic idea of this method is to increase the update performed during each iteration of some iterative algorithm

$$x_{n+1} = f(x_n). \tag{5.2}$$

After applying SOR, the iterative scheme (5.2) becomes

$$x_{n+1} = (1 - \eta) x_n + \eta f(x_n), \tag{5.3}$$

where $\eta \geqslant 1$ is an over-relaxation factor regulating the update step (when $\eta = 1$ the Equations (5.2) and (5.3) are identical). It is evident that with successive over-relaxation

the update applied to x_n

$$\|x_{n+1} - x_n\| = \|(1-\eta)\, x_n + \eta f(x_n) - x_n\| = \eta \|f(x_n) - x_n\|, \tag{5.4}$$

is η times larger compared to the original shift length that is equal to $\|x_{n+1} - x_n\| = \|f(x_n) - x_n\|$.

This approach turned out to be a very useful in other iterative algorithms. Here we promote the use of this method in acceleration of the proposed iterative geometric SVM solvers, namely SphereSVM and MNSVM.

5.3.1 Cycles in MDM Algorithm

Being vulnerable to the cycles in the update directions [36] is well known drawback of MDM algorithm. An example of such cycles is presented in Figure 5.2. Green lines forming zigzag $\mathbf{c}$, $\mathbf{c}'$, $\mathbf{c}''$, etc. represent the update steps of the BVM algorithm. The convergence to the optimal solution $\mathbf{c}^*$ is slow due to the update cycle – the algorithm alternates violating vectors and performs steps almost parallel to the optimal update direction (please note that after applying successive over-relaxation, marked with the red line, the algorithm converges faster). Such cyclings significantly decrease the efficiency of the method. Therefore, Barbero and López [35, 40] performed series of experiments aiming at detecting and removing the cycles from MDM method.

The loop detection algorithm used by Barbero and López [40] does not seem to be an appropriate approach in case of probabilistic algorithms such as SphereSVM or MNSVM. Their approach keeps track of previous update vectors $\mathbf{x}_v - \mathbf{x}_u$ and it is signaling a cycle whenever the same update vector is chosen. Since in proposed geometric algorithms the violators are chosen randomly then the probability of picking the same update vector is very small and the cycle detection is very unlikely. Moreover, this probability decreases even further when the size of the dataset is large.

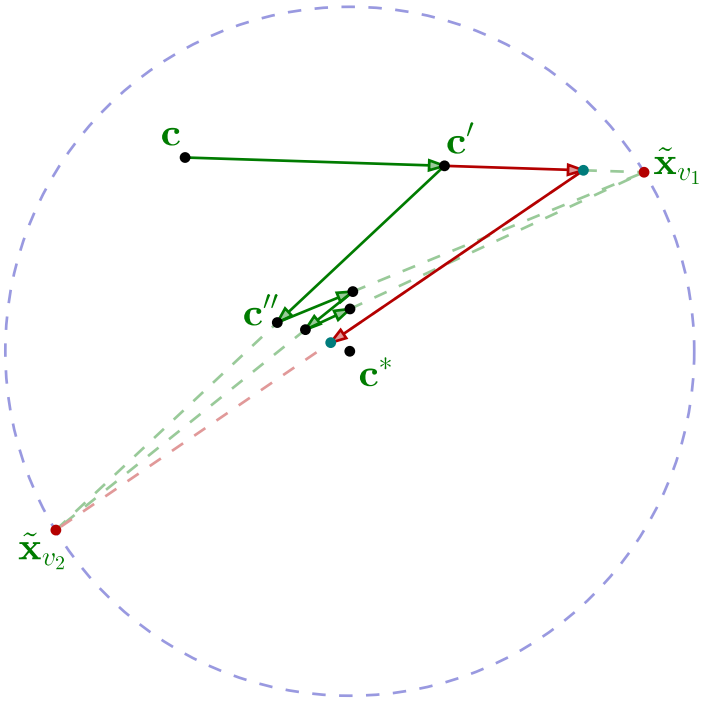

Figure 5.2: Update cycle in BVM algorithm. Green lines forming zigzag $\mathbf{c}$, $\mathbf{c}'$, $\mathbf{c}''$ etc. represent the original update scheme. Red lines represent the update step with successive over-relaxation algorithm.

5.3.2 Successive Over-relaxation

Since it is very difficult to detect an update cycle in SphereSVM and MNSVM methods due to the probabilistic nature of these algorithms, we decided to apply a different approach. Instead of detecting and removing loops form the algorithms we will try to prevent them. This prevention can be performed by use of successive over-relaxation approach.

The cycle is formed whenever an update along a vector $\mathbf{x}_v - \mathbf{x}_u$ is too small. In such case the same violating vector will be chosen again after several iterations. Increase of the update step is very likely to prevent such situations. This increase can be performed by introduction of a new parameter η into (3.3) describing the the update step that is simply a translation of the solution $\mathbf{c}$ along vector $\mathbf{x}_v - \mathbf{x}_u$

$$\mathbf{c}' = \mathbf{c} + \eta\beta(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u). \tag{5.5}$$

The parameter $\eta \in [1, 2)$ determines how much will the update be enlarged. In the original SphereSVM and MNSVM algorithms the solution $\mathbf{c}$ was shifted by $\|\beta(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u)\|$. After introduction of the over-relaxation coefficient η, the length of the displacement increased η times and is equal to $\eta\|\beta(\tilde{\mathbf{x}}_v - \tilde{\mathbf{x}}_u)\|$.

In order to apply over-relaxation expressed by (5.5) to the SphereSVM algorithm shown in Algorithm 3 it is required to change the line 8 into

$$\beta \leftarrow \min\left\{\eta\hat{\beta}, \alpha_u\right\}. \tag{5.6}$$

The same modification can be made to MNSVM algorithm listed in Algorithm 4, here the line 6 must be replaced by (5.6).

5.4 Alternative Approach to Multi-class Problems

LIBSVM and LibCVM toolkits use one-vs-one (aka pairwise) approach to deal with multi-class classification problems. For both tools single training procedure requires to obtain $\frac{n(n-1)}{2}$ models (where c is the number of classes). This way, each possible pair of classes is represented by an SVM model trained in recognizing these classes. The classification process is based on a simple voting mechanism – for a given sample, the class that was recognized by the largest number of classifiers is considered to be the true class of that sample. This method has been proven to have good classification accuracy and time performance [41].

5.4.1 All-at-once SVM Training

Several other multi-class training methods have been proposed. Especially interesting are the ones proposed by Weston and Watkins [42] or Crammer and Singer [43]. In those methods, the multi-class classification problem is not divided into several smaller binary

problems. Instead, one universal SVM model, that is capable to distinguish all the classes, is trained.

Recently, Ashraf et al. [20] proposed a multi-class SVM solver that is based on CVM algorithm and that performs only one training to obtain model describing all classes. The basic idea of the algorithm is to allow multi-dimensional vector labels. In a binary classification problems the labels are one-dimensional (they have values +1 or −1). In the case of Ashraf's method the labels were c-dimensional (where c is the number of classes). He proposed a way of choosing the labels and found simple equation describing the scalar product between the labels.

The modified L2 SVM learning problem (2.87) with multidimensional labels becomes

$$\arg\min_{\mathbf{W},b,\zeta,\rho} \frac{1}{2}\operatorname{trace}(\mathbf{W}^\intercal\mathbf{W}) + \frac{1}{2}\|\mathbf{b}\|^2 - \rho + \frac{C}{2}\sum_{i=1}^{m}\zeta_i^2, \tag{5.7}$$

subject to

$$\mathbf{y}_i^\intercal(\mathbf{W}\mathbf{x}_i + \mathbf{b}) \geqslant \rho - \zeta_i, \quad i = 1,\ldots,m. \tag{5.8}$$

Please note that the bias vector $\mathbf{b}$ has the same dimensionality as the label vectors $\mathbf{y}_i$. Moreover, the solution $\mathbf{W}$ is a matrix with the number of rows equal to the dimension of the label vector and the number of columns equal to the dimensionality of the vector $\mathbf{x}_i$.

The corresponding dual form is

$$\arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j\left(\mathbf{y}_i\cdot\mathbf{y}_j\right)\left(\mathbf{x}_i\cdot\mathbf{x}_j\right) + \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j\left(\mathbf{y}_i\cdot\mathbf{y}_j\right) + \frac{1}{C}\sum_{i=1}^{m}\alpha_i^2, \tag{5.9}$$

subject to

$$\sum_{i=1}^{m}\alpha_i = 0 \tag{5.10a}$$

$$\alpha_i \geqslant 0, \quad i = 1,\ldots,m, \tag{5.10b}$$

and the Karush-Kuhn-Tucker conditions are as follows

$$\alpha_i \left(\mathbf{y}_i^\intercal (\mathbf{W}\mathbf{x}_i + \mathbf{b}) - \rho + \zeta_i \right) = 0, \quad i = 1, \ldots, m. \tag{5.11}$$

From the KKT conditions we obtain that

$$\mathbf{W} = \sum_{i=1}^{m} \alpha_i \mathbf{y}_i \mathbf{x}_i^\intercal, \tag{5.12}$$

and

$$\mathbf{b} = \sum_{i=1}^{m} \alpha_i \mathbf{y}_i. \tag{5.13}$$

Now, the decision function for the SVM classifier is

$$d(\mathbf{x}) = \arg\max_{t=1\ldots c} \mathbf{y}_t^\intercal (\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{5.14}$$

where $\mathbf{y}_t$ for $t = 1, \ldots, c$ are the label vectors. Equation (5.14) is equivalent to

$$d(\mathbf{x}) = \arg\max_{t=1\ldots c} \sum_{i=1}^{m} \alpha_i \left(\mathbf{y}_t \cdot \mathbf{y}_i \right) \left(\mathbf{x}_i \cdot \mathbf{x}_j \right) + \sum_{i=1}^{m} \alpha_i \left(\mathbf{y}_t \cdot \mathbf{y}_i \right). \tag{5.15}$$

5.4.2 Nonlinear Multi-class Training

The nonlinear version of the approach presented in this section can be easily derived. First, the scalar products $\mathbf{x}_i \cdot \mathbf{x}_j$ from (5.9) must be replaced by a kernel function $k\left(\mathbf{x}_i, \mathbf{x}_j\right)$ defining a dot product in some feature space Φ. The resulting criterion is

$$\arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \left(\mathbf{y}_i \cdot \mathbf{y}_j \right) k\left(\mathbf{x}_i, \mathbf{x}_j\right) + \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \left(\mathbf{y}_i \cdot \mathbf{y}_j \right) + \frac{1}{C} \sum_{i=1}^{m} \alpha_i^2. \tag{5.16}$$

Similarly, the decision function (5.15) becomes

$$d(\mathbf{x}) = \arg\max_{t=1\ldots n} \sum_{i=1}^{m} \alpha_i \left(\mathbf{y}_t \cdot \mathbf{y}_i\right) k\left(\mathbf{x}_t, \mathbf{x}_i\right) + \sum_{i=1}^{m} \alpha_i \left(\mathbf{y}_t \cdot \mathbf{y}_i\right). \tag{5.17}$$

5.4.3 Label Vector Selection

We propose use of $c-1$ dimensional labels that satisfy $\|\mathbf{y}_i\| = 1$ and that are distributed in form of a symmetric multidimensional pyramid. From the algorithm's point of view it is not necessary to know exact coordinates of the label vectors. It is enough to know the scalar products of the label vectors that in case of pyramid-like vector structure are equal to

$$\mathbf{y}_i \cdot \mathbf{y}_j = \begin{cases} 1 & \mathbf{y}_i = \mathbf{y}_j \\ -\frac{1}{c-1} & \mathbf{y}_i \neq \mathbf{y}_j \end{cases}. \tag{5.18}$$

For binary problems, where $c = 2$, the scalar product defined by (5.18) is equal to 1 when $y_i = y_j$ and -1 when $y_i \neq y_j$. This is consistent with the traditional binary SVM classifier. The label vectors proposed by Ashraf do not have this property – they cannot be applied to binary problems.

5.5 Bias Evaluation

All geometric L2 SVM algorithms presented in this dissertation uses the same decision function (2.109), namely

$$d(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i k\left(\mathbf{x}_i, \mathbf{x}\right) + b, \tag{5.19}$$

where bias b can be calculated directly in the dual space (2.98)

$$b = \sum_{i=1}^{m} \alpha_i y_i. \tag{5.20}$$

This way of bias evaluation may lead to inaccurate decision functions when the quality of SVM model approximation is small (for instance, if large value of ε is used). Also, for a bad-conditioned problem settings (e.g. for large values of penalty parameter C and large variances of Gaussian kernel when the iteration is stopped before reaching the optimal solution) equation (5.20) should not be used. Fortunately, the accuracy of the bias evaluation can be improved. Instead of using the direct equation (5.20) it is possible to calculate the bias from KKT conditions. For the modified L2 SVM problem the KKT conditions are as follows

$$\alpha_i \left(y_i \left(\sum_{j=1}^{m} \alpha_j y_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) + b \right) - \rho + \zeta_i \right) = 0, \quad i = 1, \ldots, m. \tag{5.21}$$

Knowing that for all support vectors $\mathbf{x}_i \in S$ the values of the corresponding coefficients α_i are greater than 0 one can calculate bias term b from

$$b = y_i \left(\rho - \zeta_i\right) - \sum_{j=1}^{m} \alpha_j y_j k\left(\mathbf{x}_i, \mathbf{x}_j\right), \quad \forall\, \mathbf{x}_i \in S. \tag{5.22}$$

In order to increase the accuracy it is better to average the calculation of the bias over all support vector and (5.22) should be used as follows

$$b = \frac{1}{|S|} \sum_{i:\mathbf{x}_i \in S} \left(y_i \left(\rho - \zeta_i\right) - \sum_{j=1}^{m} \alpha_j y_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) \right). \tag{5.23}$$

Furthermore, because of (2.100) and (2.102), formula (5.23) can be rewritten as

$$b = \frac{1}{|S|} \sum_{i:\mathbf{x}_i \in S} \left(y_i \left(\|\mathbf{c}\|^2 - \frac{\alpha_i}{C} \right) - \sum_{j=1}^{m} \alpha_j y_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) \right), \tag{5.24}$$

where $\mathbf{c}$ is the solution of the underlying MEB or MN problem.

Experimental results suggest that bias calculated using (5.24) is less sensitive to the accuracy of the final model estimation. This usually results in better classification perfor-

mance.

5.5.1 Bias Evaluation in All-at-once Multi-class Training

Similar bias evaluation may be applied to the multi-class SVM introduced in Section 5.4.1. In order to find the label of a vector $\mathbf{x}$ using the decision function for multi-class SVM (5.15) it is actually not necessary to know the bias $\mathbf{b}$ – the value of the decision function can be evaluated if the scalar product $\mathbf{y}_t \cdot \mathbf{b}$ are known for all the labels $\mathbf{y}_t$ where $t = 1, \ldots, c$.

From the KKT conditions for the multi-class SVM (5.11) we know that for all support vectors

$$\mathbf{y}_i \cdot \mathbf{b} = \rho - \zeta_i - \mathbf{y}_i^\mathsf{T} \mathbf{W} \mathbf{x}_i, \quad \mathbf{x}_i \in S. \tag{5.25}$$

Therefore, the scalar product $\mathbf{y}_t \cdot \mathbf{b}$ can be expressed as

$$\mathbf{y}_t \cdot \mathbf{b} = \frac{1}{|S_t|} \sum_{i=1}^{m} \left(\rho - \zeta_i - \mathbf{y}_t^\mathsf{T} \mathbf{W} \mathbf{x}_i \right), \tag{5.26}$$

Where S_t is the set of support vectors $\mathbf{x}_i$ such that $\mathbf{y}_i = \mathbf{y}_t$. The equation (5.26) can be simplified to

$$\mathbf{y}_t \cdot \mathbf{b} = \frac{1}{|S_t|} \sum_{i=1}^{m} \left(\|\mathbf{c}\|^2 - \frac{\alpha_i}{C} - \mathbf{y}_i^\mathsf{T} \mathbf{W} \mathbf{x}_i \right), \tag{5.27}$$

which in case of nonlinear multi-class training becomes

$$\mathbf{y}_t \cdot \mathbf{b} = \frac{1}{|S_t|} \sum_{i=1}^{m} \left(\|\mathbf{c}\|^2 - \frac{\alpha_i}{C} - \sum_{j=1}^{m} \alpha_j \left(\mathbf{y}_i \cdot \mathbf{y}_j \right) k\left(\mathbf{x}_i, \mathbf{x}_j \right) \right). \tag{5.28}$$

5.6 Other Minimal Norm Solvers in MNSVM

The original MDM algorithm, that is used in MNSVM solver [44], mainly focuses on removing support vectors violating KKT conditions from the coreset. In each iteration two violators are found. First violator, identified by $\mathbf{x}_v$, is the vector located "in front"

of the current solution $\mathbf{c}$. This vector satisfies condition $\mathbf{x}_v \cdot \mathbf{c} < ||\mathbf{c}||^2$. Second violating vector, labeled as $\mathbf{x}_u$, is a support vector that is laying "behind" the solution and satisfies $\mathbf{x}_u \cdot \mathbf{c} > ||\mathbf{c}||^2$. Finally, during the update step, weight α_v corresponding to $\mathbf{x}_v$ is increased and weight α_u related to $\mathbf{x}_u$ is decreased (in other words – importance of the violator $\mathbf{x}_v$ is increased at the expense of the violator $\mathbf{x}_u$). Therefore, we can say that MDM is trying to remove support vectors $\mathbf{x}_u$ not supporting the solution.

The decision of selection particular violator $\mathbf{x}_u$ is driven by the belief that the larger $\mathbf{x}_i \cdot \mathbf{c}$ is for a given sample $\mathbf{x}_i$ the less this sample contributes to the final solution and it should be a priority to eliminate if from the coreset (or at least its share in the final solution should be decreased).

5.6.1 Improved MDM

Improved MDM (IMDM) proposed by Franc [45] uses different assumption. The algorithm focuses on minimization of $||\mathbf{c}||$ (which is actually, the essence of the minimal norm problem being solved). So instead of removing vectors from the coreset that do not support the solution, IMDM is trying to maximize decrease of $||\mathbf{c}||$ in subsequent update steps.

The update to the solution $\mathbf{c}$ is performed by shifting it along line connecting two violating points

$$\mathbf{c}' = \mathbf{c} + \beta\,(\mathbf{x}_v - \mathbf{x}_u)\,. \tag{5.29}$$

In MDM algorithm, first the violators $\mathbf{x}_v$ and $\mathbf{x}_u$ are selected and then the optimal value of β is calculated in way that minimizes the norm of the new solution $\mathbf{c}'$. In IMDM first the violator $\mathbf{x}_v$ is chosen and then the vector $\mathbf{x}_u$ is selected in a way that would minimize $||\mathbf{c}'||$. Since

$$||\mathbf{c}' - \mathbf{c}||^2 = 2\beta\,(\mathbf{x}_v - \mathbf{x}_u) \cdot \mathbf{c} + \beta^2 ||\mathbf{x}_v - \mathbf{x}_u||^2, \tag{5.30}$$

the maximal possible improvement is achieved for

$$\beta = -\frac{(\mathbf{x}_v - \mathbf{x}_u) \cdot \mathbf{c}}{||\mathbf{x}_v - \mathbf{x}_u||^2}. \tag{5.31}$$

If we use (5.31) in (5.30) we obtain that maximal decrease of $||\mathbf{c}'||$ is equal to

$$||\mathbf{c}' - \mathbf{c}||^2 = -\left(\frac{(\mathbf{x}_v - \mathbf{x}_u) \cdot \mathbf{c}}{||\mathbf{x}_v - \mathbf{x}_u||}\right)^2. \tag{5.32}$$

Therefore, we can say that IMDM during searching for a violator $\mathbf{x}_u$ is solving

$$\mathbf{x}_u = \arg\max_{\mathbf{x}_u \in S} \left(\frac{(\mathbf{x}_v - \mathbf{x}_u) \cdot \mathbf{c}}{||\mathbf{x}_v - \mathbf{x}_u||}\right)^2. \tag{5.33}$$

Similarly, since MDM selects $\mathbf{x}_u$ in such way as if to maximize $\mathbf{x}_u \cdot \mathbf{c}$ it is actually solving the following problem

$$\mathbf{x}_u = \arg\max_{\mathbf{x}_u \in S} \left((\mathbf{x}_v - \mathbf{x}_u) \cdot \mathbf{c}\right)^2. \tag{5.34}$$

Likeness of these two methods is clearly visible in Equations (5.33) and (5.34).

5.6.2 Generalized IMDM

It is possible to generalize both methods by introduction of coefficient $k \in [0, 1]$. The generalized IMDM algorithm uses the following criterion

$$\mathbf{x}_u = \arg\max_{\mathbf{x}_u \in S} \left(\frac{(\mathbf{x}_v - \mathbf{x}_u) \cdot \mathbf{c}}{||\mathbf{x}_v - \mathbf{x}_u||^k}\right)^2, \tag{5.35}$$

to select the violator $\mathbf{x}_u$. When $k = 0$ then (5.35) is equivalent to the standard MDM since (5.35) is the same as (5.34). Whereas, if $k = 1$ then criterion (5.35) is identical to (5.33) and general MDM becomes the same as Improved MDM proposed by Franc.

5.6.3 MNSVM with different Minimal Norm Problem Solvers

Both Minimal Norm problem solvers presented in Sections 5.6.1 and 5.6.2 may be used with MNSVM algorithm. The only change that must be introduced in respect to the original MNSVM method is the way the violating vectors are found. Instead of searching for a vector $\mathbf{x}_u$ satisfying condition (4.4) (that is equivalent to (5.34)) the criterion (5.34) or (5.35) should be used. In this way we may hope to obtain version of MNSVM that has faster convergence compared to the original approach.

5.7 Model Selection based on Pattern Search

Same as in many other classifier designs, the training of SVM includes a search for the best values of the so called hyper-parameters of an SVM model. These are the penalty parameter C, the variances of Gaussian kernel or the order (degree) of the polynomial one. Finding the most suitable values for a given problem, in respect to the CPU time needed, is the most expensive part of the training (i.e. learning). In this section we will focus on parameters selection for SVM training with Gaussian kernels. Such training requires two parameters – the regularization parameter C and the kernel parameter γ.

Let us define a finite set of all possible values of parameter γ as

$$\Gamma = \{\gamma_i\}, \quad i = 1, \ldots, n_\gamma, \tag{5.36}$$

where n_γ is the number of all possible values of γ and elements of Γ satisfy $\gamma_i > \gamma_j$ for $i > j$. Then, let the finite set of all possible values of parameter C be defined as

$$\Theta = \{C_i\}, \quad i = 1, \ldots, n_C, \tag{5.37}$$

where n_C is the number of distinct values of C. The set Θ has the property that $C_i > C_j$ if $i > j$. Now, the search space Ω for the model selection problem is the Cartesian product

of all possible values of parameters γ and C

$$\Omega = \Gamma \times \Theta. \tag{5.38}$$

The model selection problem can be mathematically described as

$$\omega^* = \arg\max_{\omega \in \Omega} P(\omega), \tag{5.39}$$

where ω^* are the optimal model parameters and $P(\omega)$ denotes prediction accuracy for SVM model obtained using parameters ω.

5.7.1 Grid Search Method

The Grid Search (GS) method, called also Combinatorial Search, is a widely used model selection approach. It requires checking all combinations of the training parameters in order to select the one that maximizes classification accuracy. Although such approach provides reliable parameter values it cannot be used when the number of parameter combinations is large or when the training time is long. This method scales exponentially with the number of parameters.

5.7.2 Pattern Search Method

There are other, possibly faster, ways to obtain good model parameters. One of such methods is the Pattern Search (PS) algorithm [46, 47]. It can be described as a greedy numerical optimization method. An important property of this algorithm is that it does not impose any constraints on the optimized function which makes it applicable for problems like model selection.

We can define a pattern as a set of offsets

$$P = \left\{\left(p_\gamma, p_C\right)_i\right\}. \tag{5.40}$$

An example of a simple pattern may be the cross-pattern defined as

$$P_+ = \{(0,0), (-1,0), (1,0), (0,1), (0,-1)\}. \tag{5.41}$$

The pseudo-code of the Pattern Search algorithm is presented in Algorithm 6. Starting from the center of the search grid, it is searching for the optimal set of parameters by analyzing the neighboring parameter combinations and selecting the most promising ones. The neighborhood is defined by the pattern P and the scale θ affecting the size of that pattern. At the beginning, the steps made by the algorithm are relatively large but as the algorithm progresses they become smaller (the step size is controlled by variable θ). Whenever no improvement can be made by a shift of the current pattern towards better parameters, the scale θ is halved. The algorithm stops when the scale θ is decreased below 1. The visualization of this procedure is presented in Figure 5.3.

Pattern Search does not guarantee obtaining optimal training parameters. Therefore random restarts are needed in order to protect oneself against obtaining local optimum.

In our implementation we decided to use deterministic restarts. As the starting point for the subsequent PS runs we select the points having the largest uncertainty. The uncertainty measure is defined as

$$U\left(\gamma_{\kappa_\gamma}, C_{\kappa_C}\right) = \min_{\left(\gamma_{\kappa'_\gamma}, C_{\kappa'_C}\right)\in\Omega'} \left(|\kappa_\gamma - \kappa'_\gamma| + |\kappa_C - \kappa'_C|\right), \tag{5.42}$$

where Ω' is a set of training parameters for which training and testing has already been performed. Function $U(\gamma_{\kappa_\gamma}, C_{\kappa_C})$ express uncertainty that a given combination of parameters has meaningfully better classification performance than the combinations already

Algorithm 6 Pattern Search

Require: $\Gamma = \{\gamma_i\},\ i = 1, \ldots, n_\gamma$ {set of all possible values of γ}
Require: $\Theta = \{C_i\},\ i = 1, \ldots, n_C$ {set of all possible values of C}
Require: $P = \left\{\left(i_\gamma, i_C\right)\right\}$ {the pattern}
Ensure: $\omega \in \Omega$ {combination of parameters γ and C}
1: $\left(\kappa_\gamma, \kappa_C\right) \leftarrow \left(\left\lceil \frac{n_\gamma}{2} \right\rceil, \left\lceil \frac{n_C}{2} \right\rceil\right)$
2: $\theta \leftarrow \min\left\{\left\lceil \frac{n_\gamma}{2} \right\rceil, \left\lceil \frac{n_C}{2} \right\rceil\right\}$
3: **while** $\theta > 0$ **do**
4: $\left(p_\gamma, p_C\right) = \arg\max_{\left(p_\gamma, p_C\right) \in P}$ accuracy for $\left(\gamma_{\kappa_\gamma + \theta p_\gamma}, C_{\kappa_C + \theta p_C}\right)$
5: **if** $\left(p_\gamma, p_C\right) = (0, 0)$ **then**
6: $\theta \leftarrow \left\lfloor \frac{\theta}{2} \right\rfloor$ {decrease scale}
7: **else**
8: $\left(\kappa_\gamma, \kappa_C\right) \leftarrow \left(\kappa_\gamma + \theta p_\gamma, \kappa_C + \theta p_C\right)$ {shift towards best parameters}
9: **end if**
10: **end while**
11: $\omega = \left(\gamma_{\kappa_\gamma}, C_{\kappa_C}\right)$

tested.

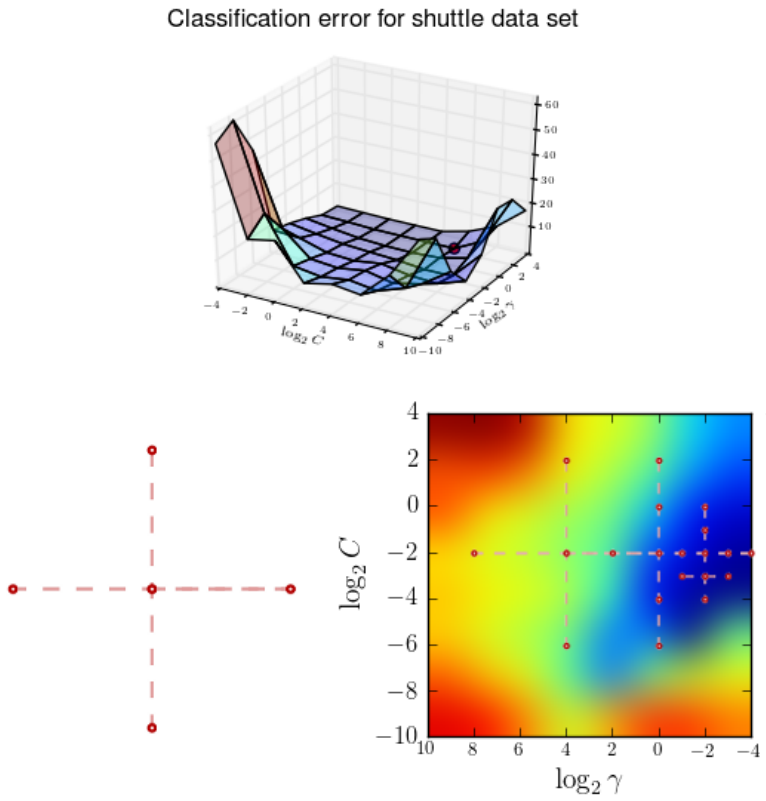

Figure 5.3: Pattern Search for *shuttle* dataset. Figure shows dependency of the classification error on the values of the training parameters C and γ (upper part), the shape of the cross pattern (lower left) and the steps of the PS algorithms (lower right) starting from $C = 2^{-2}$, $\gamma = 2^4$ and $\theta = 2$ and converging to $C = 2^{-3}$ and $\gamma = 2^{-2}$ (red points represent combinations of parameters C and γ for which SVM training was performed).

Chapter 6

Role of the Bias in the Geometric Approach to SVM

It was shown by Vogt [48] and Kecman et al. [49, 50] that for positive definite kernels it is possible to train SVM models without the bias term b. The bias may be implicitly included into kernel so there is no need to use it in the decision function. In the case of the L1 SVM training, the problem defined in (2.10) can be rewritten as

$$\arg\min_{\mathbf{w},\zeta} \|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\zeta_i, \tag{6.1}$$

subject to

$$y_i\left(\phi(\mathbf{x}_i)\cdot\mathbf{w}\right) \geqslant 1-\zeta_i, \quad i=1,\ldots,m, \tag{6.2}$$

and

$$\zeta_i \geqslant 0, \quad i=1,\ldots,m, \tag{6.3}$$

where $\varphi(\mathbf{x})$ represents a mapping of a vector $\mathbf{x}$ into some feature space Φ defined by kernel $k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)$. That leads to the following decision function

$$d(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i k\left(\mathbf{x}_i, \mathbf{x}\right). \tag{6.4}$$

Similarly for L2 SVM it is possible to transform optimization problem defined in (2.31) into

$$\arg\min_{\mathbf{w},\zeta} \|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \zeta^2, \tag{6.5}$$

subject to

$$y_i \left(\phi(\mathbf{x}_i) \cdot \mathbf{w}\right) \geqslant 1 - \zeta_i, \quad i = 1, \ldots, m, \tag{6.6}$$

which leads to the same decision function as in (6.4).

It was presented by Huang and Kecman in [51] that removing the bias term from SVM models very often results in better accuracy. This is the main reason why we investigate the application of this technique in SphereSVM and MNSVM algorithms. Additional premise supporting this research is the fact that the geometric SVM criterion (2.87) without bias term is closer to the original L2 SVM optimization problem (2.31). Since the geometric algorithms without bias will be focused on minimization of $\|\mathbf{w}\|^2$ instead of $\|\mathbf{w}\|^2 + b^2$, approach not involving bias may result in better decision boundary (i.e. having larger separation margin $\frac{2}{\|\mathbf{w}\|}$).

6.1 Geometric Approach without Bias Term

Removing the bias term b from the optimization problem (2.87), used by SphereSVM and MNSVM algorithms, results in the following minimization problem

$$\arg\min_{\mathbf{w},\zeta,\rho} \frac{1}{2}\|\mathbf{w}\|^2 - \rho + \frac{C}{2} \sum_{i=1}^{m} \zeta_i^2, \tag{6.7}$$

subject to

$$y_i \left(\phi(\mathbf{x}_i) \cdot \mathbf{w} \right) \geqslant \rho - \zeta_i, \quad i = 1, \ldots, m. \tag{6.8}$$

Similarly as it was performed in Section 2.3, it can be derived that this problem is equivalent to the following

$$\arg\min_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \ddot{k} \left(\mathbf{x}_i, \mathbf{x}_j \right). \tag{6.9}$$

subject to

$$\sum_{i=1}^{m} \alpha_i = 1, \tag{6.10a}$$

$$\alpha_i \geqslant 0, \quad i = 1, \ldots, m. \tag{6.10b}$$

The kernel $\ddot{k} \left(\mathbf{x}_i, \mathbf{x}_j \right)$, representing a mapping into feature space $\ddot{\Phi}$, is defined as

$$\ddot{k} \left(\mathbf{x}_i, \mathbf{x}_j \right) = y_i y_j k \left(\mathbf{x}_i, \mathbf{x}_j \right) + \frac{\delta_{ij}}{C}. \tag{6.11}$$

Both problems, the original one (2.87) having the bias term and the newly introduced one (6.7), can be reduced to the same minimal enclosing ball (or minimal norm) problems. In fact (2.122) and (6.9) representing the dual space optimization criterion differ only in the kernel. Compared with the kernel $\tilde{k} \left(\mathbf{x}_i, \mathbf{x}_j \right)$ for the original optimization criterion (2.114), kernel $\ddot{k} \left(\mathbf{x}_i, \mathbf{x}_j \right)$ lacks the component $y_i y_j$.

6.2 Properties of the Feature Space

When the model contains a bias term b, the feature space $\tilde{\Phi}$ is defined by the following mapping

$$\tilde{\varphi}(\mathbf{x}_i) = \begin{bmatrix} y_i\varphi(\mathbf{x}_i) \\ y_i \\ \frac{\mathbf{e}_i}{\sqrt{C}} \end{bmatrix}, \tag{6.12}$$

where $\varphi(\mathbf{x})$ is the original mapping associated with the kernel k, y_i is the label of the vector $\mathbf{x}_i$ and $\mathbf{e}_i$ is an m dimensional vector with all zeros except that the i-th entry equals to 1. For training without bias, removal of the component $y_i y_j$ from the kernel $\tilde{k}$ is equivalent to removal feature y_i from the mapping $\tilde{\varphi}(\mathbf{x})$ and it leads to the following projection

$$\ddot{\varphi}(\mathbf{x}_i) = \begin{bmatrix} y_i\varphi(\mathbf{x}_i) \\ \frac{\mathbf{e}_i}{\sqrt{C}} \end{bmatrix}. \tag{6.13}$$

This elimination causes the decrease of the condition number of the kernel matrix since the most significant eigenvalue of the kernel matrix is reduced. Moreover, the reduction of the main principal component leads to more balanced distribution of the datapoints in the feature space $\ddot{\Phi}$. For this reason, we can expect that the number of support vectors in models generated by original approach involving the bias term will be smaller than in the case of the models generated by the "bias-less" methods. The intuition tells that the ball enclosing data points stretched along some direction will require smaller number of supporting vectors than the ball enclosing evenly distributed points (see Figure 6.1).

Since the removal of the bias term is likely to increase the number of support vectors, it is expected that the training time will be increased as well. Namely, not only the minimal number of iterations (equal to the number of support vectors), but also the number of kernel evaluations required to find a violator will be greater. Of course one can not be absolutely certain that the approach not involving bias term will be slower since the

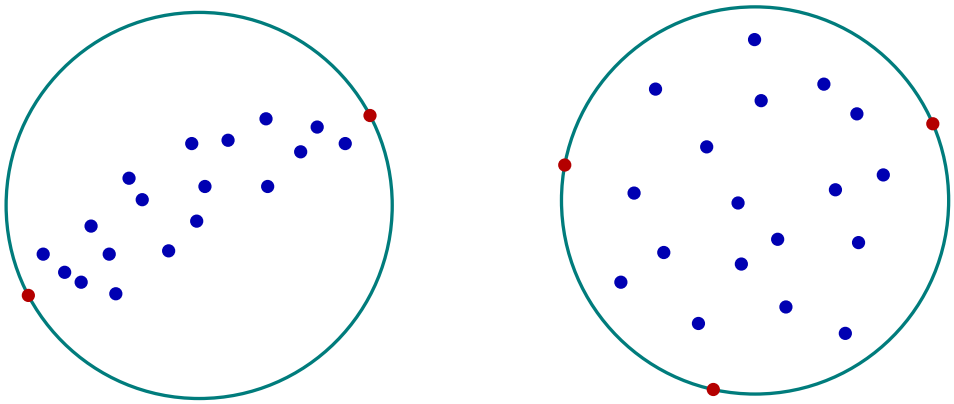

Figure 6.1: Visualization of the minimal enclosing ball in the feature space for "stretched" dataset (with bias, on the left side) and evenly distributed one (without bias, on the right side).

distribution of the datapoints in the feature space is not the only criterion that affects the speed of the algorithms.

Chapter 7

Experiments and Results

7.1 Geometric Support Vector Machines

7.1.1 Datasets and Experimental Environment

All results presented in this section were obtained using double cross-validation procedure [52, 53, 54]. The double cross-validation is a very rigorous scheme for assessing a classification model's performance. Here, we evaluate the generalization performances of the SVM models by using the double cross-validation procedure which is structured as the two loop algorithm. In the outer loop, the data set is separated into J_1 roughly equal-sized parts (in this paper $J_1 = 5$ was used in all the experiments). Each part is held out in turn as the test set, and the remaining 4 parts are used as the training set. In the inner loop, J_2-fold cross-validation is performed over the training set only to determine the best values of hyper-parameters (here, $J_2 = 5$). The best model obtained in the inner-loop is then applied on the test set. The double cross-validation procedure ensures that the class labels of the test data will not be seen when tuning the hyper-parameters, which is consistent with the real-world applications scenario. Obviously such a rigorous procedure is done in many runs, but if the main goal is to compare different classification models on the same data sets and under same conditions fairly, double cross-validation must be used.

First, the datasets were normalized by linear transformation of the feature values into the $[0, 1]$ range. Then, the training process, that involved searching for the best model parameters using a grid search method, was performed. The parameters were selected among 64 possible combinations of the regularization parameter C and γ which is a coefficient of the Gaussian kernel $\varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2}$. There were 8 possible values for parameter C

$$C \in \{4^n\}, \quad n = -2, \ldots, 5, \tag{7.1}$$

and 8 possible values of γ

$$\gamma \in \{4^n\}, \quad n = -5, \ldots, 2. \tag{7.2}$$

The tolerance parameter ε used in the stopping criterion was set to $\varepsilon = 10^{-3}$ for L1 SVM and L2 SVM algorithms. As it was shown in [6], for SMO based algorithms the value of ε does not affect the accuracy or the time performance significantly. In other words, neither decrease of ε improves accuracy nor reasonable increase of its value speeds up the training procedure in a way that could change the results substantially. Therefore, we believe that this setting is a good trade-off between accuracy and the time required to train the model. All geometric algorithms presented here (i.e. BVM, SphereSVM and MNSVM) use a stopping criterion that is based on geometrical properties of the MEB problem. Since this stopping criterion is different than the one used in SMO algorithms we had to use a different strategy for determining correct value of the parameter ε. Therefore, the heuristic proposed by Tsang was used. This heuristic allows to estimate ε parameter based on the value of the regularization coefficient C

$$\varepsilon = \frac{\frac{2 \cdot 10^{-6}}{\tau + 1} + \frac{1}{N_S C}}{\tau + 1 + \frac{1}{C}}, \tag{7.3}$$

where N_S is the expected number of support vectors (we assumed that $N_S = 15000$) and $\tau = k(\mathbf{x}, \mathbf{x}) = 1$. In our case, values of ε were in the range $[10^{-6}, 10^{-3}]$ depending on the

value of C (smaller ε for larger C). More information regarding dependency of the ε parameters for both methods can be found in [37, 55].

The selection of the best model's parameters has been done by using 5-fold cross-validation applied to the previously selected training sets. After the best parameters were chosen, one additional SVM model was trained using entire training dataset. Finally, this model was assessed on the test dataset.

The double CV experimental environment used here is the only objective procedure for comparing performances of various classification algorithms and it is suggested as the required environment for the models comparisons in the future.

The datasets used in our experiments and the results obtained are "divided" into three groups dubbed as small, medium and large here. Note, that these are mostly the same datasets as in [9] where BVM algorithm has been introduced. However, the experimental environment in [9] was not as strict as the double CV used here. The three adjectives do not necessarily describe the size of datasets only. They also include the complexity of the classification tasks which is usually reflected in the percentage of the data becoming the support vectors as well as in the number of classes m_C because the number of classification models which have to be designed equals $\frac{m_C(m_C-1)}{2}$ for a pairwise multi-class classification. All the datasets can be downloaded from the LIBSVM[1] and LibCVM[2] sites.

7.1.1.1 Visualization of Statistical Properties

The measurements presented in this section are not precise and are subject to uncertainty. In order to mark this uncertainty, various visualizations shown below contain error bars representing the standard error (SE_x) of the sample mean estimate $\mu_x = E[X]$. The standard error is calculated as follows

$$SE_x = \frac{\sigma_x}{\sqrt{n}}, \tag{7.4}$$

[1] available at http://www.csie.ntu.edu.tw/ cjlin/libsvm/
[2] avaliable at http://c2inet.sce.ntu.edu.sg/ivor/cvm.html

Dataset	# of classes	Dimension	# of patterns
small datasets			
iris	3	4	150
wine	3	13	178
sonar	2	60	208
glass	6	9	214
breast-cancer	2	10	638
diabetes	2	8	768
medium datasets			
optdigits	10	64	5,620
satimage	6	36	6,435
usps	10	256	9,298
pendigits	10	16	10,992
reuters	2	8315	11,069
letter	26	16	20,000
large datasets			
adult	2	123	48,842
w3a	2	300	49,749
shuttle	7	7	58,000
web (w8a)	2	300	64,700
ijcnn1	2	22	141,691
intrusion	2	127	5,209,460

Table 7.1: Datasets used in experiments, number of classes, number of features (dimension) and number of samples.

where $\sigma_x = \sqrt{E[X^2] - (E[X])^2}$ is the standard deviation and n is the probe size.

7.1.2 Performance of the Sphere Support Vector Machines

In this section we present comparison of our algorithms to both classical L1 and L2 SVMs and the BVM algorithm. The LIBSVM [4] software is used as the reference implementation of the L1 and L2 SVMs, whereas the BVM implementation is taken from the LibCVM [7, 9] package. We compared three different versions of SphereSVM

- SphereSVM – This version uses the same probabilistic speedup settings as the BVM

algorithm. In each iteration the algorithm is searching for a violator among 59 candidate vectors and this search is repeated up to 10 times.

- SphereSVM-590 – This configuration preserves the same stopping probability as the BVM and SphereSVM algorithms but it uses new drawing scheme proposed in Section 5.1. In each iteration up to 590 attempts of finding a violating vector are performed. In other words, the algorithm stops when the probability of finding appropriate violator $\tilde{\mathbf{x}}_v$ is estimated to be less than $\frac{1}{590}$.

- SphereSVM-100 – This configuration is very similar to SphereSVM-590 – the algorithms performs at most 100 attempts of finding a violator. SphereSVM-100 is more likely to stop prematurely therefore it is supposed to have lower accuracy but better time performance.

Our experiments were performed using 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. Although the implementation of the algorithms that we show in this paper does not support multi-threaded execution, we utilized the multi-core environment by decomposing the nested cross-validation procedure into several independent processes. In other words, double cross-validation was performed by parallel execution of the independent training and testing processes.

7.1.2.1 Medium Datasets

Figure 7.1 shows accuracies obtained during nested cross-validation for *optdigits*, *satimage*, *usps*, *pendigits*, *routers* and *letter* datasets. It can be readily seen that the accuracies of all six models are very similar. More precisely, they are equal within the standard error for four datasets, namely *optdigits*, *satimage*, *pendigits* (not for SphereSVM-100) and *reuters*. For *usps* dataset only, L1 and L2 SVMs are just slightly better (less than 0.5%) than SphereSVM, and for the *letter* data only SphereSVM-100 is below the standard error difference for

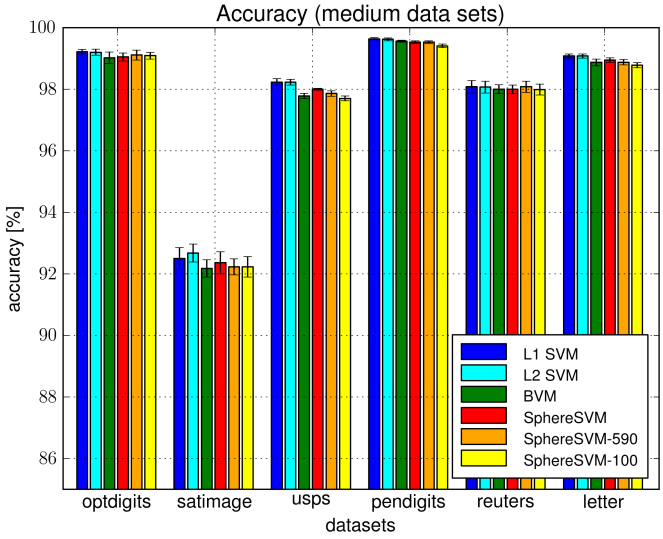

Figure 7.1: Medium datasets – accuracy obtained during nested cross-validation. For each dataset the bars represent the accuracy of (bars from left to right): L1 SVM, L2 SVM, BVM, SphereSVM, SphereSVM-590 and SphereSVM-100.

about 0.4%. As for the later, this tiny difference in accuracy is compensated by faster training stage. Overall, we can say that the accuracy of the SphereSVM-100 seems to be competitive to the other algorithms.

Figure 7.2 presents the total time of the nested cross-validation procedure. One can readily notice that original BVM is slower than SphereSVM which consecutively is slower than SphereSVM-590 and SphereSVM-100 algorithms. One can also see that for smaller datasets Sphere-100 performs the best in terms of training time followed by LIBSVM's implementation of L1 SVM and SphereSVM-590.

The training time for the optimal parameters C and σ is shown in Figure 7.3. All versions of SphereSVM algorithm are always faster than both, BVM and traditional SVMs based on SMO approach. SphereSVM is approximately 10-60% faster than L1 SVM from LibSVM library. But even greater improvement can be observed for SphereSVM-100 algorithm. Here, the speedup is in range 60%-300% (and what is interesting – the larger the dataset is the larger the speedup is).

The average percent of support vectors (calculated as the average percent of support

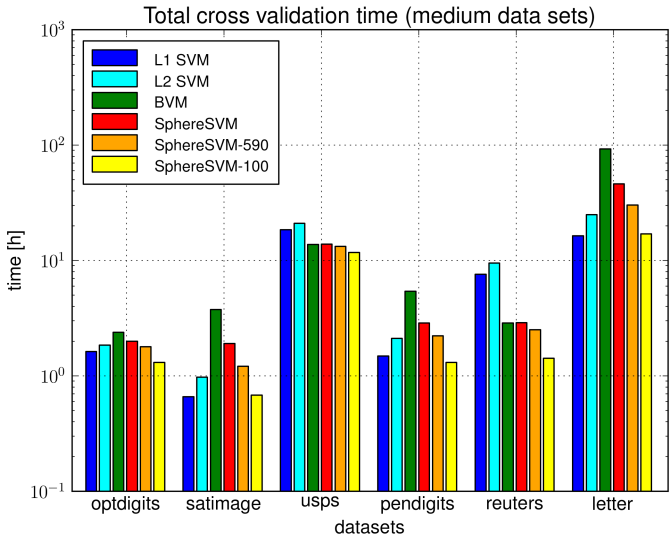

Figure 7.2: Medium datasets – total nested cross-validation time. For each dataset the bars represent the nested cross-validation time of (bars from left to right): L1 SVM, L2 SVM, BVM, SphereSVM, SphereSVM-590 and SphereSVM-100.

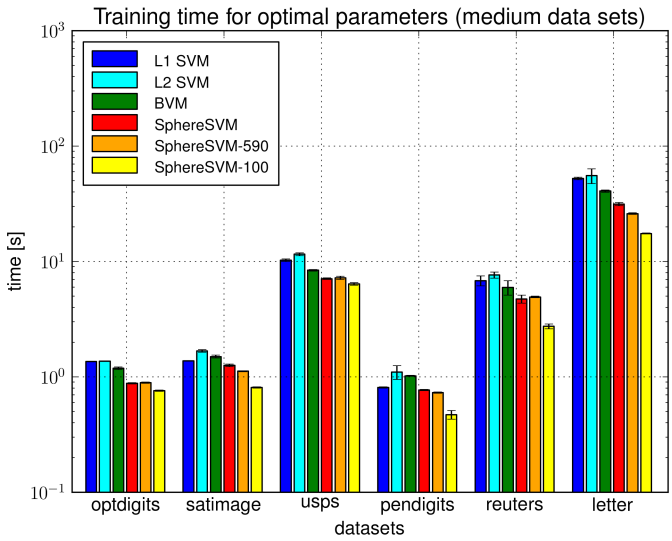

Figure 7.3: Medium datasets – training time for optimal parameters. For each dataset the bars represent the training time of (bars from left to right): L1 SVM, L2 SVM, BVM, SphereSVM, SphereSVM-590 and SphereSVM-100.

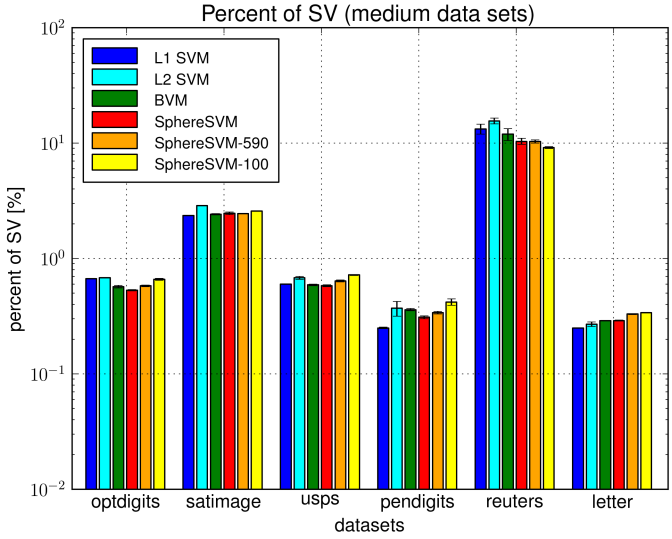

Figure 7.4: Medium datasets – average percent of support vectors for each of the models obtained in one-vs-one training. For each dataset the bars represent the model size (bars from left to right): L1 SVM, L2 SVM, BVM, SphereSVM, SphereSVM-590 and SphereSVM-100.

vectors of the models resulting from the one-vs-one training) is presented in Figure 7.4. All algorithms have similar percentage of support vectors. Since the time required to evaluate the decision function is proportional to the size of the model, we can expect that none of these methods will have significant advantage when it comes to the class prediction speed.

7.1.2.2 Large Datasets

Figures 7.5, 7.6, 7.7 and 7.8 show results of the nested cross-validation obtained for large datasets *adult*, *w3a*, *shuttle*, *web*, *ijcnn1* and *intrusion*. Note that for the *intrusion* dataset we present only results obtained by algorithms based on enclosing ball approach, including SphereSVM. The reason is simple, both L1 and L2 SVM were unable to complete the learning process in a reasonable time. We decided to abort their training after approximately 60 hours because none of the algorithms were able to finish cross-validation even for a single set of parameters. Since our nested cross-validation procedure consist of searching for an

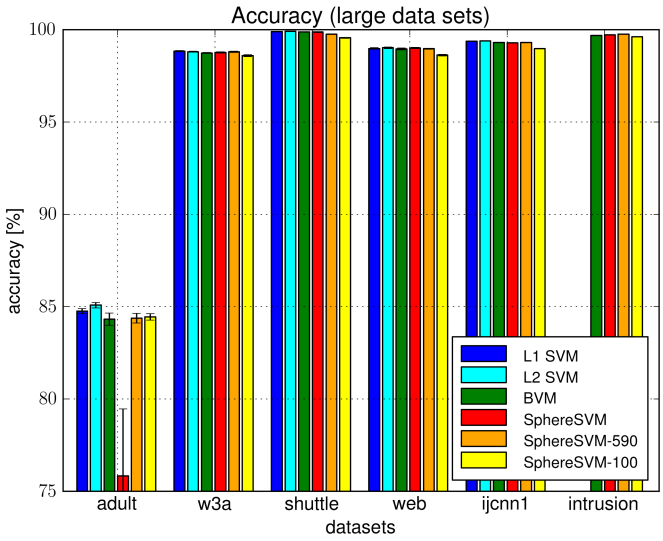

Figure 7.5: Large datasets – accuracy obtained during nested cross-validation. For each dataset the bars represent the accuracy of (bars from left to right): L1 SVM, L2 SVM, BVM, SphereSVM, SphereSVM-590 and SphereSVM-100.

optimal set of parameters among 64 combinations of C and γ values, we could roughly estimate that the learning process would not have finished within 160 days, which is a huge difference compared to 3 days required by SphereSVM- 590.

Figure 7.5 shows the accuracy obtained by the nested cross-validation procedure for large datasets. For *w3a*, *shuttle*, *web*, *ijcnn* and *intrusion* there is no significant difference between the methods (except for SphereSVM-100). The accuracy of the SphereSVM-100 is approximately 0.3% lower compared to other algorithms but, it is usually more than 10 times faster than the other methods (see Figure 7.6). One can notice that SphereSVM achieved much lower accuracy for the *adult* dataset. The reason for that is too big value of the tolerance parameter ε. We obtained competitive accuracy after decreasing its value to 10^{-6}. This is the only time when heuristic used by Tsang for determining the value of ε failed.

The learning times presented in Figure 7.6 allow to conclude that all versions of the SphereSVM are faster than their predecessor BVM (more precisely, the SphereSVM-590 is usually 4 to 5 times more efficient than BVM). Moreover the capabilities of the SphereSVM

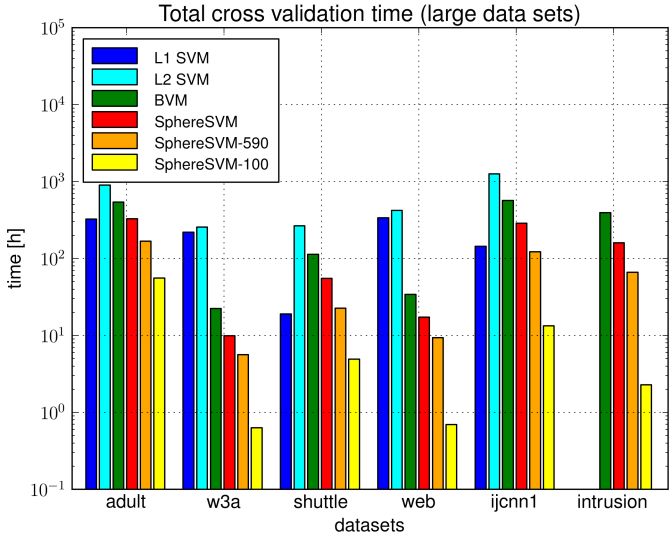

Figure 7.6: Large datasets – total nested cross-validation time. For each dataset the bars represent the nested cross-validation time of (bars from left to right): L1 SVM, L2 SVM, BVM, SphereSVM, SphereSVM-590 and SphereSVM-100.

approach are even more evident in comparison of SphereSVM-100 and BVM. The former is up to two orders of magnitude faster than the later (although, as mentioned earlier, its accuracy is approximately 0.3% lower). Furthermore, SphereSVM approach is more than competitive to L1 SVM. SphereSVM-590 is at least as fast as L1 SVM (except for the *shuttle* dataset) and in case of datasets *w3a* and *web* it is even more than 35 times faster.

The training time for optimal parameters is presented in Figure 7.7. Similarly as in the case of medium datasets, SphereSVM-590 and SphereSVM-100 are always faster than BVM, L1 and L2 SVM. Furthermore, SphereSVM-100 shows outstanding speedup when compared to LibSVM implementations (for example 50-times time improvement for *web* dataset). Unfortunately, the price of this speedup is a slightly lower accuracy.

As for the models' sizes, Figure 7.8 shows that the models generated by all three SphereSVM algorithms are smaller compared to the models obtained by BVM algorithm. This is caused by the difference in the way how both algorithms update the weight vector $\boldsymbol{\alpha}$. The update scheme applied in SphereSVM allows reduction of the number of support vectors (removal of the support vector is performed in line 10 of the Algorithm 3 whenever

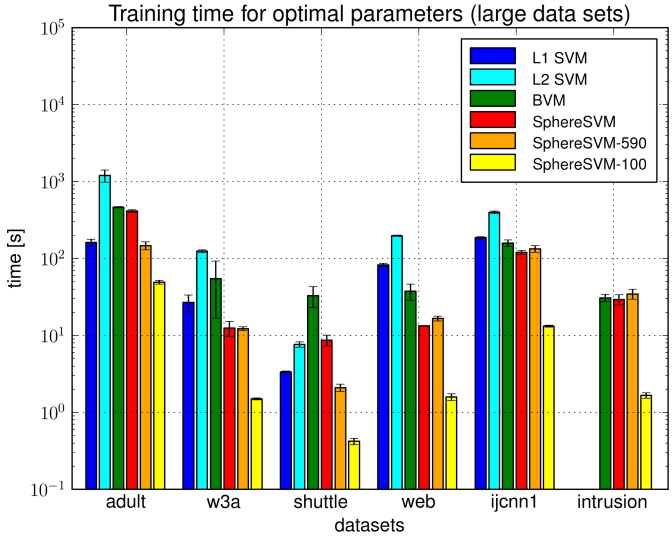

Figure 7.7: Large datasets – training time for optimal parameters. For each dataset the bars represent the training time of (bars from left to right): L1 SVM, L2 SVM, BVM, SphereSVM, SphereSVM-590 and SphereSVM-100.

$\beta = \alpha_u$). In contrast, BVM algorithm cannot remove a support vector from the coreset; once a vector $\tilde{\mathbf{x}}_i$ becomes a support vector it is not possible to decrease its weight α_i to 0 (although α_i value may asymptotically approach to 0).

Figure 7.9 shows how the training time of SphereSVM algorithm depends on the complexity of the dataset, measured as the number of support vectors required to build the model (SphereSVM-590 algorithm was used). A linear correlation between the training time and the number of support vectors[3] is clearly evident.

7.1.2.3 Draw Scheme for SphereSVM

From all the Figures presented so far it is easy to realize that SphereSVM-100 (SphereSVM performing at most 100 attempts to find a vector violating the stopping condition) is by far the fastest algorithm while still producing similar accuracy as the other methods. This is particularly pronounced for large datasets (*web*, *w3a* and *intrusion*) where the training

[3] linear regression of the logarithm of the model size vs. the logarithm of the training time was performed (p-value=0.007 and R^2=0.54).

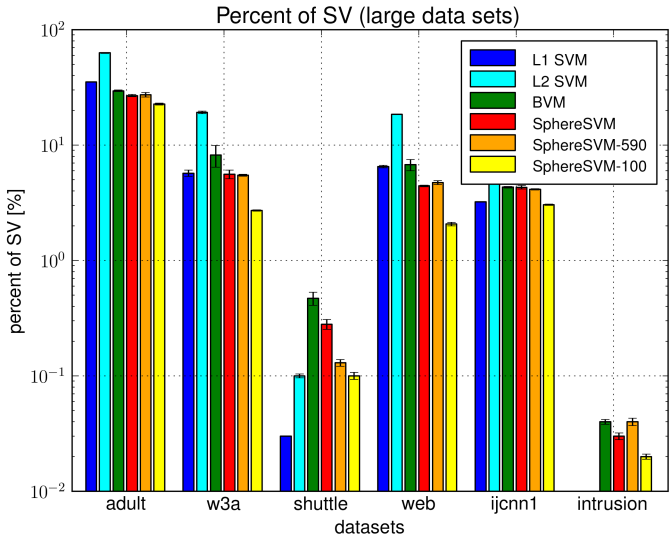

Figure 7.8: Large datasets – average percent of support vectors for each of the models obtained in one-vs-one training. For each dataset the bars represent the model size (bars from left to right): L1 SVM, L2 SVM, BVM, SphereSVM, SphereSVM-590 and SphereSVM-100.

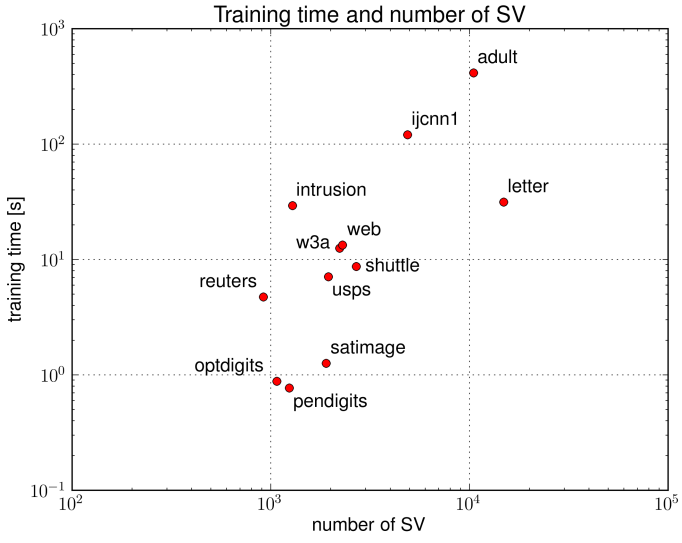

Figure 7.9: Dependency of the training time of the SphereSVM algorithm upon the number of support vectors for best hyper-parameters values

speedup is up to the three orders of magnitude. In the case of *intrusion* both classic SVM algorithms (L1 and L2 implemented in LIBSVM) never converged and the speedup cannot be expressed.

This raises a question what is a good number of draws for SphereSVM. Figures 7.10, 7.11 and 7.12 present dependencies of accuracy, training time and percent of support vectors upon the number of draws for medium datasets. By "number of draws" we mean the maximal number of random vector draws N_d performed in one iteration (in other words, in each iteration, the algorithm performs up to N_d draws from the entire dataset until it finds a violator $\tilde{\mathbf{x}}_v$ laying farther than $(1 + \hat{\varepsilon})R$ from the center). The algorithms dubbed SphereSVM-590 and SphereSVM-100 (both performing respectively up to 590 and 100 attempts to find a stopping criterion violator) presented in the Section 7.1.2 are the examples of the new draw scheme presented here.

Figure 7.10 shows the dependency between the classification accuracy and the number of draws for *optdigits*, *pendigits*, *reuters* and *satimage* datasets. It can be observed that increasing N_d beyond 100 does not affect the performance of the algorithm. Although, for these datasets even $N_d = 100$ seems to be enough to obtain maximal performance, one can expect that for more complex datasets (having more support vectors) this number should be increased. This expectancy is partially confirmed by the results presented in Figure 7.5, where for the largest datasets the accuracy of the SVM model trained with parameter $N_d = 100$ is slightly smaller than for the rest of the models.

Figure 7.11 visualizes the dependency between the nested cross-validation time and the number of random draws. It is an interesting observation that the training time increases linearly when N_d parameter is large enough ($N_d > 100$ for *optidigits*, *pendigits* and *satimage* and $N_d > 600$ for *reuters*).

The results presented in Figure 7.12 suggest that, for *optdigits*, *pendigits* and *satimage*, increasing the number of draws beyond 100 does not affect the size of the model. The exception here is *reuters* dataset, which achieves stable number of support vector for

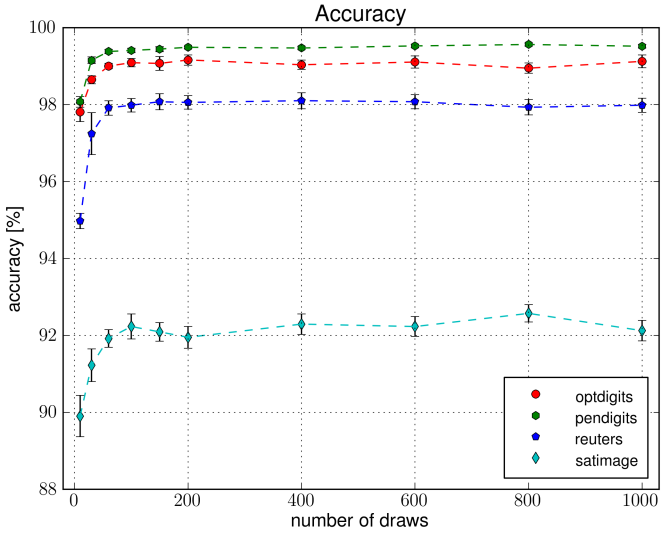

Figure 7.10: Dependency of the classification accuracy upon the maximal number of draws allowed during one iteration.

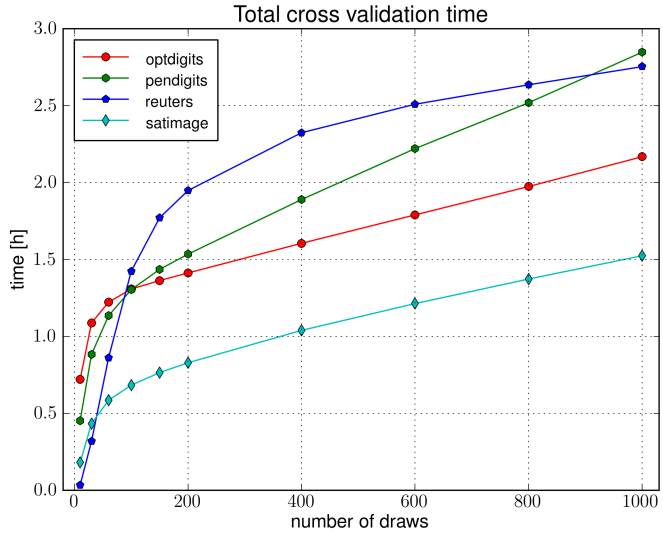

Figure 7.11: Dependency of the total nested cross-validation time upon the maximal number of draws allowed during one iteration.

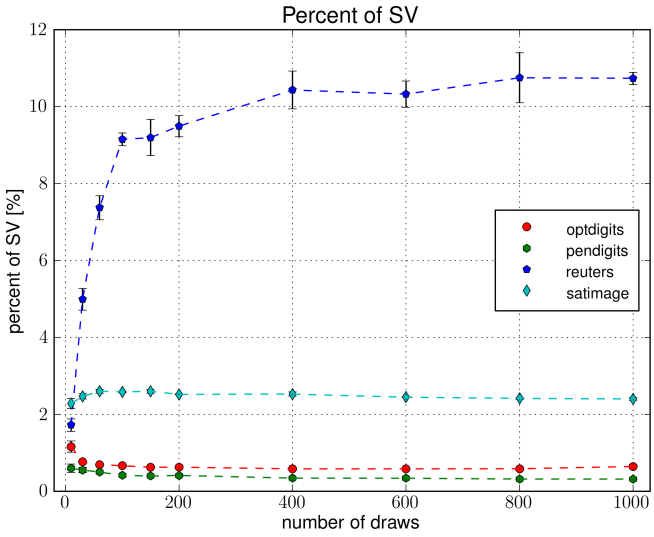

Figure 7.12: Dependency of the percent of support vectors upon the maximal number of draws allowed during one iteration.

$N_d \gg 100$.

7.1.3 Performance of the Minimal Norm SVM

Our experiments were performed using a computer cluster equipped with E5520 Intel Xeon CPUs. The double cross-validation was performed by parallel execution of the independent training and testing processes. We used the LIBSVM [4] package as the reference implementation of the L1 and L2 SVMs based on SMO approach. The BVM implementation is taken from the LibCVM [7, 9] software. We applied the same double cross-validation procedure as the one described in Section 7.1.1.

Figure 7.13 shows the total time of the nested cross validation procedure. Results allow us to conclude that MNSVM is approximately two times more efficient than BVM. In case of datasets *w3a* and *web* our approach is even more than 35 times faster. Note that we do not present results for L1 and L2 SVM obtained for the *intrusion* dataset. The reason is that, both SVM algorithms based on SMO were unable to complete the training within reasonable time. Since the algorithms were not able to finish cross-validation

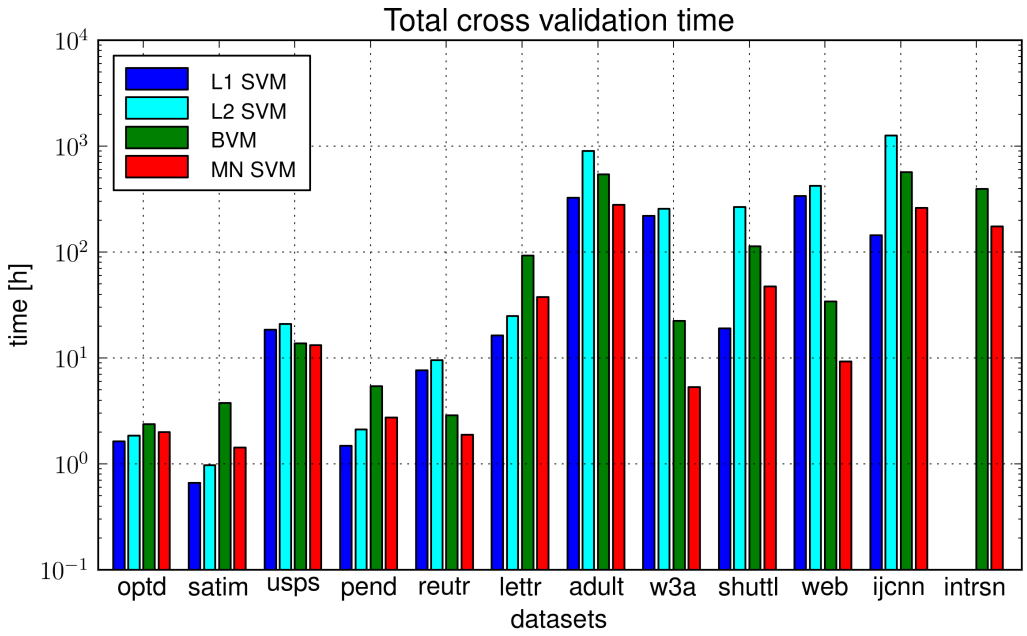

Figure 7.13: Total nested cross validation time. For each dataset the bars represent training time of (bars from left to right): L1 SVM, L2 SVM, BVM, MNSVM.

even for a single set of parameters C and γ, we were forced to abort their training (after approximately 60 hours). Simple calculations lead to rough estimation, that the learning process would not have finished within 160 days, which is a huge difference compared to 7 days required by MNSVM.

If we analyze the training times for optimal parameters (see Figure 7.14) we will see that almost always MNSVM algorithm is faster than its competitors. It loses with L1 SVM only for *shuttle* and *adult* (here, BVM is slightly faster as well). Usually, MNSVM is approximately two times faster than the fastest of the SMO-based algorithms – L1 SVM. In the case of the *web* dataset a sevenfold speedup was achieved.

Figure 7.15 presents accuracies obtained for all datasets during nested cross validation. It can be readily seen that the results of all four models are very similar in terms of the accuracy. The error rate for MNSVM is usually higher than the one of L1 and L2 SVM but the difference is almost always smaller than 0.5% (excluding *adult* dataset[4]). According to

[4]The reason for that is too big value of the tolerance parameter ε. We obtained competitive accuracy after decreasing its value.

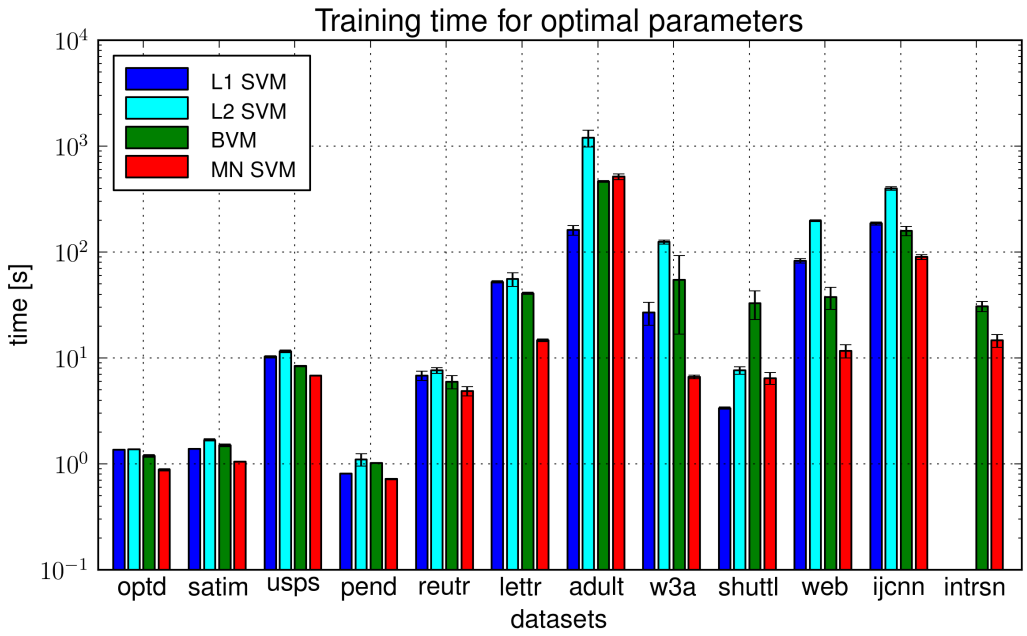

Figure 7.14: Training time for optimal parameters. For each dataset the bars represent training time of (bars from left to right): L1 SVM, L2 SVM, BVM, MNSVM.

the standard error range of our measurements – the significant difference can be observed for datasets *usps*, *letter*, *adult*, *shuttle* and *ijcnn1*. Here, both algorithms BVM and MNSVM have evidently higher error rate. BVM resulted in significantly better models only for datasets *letter* and *adult* compared to MNSVM, but on the other hand MNSVM was more accurate for *usps* dataset. Since both algorithms optimize the same cost function, we strongly believe that further experiments will lead to conclusion that the accuracy of both algorithms is the same. Fortunately, the small differences in accuracy are compensated by shorter training time required by our geometrical method.

The average percent of support vectors[5] is shown in Figure 7.16. All the models obtained by different algorithms have similar number of support vectors. It can be observed that the models created by MNSVM approach are always smaller compared to the ones generated by BVM algorithm. The reason for that is the way the update step is performed. Namely, MNSVM algorithm allows to reduce the number of support vectors

[5]By the percent of support vectors we mean the ratio of the number of non-zero coefficients α_i to the total number of training patterns used in one-vs-one training.

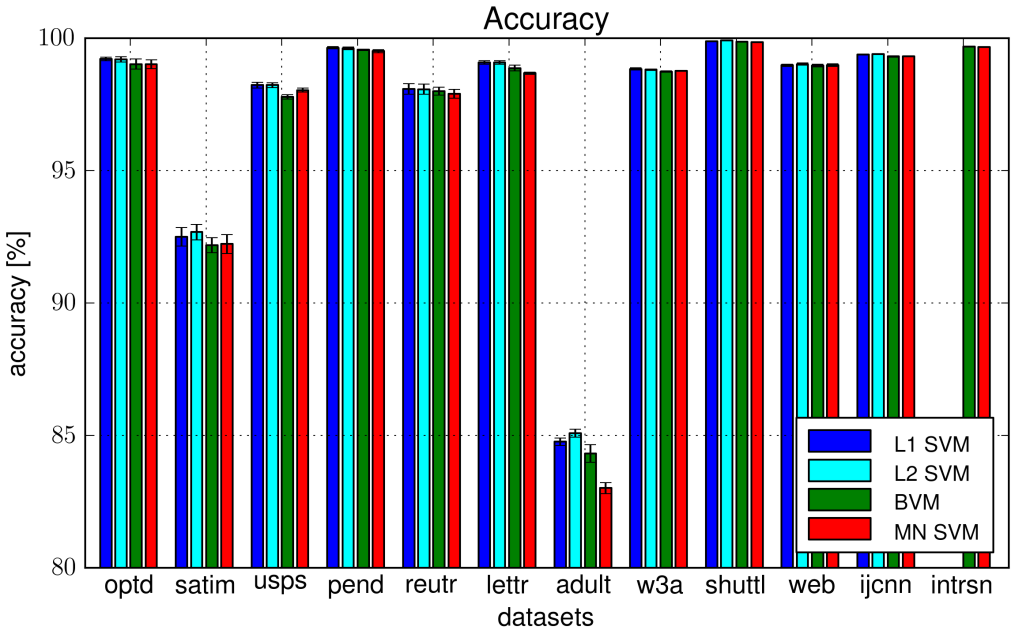

Figure 7.15: Accuracy obtained during nested cross validation. For each dataset the bars represent accuracy of (bars from left to right): L1 SVM, L2 SVM, BVM, MNSVM.

(by decreasing the weight α_u for the violator $\tilde{\mathbf{x}}_i$ to 0). In case of the BVM algorithm it impossible to remove a support vector from the coreset.

The performance of the MNSVM algorithm has been verified on artificial "checkers" dataset. This two-dimensional dataset consist of two classes that are distributed in form of the checker board having four rows and four columns. Using such simulated data allowed us to test the performance of the algorithm against extremely large training problems. Figure 7.17 shows how the training time depends upon the number of training samples (the number of the data varies from 100 to 10,000,000). The linear character of the time complexity can be observed when the number of training patterns is less than 30,000. When the size of the data is between 30,000 and 100,000 the training time grows disproportionately faster than the problem size. The reason for that phenomenon is the limited size of the kernel cache. When the number of support vectors grows above the maximal cache capacity then the overall cache performance decreases since some of the kernel values must be discarded and the cache-misses become more frequent which

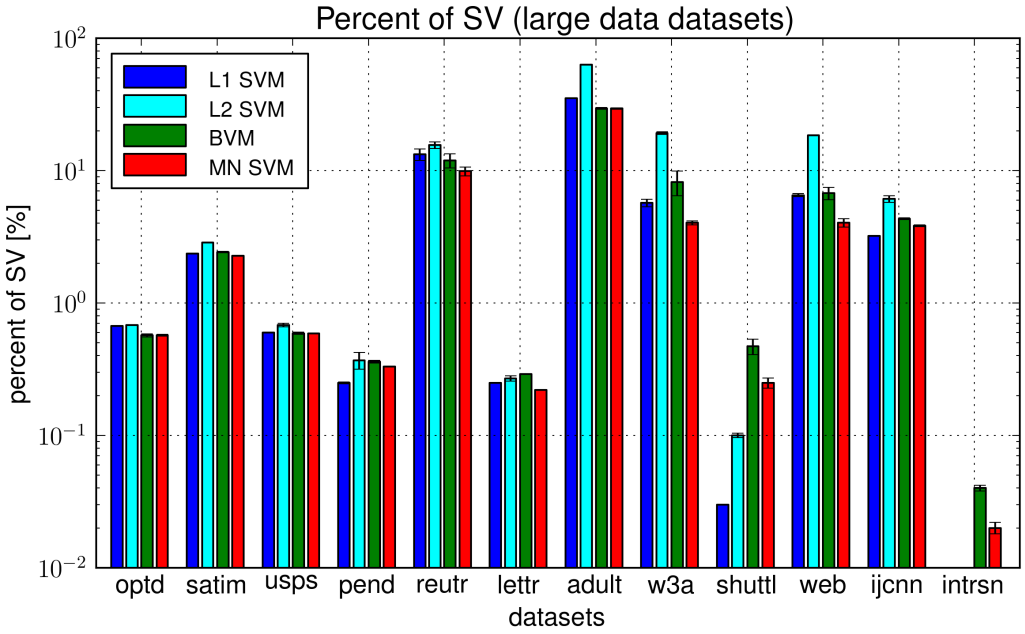

Figure 7.16: Average percent of support vectors obtained in one-vs-one training. For each dataset the bars represent the model size for (bars from left to right): L1 SVM, L2 SVM, BVM, MNSVM.

leads to additional kernel re-computations. When the data size grows above 100,000 the dependency between the number of training samples and the training time becomes sub-linear – the learning time grows slower than the increase of the dataset. For the number of training samples above 1,000,000 the learning time is approximately constant. This can be explained by the fact that the complexity of the MNSVM with probabilistic speedup technique is not dependent upon the size of the data. The number of iterations is limited by a function of the tolerance parameter ε.

7.1.4 Comparison of SphereSVM and Minimal Norm SVM

In this section we compare SphereSVM and MNSVM using the nested cross-validation procedure described in Section 7.1.1. Both presented here algorithms were embedded into LibCVM toolkit. To allow fair comparison, we use MEB-based stopping criterion for both methods.

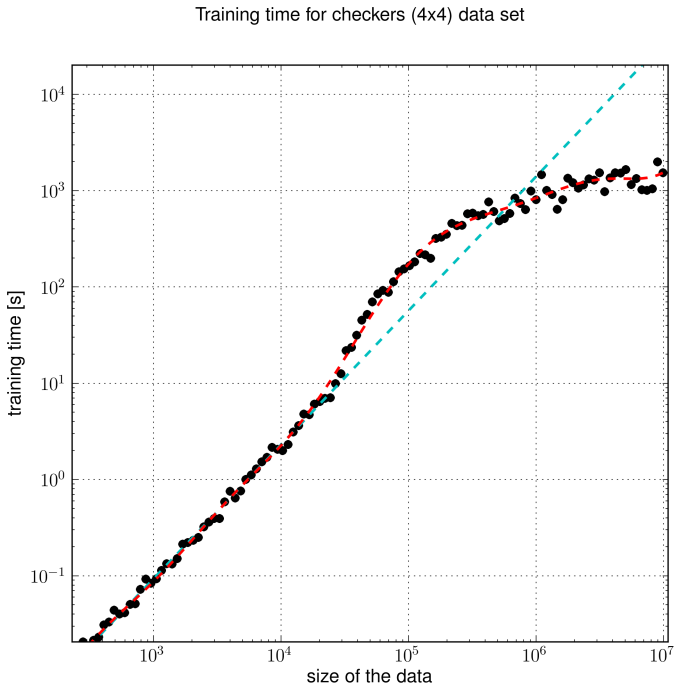

Figure 7.17: MNSVM training time for "checkers" data set.

The accuracy obtained by L1 and L2 SVM (LIBSVM) and BVM, SphereSVM, MNSVM (LibCVM) is presented in Figure 7.18. All these algorithms have similar performance although for *satimage*, *usps*, *letter* and *adult* the geometric algorithms seem to be slightly less accurate. In addition to comparing accuracies we have also run a series of Wilcoxon signed-rank tests [56] to perform pair-wise comparisons of algorithms' performance. The null-hypothesis is that the algorithms perform at the same accuracy level. The significance level was determined by a p-value of less than 0.05. Both SMO-based algorithms (L1 and L2 SVM) and GSVM implementation of MNSVM (see Section 7.8) are significantly more accurate than the geometric approaches that use MEB stopping criterion (p-values for all comparisons were within the range $p \in [0.0033, 0.0059]$). Nonetheless, the accuracy of SphereSVM and MNSVM is the same for all datasets except *adult* where high error rate for SphereSVM is a result of inappropriate adjustment of the tolerance parameter ε[6]. There

[6]The value of the parameter ε was determined by a heuristic proposed and implemented by Tsang in LibCVM tool.

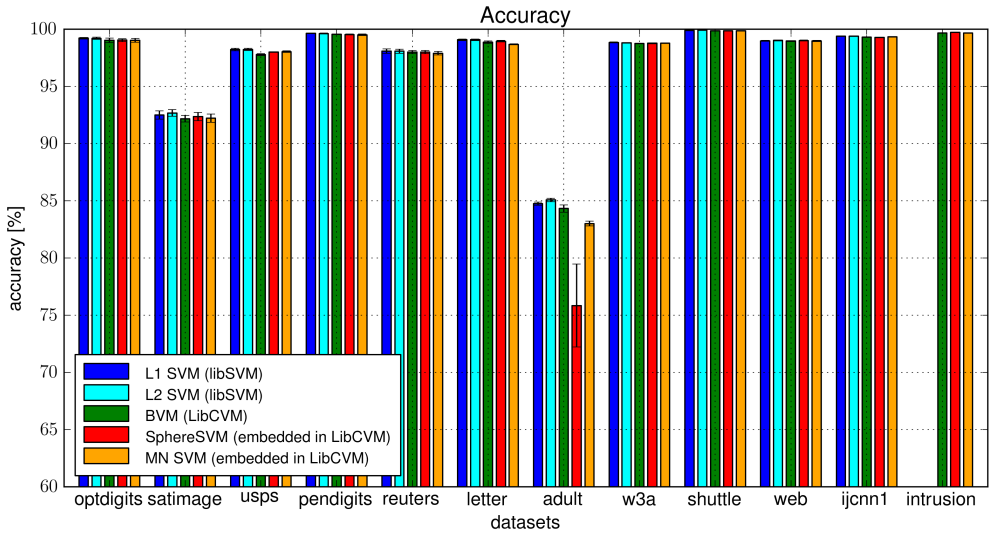

Figure 7.18: Comparison of accuracy obtained by various SVM algorithms. For each dataset the bars represent the accuracy obtained by L1 SVM, L2 SVM, BVM, SphereSVM and MNSVM (bars from left to right).

is no sufficient evidence to reject the null-hypothesis that the algorithms perform at the same level in case of comparison of SphereSVM and MNSVM with BVM method ($p = 0.16$ and $p = 0.42$ respectively).

Figure 7.19 depicts the total nested cross-validation time for the SVM algorithms. MNSVM is almost always faster than its predecessor SphereSVM. It is especially visible on the datasets like *web* or *reuters* where MNSVM is approximately two times faster. Moreover, both introduced in this dissertation algorithms, SphereSVM and MNSVM, are always faster than BVM.

Comparison of the training time for optimal parameters, that is presented in Figure 7.20, reveals that MNSVM is almost always the fastest SVM algorithm. Only in the case of *adult* and *shuttle* datasets it looses with L1 SVM and for *optdigits* and *reuters* it achieves the same performance as SphereSVM.

The percent of support vectors found by the algorithms is presented in Figure 7.21. The sizes of the models generated by all SVM algorithms seem to be similar. It is interesting that for larger datasets the models obtained by MNSVM are slightly smaller than the ones

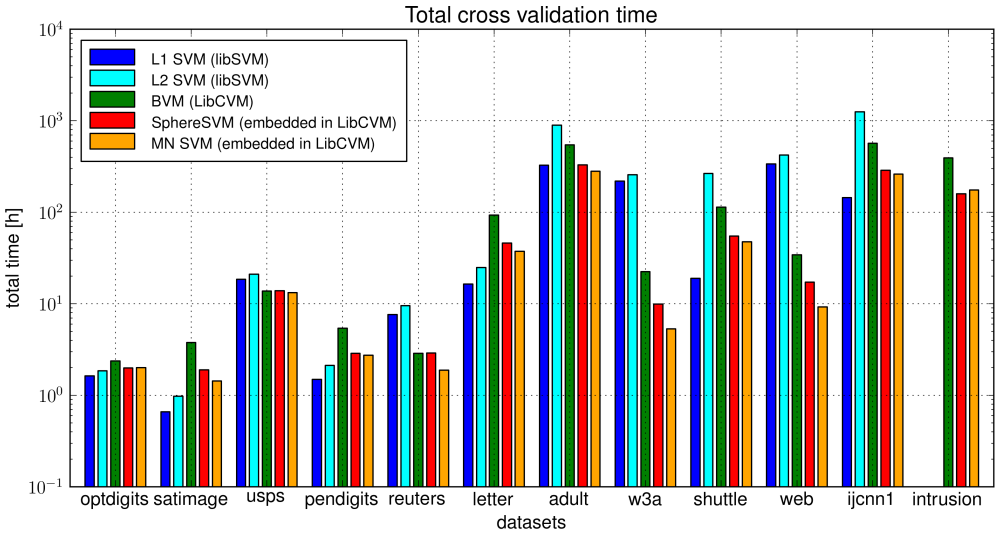

Figure 7.19: Total nested cross-validation time for various SVM algorithms. For each dataset the bars represent the cross-validation time obtained by L1 SVM, L2 SVM, BVM, SphereSVM and MNSVM (bars from left to right).

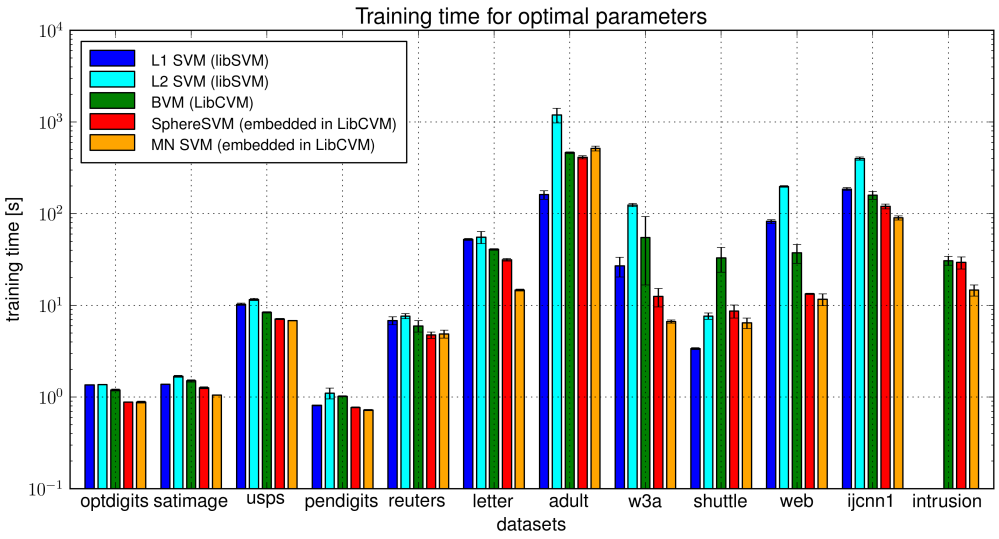

Figure 7.20: Training time for optimal parameters for various SVM algorithms. For each dataset the bars represent the learning time obtained by L1 SVM, L2 SVM, BVM, SphereSVM and MNSVM (bars from left to right).

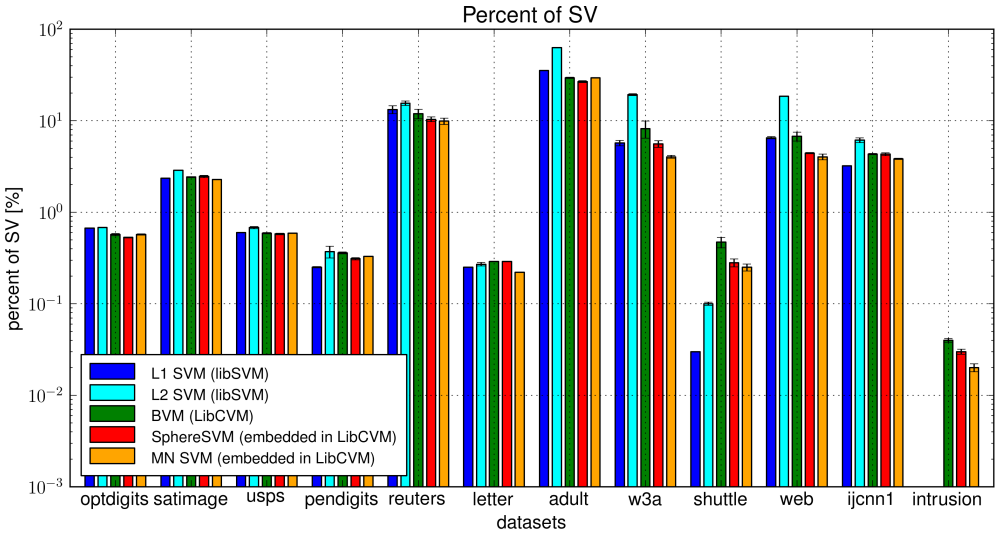

Figure 7.21: The model size for various SVM algorithms. For each dataset the bars represent the percentage of support vectors of the models generated by L1 SVM, L2 SVM, BVM, SphereSVM and MNSVM (bars from left to right).

generated by SphereSVM.

7.2 Geometric SVM without Bias

7.2.1 SphereSVM without Bias

The accuracy of the two versions of SphereSVM algorithm – with and without bias – is presented in Figure 7.22. There is no significant difference between these two methods except for the *satimage* dataset, where the approach without bias resulted in accuracy greater by 0.23% (but even this improvement is still within a standard error range).

The Figure 7.23 shows the total time required for nested cross-validation procedure. It is very difficult to draw any conclusions based on this plot. Neither of the algorithms seems to be faster than the other. The cross-validation time seems to be independent upon the size of the dataset. On the other hand it is unsettling that for *w3a* and *web* datasets SphereSVM without bias was more than 10 times slower than the original algorithm.

The training times for optimal parameters obtained from the cross-validation proce-

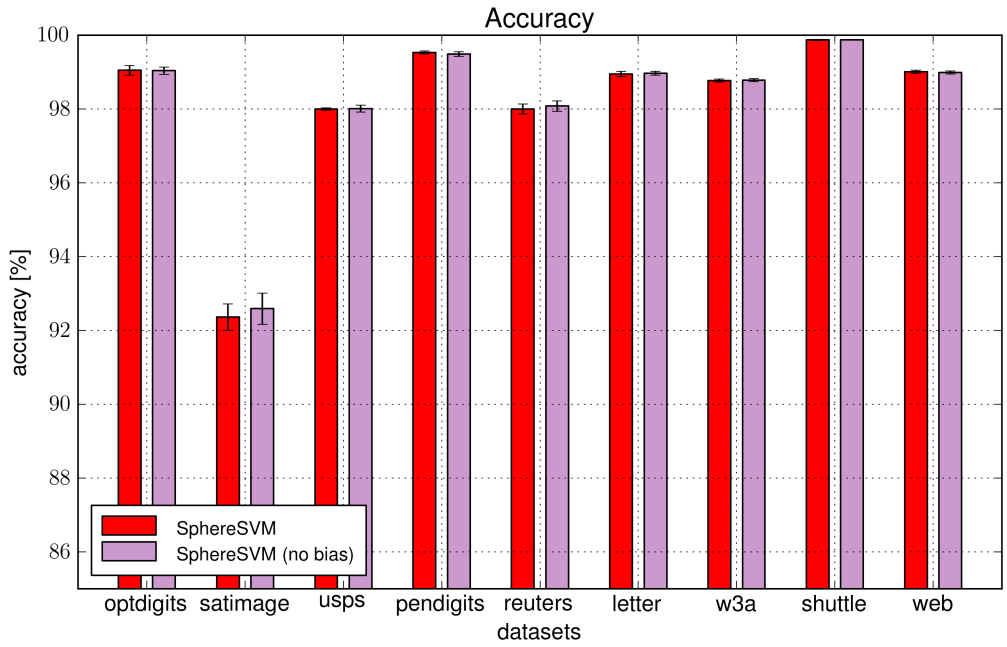

Figure 7.22: Accuracy of two SphereSVM variants (with and without bias) obtained during nested cross-validation.

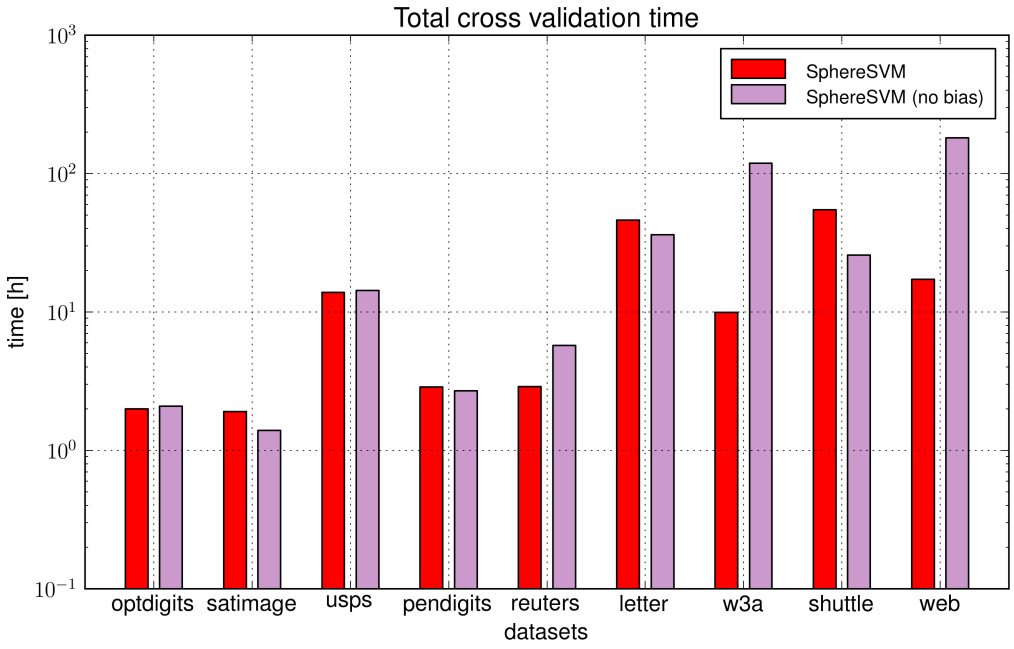

Figure 7.23: Total nested cross-validation time for two SphereSVM variants (with and without bias).

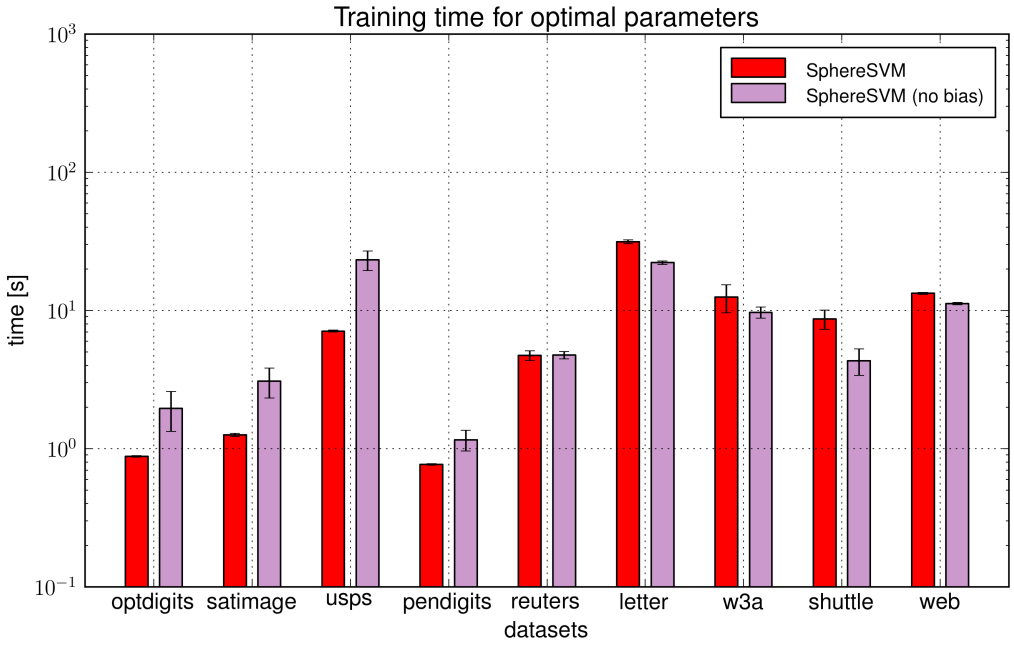

Figure 7.24: Training time for optimal parameters for two SphereSVM variants (with and without bias).

dure are presented on Figure 7.24. It is very interesting that for larger datasets (*letter*, *w3a*, *shuttle* and *web*) ShpereSVM without bias was faster, whereas for smaller datasets *optdigits*, *satimage*, *usps* and *pendigits*) its performance was not as good as the performance of the original algorithm. Moreover, it is noteworthy that the standard error of the training time is very often greater for the method not involving the bias term. This means that it may be more difficult to predict the training time required for this approach. These training time fluctuations may explain the worse cross-validation performance that was achieved by the algorithm for *w3a* and *web* datasets (see Figure 7.23).

Figure 7.25 compares the sizes of the models obtained by both versions of SphereSVM algorithm. For large datasets the method not using the bias term results in slightly smaller number of support vectors. The correlation between the results presented in Figures 7.23 and 7.25 is easily noticeable. Apparently, the smaller number of support vectors improved the time performance of the training procedure.

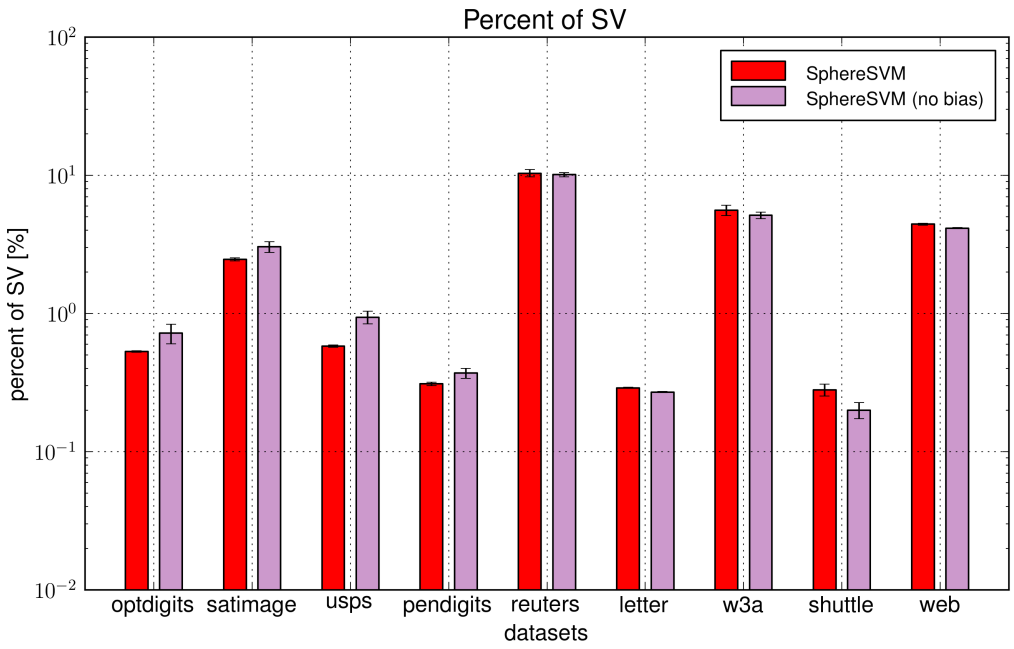

Figure 7.25: Average percent of support vectors for each of the models obtained in one-vs-one training (results obtained for two SphereSVM variants – with and without bias).

7.2.2 Minimal Norm SVM without Bias

The properties of MNSVM without bias were tested on several medium size datasets (*optdigits*, *pendigits*, *reuters*, *satimage*, *usps* and *letter*). Figure 7.26 shows the accuracy of the SVM model obtained on these data. Two versions of the algorithm were used – standard MNSVM and MNSVM without bias. For both methods the learning process was performed with optimal values of the training parameters C and σ. These values were determined in a cross-validation procedure. Moreover, the training was performed with different values of stopping parameter ε in order to check how this factor affects the accuracy and the computation time.

For relatively large values of ε (i.e. greater than 0.01) the models generated by MNSVM without bias seem to be much more accurate. When the value of ε decreases below 0.001 then the misclassification rate for both methods converges to approximately equal value (although for *optdigits* and *pendigits* datasets, SVM training without bias yields slightly

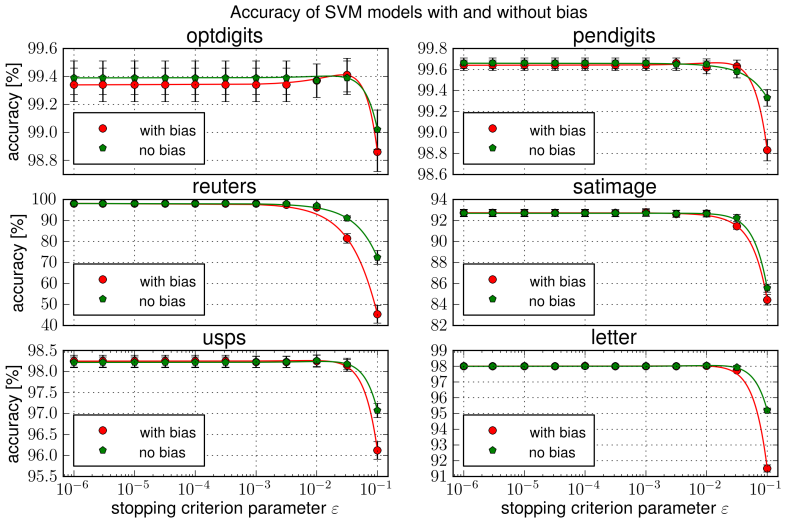

Figure 7.26: Accuracy obtained by two versions of MNSVM solvers (with and without bias) depending upon the value of the stopping criterion parameter ε.

better accuracy).

The training time for the optimal parameters of the Gaussian kernel is presented in Figure 7.27. For values of the parameter ε that are less than 0.001 the version of the MNSVM algorithm that does not use bias is always faster than the original method (approximately 10%). Furthermore, decreasing the value of ε below 0.001 does not affect the training time in a significant way – the learning time stays approximately the same (the only exceptions here are *reuters* and *letter* datasets where for the MNSVM with bias the computational time increases proportionally to $-\ln \varepsilon$).

The important conclusion resulting from Figures 7.26 and 7.27 is that by performing training with MNSVM without bias one may expect both: slightly better accuracy and shorter training time. Moreover, the reasonable value of the parameter ε for MNSVM learning is 0.001 since further decrease of ε does not improve the accuracy. Although, if one wishes to obtain very accurate model, then decreasing of ε is possible since it does not affect the training time in a significant way.

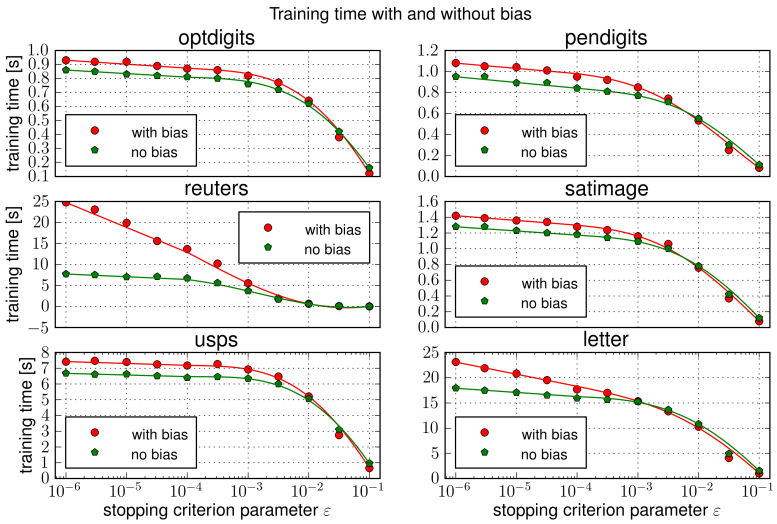

Figure 7.27: Training time for the optimal Gaussian kernel parameters obtained by two versions of MNSVM solvers (with and without bias) depending upon the value of the stopping criterion parameter ε.

7.3 Over-relaxation

7.3.1 Over-relaxation in SphereSVM

Experiments on the SphereSVM algorithm enhanced by over-relaxation technique were performed using nested cross-validation procedure with the same parameters as the ones described in Section 7.1.1. There are noticeable differences in execution time when compared with previously presented results. The reason for that is the fact that we used different hardware configuration during our experiments. The results presented below were obtained using a computer equipped with two Intel Xeon X5680 CPUs and 96 GB of RAM.

Values of parameter η were selected from the range $[1, 1.9995]$. According to our experiments, for $\eta \geqslant 2$ the convergence of the algorithm is jeopardized – this is especially noticeable when the parameter C is large. The reason is that when $\eta \geqslant 2$ then subsequent approximations of $\mathbf{c}$ obtained by the algorithm may move away from the origin. In other words, it is possible that $\|\mathbf{c}\|$ will be increasing instead of decreasing which contradicts to

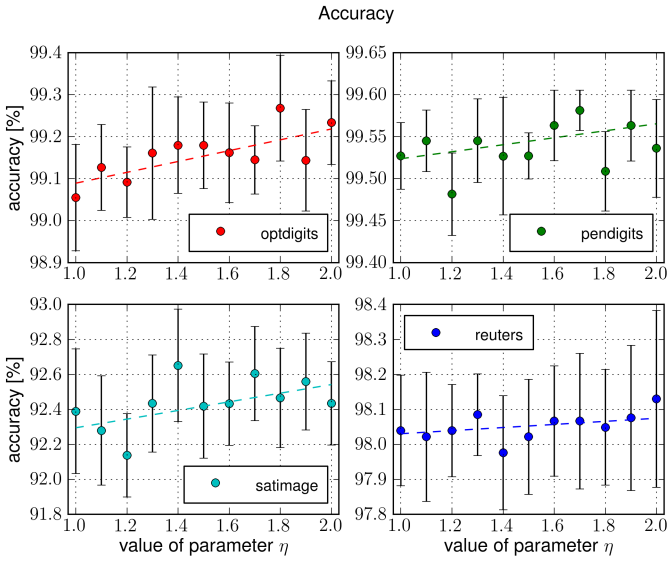

Figure 7.28: Accuracy of SphereSVM with over-relaxation depending upon parameter η.

the invariant of the proposed algorithm.

Figure 7.28 shows the accuracy of the over-relaxation method used with SphereSVM algorithm. The trend lines were calculated based on the accuracy obtained during 11 independent nested cross-validation runs for $\eta \in [1, 1.9995]$. The trend lines suggest that increase of the parameter η has a positive impact on the accuracy of the algorithm (p-value of all regression lines satisfies $p < 0.001$, moreover $R^2 = 0.52$ for *optdigits*, $R^2 = 0.23$ for *pendigits*, $R^2 = 0.13$ for *reuters* and $R^2 = 0.32$ for *satimage*). For instance, in the case of *satimage* dataset, setting this parameter to $\eta \approx 2$ improved the accuracy by 0.25% compared to the original method (having $\eta = 1$).

The dependency between the nested cross-validation time and the value of the parameter η is presented in Figure 7.29. Over-relaxation method increased the performance by 10-55%. Moreover, it is interesting that for *optdigits*, *pendigits* and *satimage* datasets the best performance was achieved for relatively large values of η (around $\eta \approx 1.98$) – see Figure 7.30.

The training time for optimal parameters, obtained by cross-validation procedure, was also improved by the over-relaxation technique (see Figure 7.31). Here, similarly to the

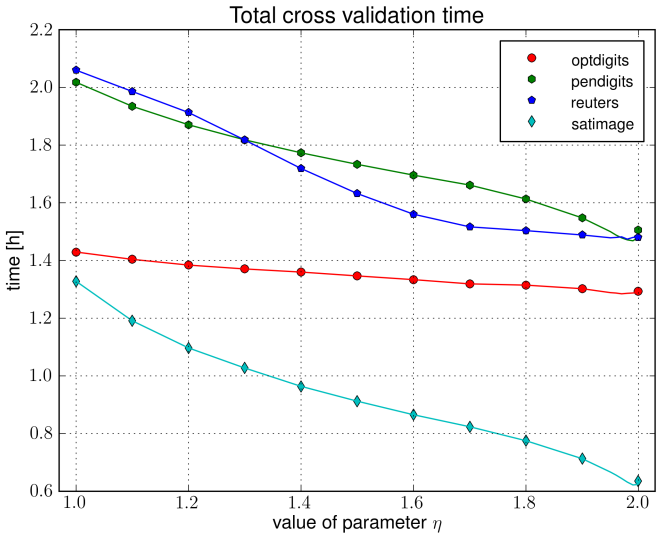

Figure 7.29: Nested cross-validation time of SphereSVM with over-relaxation depending upon parameter η.

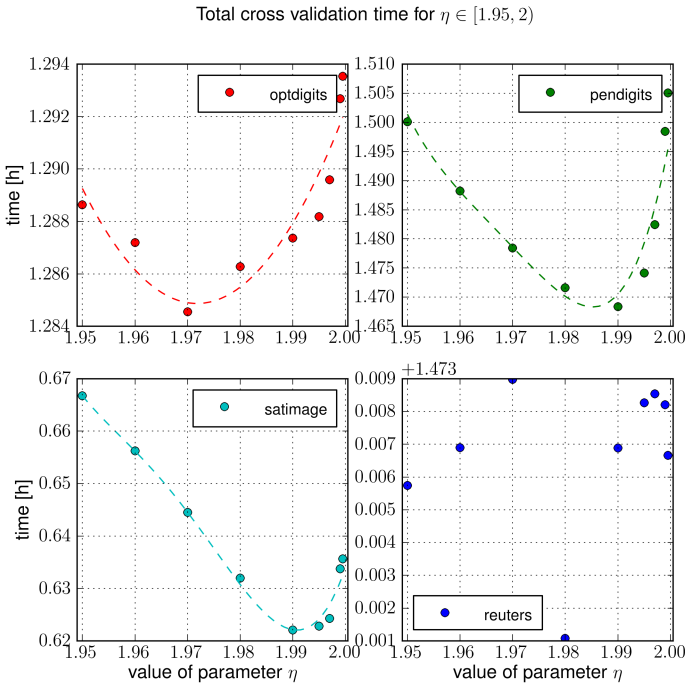

Figure 7.30: Nested cross-validation time of SphereSVM with over-relaxation depending upon parameter η.

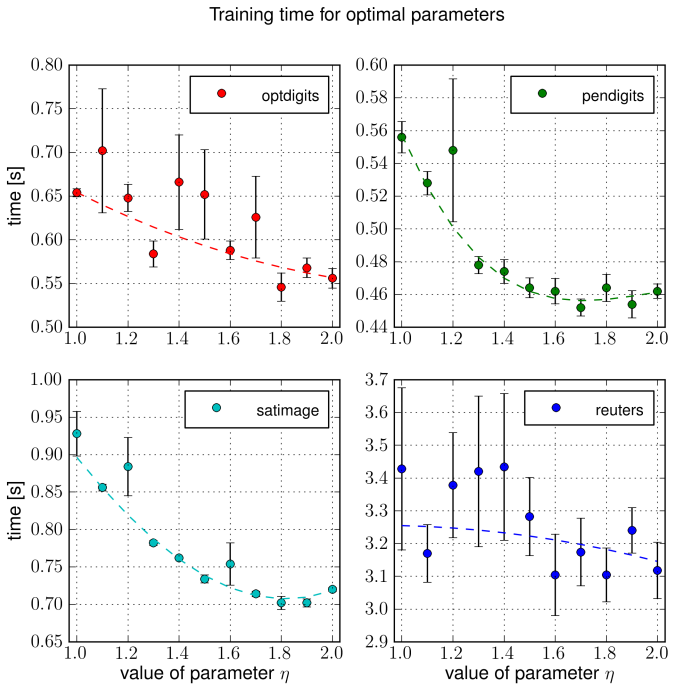

Figure 7.31: Training time for optimal parameters for SphereSVM with over-relaxation depending upon parameter η.

total cross-validation time, the optimal times were achieved for larger values of η.

Figure 7.32 shows the size of the model depending upon the parameter η. In case of *satimage* and *reuters* datasets, over-relaxation significantly decreased the number of support vectors.

7.3.2 Over-relaxation in Minimal Norm SVM

Properties of the over-relaxation method used with MNSVM algorithm were tested on several small-size datasets such as *breast-cancer*, *diabetes*, *glass*, *iris*, *sonar* and *wine*. First, the execution times of the nested cross-validation procedure were recorded for for different values of the parameter η . Then, the speedup was calculated as a percentage improvement in training time of the MNSVM with over-relaxation over the original MNSVM method.

Speedups achieved for various datasets and values of η are presented in Figure 7.33. For all training sets, the over-relaxation method with the value of η within the range

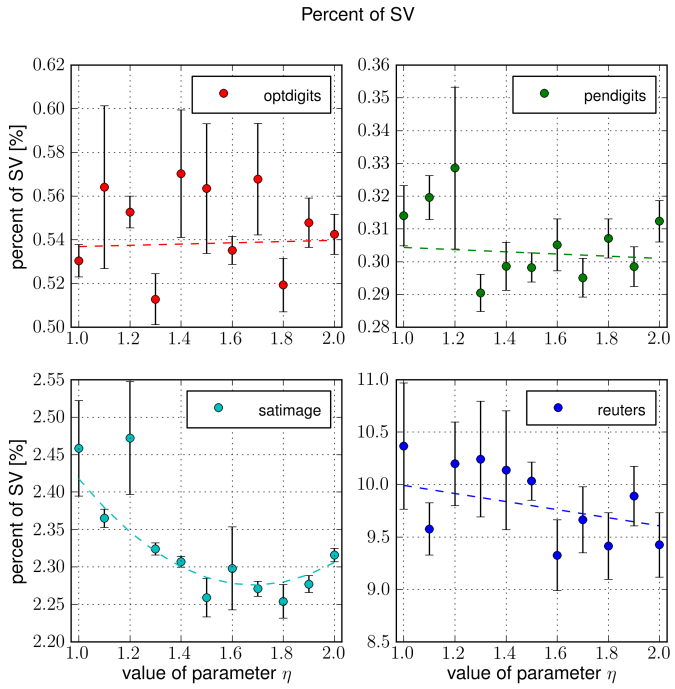

Figure 7.32: Percent of support vectors for SphereSVM with over-relaxation depending upon parameter η.

$\eta \in (1, 1.3)$ always resulted in performance improvement. The average speedup, marked with black solid line, allows to conclude that the reasonable values for the parameter η are somewhere around $\eta \approx 1.3$. However, one must not assume that $\eta \approx 1.3$ would be a good choice for all dataset. Each training set has its own optimum in a different spot. For example, using value of $\eta = 1.3$ for *breast-cancer* would result in almost no improvement. Similarly, for *diabetes* the value of η that would maximize the time performance is equal to $\eta \approx 1.75$ (which results in almost 40% speedup).

7.4 All-at-once Approach for Multi-class Problems

Below we compare two multi-class SVM training methods – one-vs-one with voting mechanism, used in LIBSVM and LibCVM, and all-at-once introduced in Section 5.4.1. The tests were performed on several multi-class datasets using nested cross-validation

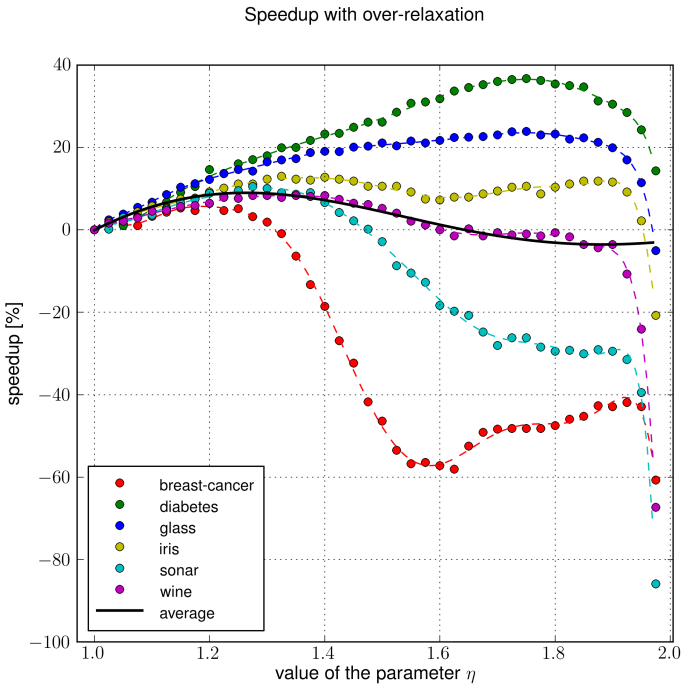

Figure 7.33: Dependency between speedup achieved by MNSVM algorithm using over-relaxation and the values of parameter η. The speedup was calculated based on the nested cross-validation times. The datasets are as follows (plots from top to bottom): *diabetes*, *glass*, *iris*, *wine*, *sonar*, *breast-cancer*. The solid black line represents the mean value of the speedups obtained for the datasets.

scheme described in Section 7.1.1. The hardware that was used during the experiments was composed of 13 server nodes equipped with four AMD Opteron 6282 SE processors and 256 GB RAM each.

The time required to complete nested cross-validation for both multi-class learning approaches is presented in Figure 7.34. It can be observed that the all-at-once learning method is much faster than the pairwise training approach. The achieved speedup is at least threefold for all the training sets except *usps*. For the *shuttle* dataset the difference is two orders of magnitude.

Unfortunately, this time improvement causes large accuracy decrease. Figure 7.35 shows the accuracy of the SVM models trained with all-at-once and pairwise methods. The one-versus-one method always has lower error rate. In the case of *shuttle* dataset, the

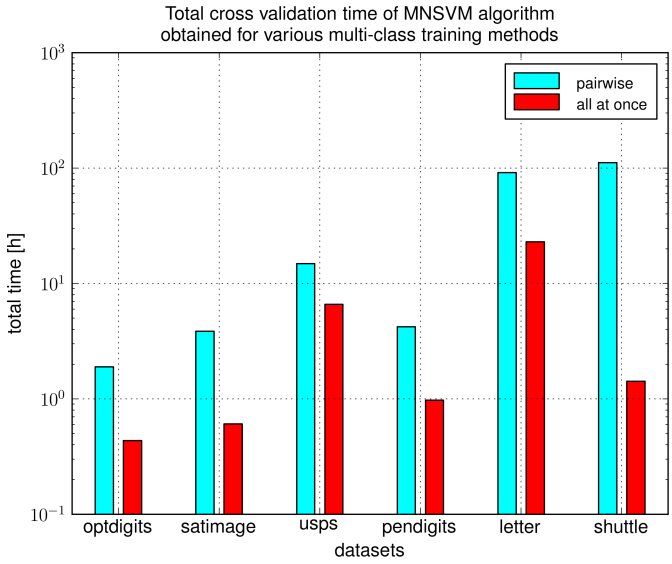

Figure 7.34: Total cross-validation time for all-at-once multi-class training. For each dataset, the bars represent the learning time obtained by pairwise (the left bar) and all-at-once method (the right bar).

difference is 7%. The difference in accuracy obtained by both algorithms does not seem to be dependent upon the number of classes. This suggest that the all-at-once method could be potentially applied to classification problems having large number of classes. This would allow to avoid training relatively large number of SVM models required by the pairwise approach.

The differences in the training time for optimal learning parameters are presented in Figure 7.36. Although all-at-once method is usually the fastest one, these results are not as impressive as in the case of total nested cross-validation training times. Best performance improvement were achieved for *shuttle* where multi-class approach is more than ten times faster than the one-vs-one training. Unfortunately, the cost of this speedup is much lower classification accuracy. The only case were pairwise classification method was more efficient was *pendigits* dataset – here, the all-at-once training was almost two times slower.

Figure 7.37 compares the percent of support vectors generated by both algorithms. Usually, the pairwise mutli-class method produces much large models (even two times

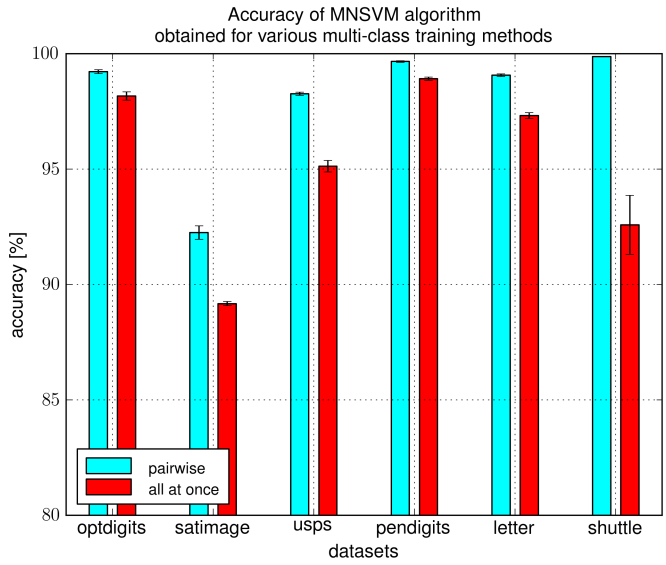

Figure 7.35: Accuracy for all-at-once multi-class training. For each dataset, the bars represent the accuracy obtained by pairwise (the left bar) and all-at-once method (the right bar).

Figure 7.36: Training time for optimal parameters for all-at-once multi-class training. For each dataset, the bars represent the training time obtained by pairwise (the left bar) and all-at-once method (the right bar).

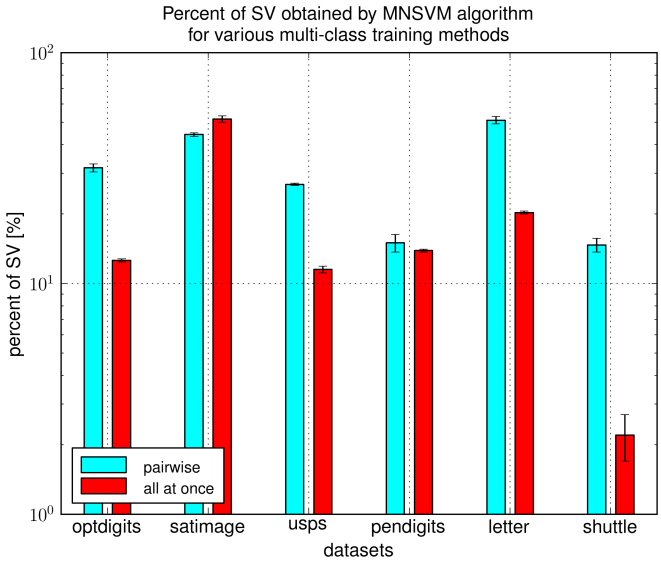

Figure 7.37: Percent of support vectors for all-at-once multi-class training. For each dataset, the bars represent the percent of support vectors obtained by pairwise (the left bar) and all-at-once method (the right bar).

larger for *optdigits*, *usps* and *letter* datasets, not mentioning *shuttle* where the model generated by all-at-once approach is seven times smaller). Since the time required to classify a sample is proportional to the number of support vectors, one may expect that the classification using models generated by all-at-once method will be faster compared to pairwise method.

7.5 Bias Evaluation Technique

Below, we present results that prove usefulness of the new bias evaluation method proposed in Section 5.5. The tests were performed on the following datasets: *optdigits*, *pendigits*, *reuters*, *satimage*, *usps* and *letter*. First, the optimal training parameters for MNSVM algorithm were found (multi-class problems were solved using the pairwise classification method). Then, for the optimal training settings, we preformed 100-fold cross-validation with various values of the tolerance parameter $\varepsilon \in \left[10^{-6}, 10^{-1}\right]$. For each such training run, the bias was evaluated using theoretic approach, where $b = \sum_{i=1}^{m} \alpha_i y_i$, and using averaging

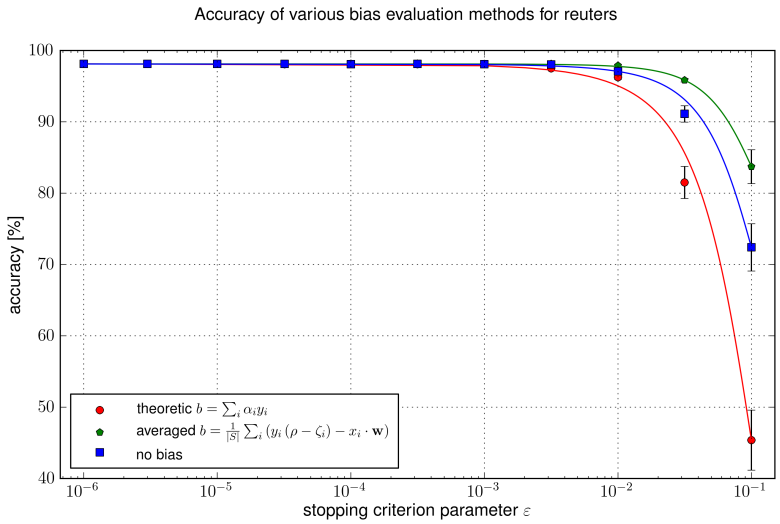

Figure 7.38: The accuracy obtained by different bias evaluation methods for *reuters* dataset.

method expressed in (5.24).

For most of the datasets there was no difference in the classification accuracy. However, we noticed that for *reuters* and *satimage* the proposed approach to bias evaluation results in noticeably better accuracy when large values of ε are used. In the case of *reuters* dataset our approach is even better than the classification method without bias – this dependency is presented in Figure 7.38.

Similar results were obtained for *satimage* dataset. In Figure 7.39 one can observe that the classification accuracy for the models that use bias calculated with the new technique seem to be slightly better than the accuracy of the original models. Unlike the results for *reuters* dataset, here the SVM training without bias seems to be the best choice when dealing with larger values of ε.

7.6 MNSVM with Improved MDM Solver

In order to test the performance of MNSVM with the update step based on Improved MDM approach we performed 100-fold cross-validation on *optdigits*, *pendigits*, *reuters*, *satimage*, *usps* and *letter* datasets. The optimal training parameters were used for each

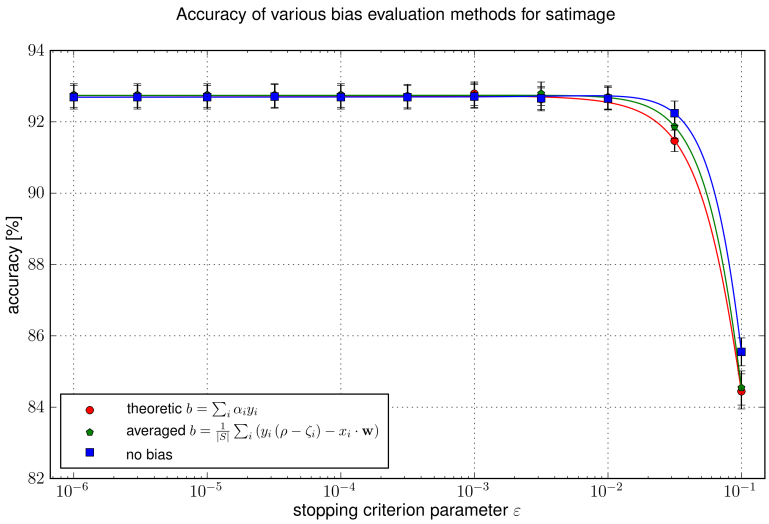

Figure 7.39: The accuracy obtained by different bias evaluation methods for *satimage* dataset.

dataset.

The classification accuracy obtained by MNSVM algorithm using the update steps based on MDM and Improved MDM algorithms are compared in Figure 7.40. The largest time improvement for Improved MDM was obtained on *reuters* training datasets – MNSVM with Improved MDM turned out to be more than two times faster than the original MNSVM algorithm. Much smaller speedup was observed for *usps* training set. In case of *letter* dataset, Improved MDM resulted in small slowdown (around 5%). There was no significant difference in training time for *optdigits*, *pendigits* and *satimage* datasets.

Figure 7.41 shows that there is no difference in accuracy for both methods – the error rates (and even standard errors) are literally the same.

The percent of support vectors obtained by both MNSVM versions is presented in Figure 7.42. MNSVM with Improved MDM update step usually results in slightly smaller models, however the difference is usually almost unnoticeable. The only notable model size decrease can be observed for *reuters* dataset where the percent of support vectors generated by Improved MDM was almost 8% lower compared to the percentage of support vectors obtained by the original MNSVM (please keep in mind that the accuracy was not

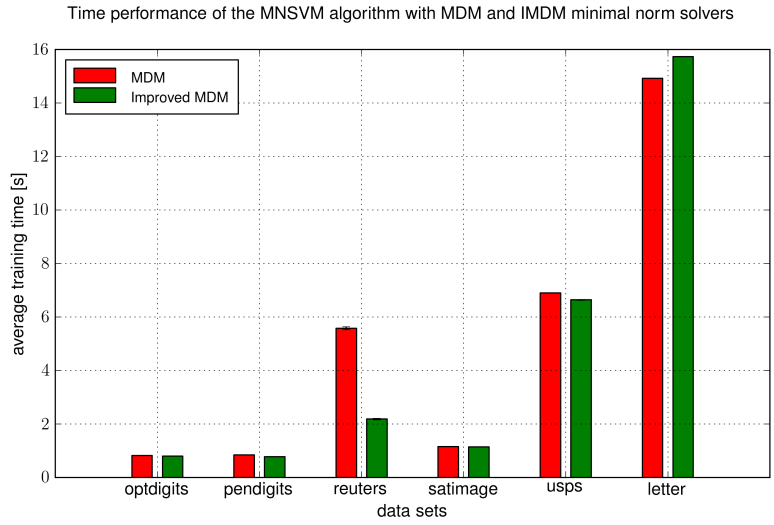

Figure 7.40: Training time for MNSVM with MDM and Improved MDM update step.

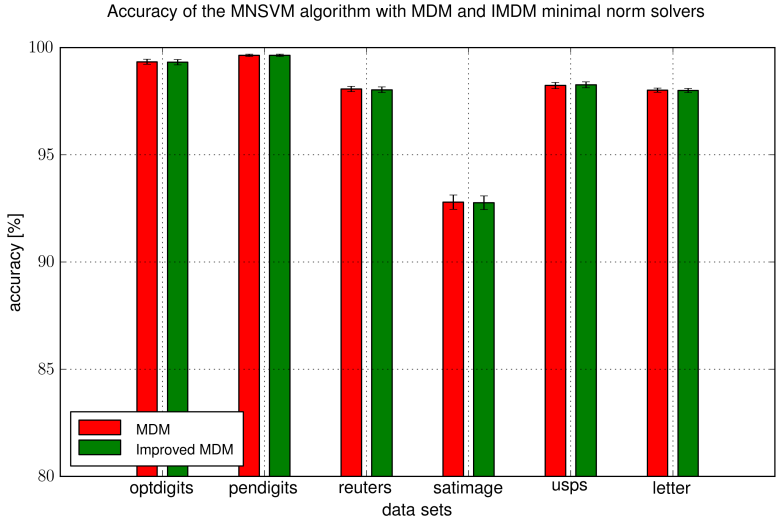

Figure 7.41: Accuracy for MNSVM with MDM and Improved MDM update step.

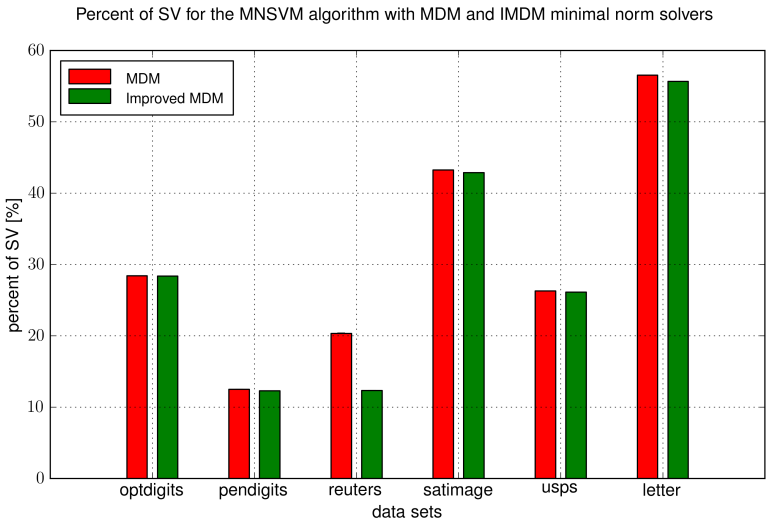

Figure 7.42: Percent of support vectors for MNSVM with MDM and Improved MDM update step.

affected).

7.6.1 Generalized IMDM

Figure 7.43 shows the training time for optimal training parameters for MNSVM with update step based on generalized IMDM presented in Section 5.6.2. The plot represents the dependency of the training time on the parameter k affecting the way how the violating vectors are selected in each iteration of the MNSVM algorithm. The results were collected during 100 training runs on *optdigits*, *pendigits*, *reuters*, *satimage*, *usps* and *letter* datasets. Almost for all training sets the optimal learning times were achieved either for $k = 0$ (which is equivalent to original MDM update scheme) or for $k = 1$ (which corresponds to the Improved MDM update step). Very small speedups for $k \notin \{0, 1\}$ can be observed for *reuters* dataset (for $k \approx 0.7$) and *usps* (when $k \approx 0.35$). Unfortunately, these results cannot be considered statistically relevant.

Obtained results suggest that Generalized IMDM approach is very unlikely to improve the time performance and it is better to use MDM or Improved MDM instead.

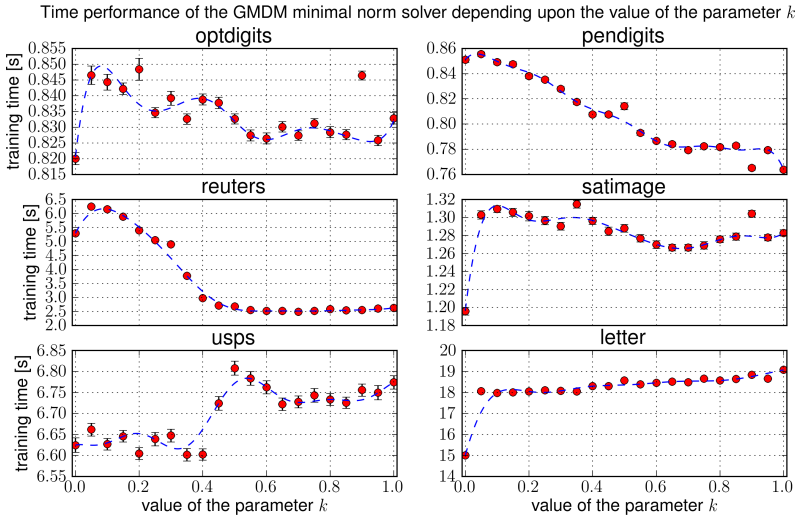

Figure 7.43: MNSVM with Generalized IMDM update scheme.

7.7 Sparse Grid Model Selection Technique

Below, we compare Combinatorial Model Selection technique (called "Grid Search") and Sparse Grid Model Selection proposed in Section 5.7.2. The same cross-validation settings were used as in Section 7.1.1. The results were obtained on AMD Opteron 6282 SE CPUs using GSVM toolkit.

The total nested cross-validation time of the MNSVM training with pattern search and grid search methods is presented in Figure 7.44. It is clearly visible that for all datasets the training with sparse grid search (pattern search) is approximately three to four times faster.

The accuracy obtained by both methods shown in Figure 7.45 indicates that despite the huge speedup achieved by the pattern search method no accuracy deterioration can be observed. Both algorithms generated models that describe the data with the same precision.

The classification results presented in Figures 7.44 and 7.45 were obtained on relatively sparse grids (the resolution[7] was 8×8). In order to investigate how sparse grid model

[7]by the grid resolution we mean the number of distinct kernel parameters C and σ – for instance, grid resolution 8×8 refers to the training settings having eight distinct values of parameter C and eight distinct

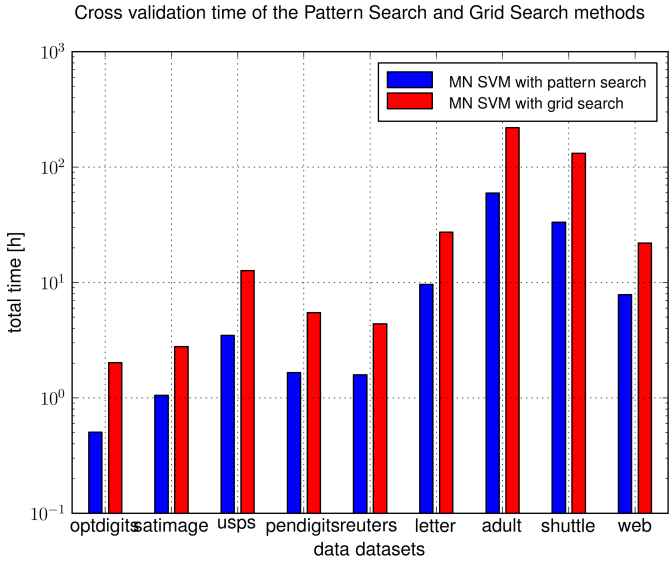

Figure 7.44: Training time for Pattern Search and Grid Search methods.

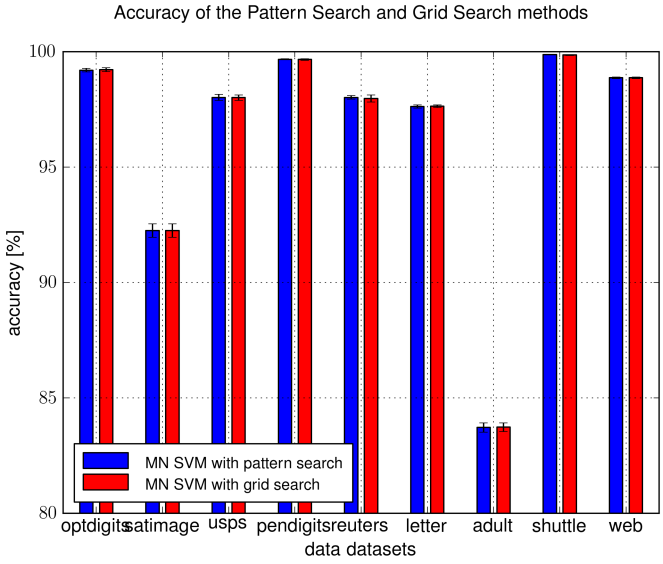

Figure 7.45: Accuracy for Pattern Search and Grid Search methods.

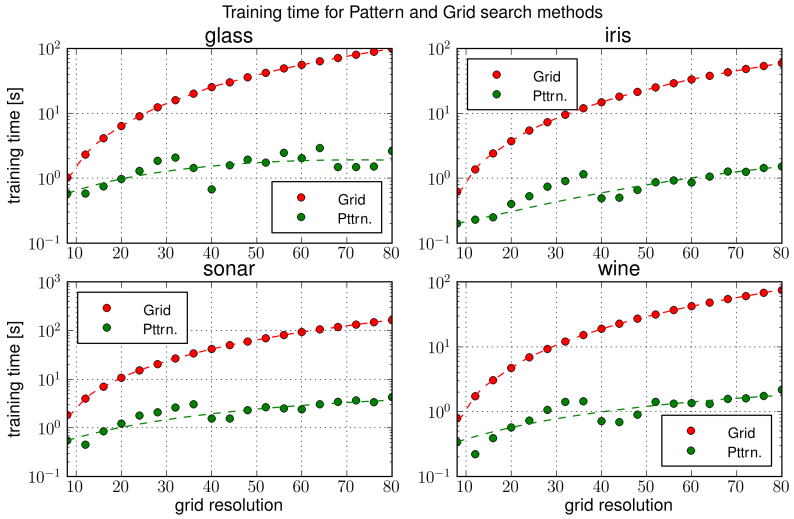

Figure 7.46: Training time of the Pattern and Grid search methods.

selection technique decreases the training time for larger grids we performed additional tests on *glass*, *iris*, *sonar* and *wine* datasets for grid resolutions ranging from 8×8 to 80×80.

Figure 7.46 compares the times required to find optimal model parameters for both model selection methods. It is apparent that the speedup achieved by pattern search increases with the density of the grid. For the largest grids the sparse grid model selection is almost two orders of magnitude faster than the combinatorial search.

It should be emphasized that for the given datasets both methods achieved the same accuracy.

7.8 GSVM toolkit

This Section compares GSVM tool with other open-source SVM frameworks such as LIBSVM and LibCVM. The same settings as the ones described in Section 7.1.1 were used during testing. The only difference is that MNSVM algorithm implemented in GSVM uses stopping criterion more appropriate for minimal norm problems (with $\varepsilon = 0.001$) instead the one implemented in LibCVM that originates from minimal enclosing ball problem.

values of parameter σ

The training runs were performed on a cluster equipped with AMD Opteron 6282 SE processors.

Figure 7.47 compares the accuracy obtained for different SVM implementations during nested cross-validation training. The accuracy of the algorithms implemented in GSVM tool are usually significantly better than the accuracy obtained by LibCVM implementation. This is mostly visible for *optdigits*, *usps*, *letter* and *adult* datasets where accuracy improvement is between 0.2% and 0.6%. Wilcoxon signed-rank test with significance level 0.05 applied in comparison of LibCVM and GSVM shows that MNSVM from GSVM is significantly more accurate than BVM implemented in LibCVM (p-value is equal to $p = 0.025$ which allows to reject the null-hypothesis that the two implementation perform at the same level of accuracy). Moreover, models generated by GSVM are competitive with the ones produced by LIBSVM. Both libraries produce similar results and for *letter* dataset GSVM shows superiority over LIBSVM toolkit. Wilcoxon test reveals that there is no sufficient evidence to conclude that there is a significant difference in accuracy between LIBSVM and GSVM ($p = 0.12$).

The total nested cross-validation training time for all methods is presented in Figure 7.48. For smaller datasets GSVM using grid search outperforms algorithms implemented in LibCVM. However, if patter search is used then GSVM is almost always an easily identifiable winner (only for *satimage* and *web* its performance is comparable with LIBSVM and LibCVM). Compared with LIBSVM, GSVM is slower in half of the experiments but for training sets like *usps* or *web* it achieves impressive several fold speedup.

The training times for optimal kernel parameters shown in Figure 7.49 reveal huge similarity between all methods. For smaller datasets GSVM is faster than LibCVM's SVM implementations. The reason for that is the implementation of the kernel cache – GSVM uses LRU[8] cache which is more efficient when dealing with smaller datasets, whereas LibCVM (and LIBSVM) uses LFU[9] cache that seems to be more effective when the cache

[8]Least Recently Used
[9]Least Frequently Used

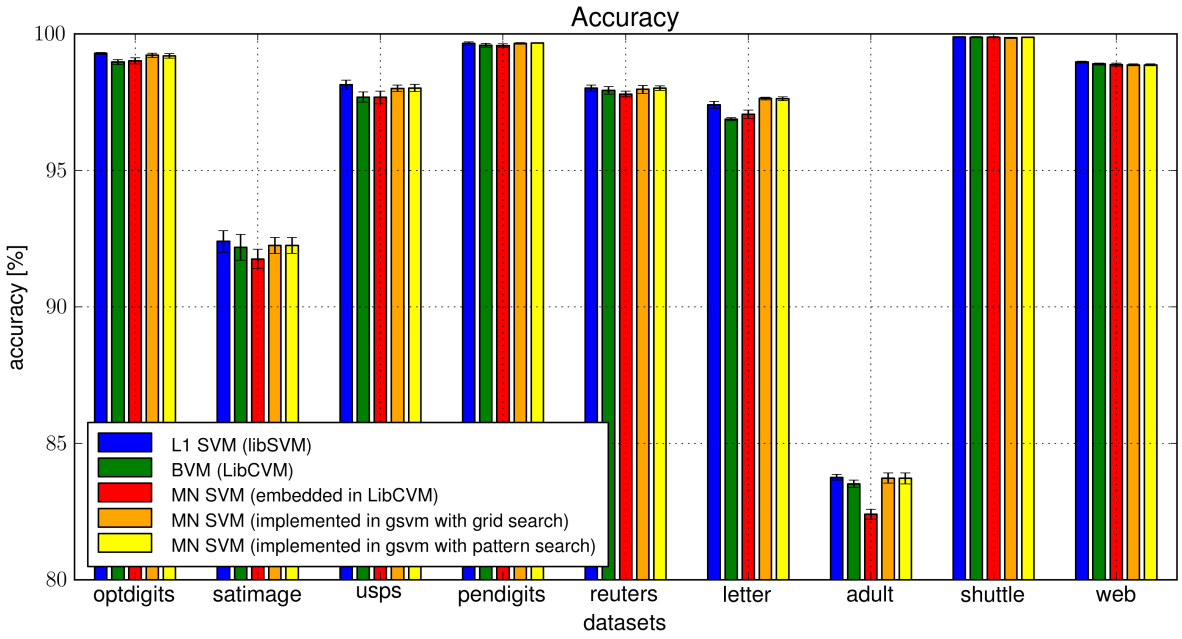

Figure 7.47: Comparison of accuracy obtained by various SVM algorithms. For each dataset the bars represent the accuracy obtained by (bars from left to right): L1 SVM, BVM, MNSVM embedded in LibCVM, and two versions of MNSVM implemented in GSVM – with grid search and with pattern search model selection.

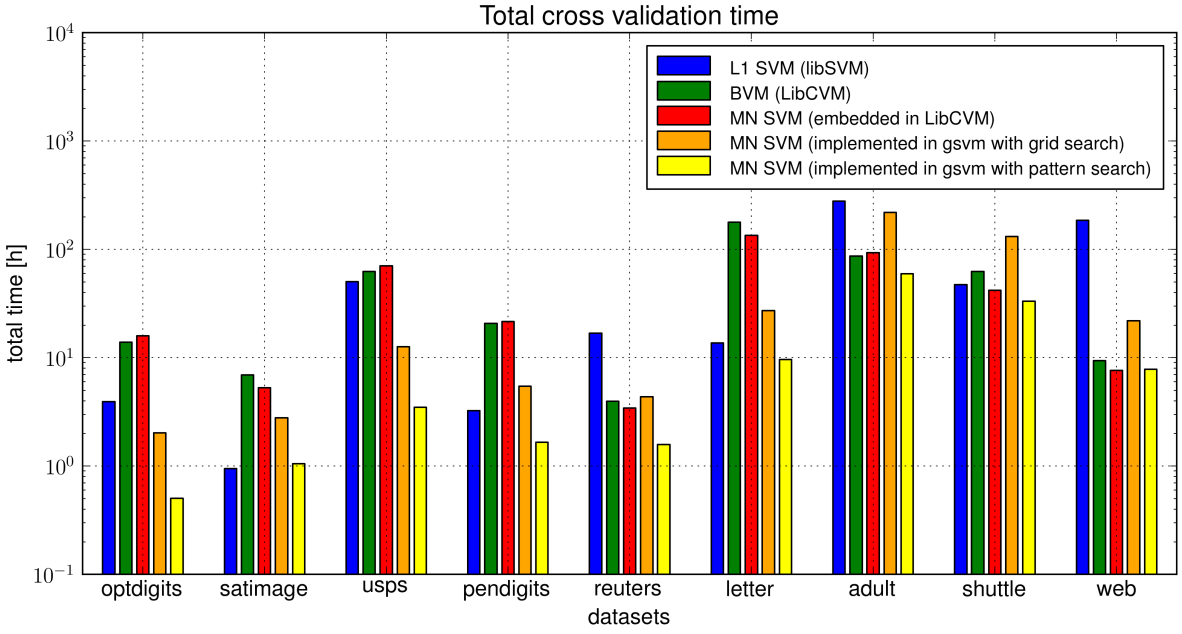

Figure 7.48: Comparison of total nested cross-validation time obtained by various SVM algorithms. For each dataset the bars represent the accuracy obtained by (bars from left to right): L1 SVM, BVM, MNSVM embedded in LibCVM, and two versions of MNSVM implemented in GSVM – with grid search and with pattern search model selection.

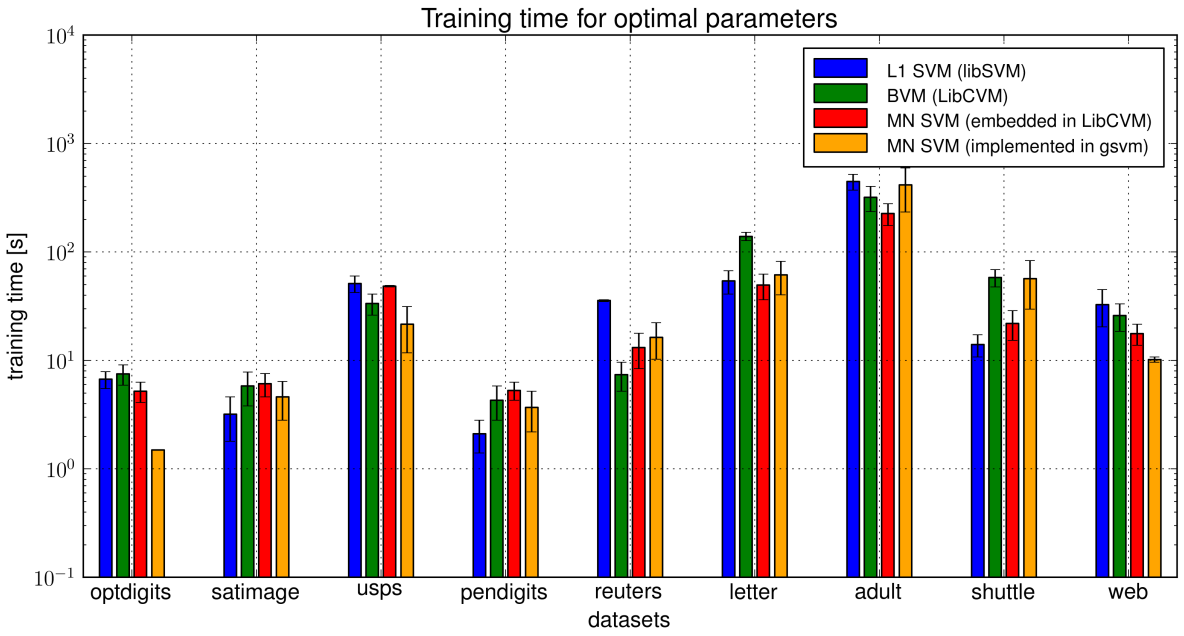

Figure 7.49: Comparison of training time for optimal parameters obtained by various SVM algorithms. For each dataset the bars represent the accuracy obtained by (bars from left to right): L1 SVM, BVM, MNSVM embedded in LibCVM, and two versions of MNSVM implemented in GSVM – with grid search and with pattern search model selection.

capacity does not allow to store the kernel values for all support vectors[10]. Moreover GSVM is significantly faster than LIBSVM for four datasets (*optdigits*, *usps*, *reuters* and *web*) whereas definite superiority of LIBSVM implementations were shown only in case of *shuttle* dataset (for the rest of the training sets the difference between classifiers is within standard error range and no precise conclusions can be drawn).

Figure 7.50 shows the percent of support vectors used in SVM models trained by the algorithms. The fact that algorithms implemented in GSVM usually produce larger models than the ones from LibCVM is easily noticeable. The reason for that is more accurate stopping criterion used in GSVM. Our geometric implementation uses more precise conditions to determine whether a current solution is close enough to the true solution. This usually results in larger number of iterations (and more support vectors). Please note that even though the models are larger, GSVM attains competitive time-

[10]Currently we are working on improving the performance of the GSVM cache so that it would be more efficient for large datasets

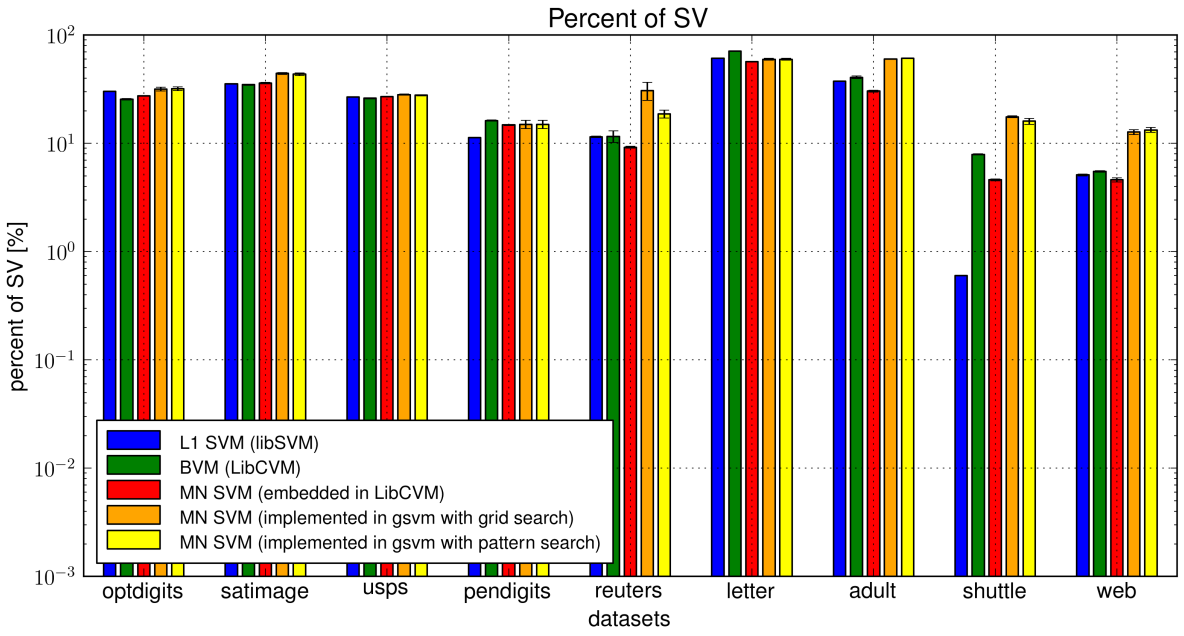

Figure 7.50: Comparison of percent of support vectors obtained by various SVM algorithms. For each dataset the bars represent the accuracy obtained by (bars from left to right): L1 SVM, BVM, MNSVM embedded in LibCVM, and two versions of MNSVM implemented in GSVM – with grid search and with pattern search model selection.

performance compared with LIBSVM (see Figure 7.49).

Chapter 8

Conclusions and future work

The novel L2 SVM classification algorithms developed in the dissertation and dubbed SphereSVM and MNSVM are aimed at classifying large and very large datasets. They show a significant speedup with respect to all three L1 and L2 SVM implementations in LIBSVM and Ball Vector Machines approach. While achieving a speedup going over few orders of magnitude for complex datasets, SphereSVM and MNSVM are still attaining comparative accuracy as the other three algorithms.

The comparisons have been performed within the double (nested) cross validation and thus accuracy estimates are obtained on the samples not seen by the classifiers during the training phase. Such a rigorous experimental environment produces the accuracy estimates which can be expected in a real life applications of all the models.

The over-relaxation scheme is proposed and implemented for the geometrical SVM solvers. It turns out that it is a very successful improvement that can significantly decrease the learning time. Further improvements such as training without bias or application of Improved MDM solver to MNSVM training has been introduced. Together with proposed multi-class solver and sparse grid search approach for model selection, the algorithms constitute very efficient alternative for solving large classification problems (especially, that the mentioned methods has been published as an open-source framework GSVM).

The proof of a convergence of the SphereSVM is given and it states that the time complexity of the algorithm depends upon the tolerance parameter ε only. However, as it is often the case with bounds, this one is also loose. In all our simulations we have got results in much shorter time than given by the theoretical bound. Finally, it is worth mentioning that proposed geometric SVM training methods usually generate sparser models (having less support vectors) than the BVM which is the result of an efficient elimination of support vectors. As of now, it seems that both SphereSVM and MNSVM may well be the recommended sequential classification approach when data size goes into ultra-large domains (say, when the number of samples comes to and crosses over into few millions).

Appendix A

GSVM - Command Line Tool for Geometric SVM Training

GSVM is a command-line tool that performs SVM training using MNSVM algorithm. It implements several different variants of MNSVM algorithm and is able to perform model selection and n-fold cross-validation (including nested cross-validation).

```
Usage: gsvm [OPTION]... [FILE]
Perform SVM training for the given data set [FILE].


Available options:
  -h [ --help ]
                                produce help message containing description
                                of all available training options
  -c [ --c-low ] arg (=0.0625)
                                the value of the penalty parameter C
                                (in case of model selection training
                                the argument determines the lower bound of C)
  -C [ --c-high ] arg (=1024)
```

```
                              in case of model selection training,
                              the argument determines the upper bound of
                              the penalty parameter C
-g [ --gamma-low ] arg (=0.0009765625)
                              the value of the Gaussian kernel parameter gamma
                              (in case of model selection training
                              the argument determines the lower bound of gamma)
-G [ --gamma-high ] arg (=16)
                              in case of model selection training,
                              the argument determines the upper bound of
                              the Gaussian kernel parameter gamma
-r [ --resolution ] arg (=8)
                              the number of C and gamma parameters used
                              during model selection
-o [ --outer-folds ] arg (=1)
                              the number of nested cross-validation
                              outer folds, if arg > 1 then double
                              cross-validation is used
-i [ --inner-folds ] arg (=10)
                              the number of inner folds (the parameter k for
                              k-fold cross-validation)
-e [ --epsilon ] arg (=default)
                              tolerance parameter for the stopping criterion, if
                              "default" is used than the default value for
                              the stopping criterion will be evaluated:
                              - 0.001 if minimal norm stopping criterion is used
                              - heuristic value if MEB-based stopping criterion
```

```
                                   is used
-E [ --eta ] arg (=1)
                                parameter for over-relaxation (eta should be
                                in range [1, 2), if eta = 1 then no
                                over-relaxation is used
-b [ --bias ] arg (=theoretic)
                                type of the bias evaluation strategy:
                                - theoretic - bias is evaluated as
                                   b = \sum \alpha_i y_i
                                - average - bias is evaluated from KKT conditions
                                - nobias - training without bias (b = 0)
-d [ --draw-number ] arg (=600)
                                parameter of the probabilistic speedup - the number
                                of random draws during violator search
-u [ --multiclass ] arg (=allatonce)
                                type of the multi-class training approach
                                - allatonce - all-at-once training (only one model)
                                - pairwise - pairwise training with voting mechanism
-m [ --model-selection ] arg (=pattern)
                                type of the model selection method
                                - grid - grid search
                                - pattern - sparse grid search (based on
                                   pattern search)
-t [ --matrix-type ] arg (=sparse)
                                data representation type
                                - sparse - sparse matrix
                                - dense - dense matrix
```

```
-p [ --stop-criterion ] arg (=adjmn)
                              stopping criterion
                              - adjmn - adjusted minimal norm stopping criterion
                                 (the value of epsilon is scalled depending on
                                 the value of the penalty parameter C)
                              - mn - minimal stopping criterion
                              - meb - minimal enclosing ball stopping criterion
-z [ --optimization ] arg (=mdm)
                              minimal norm problem solving strategy
                              - mdm - traditional MDM approach
                              - imdm - Improved MDM algorithm
                              - gmdm - generalized IMDM approach
-k [ --gmdm-k ] arg (=0.5)
                              parameter for generalized IMDM solver
-s [ --randomizer ] arg (=fair)
                              approach to selection violator candidates
                              - simple - basic random number generator
                              - fair - first a random class is picked, then
                                 random sample belonging to that class
-S [ --cache-size ] arg (=200)
                              cache size (in MB)
-I [ --input ] arg
                              name of the training data set, the input file must
                              be in sparse LIBSVM format
```

Some examples of gsvm toolkit usage are presented below

- 10-fold cross validation with cache size set to 5 MB

```
gsvm -c 100.0 -g 1.0 -i 10 -S 5 iris.dat
```

- nested cross validation, for five values of parameter C from the range $C \in [1, 10]$ and five values of parameter γ from the range $\gamma \in [0.1, 10]$; 10 inner and outer folds are used with model selection based on grid search

```
gsvm -c 1.0 -C 10.0 -g 0.1 -G 10.0 -i 10 -o 10 -r 5 -m grid iris.dat
```

- pairwise multi-class training on dense data without bias

```
gsvm -c 1.0 -g 0.1 -u pairwise -b nobias -t dense iris.dat
```

Bibliography

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

[2] E. Osuna, R. Freund, 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*, pp. 276 –285, 1997.

[3] T. Joachims, "Making large-scale support vector machine learning practical," in *Advances in kernel methods*, pp. 169–184, MIT Press, 1999.

[4] C.-H. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 27:1–27:27, 2011.

[5] Q. Li, R. Salman, E. Test, R. Strack, and V. Kecman, "GPUSVM: a comprehensive CUDA based support vector machine package," *Central European Journal of Computer Science*, vol. 1, no. 4, pp. 387–405, 2011.

[6] Q. Li, *Fast parallel machine learning algorithms for large datasets using Graphic Processing Unit*. PhD thesis, Virginia Commonwealth University, 2011.

[7] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, "Core Vector Machines: Fast SVM Training on Very Large Data Sets," *Journal of Machine Learning Research*, vol. 6, pp. 363–392, 2005.

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

[9] I. W. Tsang, A. Kocsor, and J. T. Kwok, "Simpler core vector machines with enclosing balls," in *Proceedings of the 24th international conference on Machine learning - ICML '07*, (New York, New York, USA), pp. 911–918, ACM Press, 2007.

[10] J. C. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," *Advances in Kernel Methods Support Vector Learning*, vol. 208, no. MSR-TR-98-14, pp. 1–21, 1998.

[11] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *Electronic Computers, IEEE Transactions on*, no. 3, pp. 326–334, 1965.

[12] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. K. Murthy, "A fast iterative nearest point algorithm for support vector machine classifier design.," *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 11, pp. 124–36, Jan. 2000.

[13] V. Franc and V. Hlaváč, "An iterative algorithm learning the maximal margin classifier," *Pattern Recognition*, vol. 36, pp. 1985–1996, Sept. 2003.

[14] D. J. Crisp and C. J. C. Burges, "A Geometric Interpretation of nu-SVM Classifiers," in *Advances in Neural Information Processing Systems*, vol. 12, pp. 223–229, 2000.

[15] M. E. Mavroforakis and S. Theodoridis, "A geometric approach to support vector machine (SVM) classification.," *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 17, pp. 671–82, May 2006.

[16] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, "Very large SVM training using core vector machines," in *Proc. 10th Int. Workshop Artif. Intell.*, pp. 349—-356, 2005.

[17] M. Badoiu and K. L. Clarkson, "Smaller Core-Sets for Balls," in *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, no. 1, pp. 1–2, 2003.

[18] I. W. Tsang, J. T. Kwok, and J. M. Zurada, "Generalized core vector machines.," *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 17, pp. 1126–40, Sept. 2006.

[19] A. J. Smola and B. Schölkopf, "Sparse Greedy Matrix Approximation for Machine Learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 911—-918, 2000.

[20] S. Asharaf, M. N. Murty, and S. K. Shevade, "Multiclass core vector machine," *Proceedings of the 24th international conference on Machine learning - ICML '07*, pp. 41–48, 2007.

[21] J. López, A. Barbero, and J. R. Dorronsoro, "An MDM solver for the nearest point problem in Scaled Convex Hulls," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pp. 1–8, IEEE, 2010.

[22] B. N. Kozinec, "Recurrent algorithm separating convex hulls of two sets," *Learning algorithms in patter recognition*, pp. 43–50, 1973.

[23] B. F. Michell, V. F. Demyanov, and V. N. Malozemov, "Finding the point of polyhedron closest to the origin," *SIAM JControl*, vol. 12, pp. 19–26, 1974.

[24] V. N. Vapnik and A. Lerner, "Pattern recognition using generalized portrait method," *Automation and Remote Control*, vol. 24, pp. 774–780, 1963.

[25] V. N. Vapnik and A. J. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability and its Applications*, vol. XVI, no. 2, pp. 264–280, 1971.

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

[27] C. Cortes and V. N. Vapnik, "Support-vector networks," *Machine learning*, vol. 297, pp. 273–297, 1995.

[28] A. Shigeo, *Support vector machines for pattern classification*. 2010.

[29] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[30] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. Bartlett, "New support vector algorithms," Jan. 2000.

[31] P. Kumar, J. S. B. Mitchell, and E. A. Yildirim, "Approximate minimum enclosing balls in high dimensions using core-sets," *Journal of Experimental Algorithmics*, vol. 8, p. 1.1, Jan. 2003.

[32] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, "Geometric approximation via coresets," in *Combinatorial and Computational Geometry, MSRI*, pp. 1–30, 2005.

[33] P. Kumar and A. E. Yildirim, "Minimum-volume enclosing ellipsoids and core sets," *Journal of Optimization Theory and Applications*, pp. 1–21, 2005.

[34] M. Badoiu and K. L. Clarkson, "Optimal Core-Sets for Balls A Lower Bound for Core-Sets," in *DIMACS Workshop on Computational Geometry*, pp. 3–6, 2002.

[35] A. Barbero, J. López, and J. R. Dorronsoro, "An accelerated MDM algorithm for SVM training," in *Advances in Computational Intelligence and Learning, Proceedings of ESANN 2008 Conference*, no. April, pp. 421–426, 2008.

[36] J. López, *On the Relationship among the MDM, SMO and SVM-Light Algorithms for Training Support Vector Machines*. PhD thesis, 2008.

[37] G. Loosli and S. Canu, "Comments on the core vector machines: Fast svm training on very large data sets," *The Journal of Machine Learning Research*, vol. 8, pp. 291–301, 2007.

[38] A. Agresti and B. A. Coull, "Approximate Is Better than "Exact" for Interval Estimation of Binomial Proportions," *The American Statistician*, vol. 52, pp. 119–126, 1998.

[39] D. Young, "Iterative methods for solving partial difference equations of elliptic type," no. 2, pp. 92–111, 1950.

[40] A. Barbero, J. López, and J. R. Dorronsoro, "Cycle-breaking acceleration of SVM training," *Neurocomputing*, vol. 72, pp. 1398–1406, Mar. 2009.

[41] C. Hsu and C. Lin, "A comparison of methods for multiclass support vector machines," *Neural Networks, IEEE Transactions on*, vol. 13, no. 2, pp. 415–425, 2002.

[42] J. Weston and C. Watkins, "Multi-class support vector machines," *Pattern Recognition*, pp. 0–9, 1998.

[43] K. Crammer and Y. Singer, "On the learnability and design of output codes for multiclass problems," *Machine Learning*, no. 1995, pp. 201–233, 2002.

[44] R. Strack and V. Kecman, "Minimal Norm Support Vector Machines for Large Classification Tasks," in *11th International Conference on Machine Learning and Applications*, 2012.

[45] V. Franc, *Optimization algorithms for kernel methods*. PhD thesis, Czech Technical University in Prague, 2005.

[46] M. Momma and K. Bennett, "A pattern search method for model selection of support vector regression," in *Proceedings of the SIAM international conference on data mining*, p. 50, Citeseer, 2002.

[47] T. G. Kolda, R. M. Lewis, and V. Torczon, "Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods," *SIAM Review*, vol. 45, no. 3, p. 385, 2003.

[48] M. Vogt, "SMO algorithms for support vector machines without bias term," *Technische Univ. Darmstadt, Inst. Automat. Contr., Lab.*, pp. 1–8, 2002.

[49] V. Kecman, M. Vogt, and T. M. Huang, "On the Equality of Kernel AdaTron and Sequential Minimal Optimization in Classification and Regression Tasks and Alike Algorithms for Kernel Machines," *Proc. of the 11th European Symposium on Artificial Neural Networks, ESANN 2003*, pp. 215–222, 2003.

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

[51] T. M. Huang and V. Kecman, "Bias Term b in SVMs Again," in *Proc. of 12 the European Symposium on Artificial Neural Networks*, pp. 441–448, 2004.

[52] S. Varma and R. Simon, "Bias in error estimation when using cross-validation for model selection," *BMC Bioinformatics*, vol. 7, p. 91, 2006.

[53] T. Scheffer, *Error estimation and model selection*. PhD thesis, Technische Universität Berlin, 1999.

[54] T. Yang and V. Kecman, *Machine Learning by Adaptive Local Hyperplane Algorithm: Theory and Applications*. Saarbrücken, Germany: VDM-Verlag, 2010.

[55] I. W. Tsang and J. T. Kwok, "Authors' Reply to the "Comments on the Core Vector Machines: Fast SVM Training on Very Large Data Sets"," *Journal of Machine Learning Research*, pp. 1–14, 2007.

[56] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

Vita

Robert Strack received his M.S. Eng. degree in Computer Science from AGH University of Science and Technology, Krakow, Poland, in 2007. He is now working towards his Ph.D. degree in Computer Science at Virginia Commonwealth University, Richmond, USA. His research is oriented towards Machine Learning and Data Mining algorithms and his field of interest includes support vector machines classification and parallel computing.

- Journal papers
  - **Strack R.**, Kecman V., Li Q., Strack B., Sphere Support Vector Machines for Large Classification Tasks, Neurocomputing, Vol. 101, pp. 59–67, 2013
  - Li Q., Salman R., Test E., **Strack R.**, Kecman V., Parallel Multi-task Cross Validation for Support Vector Machine Using GPU, Journal of Parallel and Distributed Computing, Vol. 73, Issue 3, pp. 293-302, 2012
  - Li Q., Salman R., Test E., **Strack R.**, Kecman V., GPUSVM: A Comprehensive CUDA Based Support Vector Machine Package, Cent. Eur. J. Comp. Sci., 1(4), pp. 387-405, 2011
  - Salman R., Kecman V., Li Q., **Strack R.**, Test E., Fast K-means algorithm clustering, International Journal of Computer Networks & Communications (IJCNC), Vol.3, No.4, pp. 17-31, 2011
- Conference papers
  - **Strack R.** and V. Kecman, Minimal Norm Support Vector Machines for Large Classification Tasks, Proc. of the 11th IEEE international conference on machine learning applications (ICMLA 2012), vol.1, pp. 209 - 214, Boca Raton, FL, 2012
  - Test E., Kecman V., **Strack R.**, Li Q., Salman R., Feature Ranking for Pattern Recognition: A Comparison of Filter Methods, IEEE SouthEast Conference Paper, DOI 10.1109/ SECon.2012.6196888, pp. 1 - 5, 2012 Orlando, FL, March 2012
  - Salman R., Kecman V., Li Q., **Strack R.**, Test E., Two-Stage Clustering with k-means Algorithm, in A. Özcan, J. Zizka, and D. Nagamalai (Eds.): Communications in Computer and Information Science, 1, Volume 162, Recent Trends in Wireless and Mobile Networks, Part 1, pp. 110-122, Springer-Verlag, Berlin, Heidelberg, 2011
- Invited talks
  - Kecman V., Li Q., **Strack R.**, Mining Ultra-Large Datasets by Kernel Machines, 11th International Conference on Intelligent Systems Design and Applications, ISDA 2011, Cordoba, Spain, 2011

- Seminars
  - 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