Graz University of Technology

Master's Thesis

Bi-level Optimization for Support Vector Machines

Author:
Teresa Klatzer

Supervisor:
Univ.-Prof. Dr. Thomas Pock

Faculty of Computer Science and Biomedical Engineering
Institute for Computer Graphics and Vision

Graz, October 2014
Version 1.1

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place, Date: ____________________________

Signature: ____________________________

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort, Datum: ____________________________

Unterschrift: ____________________________

Abstract

This thesis deals with an efficient approach for learning the optimal hyper-parameters for Support Vector Machines (SVMs). The common method to determine hyper-parameters is grid search. Grid search typically involves the definition of a discretized "grid" of possible parameter values with a certain resolution and a search for the values that result in the minimal validation error of the learned model. A major limitation of grid search is that the search space grows exponentially in the parameters which makes the approach only practical for determining very few hyper-parameters. Additionally, grid search operates on discrete parameter values which leads to suboptimal solutions. In this thesis we develop an approach to use bi-level optimization for learning the optimal hyper-parameters and solve both major shortcomings of grid search in an efficient and elegant way. Bi-level learning is an optimization method where one optimization problem has another optimization problem as its constraint. The goal of the bi-level program is to find optimal hyper-parameters such that the validation error (the higher level objective) is minimized, while the optimal training problem is solved for the underlying SVM (the lower level objective). We use Lagrange multipliers to solve the bi-level problem and formulate the solution for several variants of the SVM (linear, kernel, multiple kernel). We can show that, using this method, the model selection problem (i.e. selection of hyper-parameters) can be solved also for a large number of hyper-parameters. The bi-level approach exploits the continuity of the hyper-parameters which allows for better solutions than with grid search. In the experiments, we investigate different properties of the bi-level approach and try to give insights into the advantages of this method. We find that highly parametrized kernel SVMs perform best compared to simpler models which is a clear advantage of bi-level optimization against grid search for model selection.

Kurzfassung

In dieser Masterarbeit geht es darum, eine effiziente Methode zu entwickeln, um optimale Hyper-Parameter für Support Vektor Maschinen (SVMs) zu bestimmen. Die klassische Methode dafür ist die Rastersuche. Diese Methode besteht aus der Definition eines diskretisierten "Rasters" von möglichen Parameterwerten mit einer bestimmten Auflösung, und der Suche nach denjenigen Werten, die am gelernten Modell den minimalen Validierungsfehler ergeben. Der größte Nachteil dieses Ansatzes ist, dass der Suchraum exponentiell mit der Anzahl der Parameter wächst, weswegen die Rastersuche nur zum Bestimmen sehr weniger Parameter geeignet ist. Außerdem können mittels Rastersuche nur diskretisierte Werte gefunden werden, was zu suboptimalen Lösungen führt. In dieser Arbeit entwickeln wir einen Ansatz, mittels Bi-level Optimierung die optimalen Hyper-Parameter zu lernen und lösen beide Mankos der Rastersuche zugleich auf effiziente und elegante Weise. Bi-level Lernen ist ein Verfahren der Optimierung, wobei ein Optimierungsproblem in ein zweites verschachtelt ist. Das Ziel des Bi-level Programms ist es, Hyper-Parameter zu finden, die den Fehler auf den Validierungsdaten minimieren (äußeres Problem), während das optimale Trainingsproblem für die darunterliegende SVM gelöst wird (inneres Problem). Wir verwenden Lagrange Multiplikatoren um das Bi-level Problem zu lösen und formulieren die Lösung für verschiedene Varianten der Support Vektor Maschine (linear, Kernel, Multikernel). Es kann gezeigt werden, dass mittels Bi-level Lernen Modellselektion (i.e. die Auswahl von Hyper-Parametern) für eine große Zahl an Parametern durchgeführt werden kann. Die vorgeschlagene Methode nützt zusätzlich aus, dass die Parameter kontinuierlich sind, und daher können auch bessere Lösungen gefunden werden als mittels Rastersuche. In den Experimenten werden verschiedene Eigenschaften des Bi-level Ansatzes untersucht und die Vorteile des Verfahrens werden diskutiert. Wir können zeigen, dass Kernel Support Vektor Maschinen mit vielen Parametern, verglichen mit einfacheren Modellen, die besten Ergebnisse liefern und untermauern dadurch die Vorteile der Verwendung einer Bi-level SVM zur Modellselektion gegenüber der Rastersuche.

Acknowledgements

First of all, I would like to express my gratitude to my parents who supported me through all those years of study, both financially and personally, and never questioned the way I took. Next, I would particularly like to thank my thesis supervisor Thomas Pock for his guidance and indispensable support, the many answers to my questions and his valuable feedback on my work. Thanks must also go to some other people from the institute who gave me important input for my thesis such as René Ranftl, I would like to thank him for sharing his expertise in optimization algorithms and an implementation of the RPROP algorithm with me, and Yunjin Chen for sharing with me his expertise in matrix calculus. Furthermore, I would like to thank all my friends and family for their open ears and some of them for giving feedback on my thesis, and all others who were involved and are not specifically mentioned here.

Finally, I would like to thank Julian for his love and support.

Contents

List of Figures

List of Tables

Symbols

Symbol	**Description**
(x_i, y_i)	Training or test example i with corresponding target
$y_i \in \{-1, 1\}$	Class label for example i
$x_i \in \mathbb{R}^{1 \times D}$	Feature vector of training example i with D dimensions
$f(x)$	Classification function of the SVM
$w \in \mathbb{R}^{1 \times D}$	Weight vector of the SVM
$\tilde{w} \in \mathbb{R}^{1 \times D}$	The same as w, only that the last dimension is set to zero
$b \in \mathbb{R}^{+}$	Bias
$c \in \mathbb{R}^{+}$	Regularization parameter of the SVM
$\bar{c} \in \mathbb{R}^{1 \times D}$	Vector containing multiple regularization parameters c_d
ξ_i	Slack variables
t_i	Place holder, $t_i = y_i(\langle w, x_i \rangle + b)$
$\lVert . \rVert_2$	l_2 norm
$\ell(.)$	Loss function of the SVM, Hinge loss or its approximations
$\Phi(x_i)$	Non-linear feature transform on example x_i
$k(x_i, x_j)$	Kernel function on examples x_i and x_j
$\gamma \in \mathbb{R}^{+}$	Kernel parameter for the RBF kernel
Γ	Covariance matrix, symmetric and positive definite
γ_d	Dimension-wise kernel parameter, scalar
$E(.)$	Lower level problem, the SVM's energy function
λ	Lagrange multiplier
ϑ	Vector containing hyper-parameters for the SVM, e.g. c, γ ...
$\tilde{\boldsymbol{I}} \in \mathbb{R}^{D \times D}$	Identity matrix, last value of diagonal is zero
N	Number of training examples
D	Number of dimensions (for linear SVM including the bias, $D = d + 1$)
L	Number of validation examples
$X \in \mathbb{R}^{N \times D}$	Matrix containing the training examples
$y \in \mathbb{R}^{N \times 1}$	Vector containing the labels of the training examples
$\Xi \in \mathbb{R}^{L \times D}$	Matrix containing the validation examples ξ_l

$\eta \in \mathbb{R}^{L\times 1}$	Vector containing the labels of the validation examples
$H(.)$	Higher level problem of the bi-level framework, validation error function
$\mathcal{L}(.)$	Lagrangian function of $H(.)$, unconstrained bi-level objective
$\mathcal{G}(.)$	Optimality system of $\mathcal{L}(.)$, contains the gradients of $\mathcal{L}(.)$
$\ell'(.)$, $\ell^{(1)}(.)$	First derivative of the loss function
$\ell''(.)$, $\ell^{(2)}(.)$	Second derivative of the loss function
t	Place holder for Xw^T
μ	Scaling factor for the modified Hinge loss function
ϵ	Parameter for the quartic Hinge loss approximation
T	Number of folds for cross-validation
$H_T(.)$	Validation error over all T folds of cross-validation
$\mathcal{L}_T(.)$	Lagrangian function over all folds of cross-validation
$\alpha \in \mathbb{R}^{N\times 1}$	Parameter vector of the kernel SVM
$K \in \mathbb{R}^{N\times N}$	Kernel matrix of the training examples
$\mathcal{K} \in \mathbb{R}^{L\times N}$	Kernel matrix of the validation vs. training examples
$k_j \in \mathbb{R}^{1\times N}$	One row of the training kernel matrix
$\kappa_l \in \mathbb{R}^{1\times N}$	One row of the validation kernel matrix
$K_{in} \in \mathbb{R}^{N\times N}$	Inner derivative of K
$\mathcal{K}_{in} \in \mathbb{R}^{L\times N}$	Inner derivative of $\mathcal{K}$
$\bar{K} \in \mathbb{R}^{N\times N}$	Point-wise product of the inner derivative K_{in} and K
$\bar{\mathcal{K}} \in \mathbb{R}^{L\times N}$	Point-wise product of the inner derivative $\mathcal{K}_{in}$ and $\mathcal{K}$
$K_D \in \mathbb{R}^{N\times N}$	Kernel matrix of the training examples for SVM with multiple γ
$\mathcal{K}_D \in \mathbb{R}^{L\times N}$	Kernel matrix of the validation examples for SVM with multiple γ
$\mathcal{K}_{in,d} \in \mathbb{R}^{L\times N}$	Inner derivative of $\mathcal{K}_D$ of dimension d
$\bar{\mathcal{K}}_d \in \mathbb{R}^{L\times N}$	Place holder for $\mathcal{K}_D \odot \mathcal{K}_{in,d}$
$\Xi_t \in \mathbb{R}^{L\times D}$	Validation examples of one cross-validation fold t
$\alpha_t \in \mathbb{R}^{N\times 1}$	Parameter vector of the kernel SVM for one cross-validation fold t
$\eta_t \in \mathbb{R}^{L\times 1}$	Labels of the validation examples for one cross-validation fold t
$\mathcal{K}_t \in \mathbb{R}^{L\times N}$	Validation kernel matrix for one cross-validation fold t
$p = 1, ..., P$	Number of kernels/partitions for multiple kernel SVM
D_p	Dimension of one partition for the multiple kernel SVM
$x_i^p \in \mathbb{R}^{D_p\times 1}$	Feature subset p of training example i for multiple kernel SVM
$k_\beta(x_j, x_i)$	One kernel element for multiple kernel SVM, $\sum_{p=1}^{P} \beta_p k_p(x_j^p, x_i^p)$
$\beta \in \mathbb{R}^{+}$	Weighting parameter for multiple kernel SVM
$K_\beta \in \mathbb{R}^{N\times N}$	Multiple kernel matrix
$k_{\beta j} \in \mathbb{R}^{1\times N}$	One row of the multiple kernel matrix
$\mathcal{K}_\beta \in \mathbb{R}^{L\times N}$	Multiple kernel matrix for validation
$\kappa_{\beta l} \in \mathbb{R}^{1\times N}$	One row of the multiple kernel matrix for validation
γ_p	Kernel parameter for feature subset p

β_p	Weighting parameter for feature subset p
$\bar{\gamma} \in \mathbb{R}^{P\times 1}$	Vector summarizing the multiple kernel parameters
$\bar{\beta} \in \mathbb{R}^{P\times 1}$	Vector summarizing the weighting parameters for multiple kernel SVM
$t_\beta \in \mathbb{R}^{N\times 1}$	Short cut for $K_\beta \alpha$
$K_\beta^p \in \mathbb{R}^{N\times N}$	Multiple kernel matrix K_β parametrized with kernel parameter γ_p
$\mathcal{K}_\beta^p \in \mathbb{R}^{L\times N}$	Kernel matrix $\mathcal{K}_\beta$ for validation with kernel parameter γ_p
$\bar{K}_\beta^p \in \mathbb{R}^{N\times N}$	Short cut for $K_{in} \odot K_\beta^p$
$\bar{\mathcal{K}}_\beta^p \in \mathbb{R}^{L\times N}$	Short cut for $\mathcal{K}_{in} \odot \mathcal{K}_\beta^p$

Chapter 1

Introduction

1.1 Motivation

This master's thesis is centred around a thriving discipline in computer science, namely machine learning, and more specifically it deals with the problem of classification. One of the main pillars of machine learning is the identification of patterns in data, to draw conclusions from the acquired knowledge and to assign a human understandable meaning to them. Some application scenarios are that we want to identify specific persons as a means of access authorization, or to train a robot to perceive its environment in terms of real objects instead of raw images, or to detect anomalies in user behaviour in order to identify frauds. There are many methods and tools one can use to achieve this kind of tasks, and this master's thesis focuses on a specific method for binary classification, namely Support Vector Machines.

Classification is a means to divide data into categories based on common characteristics. For example, take the problem of deciding whether a person suffers from diabetes or not. This is an instance of a binary classification problem, and can be solved by regarding a set of typical laboratory values such as different blood glucose data, and personal information such as the age, sex, body mass index, diastolic blood pressure and physical activity level. We call such information "features" or "attributes". Let (x_i, y_i) be a training example for solving the diabetes problem; then x_i is a vector containing the previously stated information (mapped to real values) for a person i. All persons having diabetes constitute the positive class with $y_i = 1$, and persons not having diabetes represent the negative class with $y_i = -1$. We call y_i the target values or labels. Classification problems where the targets are given fall into the category of supervised learning. During training of the machine learning algorithm, several training examples are used to learn a function $f(x)$ that takes a new unseen example of data as input and generates a correct output y determining the category for the specific example. If a lot of previously unseen data can be correctly classified the algorithm shows good generalization, see e.g. [9].

Support Vector Machines (SVMs) are a popular supervised learning method for classification. They show empirically good performance and have successful applications in many fields such as bioinformatics, text and image recognition and many more.

In general, the learning problem can be seen as an optimization problem. The amount of misclassification described by some error measure on the training data is minimized and also the complexity of the classifier. Many machine learning algorithms consist of a convex optimization problem with one or more hyper-parameters. SVMs fall in that category as well, and common hyper-parameters are the regularization parameter or kernel parameters, etc.

The choice of those parameters is traditionally left to the user, typically the problem is solved by cross-validation and grid search methods. This procedure can be very tedious, and grid search is only practical if there are very few parameters to choose [6]. Other attempts to facilitate for parameter selection other than grid search have been made in [15] where they seek to minimize smoothed estimates of the generalization error of the SVM w.r.t. the hyper-parameters by gradient methods. Another approach using different generalization error bounds has been investigated in [19].

That is the starting point of this thesis. We want to examine an approach to circumvent the arduous search for "good" parameters - by bi-level optimization. Similar to ideas in e.g. [15], we take a performance measure of the SVM, in our case the validation error, and minimize the error w.r.t. the hyper-parameters of the SVM. This sounds rather straight forward, however, this is not directly possible because the validation error does not explicitly depend on the hyper-parameters of the SVM. In this work we show how to solve this problem via bi-level optimization techniques. Bi-level optimization makes it possible to determine many parameters at once using standard optimization techniques and to exploit the fact that the hyper-parameters are continuous.

The idea of using bi-level optimization for determining hyper-parameters is not entirely new. Bennett, Kunapuli *et al.* [6, 27, 7] have investigated a similar approach, but they use different methods to deal with the optimization problem and only use available standard solvers, which limits them basically to experiments with a linear SVM.

However, SVMs can be extended to deal with high dimensional non-linear data sets (e.g images) by using kernels. Of course, those kernels contain additional parameters which can be determined by bi-level optimization. In the end, solving the kernelized SVM model with a bi-level approach should accomplish the task of selecting the regularization parameters as well as the kernel parameters automatically, see [27]. The problem of determining hyper-parameters is also referred to as model selection in the machine learning community.

Our contribution adopts their ideas of bi-level optimization to automate model selection, but follows a different way when dealing with the mathematics, and includes the implementation of several configurations of bi-level SVMs including kernels. Furthermore, we apply the developed algorithms on several data sets for evaluation and analyse properties and possible advantages or limitations of our approach.

The thesis is organized as follows: In the remainder of Chapter 1 we draw connections to related work, and give a short introduction to Support Vector Machines, kernels, and bi-level optimization in general. Chapter 2 contains the derivations of the bi-level solution for the linear SVM. In Chapter 3 we deal with the bi-level solution of the bi-level kernel SVM in different settings. Chapter 4 closes the theoretical part and contains explanations of the used optimization algorithms. Next, Chapter 5 and Chapter 6 deal with experiments and the evaluation of this work. The first chapter covers mainly experiments with small scale datasets to show some important properties of our approach and includes also a comparison of the bi-level approach with grid search, the second chapter discusses experiments with image classification benchmark data sets and the corresponding results. Finally, in Chapter 7 there is a conclusion and some comments on possible future work.

1.2 Relation to Existing Work

As already mentioned, Bennet *et al.* [6] aim at solving the model selection problem with bi-level optimization, and their starting point is a linear SVM with the regularization parameter c. They argue, that it might be feasible to use exhaustive search over a discretized grid of possible parameter values for determining this parameter in the linear case, but this is definitely not the case when the complexity of the model and therefore the number of parameters grows. The model selection process is usually based on the evaluation of the validation error or out-of-sample estimate. The principle of the bi-level approach is to look for those hyper-parameters that yield the lowest validation error, while the optimal learning problem is solved for the training set. The learning problem is called inner or lower level problem, and is solved for a fixed instance of hyper-parameters, and the outer or higher level problem is the error function on the validation set.

From an optimization point of view, the bi-level problem is an optimization problem with constraints, and those constraints themselves contain a parametrized optimization problem. There is a connection to mathematical programs with equilibrium constraints, so-called MPECs. MPECs are a generalization of bi-level problems, only that the lower-level constraints are finite-dimensional parametric variational inequalities, see [20], page 66f. Bennet *et al.* [27, 7] transform the present bi-level problem into an instance of a MPEC problem by substituting the lower level problem into a system of equations of its primal and dual feasibility, complementary slackness and KKT (Karush-Kuhn-Tucker) conditions. Due to the complementarity constraints, the optimization problem is non-convex.

In [6] off-the-shelf non-linear programming solvers are used for the experiments. Bennet, Kunapuli *et al.* also published papers investigating this topic further [27, 7]. In [7] they claim that their model is restricted to linear data sets which is a major drawback. They suggest to use a kernelized version of the SVM and state that the modified problem can be reformulated into a MPEC problem, too [27]. However, they state that the development of efficient solvers is still outstanding - with the available solvers they only succeed to solve very small problems.

Following up to their statement, the main contributions of this thesis are

- the development of a suitable mathematical model to solve the bi-level problem for linear as well as kernel SVMs,
- to develop software to solve the bi-level problems, and
- to investigate the advantages and limitations of the method with experiments.

1.3 Support Vector Machine

The basic building block of this work is the so-called Support Vector Machine (SVM). It is a very popular supervised learning method which has been studied heavily during the last two decades. One of the first references describing SVMs in their current form was Cortes and Vapnik [17] as well as Vapnik [45] in 1995, including already the soft-margin SVM and the extension to regression as well as the concept of kernel SVMs using polynomial and radial basis function (RBF) kernels.

In this work, the focus is on Support Vector Machines used for binary classification. To start with, we take a step back in history to 1982 where Vapnik [44] introduced the optimal hyperplanes method for linear separation of data without errors. In a more recent publication of 1992 [10], a training algorithm for large margin classifiers was presented.

Let

$$(x_1, y_1), ..., (x_N, y_N) \quad \text{with} \quad y_i \in \{-1, 1\}$$

be a set of training data and x_i a feature vector as previously described in section 1.1. The optimal hyperplane is given by the vector w which has the same length as the training examples x_i. There is also an offset or bias, b. The data is linearly separable if the following inequalities are satisfied:

$$\begin{aligned} \langle w, x_i \rangle + b &\geq 1 \quad &\text{if} \quad y_i = 1 \\ \langle w, x_i \rangle + b &\leq -1 \quad &\text{if} \quad y_i = -1. \end{aligned} \tag{1.1}$$

The optimal hyperplane fulfills

$$\langle w, x_i \rangle + b = 0.$$

For an example, see fig. 1.1: This version of the SVM is also called hard-margin SVM. The data is separated with maximal margin, inside the margin no data points are allowed. The data points lying on the margins (marked with thin blue lines) are the so-called support vectors, fullfilling following equation:

$$y_i(\langle w, x_i \rangle + b) = 1.$$

The thick blue line marks the optimal separating hyper-plane. Using the given infor-

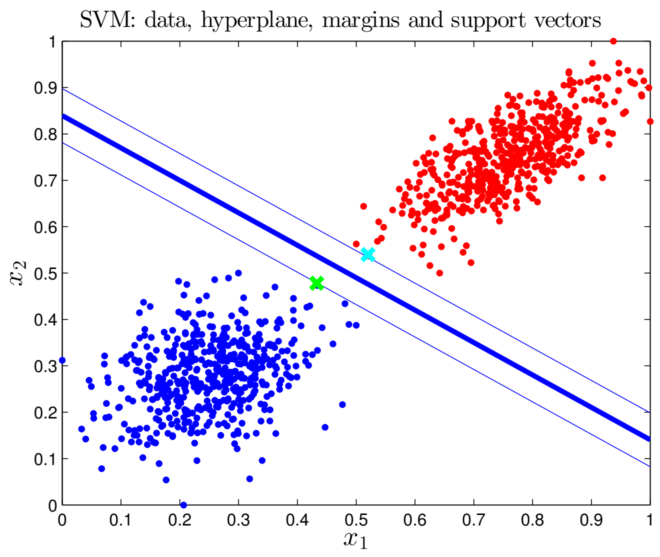


FIGURE 1.1: Toy example showing a linear separable data set, the hard-margin SVM classifier, its margins and the support vectors lying on the margins. The thick blue line marks the separating hyperplane, the thin blue lines mark the margins and the crosses mark the support vectors.

mation, we can state the optimization problem to determine the vector w defining the hyperplane:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|_2^2 \\ s.t. \quad & y_i(\langle w, x_i \rangle + b) \geq 1 \end{aligned}$$

with the constraint following from eq. 1.1. The decision function that decides the category of a new example x_j is given by

$$y_i = \operatorname{sign}(\langle w, x_j \rangle + b).$$

In the real world, most data sets are not linearly separable. Therefore, the concept of the soft-margin Support Vector Machine has been introduced in [17]. The principle of the soft-margin SVM is to minimize the number of errors i.e. the number of examples being inside the margin or on the other side of the hyperplane when perfect separation of the given data into two classes is not possible. This leads to a different formulation of the minimization problem stated before by introducing so-called non-negative slack variables ξ_i:

$$\begin{aligned} &\min_{w,\xi} \frac{c}{2}\|w\|_2^2 + \sum_{i=1}^{N} \xi_i \\ s.t. \quad & y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i \\ & \xi_i \geq 0. \end{aligned} \tag{1.2}$$

The parameter w gives the optimal soft-margin hyperplane, and the regularization parameter c serves as a trade-off parameter between maximizing the margin and minimizing the misclassification error. Let $t_i = y_i(\langle w, x_i \rangle + b)$, then the constraints of the soft-margin SVM can be written as shown in figure 1.2.

Minimizing over ξ_i, we can interpret the constraints in eq. 1.2 in terms of a loss function, also called Hinge loss function $\ell(w, b, x_i, y_i) = \max(0, 1 - t_i)$. Because the Hinge loss function is not continuously differentiable, following loss functions are frequently used:

- the squared Hinge loss function: $\ell(w, b, x_i, y_i) = \max(0, 1 - t_i)^2$,
- the logarithmic loss function: $\ell(w, b, x_i, y_i) = \log(1 + \exp(-t_i))$.

The different loss functions are shown in fig. 1.3.

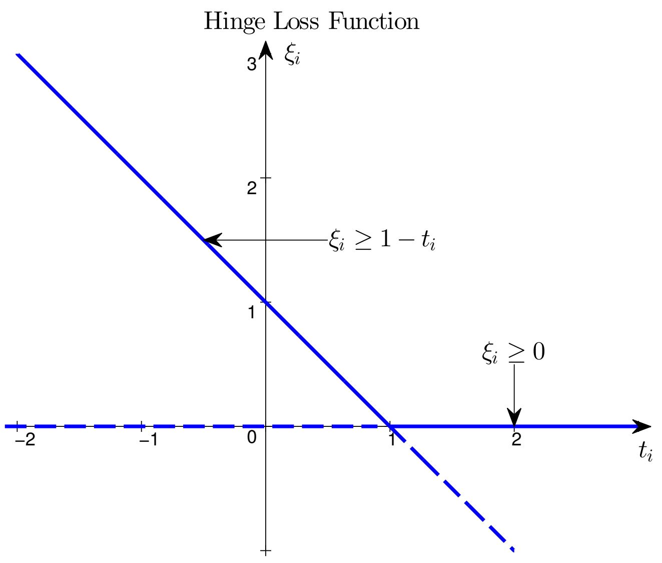


FIGURE 1.2: Constraints of the soft-margin SVM. The solid line shows the resulting Hinge loss function

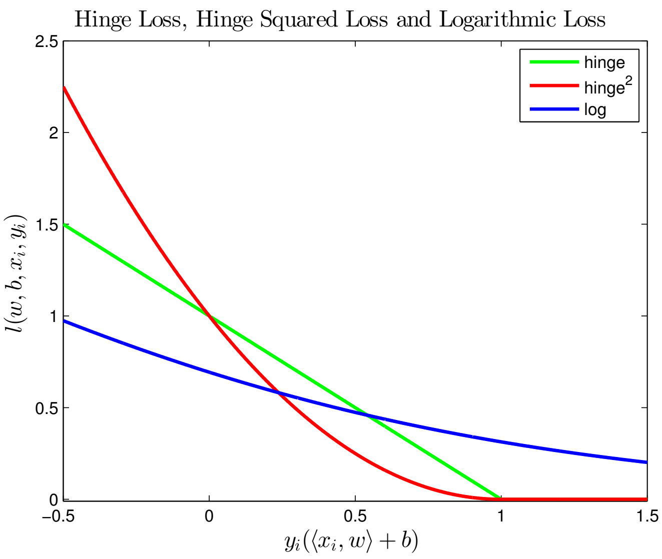


FIGURE 1.3: Variants of loss functions. The Hinge loss function is shown in green, the squared Hinge loss is depicted in red, and the logarithmic loss is shown in blue.

Using the concept of loss functions, we can write the soft-margin objective shown in equation 1.2 in a different way:

$$E(c, w) = \frac{c}{2}\|w\|_2^2 + \sum_{i=1}^{N} \ell(w, b, x_i, y_i). \tag{1.3}$$

This equation can also be seen as an objective function that is minimized when solving the SVM.

To summarize, the soft-margin SVM allows errors in the classification but penalizes them via a loss function. Note that eq. 1.2 is a constrained formulation of the SVM [14].

Solving the soft-margin SVM determines the optimal hyperplane that minimizes the number of errors on the training set and at the same time separates the training examples with maximal margin with the regularization parameter c. The Hinge loss function can be seen as a misclassification measure: It minimizes the distance of each misclassified training example from the trained classifier margin [27]. See the example in fig. 1.4: Suppose we want to learn the optimal soft-margin hyperplane to separate the two classes marked in red and blue dots respectively. The separating hyperplane (marked with the thick blue line) results from minimizing the energy function given in equation 1.3. The thin blue lines show the margin outside of which data points on the 'right' side of the classifier are not penalized. Examples which are correctly classified and are outside of the margin, do not get a penalty (the loss function is 0 there). The penalty is between 0 and 1 between margin and hyperplane, and increases linearly when an example is on the negative side of the hyperplane.

Summarized, we state that

$$y_i(\langle x_i, w\rangle + b) \begin{cases} > 1 & \text{if the example lies outside of the positive margin} \\ = 1 & \text{if the example lies on the positive margin} \\ \in (0,1) & \text{if the example lies inside the positive margin} \\ = 0 & \text{if the example lies on the optimal hyperplane} \\ \in (-1,0) & \text{if the example lies inside the negative margin} \\ = -1 & \text{if the example lies on the negative margin} \\ < -1 & \text{if the example lies outside of the negative margin.} \end{cases} \tag{1.4}$$

Examples lying on the hyperplane or the margins are support vectors of the soft-margin SVM and are marked with circles and crosses in fig. 1.4 respectively.

The concept of SVMs evolved over time and there have been proposed diverse extensions such as structured prediction for arbitrary input and output domains (see [3] for different kinds of approaches to deal with structured data), Support Vector Regression where the SVM is used for function estimation with a tube parameter ε, or the concept of single class SVMs for novelty/outlier detection e.g. to identify examples that are "hard" to classify due to mislabelling or similar [39]. Furthermore, we often want to discriminate between more than two classes. A common method to perform multi-class classification is to learn N 1-to-rest classifiers, with N the number of classes.

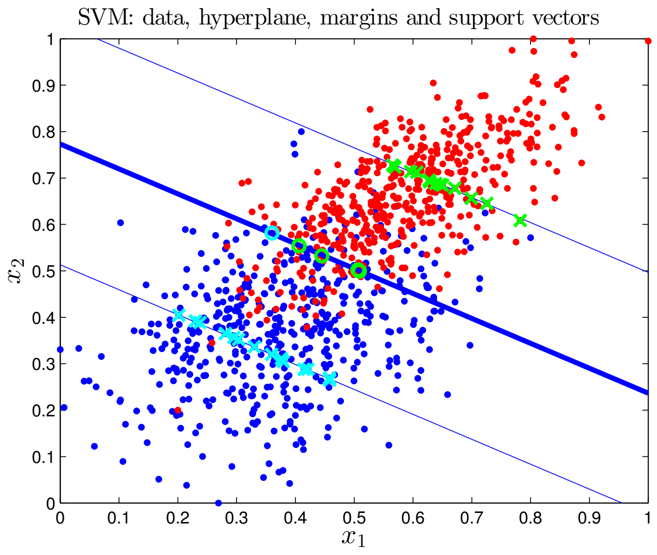


FIGURE 1.4: Toy example showing a non linearly separable data set, the SVM classifier, its margins and the support vectors. The thick blue line depicts the hyperplane, the thin blue lines depict the margins, the crosses and circles mark the support vectors.

1.4 Kernels

However, with a linear classifier we can only handle simple, low-dimensional data. But of course, there is a solution to this problem: We can use kernel functions to handle higher dimensional data and linearly not separable data sets. The input data can be transformed via a kernel function in a higher dimensional feature space where the data instances are linearly separable (see fig. 1.5).

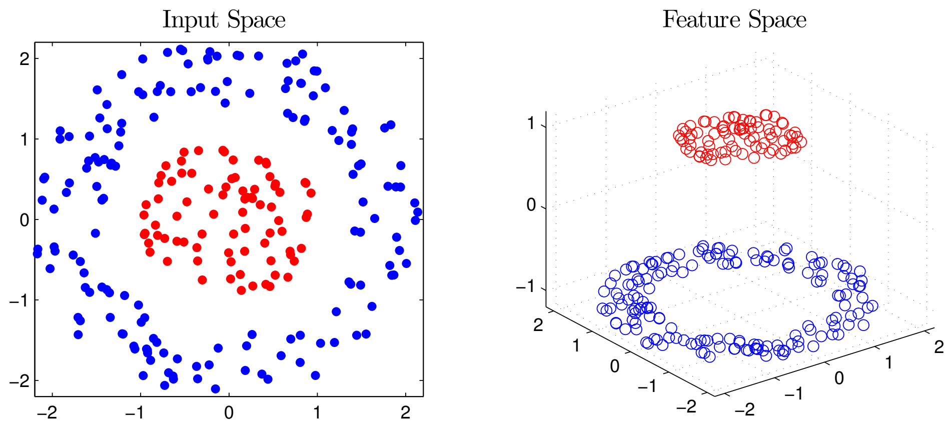


FIGURE 1.5: Example. Transformation of linearly not separable input data to feature space

The kernel function is able to describe pairwise similarities between instances of data by using the inner products between the data points. The inner products implicitly define the feature space, computing the exact transformation is not needed (see page 25f [40]). Schölkopf describes kernels in his book [39] on page 32, figure 2.2 this way: Each pattern

(x_i, x_j) is represented by a ”kernel shaped *function* sitting on the pattern. In this sense, each pattern is represented by its similarity to *all* other patterns” (see figure 1.6).

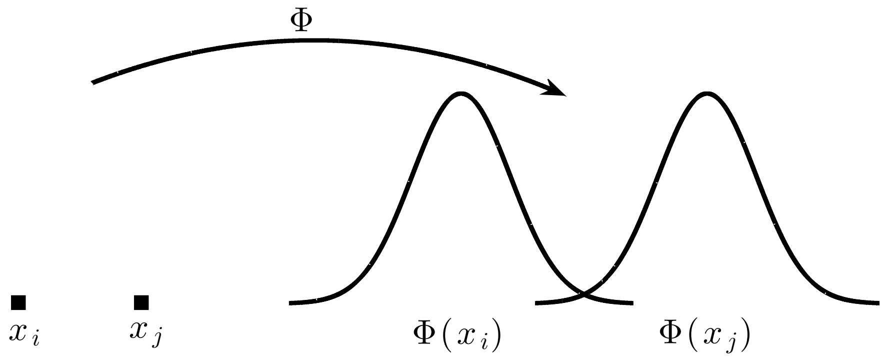


FIGURE 1.6: See [39], p. 32, fig. 2.2. Idea of a kernel function.

Typical kernel functions are the isotropic radial basis function kernel parameterized with γ_i such as in equation 1.5 or the anisotropic version parameterized with a covariance matrix Γ such as in equation 1.6.

$$k_\Phi(x_i, x_j) = \sum_{i=1}^{n} \exp(-\gamma_i \|x_i - x_j\|_2^2) \tag{1.5}$$

$$k_\Phi(x_i, x_j) = \sum_{i=1}^{n} \exp(-(x_i - x_j)^T \Gamma (x_i - x_j)) \tag{1.6}$$

We require that the kernel parameters $\gamma_i > 0$ and that the covariance matrix Γ is symmetric and positive definite.

Next, we discuss the concept of bi-level optimization.

1.5 Bi-level Optimization

Bi-level optimization is a hierarchical mathematical concept involving a higher level optimization problem having as constraint another optimization problem, the lower level problem. Let us first have a look at some historical background information about bi-level optimization: Bi-level optimization is related to Stackelberg games, a concept from economics introduced in 1952. It is also called leader follower game, where the leader commits to a strategy knowing all about the possible steps the follower might take, and the follower observes the leader and chooses its own strategy such that the follower's benefit is maximized (see e.g. [42] for a good overview). The leader's decision can be seen as the higher level problem, the follower's decision as the lower level problem. Bi-level optimization as such can possibly be traced back to a related concept introduced by Bracken and McGill in 1973 in the context of mathematical programming. Bi-level problems were described as mathematical programs with optimization problems in the constraints [12].

In the context of model selection (i.e. choice of optimal hyper-parameters for a machine learning algorithm), the higher level problem corresponds to finding the optimal hyper-parameters that yield the smallest error on test or validation data, but requires that the lower level learning problem is solved using those hyper-parameters. A common method to determine "good" hyperparameters is the procedure of grid search. Grid search typically involves the definition of a "grid" over a certain range of parameter values with a certain resolution and a search for the best values that minimize the test or validation error of the learned model. The crucial disadvantage of grid search is that the complexity grows exponentially in the number of parameters. Additionally, grid search operates on discrete parameter values and does not take advantage of the continuity of the parameters [6]. With the bi-level approach, we can tackle both major shortcomings of grid search in an efficient and elegant way.

Bi-level optimization is a generic concept and has diverse applications other than in the machine learning context. It has been applied e.g. for solving inverse problems in image processing such as diverse methods for image restoration. We want to name a few examples: Pock and Kunisch [28] have used the bi-level appraoch to learn parameters for variational de-noising models. In [16, 4] we find approaches to bi-level learning of the parameters of Markov Random Field (MRF) models for image de-noising. Similar work on learning MRFs by Tappen can be found in [38, 43] with applications to de-noising as well as in-painting. A general bi-level approach to solve energy functionals typical for tasks such as de-noising or image labelling is proposed in [18]. The use of bi-level optimization for training of a non-parametric image restoration framework is suggested in [26]. To state a final example, a bi-level programming approach for learning a dictionary of analysis sparsity priors with application in signal denoising can be found in [35].

Chapter 2

Bi-Level Solution for the Linear Support Vector Machine

In this section the simplest version of the bi-level solution is presented. The goal is to learn the optimal regularization hyper-parameter for the linear soft-margin SVM. In section 2.1 we derive the solution of the general bi-level problem using implicit differentiation and a Lagrange multiplier λ for the lower level constraint. The whole problem is differentiated to the variables w, ϑ and λ and then the resulting gradients are used for optimization (e.g. with a gradient descent method). As we need a twice differentiable loss function ℓ we show different approximations to the Hinge loss function in section 2.2. Finally, we extend our model to cross-validation (section 2.3) and to optimize multiple regularization parameters (section 2.4).

2.1 Derivations

In the previous chapter we discussed the Support Vector Machine and bi-level problems, now it is time to put everything together. To begin with, we formulate the bi-level problem in its general form. We introduce the parameter vector ϑ which contains the specific hyper-parameters we want to learn. In the linear case, the parameter vector ϑ contains only the regularization parameter c. This simplifies notation because later, when we extend our model we will have different sets of parameters, not only the regularization parameter c of the linear SVM.

In the following we show the constrained bi-level problem with $H(w(\vartheta), \Xi, \eta)$ as the upper level problem and the SVM's energy function $E(w, \vartheta, X, y)$ as the lower level

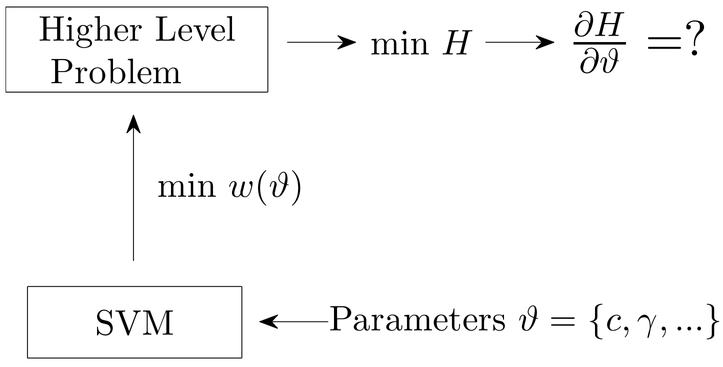

Higher Level Problem
min H
$\frac{\partial H}{\partial \vartheta} = ?$
min $w(\vartheta)$
SVM
Parameters $\vartheta = \{c, \gamma, ...\}$

Figure 2.1: Schematic description of the bi-level problem

problem:

$$\begin{aligned} &\min_{\vartheta} H(w(\vartheta), \Xi, \eta) \\ s.t. \quad & w(\vartheta) \in \arg\min_{w} E(w, \vartheta, X, y). \end{aligned} \tag{2.1}$$

The parameter vector $w(\vartheta)$ can be determined from solving the optimality condition of the SVM's energy function E to the parameters w analytically if possible, or with some iterative method. We can replace the lower level problem in equation 2.1 with its optimality condition:

$$\begin{aligned} &\min_{\vartheta} H(w(\vartheta), \Xi, \eta) \\ s.t. \quad & \partial_w E(w, \vartheta, X, y) \ni 0. \end{aligned} \tag{2.2}$$

In our case, the higher level problem $H(.)$ is the error of the SVM evaluated on a validation data set for the current set of hyper-parameters ϑ. In figure 2.1 we show a schematic view on the bi-level optimization process. The first step is to calculate the solution of the SVM with a given parameter vector ϑ. The parameter vector contains the regularization parameter c or kernel parameters γ etc. The resulting minimized $w(\vartheta)$ is used for evaluating the classifier on the validation set. Based on this result, we want to know how we need to change the parameters in order to achieve a better classification performance so we would like to determine the gradient $\frac{\partial H}{\partial \vartheta}$. This is not obvious, since the higher level objective does not directly depend on the parameters we want to optimize, but only through the lower level constraint. We apply implicit differentiation to solve this issue.

In summary, the upper level problem H aims at minimizing the regularization parameter c of the linear SVM such that the validation error is minimal. Since H does only depend implicitly on c we need to solve this by implicit differentiation. We assume some properties of H and E for our approach to work: The energy function E needs to be twice differentiable, for that reason we in introduced twice differentiable loss functions as replacement for the Hinge loss function (see section 2.2). The validation error H needs to be once differentiable. For that reason, we choose the mean squared error as error measure:

$$H(w(c), \Xi, \eta) = \frac{1}{2L} \|\Xi w^T - \eta\|_2^2 \tag{2.3}$$

Figure 2.2 shows that the validation error varies with the regularization parameter c, and that $H(w(c), .)$ is not a convex function in c (only in w for a fixed c). To create this figure, we evaluated H for different regularization parameters c from the range $[1, 10^4]$ on the Iris data set. In section 5.4 we consider a different function H and show some results.

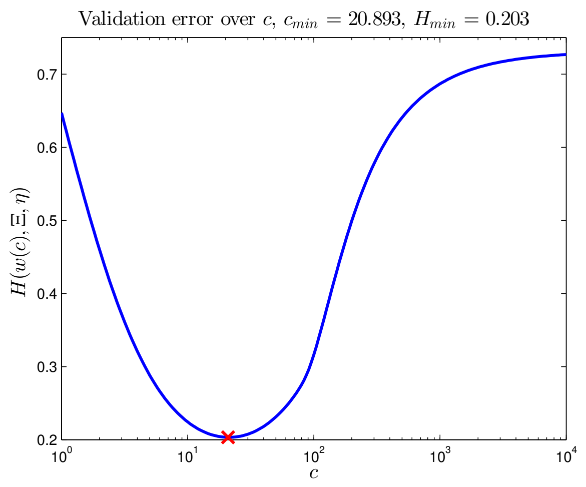


FIGURE 2.2: One dimensional grid search showing the validation error H over c for the Iris data set

For the detailed derivations, we first need to state the training objective function of the linear soft-margin SVM classifier. This objective function is used as the lower level problem of the bi-level framework:

$$E(w, c, X, y) = \frac{c}{2} \|\tilde{w}\|_2^2 + \sum_{i=1}^{N} \ell(w, x_i, y_i). \tag{2.4}$$

Let us define the matrix containing the training examples with $X \in \mathbb{R}^{N \times D}$. One row of X contains one D-dimensional training example x_i, in total there are N training examples. The vector containing all the labels y_i belonging to the examples x_i we call $y \in \mathbb{R}^{N \times 1}$. The labels can take the values $\{-1, 1\}$ stating whether one example belongs to the positive or negative class.

We evaluate the higher level problem on a validation set to get an estimate of the performance of the bi-level SVM. Therefore we define the matrix containing the examples used for validation $\Xi \in \mathbb{R}^{L \times D}$ with L the number of validation examples. One row of Ξ contains one D-dimensional validation example ξ_l. The vector containing the respective validation labels is defined by $\eta \in \mathbb{R}^{L \times 1}$.

For a simpler form of notation we include the bias directly in our data matrices. To achieve that, we define the weight vector of the SVM classifier with $w \in \mathbb{R}^{1 \times D}$, including bias b in the end. The original data has d dimensions and by adding the ones, we get $D = d + 1$ dimensions. Additionally, we add a column of ones at the end of X and Ξ respectively. Note the notation $\tilde{w}$: The vector $\tilde{w}$ is the same as w, only that the last element of w is zero. We use $\tilde{w}$ in the regularization term because the last element of w is the bias included in the weight vector of the SVM and this value is not regularized.

Typically, the loss function ℓ is the Hinge loss function, but due to the nature of the optimization algorithm we use, we need a twice differentiable approximation. See chapter 2.2 for details.

We continue our derivations by reformulating the bi-level problem in eq. 2.1 using the optimality condition for the lower level problem in eq. 2.2 to its unconstrained form using a Lagrange multiplier λ .

$$\mathcal{L}(w, \vartheta, \lambda) = H(w(\vartheta), \Xi, \eta) + \langle \lambda, \frac{\partial E}{\partial w} \rangle \tag{2.5}$$

To calculate the gradient of $\mathcal{L}$ to the parameters ϑ we need to identify the gradient $\frac{\partial \mathcal{L}}{\partial \vartheta}$ which we use instead of $\frac{\partial H}{\partial \vartheta}$ for determining the optimal parameters ϑ.

The optimality condition of equation 2.5 is given by:

$$\mathcal{G}(w, \vartheta, \lambda) = \begin{pmatrix} \frac{\partial H}{\partial w} + \frac{\partial^2 E}{\partial w^2} \lambda \\ \frac{\partial^2 E}{\partial w \partial \vartheta} \lambda \\ \frac{\partial E}{\partial w} \end{pmatrix} = 0. \tag{2.6}$$

From equation 2.6 it can easily be seen that we require $E(.)$ to be twice differentiable w.r.t. w and ϑ and that $H(.)$ is once differentiable w.r.t. w.

First of all, we eliminate the last equation in eq. 2.6 by solving the SVM for a given ϑ with sufficient accuracy, see Chapter 4. By solving the SVM, we obtain the optimal w^* which we will use in the following.

From the first line in eq. 2.6 we can calculate the solution for the Lagrange multiplier.

$$\frac{\partial H}{\partial w^*} + \frac{\partial^2 E}{\partial w^{*2}}\lambda = 0$$

$$\frac{\partial^2 E}{\partial w^{*2}}\lambda = -\frac{\partial H}{\partial w^*}$$

Solving for λ yields:

$$\lambda = -\left(\frac{\partial^2 E}{\partial w^{*2}}\right)^{-1}\frac{\partial H}{\partial w^*}.$$

Substituting back into the second equation in 2.6 we can state the gradient of the Lagrangian with respect to ϑ:

$$\boxed{\frac{\partial \mathcal{L}}{\partial \vartheta} = -\frac{\partial^2 E}{\partial w^* \partial \vartheta}\left(\frac{\partial^2 E}{\partial w^{*2}}\right)^{-1}\frac{\partial H}{\partial w^*}.} \tag{2.7}$$

Using the function $H(.)$ from equation 2.3 (the mean squared error on the validation data set) we restate the bi-level problem for the linear case where $\vartheta = c$:

$$\min_{\vartheta} H(w(c), \Xi, \eta) = \min_{\vartheta}\left\{\frac{1}{2L}\|\Xi w^T - \eta\|_2^2\right\}$$

$$s.t. \quad w(c) \in \arg\min_{w}\left\{\frac{c}{2}\|\tilde{w}\|_2^2 + \sum_{i=1}^{N}\ell(w, x_i, y_i)\right\}.$$

To solve the optimization problem for c we need the following components:

$$\frac{\partial H}{\partial w} = \frac{1}{L}\Xi^T(\Xi w^T - \eta)$$

$$\frac{\partial^2 E}{\partial w^2} = c\tilde{\boldsymbol{I}} + X^T \operatorname{diag}(\ell''(Xw^T, y))X$$

$$\frac{\partial^2 E}{\partial w \partial c} = \tilde{w}$$

$$\frac{\partial E}{\partial w} = c\tilde{w} + \ell'(Xw^T, y)^T X$$

Inserting the results into equation 2.7 gives us the desired gradient w.r.t. c:

$$\frac{\partial \mathcal{L}}{\partial \vartheta} = -\tilde{w}\left(c\tilde{\boldsymbol{I}} + X^T \operatorname{diag}(\ell''(Xw^T, y))X\right)^{-1}\frac{1}{L}\Xi^T(\Xi w^T - \eta)$$

A short note on the variables used in the previous equations: $\ell''(.)$ is the second derivative of the loss function. This result shows us that we indeed need a twice differentiable approximation to the Hinge loss function. As mentioned beforehand, we include the bias in the w vector, but we do not want to regularize this value. Therefore, we introduced a modified weight vector for the derivations $\tilde{w}$ which is the same as w, only that the last entry of w is set to zero. For the same reason, inspired by the work of Zhang *et al.* [51], we introduce the diagonal matrix $\tilde{\boldsymbol{I}} \in \mathbb{R}^{D \times D}$ which is the identity matrix except that the last element in the diagonal is set to zero.

Next, we will introduce the loss function approximations we used in this work.

2.2 Loss Functions

As we have already mentioned, we need twice continuously differentiable approximations to the Hinge loss function because it is not differentiable:

$$\ell(t, y) = \max(0, 1 - yt)$$

Note that we use the abbreviation $t = Xw^T$ in the linear case. We require the approximations to be twice continuously differentiable because of the nature of the optimization procedure we use to solve the bi-level problem. The simplest of the frequently used loss functions is the logarithmic loss as discussed in section 1.3. Strictly speaking, when using the logarithmic loss $\ell(t, y) = \log(1 + e^{-yt})$, the SVM is actually a logistic regression problem. There is however a modified version of the logarithmic loss as proposed by Zhang *et al.* [51] which has a much better fit to the actual Hinge loss function. As a second approximation we did a quartic approximation of the Hinge loss function.

Table 2.1 shows a summary of the two different loss functions and their first and second derivatives. The inner derivatives of the functions are not included.

	Mod. Logarithmic	Quartic Approximation
$\ell(t, y)$	$\frac{1}{\mu}\log(1 + e^{-\mu(yt-1)})$	$\begin{cases} 1 - yt & \text{if (1)} \\ a + byt + c(yt)^2 + d(yt)^3 + e(yt)^4 & \text{if (2)} \\ 0 & \text{if (3)} \end{cases}$
$\ell'(t, y)$	$-\dfrac{y}{e^{\mu(yt-1)} + 1}$	$\begin{cases} -y & \text{if (1)} \\ (b + 2cyt + 3d(yt)^2 + 4e(yt)^3)y & \text{if (2)} \\ 0 & \text{if (3)} \end{cases}$
$\ell''(t, y)$	$\dfrac{\mu e^{\mu(yt-1)}}{(e^{\mu(yt-1)} + 1)^2}$	$\begin{cases} 0 & \text{if (1)} \\ 2c + 6dyt + 12e(yt)^2 & \text{if (2)} \\ 0 & \text{if (3)} \end{cases}$

Table 2.1: Different approximations to Hinge loss and their derivatives. Note the additional information for the quartic approximation: (1) $1-\epsilon \leq yt$, (2) $1-\epsilon < yt < 1+\epsilon$ and (3) $yt \geq 1+\epsilon$. The solutions for the coefficients of the quartic approximation are as follows: $a = \frac{3\epsilon^4+8\epsilon^3+6\epsilon^2-1}{16\epsilon^3}$, $b = -\frac{2\epsilon^3+3\epsilon^2-1}{4\epsilon^3}$, $c = \frac{3(\epsilon^2-1)}{8\epsilon^3}$, $d = \frac{1}{4\epsilon^3}$, $e = -\frac{1}{16\epsilon^3}$.

The modified logarithmic loss has a parameter μ determining the fit to the original Hinge loss function, the bigger μ is, the closer it is to Hinge loss. The second approximation is a piecewise defined quartic approximation to the Hinge loss with a parameter ϵ. The smaller the ϵ parameter, the closer the approximation fits to the original Hinge loss function.

See figures 2.3 to 2.5 to get a feeling how the loss function approximations look like.

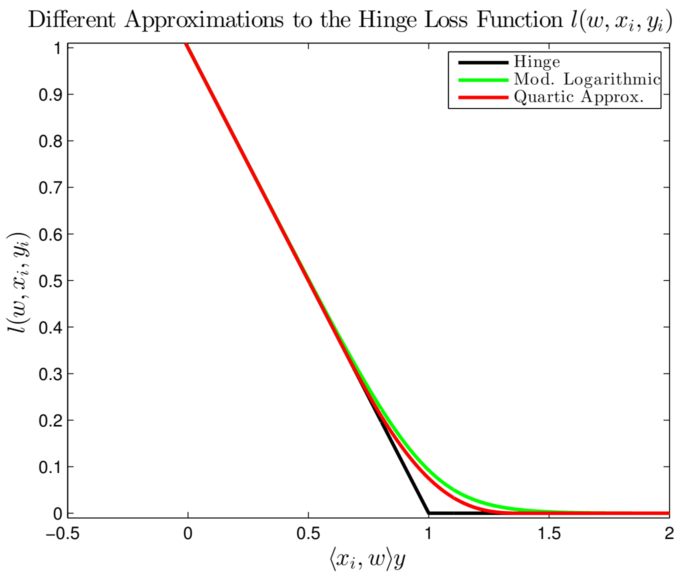


FIGURE 2.3: Different approximations of the Hinge loss function, $\mu = 7.5$, $\epsilon = \frac{2}{5}$.

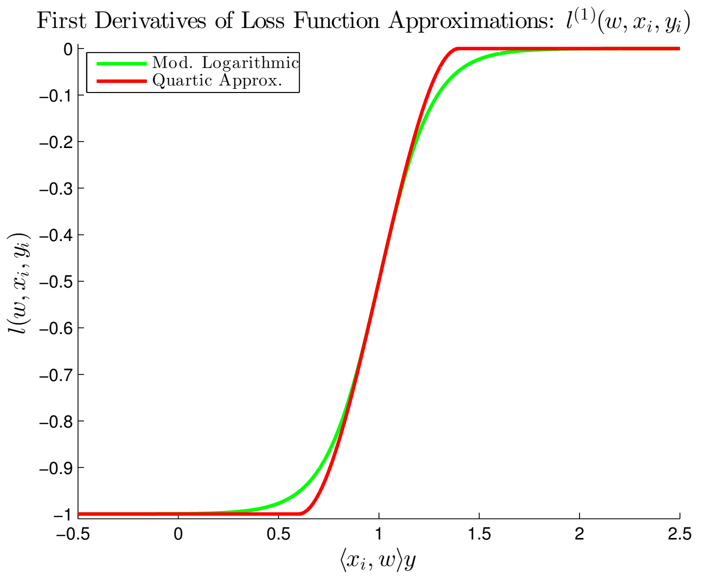


FIGURE 2.4: First derivatives of the Hinge loss approximations, $\mu = 7.5$, $\epsilon = \frac{2}{5}$.

The plots show a qualitative comparison of the modified logarithmic approximation and the quartic approximation. We chose the parameters ϵ and μ such that the approximations show a comparable peak in figure 2.5. What we observe in figure 2.3 is that the

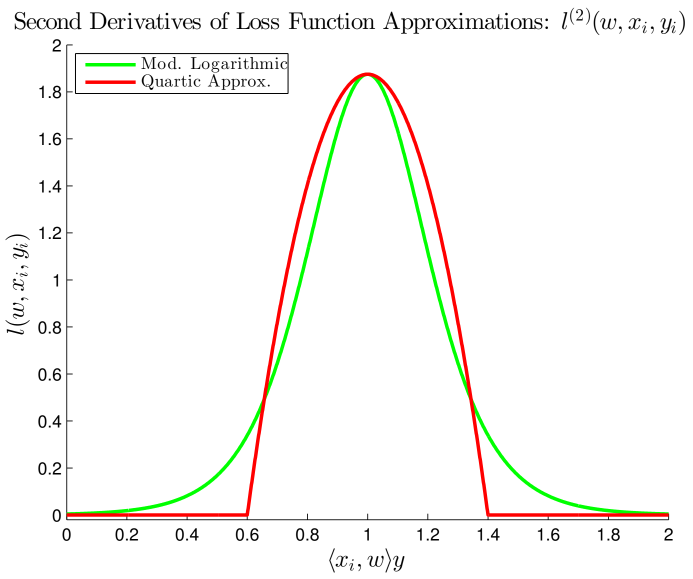


FIGURE 2.5: Second derivatives of the Hinge loss approximations, $\mu = 7.5$, $\epsilon = \frac{2}{5}$.

logarithmic function is not as exact as the piece-wise defined quartic approximation; it does only get close to the Hinge loss where the quartic approximation is already equal to Hinge loss (outside of the interval $[1-\epsilon, 1+\epsilon]$). The difference is visually not very prominent, but the difference between both approximations increases when looking at the first (fig. 2.4) or second derivatives (fig. 2.5). At the interval boundaries the logarithmic function is smoothly decaying to 0 and -1 respectively, and the peak of the second derivative is narrower than that of the quartic approximation. The results are not surprising, given the indefinitely differentiable logarithmic function as opposed to the other approach which consists of a quartic approximation in the close environment of the discontinuity of the Hinge loss function and which is otherwise the exact Hinge loss.

Next we will show how we applied cross-validation to our bi-level SVM.

2.3 Cross Validation

Cross-validation is a widely used technique to prevent a machine learning algorithm from over-fitting and to ensure a good generalization performance. For cross-validation the training set is partitioned T times into a training set and a validation set and then the bi-level problem is solved for each partition of training/validation data. The resulting parameters that give the best result are then used to retrain the model (= solve the SVM) and this gives us then the final classifier.

On our case of bi-level optimization, cross-validation is included in the optimization procedure because we do not select the model parameters "by hand" but let our bi-level program do the job for us. Say we divide our data in $t = 1, ..., T$ folds, then we need to solve T times our lower level problem for one step towards the optimal hyper-parameters.

To state an example, if we have 30 training examples and select $T = 5$ folds, we have 24 training and 6 validation examples. During the cross-validation process the examples are shuffled in $T = 5$ different configurations yielding different validation error values. The sum of the error values is minimized, see eq. 2.8 and 2.10.

The general bi-level problem for cross-validation is:

$$\min_{\vartheta} H_T(w_1(\vartheta), ..., w_T(\vartheta), \Xi_1, ..., \Xi_T, \eta_1, ..., \eta_T) \tag{2.8}$$

$$s.t. \quad w_t \in \arg\min_{w_t} E(w_t, \vartheta, X_t, y_t) \quad \text{for} \quad t = 1, ..., T.$$

The optimality condition of the t-th lower level problem is:

$$\partial_{w_t} E(w_t, \vartheta, X_t, y_t) \ni 0 \tag{2.9}$$

The higher level objective is now a sum over the validation errors of all folds, see the equation for the linear case:

$$H_T(w_1(c), ..., w_T(c), \Xi_1, ..., \Xi_T, \eta_1, ..., \eta_T) = \sum_{t=1}^{T} H_t(w_t(c), \Xi_t, \eta_t) \tag{2.10}$$

with

$$H_t(w_t(c), \Xi_t, \eta_t) = \frac{1}{2L_t} \|\Xi_t w_t^T - \eta_t\|_2^2.$$

Analogous to before, we can state the unconstrained bi-level problem $\mathcal{L}_T$ for cross-validation using equations 2.8 and 2.9:

$$\begin{aligned} &\mathcal{L}_T(w_1, ..., w_T, \vartheta, \lambda_1, ..., \lambda_T) = \\ &H_T(w_1(\vartheta), ..., w_T(\vartheta), \Xi_1, ..., \Xi_T, \eta_1, ..., \eta_T) + \sum_{t=1}^{T} \left\langle \lambda_t, \frac{\partial E(w_t, ...)}{\partial w_t} \right\rangle \end{aligned} \tag{2.11}$$

The optimality condition of equation 2.11 is given by:

$$\mathcal{G}_T(w_1,...,w_T,\vartheta,\lambda_1,...,\lambda_T)=\begin{pmatrix}\frac{\partial H_1}{\partial w_1}+\frac{\partial^2 E}{\partial w_1^2}\lambda_t\\ \vdots\\ \frac{\partial H_T}{\partial w_T}+\frac{\partial^2 E}{\partial w_T^2}\lambda_t\\ \sum_{t=1}^{T}\frac{\partial^2 E}{\partial w_t\partial\vartheta}\lambda_t\\ \frac{\partial E}{\partial w_1}\\ \vdots\\ \frac{\partial E}{\partial w_T}\end{pmatrix}=0. \tag{2.12}$$

From the last T lines of equation 2.12 by solving the t-th SVM we obtain w_t^* which we use in the first T equations to obtain the Lagrange multiplier λ_t:

$$\lambda_t=-\left(\frac{\partial^2 E}{\partial w_t^{*2}}\right)^{-1}\frac{\partial H_t}{\partial w_t^*}.$$

Consequently, instead of the original gradient $\frac{\partial\mathcal{L}}{\partial c}$ we have the sum of gradients:

$$\frac{\partial\mathcal{L}_T}{\partial c}=\sum_{t=1}^{T}\left(\frac{\partial\mathcal{L}}{\partial c}\right)_t$$

Having introduced the concept of cross-validation for the linear bi-level SVM, we continue to extend our model to deal with several regularization parameters.

2.4 Regularization Parameters for Each Dimension of the Input Data

To learn a model with more degrees of freedom using the linear SVM we extend the idea of using one single regularization parameter to using several parameters. Instead of a single regularization parameter c we now have a vector $\bar{c}\in\mathbb{R}^{1\times D}$ with one entry c_d per feature dimension of the input data (plus one due to the bias, but the last value of the $\bar{c}$ vector is set to zero).

With the following equations we want to show that we can also optimize multiple parameters with the bi-level framework. There are only subtle changes in the equations, and they affect $E(w,\bar{c},X,y)$ and the derivatives w.r.t. w only. See the equations:

$$E(w, \bar{c}, X, y) = \frac{1}{2} \sum_{d=1}^{D} c_d w_d^2 + \ell(Xw^T, y)$$

$$\frac{\partial E}{\partial w} = w \operatorname{diag}(\bar{c}) + \ell'(Xw^T, y)^T X$$

$$\frac{\partial^2 E}{\partial w^2} = \operatorname{diag}(\bar{c}) + X^T \operatorname{diag}(\ell''(Xw^T, y)X$$

We will show in Chapter 5 that it is indeed beneficial to use models with a higher number of parameters compared to models using only one or two parameters. In the next chapter, we will extend the linear bi-level SVM with kernels.

Chapter 3

Bi-Level Solution for the Kernel Support Vector Machine

In this chapter, we develop the bi-level solution for kernel Support Vector Machines. First we formulate the bi-level solution in its general form and then we specialize to different settings. We develop solutions for optimization of the regularization parameter c, then c and the kernel parameter γ combined. Next, we discuss the optimization of multiple parameters γ_d and finally we develop the bi-level solution for a multiple kernel approach. The chapter closes with a short note on cross-validation for the kernel SVM.

3.1 Bi-level Kernel SVM

In the following, we show the derivations of the bi-level kernel SVM with a simple kernel function including the optimization of both the kernel and the regularization parameters of the SVM. Again, we summarize the parameters in a vector ϑ to keep the derivations universal.

At first some remarks on the notation: In analogy with the derivations of the linear SVM found in chapter 2, we denote the training examples with x_i, $i = 1, 2, ..., N$, and the validation examples with ξ_l , $l = 1, 2, ..., L$. We denote the training targets with y_i, $i = 1, 2, ..., N$ and the validation targets with η_l, $l = 1, 2, ..., L$. One example has $d = 1, ..., D$ coordinates, with D the number of feature dimensions. Note that for the kernel SVM we do not consider any bias and therefore is D the true number of feature dimensions.

The general classification function of a kernel SVM is:

$$f(x) = \sum_{i=1}^{N} \alpha_i k(x, x_i) \tag{3.1}$$

where x is an arbitrary example. Instead of the parameter vector w that defined the optimal hyperplane for the linear SVM, the parameter vector α defines the hyperplane for the kernel SVM. We define the general loss function as follows:

$$\ell(f(x_j), y_j). \tag{3.2}$$

Putting equations 3.1, 3.2 and the SVM's energy function from eq. 2.4 together, we get the kernel SVM's training objective function:

$$\alpha(\vartheta) = \arg\min_{\alpha} \left\{ \frac{c}{2} \|f\|^2 + \sum_{j=1}^{N} \ell \left(\sum_{i=1}^{N} \alpha_i k(x_j, x_i), y_j \right) \right\}.$$

What is still missing is to resolve the squared l_2 norm of f, $\|f\|^2$:

$$\|f\|^2 = \sum_{j=1}^{N} \sum_{i=1}^{N} \alpha_j \alpha_i k(x_j, x_i) = \alpha^T K \alpha.$$

Finally we get the training objective function for the non-linear SVM:

$$\alpha(\vartheta) = \arg\min_{\alpha} \left\{ \frac{c}{2} \alpha^T K \alpha + \sum_{j=1}^{N} \ell \left(\sum_{i=1}^{N} \alpha_i k(x_j, x_i), y_j \right) \right\}.$$

Using a radial basis function (RBF) kernel, one kernel element for training is $k(x_j, x_i) = \exp(-\gamma \|x_i - x_j\|_2^2)$. Let us take a close look on the kernel matrices: We decompose the kernel Matrix K with entries $k(x_j, x_i)$ for training into rows $k_j \in \mathbb{R}^{1 \times N}$

$$k_1 = (k(x_1, x_1), k(x_1, x_2), \ldots, k(x_1, x_N))$$

which leads to the description of $K \in \mathbb{R}^{N \times N}$:

$$K = \begin{pmatrix} k_1 \\ k_2 \\ \vdots \\ k_N \end{pmatrix}.$$

Note also, that the matrix K is symmetric, meaning $K = K^T$, and positive definite which is a fundamental requirement of kernels in general.

Similarly, we decompose the kernel matrix of the validation examples into rows $\kappa_l \in \mathbb{R}^{1\times N}$

$$\kappa_1 = (k(\xi_1, x_1), k(\xi_1, x_2), \dots, k(\xi_1, x_N))$$

which leads to the description of $\mathcal{K} \in \mathbb{R}^{L\times N}$

$$\mathcal{K} = \begin{pmatrix} \kappa_1 \\ \kappa_2 \\ \vdots \\ \kappa_L \end{pmatrix}.$$

Using a RBF kernel function, one kernel element for validation is

$$k(\xi_l, x_i) = \exp(-\gamma \|\xi_l - x_i\|_2^2).$$

To sum up, let us define the matrix dimensions for important terms: The parameter vector of the SVM $\alpha \in \mathbb{R}^{N\times 1}$, one training example $x_i \in \mathbb{R}^{1\times D}$, the matrix of training examples $X \in \mathbb{R}^{N\times D}$, one validation example $\xi_l \in \mathbb{R}^{1\times D}$, the matrix of validation examples $\Xi \in \mathbb{R}^{L\times D}$, the target vectors $y \in \mathbb{R}^{N\times 1}$ and $\eta \in \mathbb{R}^{L\times 1}$.

Now we are prepared to state the bi-level problem for the kernel SVM using a generic parameter vector ϑ in the dual variable α:

$$\begin{aligned} &\min_{\vartheta} H(\alpha(\vartheta), \Xi, \eta) \\ s.t. \quad &\alpha(\vartheta) \in \arg\min_{\alpha} E(\alpha, \vartheta, X, y). \end{aligned}$$

We can substitute the lower level problem by its optimality condition:

$$\begin{aligned} &\min_{\vartheta} H(\alpha(\vartheta), \Xi, \eta) \\ s.t. \quad &\partial E_\alpha(\alpha, \vartheta, X, y) \ni 0. \end{aligned}$$

Pursuing the same approach as in chapter 2, we rewrite the bi-level problem to its unconstrained form using the Lagrange multiplier λ:

$$\mathcal{L}(\alpha, \vartheta, \lambda) = H(\alpha(\vartheta), \Xi, \eta) + \langle \lambda, \frac{\partial E}{\partial \alpha} \rangle. \tag{3.3}$$

Again, differentiating w.r.t. α, ϑ and λ we obtain the optimality system $\mathcal{G}(\alpha, \vartheta, \lambda)$ for the kernel SVM in its general form:

$$\mathcal{G}(\alpha, \vartheta, \lambda) = \begin{pmatrix} \frac{\partial H}{\partial \alpha} + \frac{\partial^2 E}{\partial \alpha^2}\lambda \\ \frac{\partial H}{\partial \vartheta} + \frac{\partial^2 E}{\partial \alpha \partial \vartheta}\lambda \\ \frac{\partial E}{\partial \alpha} \end{pmatrix} = 0. \tag{3.4}$$

Given the SVM is solved to sufficient accuracy, the last line of equation 3.4 can be eliminated and we obtain α^*. Using the first line of the optimality system given in equation 3.4 we can find the solution for the Lagrange multiplier (analogous to the linear case):

$$\frac{\partial H}{\partial \alpha^*} + \frac{\partial^2 E}{\partial \alpha^{*2}}\lambda = 0$$
$$\frac{\partial^2 E}{\partial \alpha^{*2}}\lambda = -\frac{\partial H}{\partial \alpha^*}$$

Solving for λ yields:

$$\lambda = -\left(\frac{\partial^2 E}{\partial \alpha^{*2}}\right)^{-1} \frac{\partial H}{\partial \alpha^*}. \tag{3.5}$$

Substituting back into the second equation in 3.4 we can state the gradient of the Lagrangian to ϑ using the result for λ:

$$\boxed{\frac{\partial \mathcal{L}}{\partial \vartheta} = \frac{\partial H}{\partial \vartheta} - \frac{\partial^2 E}{\partial \alpha^* \partial \vartheta}\left(\frac{\partial^2 E}{\partial \alpha^{*2}}\right)^{-1} \frac{\partial H}{\partial \alpha^*}.}$$

In general, when computing the gradient of the Lagrangian with respect to the parameters ϑ, we use the optimal α^*.

In the following we specialize to several variants of the bi-level kernel SVM.

3.1.1 Bi-level Program to Find the Optimal Regularization Parameter c

In the first version, our aim is to find the optimal regularization parameter c. Let $\vartheta = c$, then the SVM classifier can be written as follows:

$$\alpha(c) = \arg\min_{\alpha} \left\{ \frac{c}{2}\alpha^T K \alpha + \sum_{j=1}^{N} \ell(k_j \alpha, y_j) \right\} = \arg\min_{\alpha} E(\alpha, c, X, y).$$

Let us state the bi-level problem once again:

$$\begin{aligned} &\min_c H(\alpha(c), \Xi, \eta) \\ s.t. \quad &\alpha(c) \in \arg\min_\alpha E(\alpha, c, X, y) \end{aligned}$$

As higher level problem for the kernel SVM we use the mean squared error on the validation data set (compare with chapter 2):

$$H(\alpha(c), \Xi, \eta) = \frac{1}{2L}\sum_{l=1}^{L}\left(\sum_{i=1}^{N}\alpha_i k(\xi_l, x_i) - \eta_l\right)^2 = \frac{1}{2L}\sum_{l=1}^{L}(\kappa_l\alpha - \eta_l)^2 = \frac{1}{2L}\|\mathcal{K}\alpha - \eta\|_2^2.$$

We state the bi-level problem in its unconstrained form for $\vartheta = c$ using the optimality condition for the lower level problem:

$$\mathcal{L}(\alpha, c, \lambda) = H(\alpha(c), \Xi, \eta) + \langle\lambda, \frac{\partial E}{\partial \alpha}\rangle.$$

As aforementioned, we use gradient methods to solve the inner and the outer problems using the gradients $\frac{\partial E}{\partial \alpha}$ and $\frac{\partial \mathcal{L}}{\partial c}$. The gradient $\frac{\partial H}{\partial c} = 0$, the remaining components read:

$$\frac{\partial H}{\partial \alpha} = \frac{1}{L}\mathcal{K}^T(\mathcal{K}\alpha - \eta) \tag{3.6}$$

$$\frac{\partial^2 E}{\partial \alpha^2} = cK + K\,\mathrm{diag}(\ell''(K\alpha, y))K \tag{3.7}$$

$$\frac{\partial^2 E}{\partial \alpha \partial c} = \alpha^T K \tag{3.8}$$

$$\frac{\partial E}{\partial \alpha} = cK\alpha + K\ell'(K\alpha, y) \tag{3.9}$$

For the derivations of the gradients shown in equation 3.6 to 3.9 please see Appendix A.

Using the gradients above we can write the optimality system:

$$\mathcal{G}(\alpha, c, \lambda) = \begin{pmatrix} \frac{1}{L}\mathcal{K}^T(\mathcal{K}\alpha - \eta) + (cK + K\,\mathrm{diag}(\ell''(t, y))K)\lambda \\ \alpha^T K\lambda \\ cK\alpha + K\ell'(t, y) \end{pmatrix} = 0.$$

Finally, using the Lagrange multiplier λ from equation 3.5 we can state the term $\frac{\partial \mathcal{L}}{\partial c}$:

$$\boxed{\frac{\partial \mathcal{L}}{\partial c} = -\frac{\partial^2 E}{\partial \alpha^* \partial c}\left(\frac{\partial^2 E}{\partial \alpha^{*2}}\right)^{-1}\frac{\partial H}{\partial \alpha^*}} \tag{3.10}$$

which we use for the optimization procedure.

So far, we have solved the bi-level kernel SVM for a fixed kernel parameter γ, but we can find the optimal regularization parameter c via bi-level optimization. In the next step, we want to achieve bi-level optimization on both the regularization parameter c and the kernel parameter γ.

3.1.2 Bi-level Program to Find the Optimal Parameters c and γ

For solving the bi-level kernel SVM in the parameters c and γ we need a two-dimensional parameter vector

$$\vartheta = \begin{pmatrix} c \\ \gamma \end{pmatrix}$$

and the following vector of derivatives

$$\frac{\partial \mathcal{L}}{\partial \vartheta} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial c} \\ \frac{\partial \mathcal{L}}{\partial \gamma} \end{pmatrix}.$$

For the bi-level optimization process we need additionally the term $\frac{\partial \mathcal{L}}{\partial \gamma}$, the other term $\frac{\partial \mathcal{L}}{\partial c}$ we have already defined in equation 3.10:

$$\boxed{\frac{\partial \mathcal{L}}{\partial \gamma} = \frac{\partial H}{\partial \gamma} + \frac{\partial^2 E}{\partial \alpha^* \partial \gamma} \lambda.} \tag{3.11}$$

The Lagrange multiplier λ has already been defined in equation 3.5. The derivations for the terms needed to calculate the gradient $\frac{\partial \mathcal{L}}{\partial \gamma}$ given in equation 3.11 can be found in the appendix A. Here the results:

$$\frac{\partial H}{\partial \gamma} = \frac{1}{L}(\bar{\mathcal{K}}\alpha)^T(\mathcal{K}\alpha - \eta) = \frac{1}{L}\alpha^T \bar{\mathcal{K}}^T(\mathcal{K}\alpha - \eta) \tag{3.12}$$

$$\frac{\partial^2 E}{\partial \alpha \partial \gamma} = (c\bar{K}\alpha + \bar{K}\ell'(t,y) + K\,\mathrm{diag}(\ell''(t,y))\bar{K}\alpha)^T \tag{3.13}$$

Referring to the kernel matrix for training, we define $\bar{K}$ as the point-wise product of the inner derivative K_{in} and K; $\bar{K} := K \odot K_{in}$. Likewise, referring to the kernel matrix for validation, we define $\bar{\mathcal{K}}$ as the point-wise product of the inner derivative $\mathcal{K}_{in}$ and $\mathcal{K}$; $\bar{\mathcal{K}} := \mathcal{K} \odot \mathcal{K}_{in}$.

Using the gradients $\frac{\partial \mathcal{L}}{\partial c}$ and $\frac{\partial \mathcal{L}}{\partial \gamma}$ we can find optimal parameters ϑ for the SVM. The optimality system is

$$\mathcal{G}(\alpha, c, \gamma, \lambda) = \begin{pmatrix} \frac{1}{L}\mathcal{K}^T(\mathcal{K}\alpha - \eta) + [cK + K\,\mathrm{diag}(\ell''(t,y))K]\lambda) \\ \alpha^T K \lambda \\ \frac{1}{L}\alpha^T \bar{\mathcal{K}}^T(\mathcal{K}\alpha - \eta) + [c\bar{K}\alpha + \bar{K}\ell'(t,y) + K\,\mathrm{diag}(\ell''(t,y))\bar{K}\alpha]^T\lambda \\ cK\alpha + K\ell'(t,y) \end{pmatrix} = 0.$$

In order to make our model able to deal with many different kernel parameters which has several benefits, we investigate a multi-γ bi-level SVM in the next section.

3.1.3 Bi-level Program to Find the Optimal Parameters c and Several Parameters γ_d

For the multi-γ bi-level SVM we assume one kernel parameter γ_i for each feature dimension $d = 1, 2, ..., D$ of the input data. We write down one kernel element for training:

$$k(x_j, x_i) = \exp(-\sum_{d=1}^{D} \gamma_d (x_{jd} - x_{id})^2)$$

and for validation:

$$k(\xi_l, x_i) = \exp(-\sum_{d=1}^{D} \gamma_d (\xi_{ld} - x_{id})^2).$$

Let us denote the multi-γ kernel matrix with K_D and the validation kernel matrix with $\mathcal{K}_D$. For solving the lower level problem, the SVM objective, only the calculation of the kernel matrices changes versus the previous setting. For the gradients to solve the higher level problem, we need to make some modifications, though. The gradients $\frac{\partial \mathcal{L}}{\partial \gamma_d}$ are computed for each dimension and summarized in the gradient vector

$$\frac{\partial \mathcal{L}}{\partial \gamma} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial \gamma_1} \\ \frac{\partial \mathcal{L}}{\partial \gamma_2} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \gamma_D} \end{pmatrix}.$$

One element is defined by

$$\boxed{\frac{\partial \mathcal{L}}{\partial \gamma_d} = \frac{\partial H}{\partial \gamma_d} + \frac{\partial^2 E}{\partial \alpha^* \partial \gamma_d}\lambda}$$

using the Lagrange multiplier λ from equation 3.5. For the gradient $\frac{\partial H}{\partial \gamma_d}$ the inner derivations of the validation kernel matrix change compared to the gradient $\frac{\partial H}{\partial \gamma}$. We state the formula of the gradient first and then resolve the unknown variables:

$$\frac{\partial H}{\partial \gamma_d} = \frac{1}{L}(\bar{\mathcal{K}}_d \alpha)^T (\mathcal{K}_D \alpha - \eta)$$

$$\bar{\mathcal{K}}_d = \mathcal{K}_D \odot \mathcal{K}_{in,d}$$

$$\mathcal{K}_{in,d} = \begin{pmatrix} -\|\xi_{1,d} - x_{1,d}\|_2^2 & -\|\xi_{1,d} - x_{2,d}\|_2^2 & \cdots \\ \vdots & \ddots & \vdots \\ -\|\xi_{L,d} - x_{1,d}\|_2^2 & \cdots & \end{pmatrix}$$

The above modifications also apply to the gradient $\frac{\partial^2 E}{\partial \alpha \partial \gamma_d}$:

$$\frac{\partial^2 E}{\partial \alpha \partial \gamma_d} = (c\bar{K}_d \alpha + \bar{K}_d \ell'(t,y) + K_D \operatorname{diag}(\ell''(t,y))\bar{K}_d \alpha)^T.$$

3.1.4 Cross Validation for the Bi-level Kernel SVM

In section 2.3 we already described the general procedure of cross-validation applied in the bi-level learning, and the same principles apply for the bi-level kernel SVM as well. The general bi-level problem for cross-validation of the kernel SVM can we written as follows, assuming that we partition the data in $t = 1, ..., T$ folds:

$$\min H_T(\alpha_1(\vartheta), ..., \alpha_T(\vartheta), \Xi_1, ..., \Xi_T, \eta_1, ..., \eta_T)$$

$$s.t. \quad \alpha_t(\vartheta) \in \underset{\alpha_t}{\arg\min}\, E(\alpha_t, \vartheta, X_t, y_t) \quad \text{for} \quad t = 1, ..., T$$

While the kernel SVM is solved for each fold t, the higher level objective H_T is the sum over the validation errors of each fold:

$$H_T(\alpha_1(\vartheta), ..., \alpha_T(\vartheta), \Xi_1, ..., \Xi_T, \eta_1, ..., \eta_T) = \sum_{t=1}^{T} H_t(\alpha_t(\vartheta), \Xi_t, \eta_t) = \sum_{t=1}^{T} \frac{1}{2L_t}\|\mathcal{K}_t \alpha_t - \eta_t\|_2^2.$$

As explained in section 2.3, we can state the unconstrained bi-level problem $\mathcal{L}_T$ for cross-validation as follows:

$$\begin{aligned} &\mathcal{L}_T(\alpha_1, ..., \alpha_T, \vartheta, \lambda_1, ..., \lambda_T) = \\ &H_T(\alpha_1(\vartheta), ..., \alpha_T(\vartheta), \Xi_1, ..., \Xi_T, \eta_1, ..., \eta_T) + \sum_{t=1}^{T} \left\langle \lambda_t, \frac{\partial E(\alpha_t, ...)}{\partial \alpha_t} \right\rangle \end{aligned} \tag{3.14}$$

The optimality condition of equation 3.14 is given by:

$$\mathcal{G}_T(\alpha_1, ..., \alpha_T, \vartheta, \lambda_1, ..., \lambda_T) = \begin{pmatrix} \frac{\partial H_1}{\partial \alpha_1} + \frac{\partial^2 E}{\partial \alpha_1^2}\lambda_t \\ \vdots \\ \frac{\partial H_T}{\partial \alpha_T} + \frac{\partial^2 E}{\partial \alpha_T^2}\lambda_t \\ \sum_{t=1}^T \left(\frac{\partial H_t}{\partial \vartheta} + \frac{\partial^2 E}{\partial \alpha_t \partial \vartheta}\lambda_t \right) \\ \frac{\partial E}{\partial \alpha_1} \\ \vdots \\ \frac{\partial E}{\partial \alpha_T} \end{pmatrix} = 0. \tag{3.15}$$

From the last T lines of equation 3.15 by solving the t-th SVM we obtain α_t^* which we use in the first T equations to obtain the Lagrange multiplier λ_t:

$$\lambda_t = -\left(\frac{\partial^2 E}{\partial \alpha_t^{*2}}\right)^{-1} \frac{\partial H_t}{\partial \alpha_t^*}.$$

The relevant vector of gradients of the Lagrangian $\mathcal{L}_T$ w.r.t. the parameter vector ϑ contains the sums of the individual gradients over all folds as its components:

$$\frac{\partial \mathcal{L}_T}{\partial \vartheta} = \begin{pmatrix} \sum_{t=1}^T (\frac{\partial \mathcal{L}}{\partial c})_t \\ \sum_{t=1}^T (\frac{\partial \mathcal{L}}{\partial \gamma})_t \end{pmatrix}.$$

The next section is dedicated to the last extension to the bi-level kernel SVM. We consider the approach of using several kernels and develop the corresponding bi-level solution.

3.2 Multiple Kernel SVM

The subject of this section is the extension of the previous model to use multiple kernels instead of a single kernel. A good overview about multiple kernel learning algorithms is provided e.g. by Gonen *et al.* [22]. There are different application scenarios for the usage of multiple kernels: Such a model can be used to combine different subsets of heterogeneous features, to combine different feature representations or different data sources which require different measures of similarity.

Siddiquie *et al.* [41] describe in their paper how they combine multiple kernels for image classification, where they use one kernel per feature channel (texture, color, histograms of oriented gradients, saliency features). There is a similar approach by Vedaldi *et al.* [47]: they combine χ^2 kernels for different feature channels such as edge distribution, visual words and spatial feature descriptors.

We define the model as follows: Let $p = 1, 2, \ldots, P$ be the partitions (i.e. equivalent to the number of kernels used) each of which are of dimension D_p. A training example can be written as concatenation of P feature subsets: $x_i = \{x_i^1, x_i^2, \ldots, x_i^P\}$ whereas $x_i^p \in \mathbb{R}^{D_p \times 1}$. Usually, the number and size of the feature partitions and the type of kernels are predefined according to the data used. The resulting classification function is then a weighted sum of P kernels:

$$f(x) = \sum_{i=1}^{N} \alpha_i \underbrace{\left(\sum_{p=1}^{P} \beta_p k_p(x^p, x_i^p) \right)}_{k_\beta(x, x_i)} .$$

Note that we use the constraint $\beta_p \in \mathbb{R}^+$ in order to ensure that the kernel stays positive definite. The SVM's training objective function reads:

$$\alpha(\vartheta) = \arg\min_{\alpha} \left\{ \frac{c}{2} \|f\|^2 + \sum_{j=1}^{N} \ell \left(\sum_{i=1}^{N} \alpha_i k_\beta(x_j, x_i), y_j \right) \right\}$$

with

$$\|f\|^2 = \sum_{j=1}^{N} \sum_{i=1}^{N} \alpha_j \alpha_i k_\beta(x_j, x_i) = \alpha^T K_\beta \alpha.$$

One element of the new multiple kernel matrix is $k_\beta(x_j, x_i)$. The kernel matrix $K_\beta \in \mathbb{R}^{N \times N}$ consists of rows $k_{\beta j} = (k_\beta(x_j, x_1), k_\beta(x_j, x_2), \ldots, k_\beta(x_j, x_N))$. Therefore,

$$K_\beta = \begin{pmatrix} k_{\beta 1} \\ k_{\beta 2} \\ \vdots \\ k_{\beta N} \end{pmatrix} .$$

Likewise, we can define the validation kernel and its elements. One element of the validation kernel matrix is $\kappa_\beta(\xi_l, x_i)$. The validation kernel matrix $\mathcal{K}_\beta \in \mathbb{R}^{L \times N}$ consists of rows $\kappa_{\beta l} = (\kappa_\beta(\xi_l, x_1), \kappa_\beta(\xi_l, x_2), \ldots, \kappa_\beta(\xi_l, x_N))$. Therefore,

$$\mathcal{K}_\beta = \begin{pmatrix} \kappa_{\beta 1} \\ \kappa_{\beta 2} \\ \vdots \\ \kappa_{\beta L} \end{pmatrix} .$$

Simplified writing of the training objective function results in

$$\alpha(\vartheta) = \arg\min_{\alpha} \left\{ \frac{c}{2}\alpha^T K_\beta \alpha + \sum_{j=1}^{N} \ell(k_{\beta j}\alpha, y_j) \right\} = \arg\min_{\alpha} E(\alpha, \vartheta, K_\beta, y).$$

The bi-level problem adapted to the new kernel matrices is

$$\min_{\vartheta} H(\alpha(\vartheta), \mathcal{K}_\beta, \eta)$$

$$s.t. \quad \alpha(\vartheta) \in \arg\min_{\alpha} E(\alpha, \vartheta, K_\beta, y)$$

The parameter vector ϑ contains additionally the β parameter for multiple kernel learning:

$$\vartheta = \begin{pmatrix} c \\ \bar{\gamma} \\ \beta \end{pmatrix}.$$

Note that we have P parameters γ_p and β_p, one for each feature subset:

$$\bar{\gamma} = (\gamma_1, \gamma_2, \dots, \gamma_P)^T$$

$$\beta = (\beta_1, \beta_2, \dots, \beta_P)^T$$

The updated higher level problem reads:

$$H(\alpha(\vartheta), \mathcal{K}_\beta, \eta) = \frac{1}{2L}\sum_{l=1}^{L}(\kappa_{\beta l}\alpha - \eta_l)^2 = \frac{1}{2L}\|\mathcal{K}_\beta\alpha - \eta\|^2.$$

In all terms that we need to solve the bi-level problem where kernel matrices are involved, the matrices K and $\mathcal{K}$ are substituted by K_β and $\mathcal{K}_\beta$ respectively. Additionally, we need to derive the gradients $\frac{\partial \mathcal{L}}{\partial \bar{\gamma}}$ and $\frac{\partial \mathcal{L}}{\partial \beta}$. We can view the partial derivatives of $\mathcal{L}$ w.r.t. $\bar{\gamma}$ element-wise:

$$\frac{\partial \mathcal{L}}{\partial \bar{\gamma}} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial \gamma_1} \\ \frac{\partial \mathcal{L}}{\partial \gamma_2} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \gamma_P} \end{pmatrix}.$$

We assume that we use RBF kernels. In the appendix B we show the derivations of $\frac{\partial H}{\partial \gamma_p}$ and $\frac{\partial^2 E}{\partial \alpha \partial \gamma_p}$. Here the results:

$$\frac{\partial H}{\partial \gamma_p} = \frac{1}{L}\alpha^T \beta_p \bar{\mathcal{K}}_\beta^{pT}(\mathcal{K}_\beta \alpha - \eta) \tag{3.16}$$

$$\frac{\partial^2 E}{\partial \alpha \partial \gamma_p} = (c\beta_p \bar{K}_\beta^p \alpha + \beta_p \bar{K}_\beta^p \ell'(t_\beta, y) + K_\beta \operatorname{diag}(\ell''(t_\beta, y))\beta_p \bar{K}_\beta^p \alpha)^T \tag{3.17}$$

The matrix $\bar{K}_\beta^p$ is the point-wise product of the generic inner derivative of K and the multiple kernel matrix K_β parametrized with kernel parameter γ_p, $\bar{K}_\beta^p = K_{in} \odot K_\beta^p$. Equivalently for the validation kernel, $\bar{\mathcal{K}}_\beta^p = \mathcal{K}_{in} \odot \mathcal{K}_\beta^p$. The variable t_β is used as a short cut for $t_\beta = K_\beta \alpha$.

To complete the list of derivatives needed for solving the bi-level problem for the multiple kernel SVM we state the derivatives of $\mathcal{L}$ w.r.t. to β component-wise:

$$\frac{\partial \mathcal{L}}{\partial \beta} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial \beta_1} \\ \frac{\partial \mathcal{L}}{\partial \beta_2} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \beta_p} \end{pmatrix}.$$

The necessary building blocks of the derivatives are $\frac{\partial H}{\partial \beta_p}$ and $\frac{\partial^2 E}{\partial \alpha \beta_p}$ are stated in the following equations (again, please find the detailed derivations in Appendix B):

$$\frac{\partial H}{\partial \beta_p} = \frac{1}{L}\alpha^T \mathcal{K}_\beta^{pT}(\mathcal{K}_\beta \alpha - \eta) \tag{3.18}$$

$$\frac{\partial^2 E}{\partial \alpha \beta_p} = (cK_\beta^p \alpha + K_\beta^p \ell'(t_\beta, y) + K_\beta \operatorname{diag}(\ell''(t_\beta, y))K_\beta^p \alpha)^T \tag{3.19}$$

Cross-validation for the bi-level multiple kernel SVM works analogous to the descriptions in the previous sections (see sections 2.3 and 3.1.4).

In the next chapter, we discuss briefly the optimization algorithms that we used.

Chapter 4

Optimization Algorithms

This section provides a short overview of the used optimization algorithms both for solving the SVM classifier (the inner problem) and for optimizing the outer problem. For solving the SVM, we applied the FISTA algorithm [5]. For solving the bi-level outer objective, implementations of the RPROP [37] algorithm or the LBFGS-B [13, 55] algorithm were used alternatively.

It is important to note that we incorporate bounds on the hyper-parameters via the used optimization algorithms. LBFGS-B inherently supports simple bounds on the optimized variables, RPROP does not.

4.1 FISTA

The first step of selecting a suitable optimization algorithm is to analyse the structure of the optimization problem. The lower level problem has a clear structure, that falls into the class of problems that can be solved efficiently with the FISTA algorithm. FISTA stands for 'fast iterative shrinkage-thresholding algorithm' and was developed by Beck and Teboulle in 2009 [5]. The structure of the optimization problem that can be solved by their approach is as follows (throughout this section, we stick to the notation in [5]):

$$\min_x \{F(x) \equiv f(x) + g(x) : x \in \mathbb{R}^n\}$$

where $g : \mathbb{R}^n \to \mathbb{R}$ is a continuous, convex but possibly non-smooth function with a simple, in-expensive proximal map, and $f : \mathbb{R}^n \to \mathbb{R}$ is a continuous, smooth, convex function $\in C^{1,1}$ that is continuously differentiable with Lipschitz continuous gradient and a Lipschitz constant $M(f) > 0$:

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq M(f)\|x - y\|_2 \quad \text{for every} \quad x, y \in \mathbb{R}^n.$$

We analyse the SVM objective function described in equation 2.4, see below a more compact writing:

$$E(w, c, X, y) = \frac{1}{2}\|\tilde{w}\|_2^2 + \ell(Xw^T, y).$$

We have with $f(x) = \ell(Xw^T, y)$ the smooth convex and continuously differentiable function and with $g(x) = \frac{1}{2}\|\tilde{w}\|_2^2$ the possible non-smooth regularizer with proximal map which is easy to compute. The requirement of $f(x)$ to be convex and continuously differentiable is one of the reasons why we use smooth approximations to the Hinge loss function fulfilling those preconditions, see section 2.2. Note that for optimizing the bi-level problem, we need even the second derivatives of $\ell(t, y)$.

Considering the kernel SVM, we have

$$E(\alpha, \vartheta, K, y) = \frac{c}{2}\alpha^T K\alpha + \ell(K\alpha, y)$$

for the energy function. Here, the proximal map of the regularization term is not easy to compute, so we set $f(x) = E(\alpha, \vartheta, K, y)$ and $g(x) = 0$. F(x) is then a smooth convex optimization problem.

Another nice property of the FISTA algorithm is that it is a first order method which means that the algorithm only uses function values and the first derivative of the function to be optimized. This is an important enabling factor for large-scale optimization. Further, the FISTA algorithm has a proven convergence bound of $F(x_k) - F(x^*) \leq \mathcal{O}(\frac{1}{k^2})$ with k the number of optimization steps and x^* the optimal solution. Specifically, we use the FISTA algorithm with backtracking (line search) including the automatic estimation of the Lipschitz constant $M > 0$ which acts as a step size parameter.

In the following, we state the FISTA algorithm with backtracking from [5]:

Step 0 Take $M_0 > 0$, $\zeta > 1$, $x_0 \in \mathbb{R}^n$. Set $y_1 = x_0$, $\tau_1 = 1$.

Step k ($k \geq 1$) Find the smallest non-negative integers i_k such that with $\bar{M} = \zeta^{i_k} M_{k-1}$

$$F(p_{\bar{M}}(y_k)) \leq Q_{\bar{M}}(p_{\bar{M}}(y_k), y_k).$$

Set $M_k = \zeta^{i_k} M_{k-1}$ and compute

$$x_k = p_{M_k}(y_k),$$

$$\tau_{k+1} = \frac{1 + \sqrt{1 + 4\tau_k^2}}{2},$$

$$y_{k+1} = x_k + \left(\frac{\tau_k - 1}{\tau_{k+1}}\right)(x_k - x_{k-1}).$$

The quadratic approximation $Q_M(x, y)$ of $F(x)$ at a point y reads:

$$Q_M(x, y) = f(y) + \langle \nabla f(y), x - y \rangle + \frac{M}{2} ||x - y||_2^2 + g(x).$$

The key of the FISTA algorithm is $x_k = p_M(y_k)$ with

$$p_M(y) = \arg\min_x \{Q_M(x, y)\} = \operatorname{prox}_{g, \frac{1}{M}}(y - \frac{1}{M} \nabla f(y)).$$

Note that the proximal operator $p_M(.)$ is applied to the point y_k which is a particular linear combination of the previous points x_k and x_{k-1}.

Beck and Teboulle [5] point out the connection between the standard gradient algorithm for solving the problem $\min f(x)$ with the key step of FISTA. They state that one iteration of gradient descent

$$x_k = x_{k-1} - \tau_k \nabla f(x_{k-1})$$

can be interpreted as the proximal regularization of a linearisation of f at the point x_{k-1}:

$$x_k = \arg\min_x \left\{ f(x_{k-1}) + \langle \nabla f(x_{k-1}), x - x_{k-1} \rangle + \frac{1}{2\tau_k} \|x - x_{k-1}\|^2 \right\}.$$

The following sections are dedicated to the algorithms used to optimize the higher level objective. First, we discuss the LBFGS-B algorithm.

4.2 LBFGS-B

Limited memory BFGS-B is an algorithm by Byrd *et al.* [13] for solving large non-linear optimization problems with bounds on the optimized variables of the form $l_i \leq x_i \leq u_i$. We can state the optimization problem as follows:

$$\min f(x)$$

$$s.t. \quad l \leq x \leq u$$

with $f : \mathbb{R}^n \to \mathbb{R}$ a non-linear function where the gradient g can be computed. The variables to be optimized are contained in the vector x, with n the number of variables (n can be large). Vectors l and u contain the lower and upper bounds on the variables in x. The higher level objective of the bi-level problem fulfills these requirements, so we can use this algorithm for optimization.

The algorithm belongs to the class of Quasi-Newton methods which use gradient information and approximations to the Hessian matrix for optimization. There is no explicit calculation of the second derivative of f needed, therefore the method can be used when the Hessian matrix cannot easily be computed. In the case of LBFGS-B, the complexity of the approximation to the Hessian matrix is linear in the memory requirements. The algorithm uses line search for determining the optimization direction.

In the following, an outline of the LBFGS-B algorithm is given (we stick to the notation in [13]).

- At iteration $k = 0$, an initial point x_0 and an integer m which defines the number of stored limited memory BFGS matrices B_k (approximations of the Hessian matrix) are chosen. The initial matrix B_0 is set to the identity matrix.
- In the beginning of each iteration, we are given the current iterate x_k, the corresponding function value f_k, the gradient of f at step k, g_k, and a positive definite approximation to the Hessian matrix of f, B_k.
- Based on that information, we can write a quadratic approximation of f_k:

$$m_k(x) = f_k + g_k^T(x - x_k) + \frac{1}{2}(x - x_k)^T B_k (x - x_k).$$

- At first, the gradient projection method is used to determine the set of active bounds, followed by a projection of the variables x_i onto the feasible region defined by the given bounds.

- The variables x_i that are at a lower bound l_i or at an upper bound u_i are held fixed, the rest of the variables are considered free variables for the next step. Then, the algorithm approximately minimizes $m_k(x)$ for the free variables, ignoring the bounds at first and then truncating the solution to satisfy the bounds.
- This approximate feasible solution $\bar{x}_{k+1}$ is used to determine the next iterate x_{k+1} by a line search procedure ensuring that the variables stay in the feasible region.
- Finally, the gradient g_{k+1} and B_{k+1} are computed, copies of B_k older than m iterations are discarded.
- If convergence has not yet been reached, set $k = k+1$ and start the next iteration.

We used the implementation provided by Liam Steward[1]. In the next section, we discuss the RPROP algorithm as an alternative to solve the higher level objective of the bi-level optimization problem.

4.3 RPROP

RPROP stands for 'resilient propagation' and was proposed by Riedmiller *et al.* [37, 36] as a heuristic alternative to standard gradient descent with the suggested application scenario but not limited to neural networks. A comparison of different first-order learning methods has been made e.g. in [25] where they state that the RPROP algorithm is "one of the best performing first-order learning methods for neural networks". Another group of researchers applied the RPROP algorithm to a problem in the context of image segmentation [1]. This shows that RPROP can be successfully applied for different kinds of optimization problems.

In the following, we will discuss the shortcomings of standard gradient descent and show how the RPROP algorithm works. See the formula for standard gradient descent:

$$x_{k+1} = x_k - \tau \nabla f_k. \tag{4.1}$$

Let x the variable we want to optimize, $k > 0$ the current number of iterations, τ the step size of the parameter update. ∇f_k denotes the gradient of the objective function f at x_k. Note that we consider x a multi-dimensional variable, let $x \in \mathbb{R}^{D \times 1}$. During gradient descent, the variable x moves in the negative direction of the gradient of the error function until convergence. The well-known problems with standard gradient descent are that the algorithm tends to converge to local optima and usually shows slow convergence.

[1] `http://www.cs.toronto.edu/~liam/software.shtml`

A critical parameter in eq. 4.1 is τ, also considered the learning rate. If τ is too small then the algorithm might get stuck in flat regions of the error surface, and if it is to high the algorithm might start to oscillate around a (local) minimum. Thus, the step size τ is highly influential on the number of iterations needed until convergence and is usually set via an adaptive scheme. Also RPROP uses an adaptive scheme for updating the step size, but contrary to standard methods it does not consider the size of the gradient but only the sign of the gradient. The adaptation works based on the principle that when the gradient value changes its sign the algorithm may have missed a local minimum or simply took a too big step. In that case the algorithm reduces the step size. If the gradient value keeps its sign, the step size is increased in order to speed up convergence.

RPROP uses one individual "update-value" [37, 36, 1] for each dimension in x, which are summarized in the vector $\Delta_k \in \mathbb{R}^{D\times 1}$. The update-values are determined as follows (for each dimension d):

$$\Delta_k^d = \begin{cases} \min(\Delta_{k-1}^d \eta^+, \Delta_{max}) & \nabla^d f_k \nabla^d f_{k-1} > 0 \\ \max(\Delta_{k-1}^d \eta^-, \Delta_{min}) & \nabla^d f_k \nabla^d f_{k-1} < 0 \\ \Delta_{k-1}^d & \nabla^d f_k \nabla^d f_{k-1} == 0. \end{cases}$$

The gradient update term $-\tau \nabla f_k$ in eq. 4.1 is then replaced by the following:

$$x_{k+1} = x_k - \operatorname{sign}(\nabla f_k) \odot \Delta_k$$

where $\odot$ denotes the element-wise multiplication of the update-values and the sign of the derivatives of the error function. The increase and decrease parameters η^+ and η^- are usually fixed to 1.2 and 0.5 respectively. The parameters restricting the size of the update-values Δ_{min} and Δ_{max} are set e.g. to 10^{-6} and 50 [37]. One of the advantages of RPROP is that the algorithm is very robust to its internal parameters, usually the algorithm works with the stated standard values. Another advantage is that the magnitude of the gradient does not influence the size of the update step. This makes the algorithm very robust also to noisy error functions and numerical gradients [25] and resilient to initial values. RPROP shows empirically fast convergence due to the update step adaptation.

We use the RPROP algorithm to optimize the higher level objective of the bi-level framework. As already mentioned, we need to ensure that the parameters we want to optimize in the bi-level framework stay within certain bounds, at least we require that the parameters stay positive. RPROP does not inherently support bounds on the variables, however we found a way to deal with the problem. As a workaround, we check in each optimization step whether the parameter values fall in a feasible region and if not, we project the value to the closest feasible value. In our case, we restrict the parameters to be strictly positive. Additionally, for kernel parameters, we set the threshold to τ which is the smallest value which makes sense for the chosen kernel function.

In the next chapter, we will discuss the characteristics of the optimization algorithms in the context of several basic experiments.

Chapter 5

Basic Experiments and Evaluation

In this section we evaluate the bi-level SVM in all its variants developed in Chapters 2 and 3 on small scale data sets and discuss the results. First, we evaluate several choices for loss function parameters according to different criteria and select them for the further experiments. Second, we consider grid search vs. the bi-level SVM and compare both approaches by experiments. Furthermore, we present results using a bi-level SVM with multiple kernel as well as multiple regularization parameters and one experiment with a bi-level SVM using multiple kernels. Finally, we discuss shortly the use of an alternative function H instead of the mean squared error.

5.1 Selection of Loss Function Parameters

For the evaluation of loss function parameters we use the Iris data set [2]. The data set consists of three classes of Iris plants, and the four features are measurements of petal and sepal lengths of instances of the flower. In our example we learn to discriminate between the Iris Setosa and the other two (Iris Versicolor and Iris Virginica).

We consider a linear bi-level SVM for this evaluation and compare the number of SVM evaluations needed to find a solution using the LBFGS-B algorithm. We always used the same initial value for optimization ($c_0 = 9$) and the very high precision of 10^{-12} for solving the SVM. Additionally, we state the minimal validation error H_{min} and the classification rates (CR) on the training and the test data. The results are shown in table 5.1.

Modified Logarithmic					Quartic Approximation				
μ	Eval.	H_{min}	CR (train)	CR (test)	ϵ	Eval.	H_{min}	CR (train)	CR (test)
7.5	9	0.0588	91.33 %	93.50 %	0.4	47	0.0552	91.85 %	93.60 %
12	10	0.0541	92.02 %	93.84 %	0.25	11	0.0527	92.24 %	94.07 %
24	33	0.0515	92.45 %	94.12 %	0.125	10	0.0512	92.51 %	94.18 %
32	35	0.0510	92.53 %	94.15 %	0.1	19	0.0509	92.56 %	94.18 %
64	45	0.0506	92.61 %	94.17 %	-	-	-	-	-

Table 5.1: Evaluation of loss function parameters. The table shows the number of evaluations needed for linear bi-level optimization, the minimal validation error H_{min} and the corresponding training and testing classification rates (CR) for selected choices of parameters μ and ϵ for the two loss functions (modified logarithmic loss and the quartic approximation, see section 2.2).

We find that a trade-off has to be made between number of SVM evaluations needed and acceptable classification rates. When the loss function approximation is more accurate (high μ or small ϵ) the resulting error rates are getting better until we observe some saturation, but also more SVM evaluations are needed to solve the SVM. Fewer evaluations however are beneficial for the performance of the bi-level program because they result in faster computation times, and the gain of performance is not very high (less than 1% on the test classification rate). Comparing the modified logarithmic loss and the quartic approximation, we observe that the resulting classification rates for a comparable number of SVM evaluations are slightly better for the latter, but not substantially. We decide to use $\mu = 12$ for the logarithmic loss and $\epsilon = 0.125$ for the following experiments due to the low number of SVM evaluations needed and the comparably good classification rates. In figure 5.1 we show the loss functions and their first and second derivatives with the examined parameters shown in table 5.1 (apart from $\mu = 64$ due to the very high peak in the second derivative). Note that we chose the shown parameters such that the peaks of the second derivatives were approximately comparable. We argue that the classification rates using the quartic approximation are better than the rates achieved using the modified logarithmic loss because the transitions of the logarithmic loss to 0 or -1 are never as sharp as the transitions of the quartic approximation, almost independent of the chosen parameters.

5.2 Grid Search vs. Bi-level Approach

As previously mentioned, grid search is a common method to choose the hyper-parameters for a machine learning problem. Depending on the setting, whether linear or kernel SVM, the number of parameters contained in the vector ϑ is different. We assume $i = 1, ..., N$ distinct parameters that we want to optimize. Then, for grid search, we need to choose

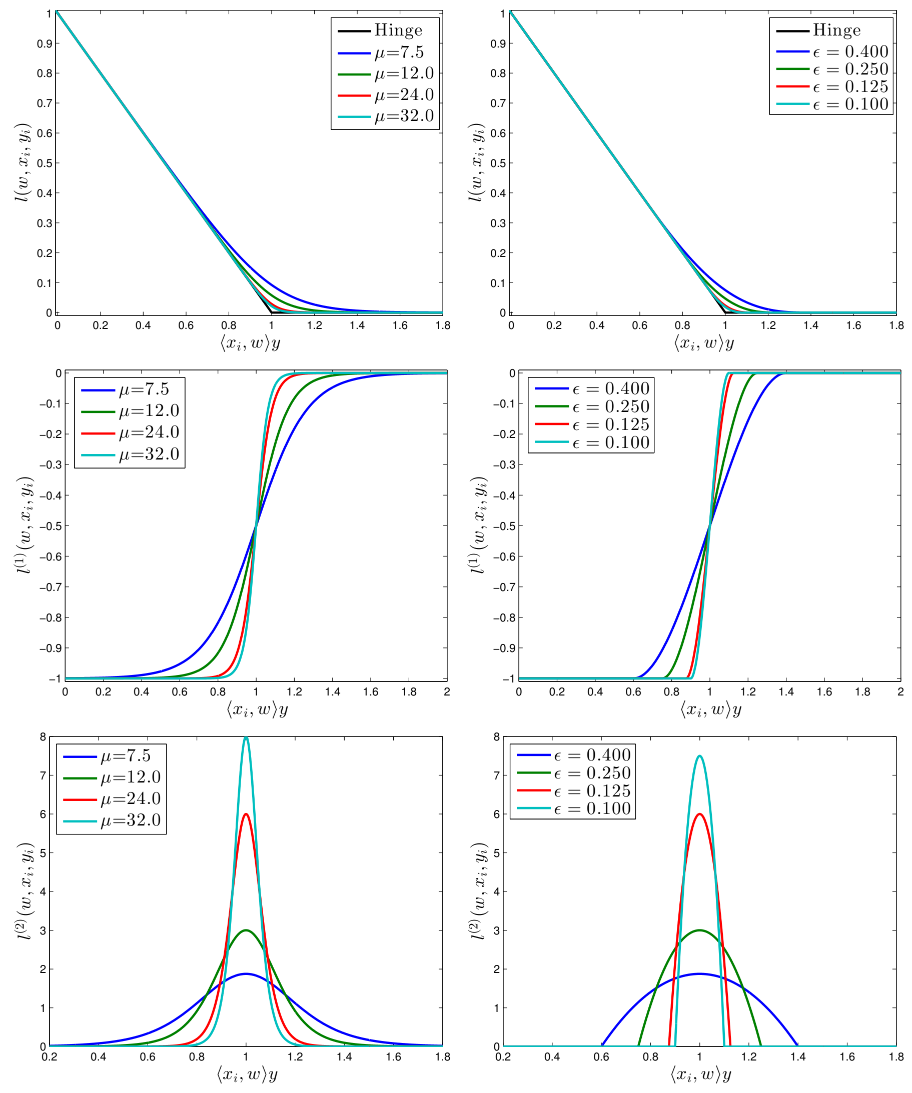


Figure 5.1: The figures show the modified logarithmic loss (left column) and the quartic loss (right column) for different parameters μ and ϵ, respectively. From top to bottom, the original functions, their first and second derivatives are shown. For the further experiments, we choose $\mu = 12$ (green curve) and $\epsilon = \frac{1}{8}$ (red curve).

a range of reasonable values for each component ϑ_i and evaluate the performance of the machine learning algorithm on each combination of values via exhaustive search. We can easily see that the grid search approach is suffering from the so-called 'curse of dimensionality' - the complexity of grid search grows exponentially with the number of parameters, see e.g. [8]. In the following subsections, we want to compare the grid search approach with our bi-level solution. Note that in sections 5.2.1 and 5.2.2 no cross-validation is used for reasons of better comparability between grid search and the bi-level optimized SVM.

5.2.1 Grid Search vs. One-dimensional Bi-level SVM

First, we compare both approaches in a one-dimensional setting. It is clear that the grid over one parameter is not expensive to compute, but we can show that with bi-level optimization we can find the optimal hyper-parameter in much fewer steps. One advantage of bi-level optimization is that the optimal parameters can be learnt up to a very high precision (which is important for applications that are very sensitive to the parameters). Additionally, the bi-level approach exploits the continuity of the hyper-parameters contrary to grid search.

A critical choice for grid search is how accurate the parameters are to be determined. For our experiment we assume a range of reasonable values for $c \in [1, 100]$. The data we use is the same as in the previous section, the Iris data set [2].

In figure 5.2 the resulting plots of the validation error $H(.)$ over the parameter c are shown for the two optimization algorithms (LBFGS-B and RPROP) and the two examined loss functions (modified logarithmic loss and the quartic approximation). The blue curves show the result of grid search, the red circles mark the iterates c_k of the optimization algorithms. In the figure, we show grid search on c with a step size of 0.01: The finer the step size, the more accurate we can determine the optimal c. In order to be able to compare the computational cost, we count for both grid search and bi-level optimization the number of SVM evaluations (i.e. computing the solution of the SVM for a given c) required to obtain the optimal hyper-parameter.

Observing the behaviour of both algorithms in the 1D case, we can see that the RPROP algorithm increases its step size gradually as long as the sign of the gradient does not change, and is reset after stepping over the minimum which comes with a change of the direction of the gradient. So the last few steps are small and relatively many additional steps are required to achieve a high accuracy compared to the LBFGS-B algorithm. Therefore it is more costly to use RPROP than LBFGS-B for the one-dimensional setting (in this example RPROP needs approximately 15 additional SVM evaluations after changing the direction). The LBFGS-B algorithm uses line search which is very effective in 1D and therefore needs very few SVM evaluations to find the optimal hyper-parameter.

When comparing the shape of the $H(.)$ function for the different loss function approximations in figure 5.2 we observe that the quartic approximation allows for slightly better error values (H_{min}) than the modified logarithmic loss, possibly due to the fact that the quartic approximation is being closer to the Hinge loss (see section 2.2). The difference in the achieved test error, however, is marginal.

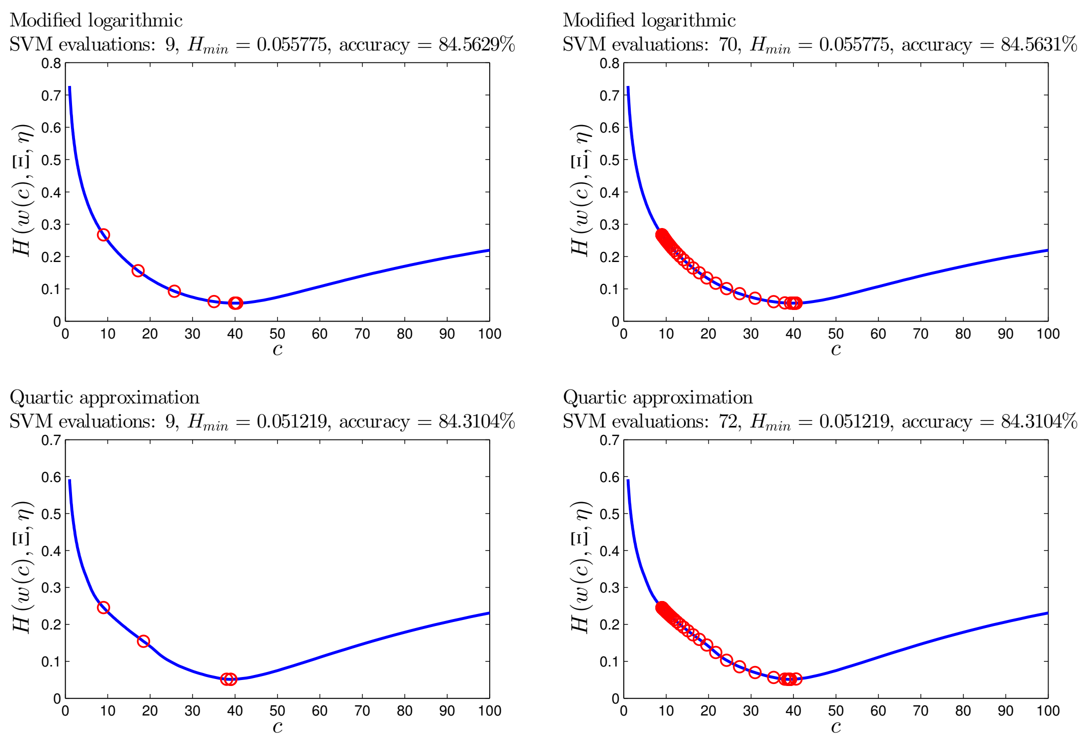


FIGURE 5.2: Iris data set. Comparison grid search and bi-level learning of the regularization parameter c. The red circles show the iteration steps of the LBFGS-B and RPROP optimization algorithms respectively, with $c_0 = 9$. The left column shows the results for the LBFGS-B algorithm, the right column the results for the RPROP algorithm. Each line shows the results for one of the two examined loss functions. Note that the number of SVM evaluations does not directly correspond to the number of red dots in the case of LBFGS-B due to the line search procedure. Additionally to the SVM evaluations, we report the determined H_{min} by the bi-level program and the achieved accuracy on a test set.

In table 5.2, we compared the number of SVM evaluations needed to find the optimal hyper-parameter c for different step sizes of grid search. We call our linear bi-level program ”bilinear” in short. As initial value for the bilinear program we chose $c_0 = 9$.

Precision	Bilinear (LBFGS-B) SVM evaluations	Bilinear (RPROP) SVM evaluations	Grid Search SVM evaluations
10^{0}	8	51	99
10^{-1}	8	57	990
10^{-2}	9	60	9900
10^{-3}	10	67	99000

TABLE 5.2: Number of SVM evaluations required to determine the regularization parameter c up to a given precision. We compare grid search with the linear bi-level program (bilinear) using LBFGS-B and RPROP alternatively. The numbers for the bilinear program are averaged over the three loss function approximations because they result in a slightly different number of evaluations until convergence.

What we observe is that the bilinear program requires significantly fewer SVM evaluations than grid search until the optimal solution for c is found. LBFGS-B is faster than RPROP due to the line search procedure which allows for large steps. When using continuous optimization a solution can be found with high precision anyway, and to be more accurate is only a matter of very few additional optimization steps (at least with LBFGS-B). As a result of this analysis, we conclude that for the one-dimensional bi-level program using the LBFGS-B algorithm for optimization is the better choice.

Next, we compare two-dimensional grid search with the basic kernel bi-level SVM ("bikernel" in short).

5.2.2 Grid Search vs. Two-dimensional Bi-level SVM

For this experiment, we consider the two-dimensional data set shown in figure 5.3, the synthetic fourclass data set provided by [23] [1].

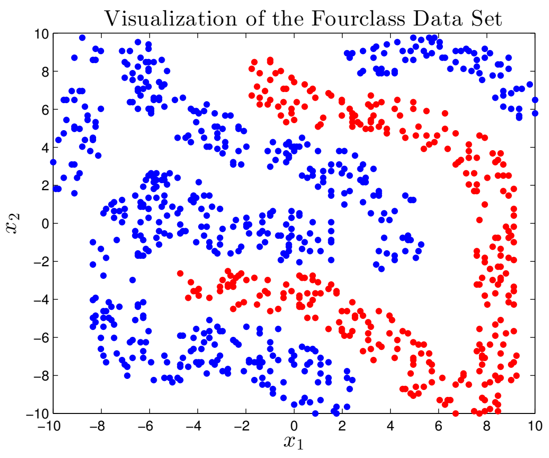


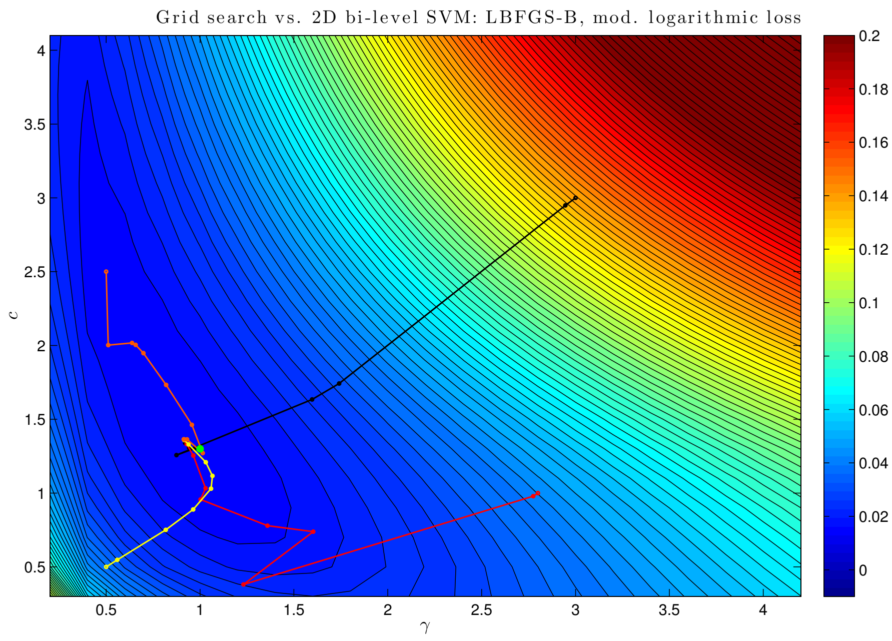
FIGURE 5.3: Visualization of the fourclass data set

The classification task here is to separate the red from the blue class with a non-linear class boundary. The data can be separated without errors, but the optimal parameters of the kernel SVM have to be determined either via grid search or via bi-level optimization.

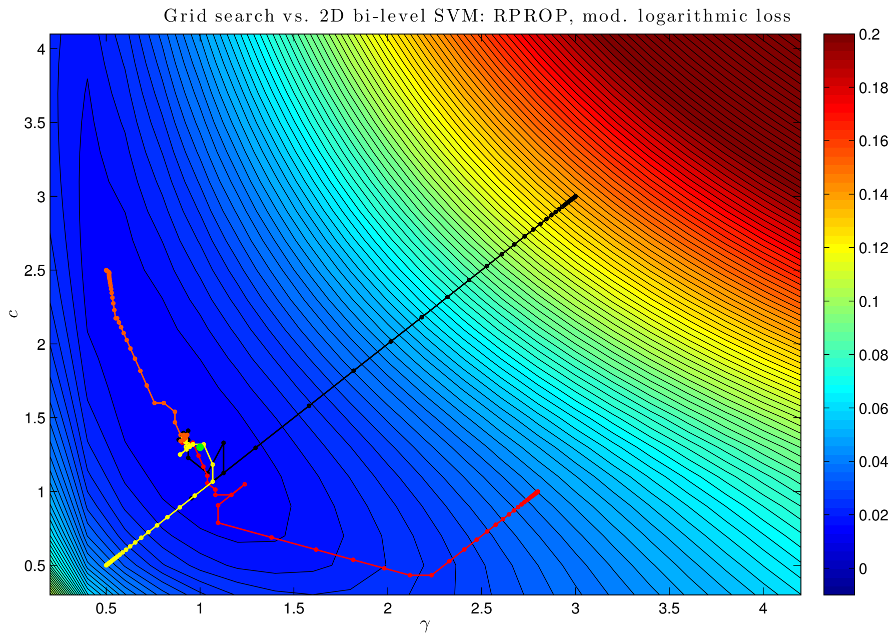
The two-dimensional grid search gives us a nice error surface of which we show the contours in figures 5.4 and 5.5 using the modified logarithmic loss and the quartic approximation respectively. To compare the grid search procedure with the two-dimensional parameter estimation via the bikernel program we chose four different starting values and show the results using either the LBFGS-B algorithm or RPROP for optimization.

For calculating the two-dimensional grid we chose the regularization parameter c from the range of $[0.1, 4.1]$ and a RBF kernel with parameters γ from the range of $[0.2, 4.2]$ with a step size of 0.2. In both figures, we observe a sizeable region that results in good

[1] downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`



(A) Optimization via LBFGS-B, modified logarithmic loss

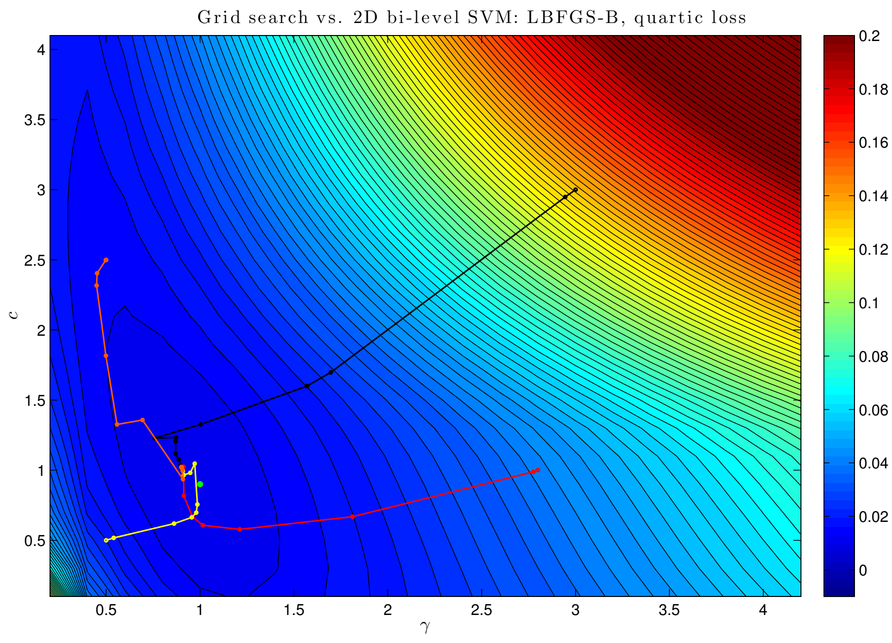


(B) Optimization via RPROP, modified logarithmic loss

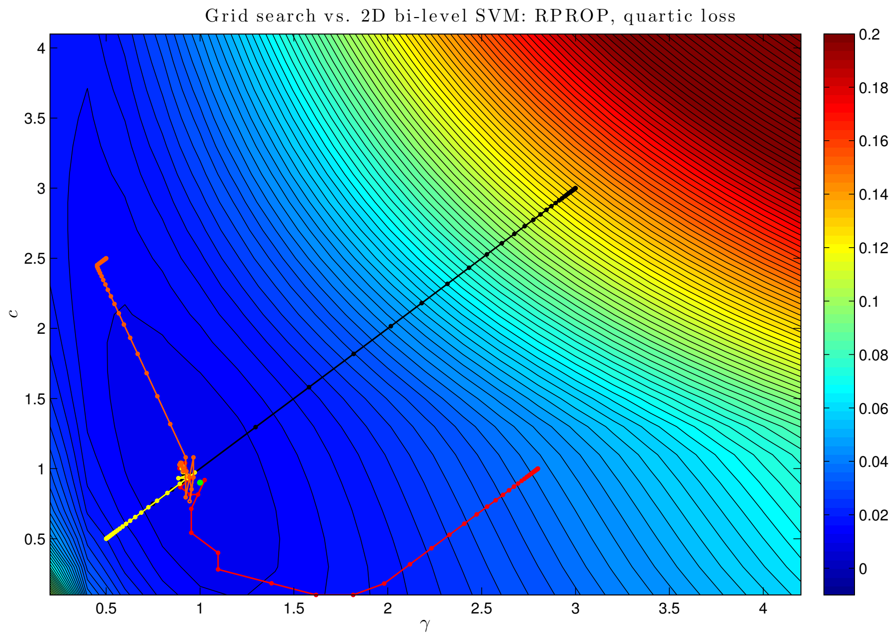
FIGURE 5.4: Comparison 2D grid search vs. 2D bi-level SVM. The loss function approximation used is the modified logarithmic loss. The contour lines show the levels of the higher level objective function $H(.)$. There is a sizeable region that results in good error values (the contour level with the darkest shade of blue shown in the plots). The orange dot marks H_{min} found with grid search.

error values (the contour level with the darkest shade of blue). The minimal value H_{min} is marked with a full orange dot.

Results of the bikernel program were reported for four different initial values, see the corresponding line plots in figures 5.4 and 5.5 as well as the summarized results in



(A) Optimization via LBFGS-B, quartic approximation



(B) Optimization via RPROP, quartic approximation

FIGURE 5.5: Comparison 2D grid search vs. 2D bi-level SVM. The loss function approximation used is the quartic approximation. The contour lines show the levels of the higher level objective function $H(.)$. There is a sizeable region that results in good error values (the contour level with the darkest shade of blue shown in the plots). The orange dot marks H_{min} found with grid search.

table 5.3. The dots show the iteration steps of the optimization process. For RPROP, the number of SVM evaluations corresponds to the iteration steps shown in the plots, but the iteration steps of LBFGS-B do not contain the SVM evaluations resulting from the line search procedure, they are given in table 5.3.

In comparison to the bilinear program, the number of additional evaluations due to line search is more significant, but it does not directly reflect in longer computation times because we use hot start for solving the SVM and for small variations in the parameters there are only few iterations necessary for the SVM to converge. Evaluating the computational cost of one SVM evaluation, see the rows $\frac{\text{Time}}{\text{Eval.}}(s)$ in table 5.3, we see that the the modified logarithmic loss needs a little less time per evaluation than the quartic approximation, but more evaluations. One evaluation by grid search and bi-level optimization consumes approximately the same time, less for the modified logarithmic loss and a little longer for the quartic loss.

We observe that the behaviour of convergence differs between both optimization methods: As discussed in Chapter 4, the LBFGS-B algorithm uses line search, so the initial steps are rather large, and the iterates show no zigzagging. Figures 5.4a and 5.5a show that the first iterates seem to jump close to the valley of the error surface with a big step, and when getting closer to the optimum the step size gets smaller.

Start value (c,γ)		LBFGS-B (Mod. Log.)	RPROP (Mod. Log.)	LBFGS-B (Quartic L.)	RPROP (Quartic L.)
(3,3) black	Eval.	16	60	21	60
	H_{min}	0.01521	0.01521	0.01098	0.01098
	Time (s)	54.55	142.52	84.41	269.10
(1,2.8) red	Eval.	13	61	20	65
	H_{min}	0.01521	0.01521	0.01098	0.01098
	Time (s)	44.79	161.14	87.19	343.57
(0.5,0.5) yellow	Eval.	13	56	19	41
	H_{min}	0.01521	0.01521	0.01098	0.01098
	Time (s)	43.67	164.86	71.35	193.60
(2.5,0.5) cyan	Eval.	14	50	18	53
	H_{min}	0.01521	0.01521	0.01098	0.01098
	Time (s)	43.31	159.46	82.43	259.30
Avg. Bi-level	Eval.	14	57	20	55
	H_{min}	0.01521	0.01521	0.01098	0.01098
	Time (s)	46.58	156.99	81.35	266.62
	$\frac{\text{Time}}{\text{Eval.}}(s)$	3.33	2.77	4.17	4.87
Grid Search	Eval.	441		441	
	H_{min}	0.01528		0.01113	
	Time (s)	1145.88		1503.17	
	$\frac{\text{Time}}{\text{Eval.}}(s)$	2.60		3.41	

TABLE 5.3: Summarized results of 2D grid search vs. bi-level SVM. The table contains results for each start value and optimization algorithm/loss function combination. The number of SVM evaluations, the minimal error H_{min} and the calculation time is shown. Finally the average values for the bi-level approach are given as well as the results from grid search for comparison.

Precision	Grid Search no. of evaluations
10^{-1}	1681
10^{-2}	160801
10^{-3}	16008001

TABLE 5.4: Grid search. Number of SVM evaluations needed to determine the regularization parameter c and the RBF kernel parameter γ up to a given precision.

The RPROP algorithm however, does not use line search and therefore tends to have a zigzagging behaviour close to the minimum. The step size is small at the beginning and increases as long as the direction of the gradient does not change. Close to the minimum, the algorithm needs many iterations to achieve a very high precision because the gradient direction changes often and the step size is quite small. The effect is amplified especially in the given example because the error surface is very flat around the minimum.

We compare computation times and used SVM evaluations in table 5.3 and come to the conclusion, that using the quartic loss - which is very close to Hinge loss - is computationally a little more expensive than using the modified logarithmic loss, but also results in better error values in the end. When a better error value is a priority, then using the quartic approximation is the best choice. When computation time is a priority, then using the modified logarithmic loss with any optimization algorithm is the better choice, or using the quartic approximation together with LBFGS-B as optimizer. In the end, whether to take RPROP or LBFGS-B for optimization does not really change anything in the results, but the LBFGS-B algorithm shows a more stable behaviour close to the minimum.

We compared also the average result from the bi-level learning with grid search which shows us that the bi-level program yields better results for the validation error H_{min}. For a grid with step size 0.2 we need already 441 SVM evaluations - more than 12 times the amount needed on average using bi-level optimization. In table 5.4 we show the number of necessary evaluations to obtain a grid with increasing accuracy (for the same range of values as in this example).

5.2.3 Two-dimensional Bi-level SVM

In this section we provide some figures to illustrate the workings of a kernel SVM with one kernel parameter γ. We used a synthetic two-dimensional data set in the form of two entangled half moons. In figures 5.6 to 5.8 the 3D plots of the RBF kernel on the half moon data set and the contour of the separating hyperplane is shown. The figures nicely illustrate how the data can be separated using a kernel SVM.

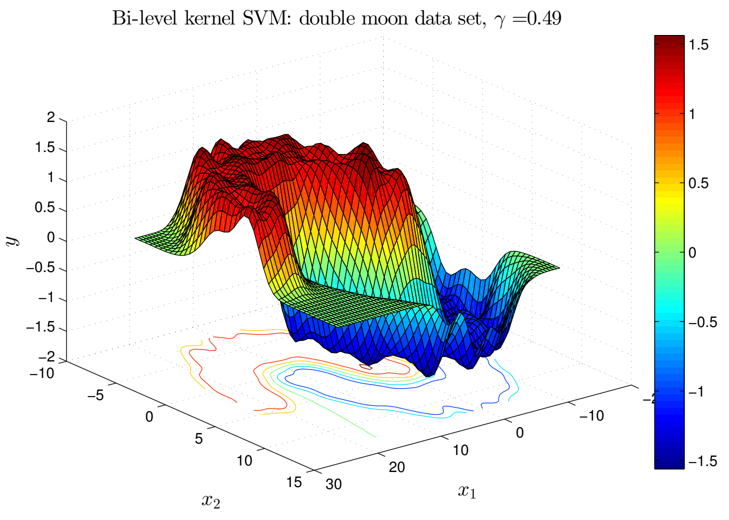


FIGURE 5.6: Bi-level kernel SVM on the half moon data set, $\gamma = 0.49$

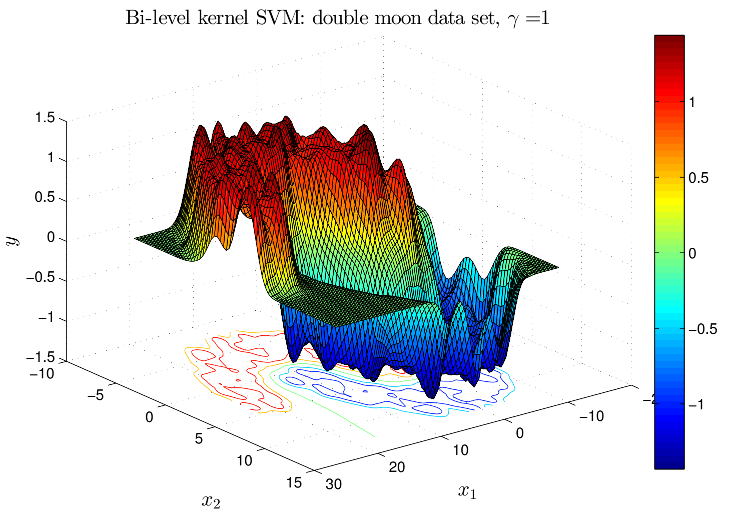


FIGURE 5.7: Bi-level kernel SVM on the half moon data set, $\gamma = 1$

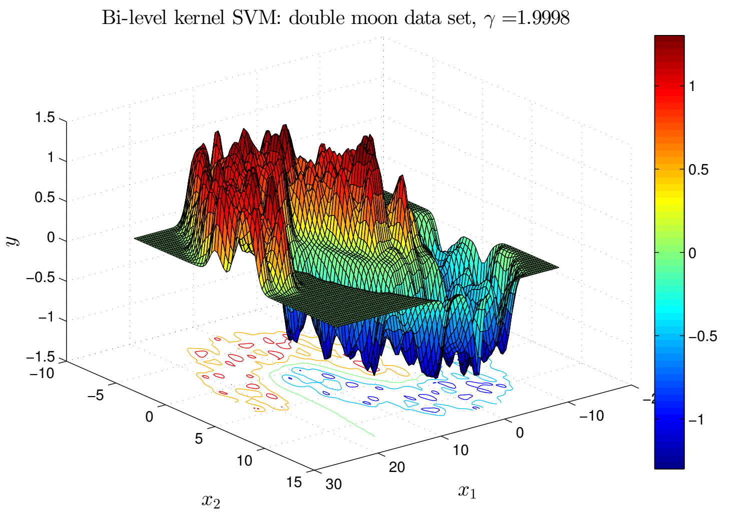


FIGURE 5.8: Bi-level kernel SVM on the half moon data set, $\gamma = 1.9998$

We can observe the influence of the parameter γ on the amount of smoothing of the Gaussian kernel functions sitting on the data points. When the kernel parameters are large, there is little smoothing. The training examples show sharp peaks as in figure 5.8 ($\gamma = 1.9998$). When the kernel parameters are small, the smoothing effect is larger, as in figure 5.6 ($\gamma = 0.49$). Note also that the gap between the classes is larger when the data is less smoothed, the classifier tries to be more accurate in this case. The optimal solution for the kernel parameter lies somewhere in between with $\gamma = 0.6134$, the corresponding optimal regularization parameter is $c = 18.4873$.

In the next section, results for experiments with increased number of parameters are reported.

5.2.4 Seeds Data Set and Multiple γ Bi-level SVM

We argue that grid search gets infeasible for a large number of hyper-parameters to choose. We want to theoretically confront the grid search approach with the bi-level approach when the number of parameters increases.

Experiment setup Because the experiment is just for demonstration purposes, we choose a small data set "Seeds" from the UCI Machine Learning repository [2]. The data set consists of three classes with 70 examples each. The task is about to differentiate three different types of wheat: Kama, Rosa and Canadian, based on 7 geometric measurements of wheat kernels. The features are: area, perimeter, compactness, length and width of the kernel, asymmetry coefficient and the length of kernel groove. Of course, the data is not linearly separable.

To highlight the differences between the bi-level approach and grid search we decided to use a multiple γ bi-level kernel SVM on this task. We use RBF kernels with one parameter γ_d per feature dimension, that makes eight parameters to optimize in total including the regularization parameter c. Not a problem for the bi-level SVM, especially because the data set is small and simple. During the grid search procedure, the SVM is solved for all possible combinations of parameters in a pre-defined range of values. That means n^8 times solving an SVM, with n the number of values per parameter used to build the grid.

In order to get close to the precision of the bi-level SVM, we might e.g. choose a grid of parameter values with step size 0.1. Usually, no prior information about the range of values that yields the optimal classifier performance is available. Also, there is no prior information about how fine-grained the grid needs to be. For that reason, one possible way to deal with the problem is to perform grid search first for a very coarse grid, with values e.g. $c \in \{1, 10, 100, 1000, 10000\}$ and $\gamma_d \in \{1, 1.5, 2, 2.5, 3\}$ and then repeat the procedure for a fine grid with step size 0.1 around the so far best combination of parameters.

We assume to have two runs of grid search, first with e.g. 5 values per parameter as suggested above, and then a second round of 10 more values with step size of 0.1 around the so far best values. Still, the range of values tested is by far not exhaustive and may be far off the optimal values. For demonstration purposes, in figure 5.9 we show exemplary results for parameter values determined by the bi-level program for the current example. We find that solving an SVM with a precision of 10^{-4} for this data set size lasts 0.3421s which is the mean value of the recorded execution time per evaluation. Given these numbers, the grid search procedure would last $(5^8 + 10^8) \cdot 0.3421s = 34343632.81$ seconds or 397.5 days. To wait for such a long time to get to know hopefully good parameters for the SVM is clearly not feasible. Additionally, the problem is a three class problem, therefore we need to learn three 1-rest classifiers which multiplies the necessary computation time by three.

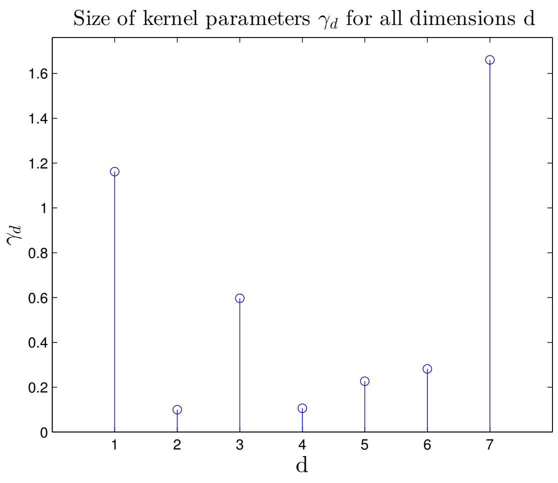


FIGURE 5.9: Seeds data set, resulting magnitude of the γ_d parameters for the 2 vs. rest classifier learnt with the LBFGS-B optimization algorithm.

Results For the experiment, the SVM was solved to a high precision of 10^{-12} or a maximum of 2000 iterations alternatively, as Hinge loss approximation the quartic loss was used, and as solver for the bi-level problem we used LBFGS or RPROP. Cross-validation was applied with $T = 5$ different training and validation sets. Start values were randomly chosen for $c_0 \in [5, 15]$ and $\gamma_{d,0} \in [0.5, 1.5]$ for all three classifiers. See the detailed results in table 5.5 including the minimal validation error returned by the bi-level program (the mean value over all folds of cross-validation), the resulting error

H and classification rates on the training and test set, and the resulting number of SVM evaluations needed for optimization.

LBFGS-B						
Classifier	H_{min}	Train: H	Train: CR	Test: H	Test: CR	Eval.
1 vs. rest	0.1145	0.0154	100%	0.1037	93.33%	441
2 vs. rest	0.0374	0.0144	100%	0.0407	100%	366
3 vs. rest	0.0766	0.0257	100%	0.0703	96.67%	331
RPROP						
Classifier	H_{min}	Train: H	Train: CR	Test: H	Test: CR	Eval.
1 vs. rest	0.1099	0.0208	99.33%	0.1075	93.33%	501
2 vs. rest	0.0383	0.0161	100%	0.0406	100%	501
3 vs. rest	0.0798	0.0251	100%	0.0699	96.67%	501

TABLE 5.5: Seeds data set, multiple γ bi-level SVM: H_{min} (the mean value over all folds of the cross-validated bi-level SVM), resulting H value and correct classification rates (CR) on the training and test set, and number of SVM evaluations needed for all three one vs. rest classifiers, classes are denoted with 1 to 3. Results are reported for optimization via LBFGS-B and RPROP respectively.

We observe that the test classification performance for both choices of optimization algorithms, either LBFGS-B or RPROP, are the same. Also the results for H are similar. However, the number of evaluations needed is significantly lower with the LBFGS-B algorithm than using RPROP. These findings suggest that for increasing numbers of parameters using the LBFGS-B algorithm is probably the better choice. For the further experiments, we will therefore stick to the LBFGS-B algorithm as solver for the bi-level problem.

5.3 Multiple Parameter Experiments

5.3.1 Multiple Parameters Bi-level SVM for Feature Selection

Chapelle *et al.* discuss in [15] an approach to automatically select multiple kernel parameters for SVMs such that the generalization error is improved. They state that the choice of kernel parameters is crucial for the performance of a SVM, and if it is possible to tune several parameters at once, one can use more sophisticated kernels instead of being restricted to very simple ones. Further, they consider the kernel parameters γ_d of the RBF kernel

$$k(x_j, x_i) = \exp(-\sum_{d=1}^{D} \gamma_d (x_{jd} - x_{id})^2)$$

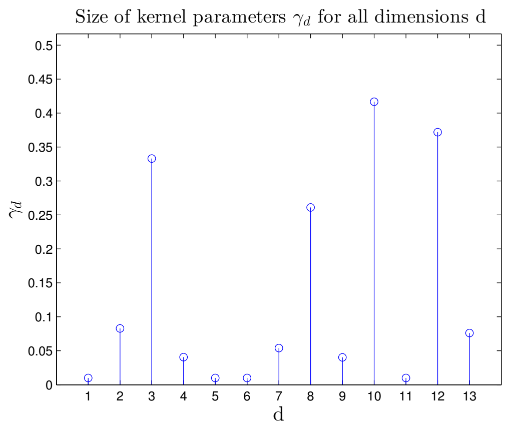
as a means of "scaling factors" for each feature dimension. By learning the weights γ_d of the different features one can determine irrelevant features with near-zero weight. We will show with the following experiments that by determining the irrelevant features we can perform a kind of feature selection. As a consequence, the dimensionality of the problem is reduced without loss of classification performance.

We consider the "Heart Disease" data set from the UCI Machine Learning Repository [2]. The data set contains 270 examples, we use 190 of them for training and 80 for testing. There are 13 discriminative features:

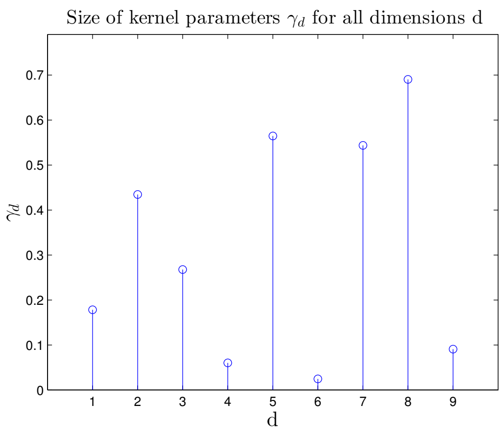
1. age
2. sex
3. chest pain type (4 values)
4. resting blood pressure
5. serum cholesterol in mg/dl
6. fasting blood sugar > 120 mg/dl
7. resting electrocardiographic results (values 0,1,2)
8. maximum heart rate achieved
9. exercise induced angina
10. oldpeak = ST depression induced by exercise relative to rest
11. the slope of the peak exercise ST segment
12. number of major vessels (0-3) coloured by flourosopy
13. thal: 3 = normal; 6 = fixed defect; 7 = reversible defect

as the basis for the decision whether a heart disease is present or not. We use RBF kernels, a precision of 10^{-4} for the lower level SVM, the quartic loss function approximation, and the LBFGS-B algorithm for optimizing the higher level objective. As initial values we choose c_0 randomly from the range $[5, 15]$ and $\gamma_{0,d}$ from the range $[0.5, 1.5]$. Depending on the initial values, we get slightly different results, but by setting a seed for the `randperm` function in Matlab we resolve this problem.

First, we determine the optimal kernel parameters γ_d for all feature dimensions with the multiple γ bi-level kernel SVM. The resulting successful classification rate on the test set is 88.75%. According to the resulting magnitudes of the parameters γ_d depicted in figure 5.10a, we identify all the features having a magnitude of 0.01 (the lower bound on the variables) as irrelevant and remove them (features number 1, 5, 6 and 11). After the features being removed, we run the multiple γ SVM again and achieve a successful classification rate on the test set of 86.25%, only 2.5% lower than before. We can see in table 5.6 that the classification rate on the training set is almost identical with full or reduced feature set. The distribution of the magnitudes of γ_d after feature selection are depicted in figure 5.10b. Interestingly, the age, cholesterol and blood sugar values are dispensable for the decision whether a person has a heart disease or not.



(A) before feature selection



(B) after feature selection

FIGURE 5.10: γ_d before and after feature selection. The resulting successful classification rates are 88.75% and 86.25% respectively.

Further, we examine the results with the same preconditions, but instead of the multiple γ bi-level SVM we use only a single γ parameter (classic bikernel SVM). The classification results on the test set using all features is worse than using several γ parameters, with 86.25% successful classification rate.

SVM version	H_{min}	Train: H	Train: CR	Test: H	Test: CR	Eval.
Multiple γ SVM (before feat. sel.)	0.2450	0.2215	84.74%	0.2276	88.75%	281
Multiple γ SVM (after feat. sel.)	0.2438	0.2131	84.74%	0.2444	86.25%	306
Bikernel SVM	0.2864	0.2492	82.11%	0.2325	86.25%	201
Multiple c SVM	0.2536	0.2442	84.74%	0.2335	87.50%	176
Bilinear SVM	0.2835	0.2605	82.63%	0.2253	85.00%	56

TABLE 5.6: Summary of results for the heart data set using several versions of SVMs

In summary, the findings suggest that the data can be better described using a complex model with multiple kernel parameters. Next, the experiment was repeated using multiple regularization parameters, with one c_d per dimension of the input data. The resulting successful classification rate on the test set was 87.50%. See figure 5.11 for the resulting magnitudes of each parameter c_d including one additional dimension showing the scaling parameter for the bias, which is not regularized according to the model presented in Section 2.4. For experimental reasons we did not manually set the regularization parameter for the bias to zero during the bi-level learning. However, the bi-level

SVM itself set the parameter to zero during the optimization process which is a nice feature.

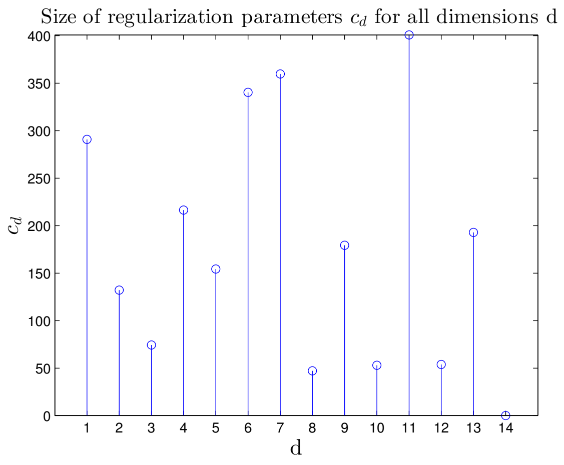


FIGURE 5.11: The magnitudes of the regularization parameters c_d for each dimension. The data point at $d = 14$ shows the value for the non-regularized bias. The resulting successful classification rate is 87.50%.

For comparison, the successful test classification rate for the bilinear SVM is 85%. In summary, the bi-level SVM with multiple γ parameters yields the best classification performance on the heart data set. Furthermore, we observe that in general using bi-level optimization to learn the SVM leads to good generalization, which is reflected in the higher classification rates on the test set compared to the training set. See table 5.6 for the summarized results.

5.3.2 Multiple kernel SVM

In the theoretical part of this thesis (see section 3.2), we have considered the bi-level learning of parameters for a multiple kernel SVM. As already discussed, application scenarios for using several kernels include problems with different feature representations or subsets of features. Here, results are reported on a small scale data set having inhomogeneous features i.e. features consisting of different kinds of measurements. We consider the "Parkinsons" data set, downloaded from the UCI Machine Learning Repository [2] and originating from [32]. The dataset is composed of voice measurements of people having the Parkinson's disease and from healthy people that are used to discriminate between both groups of persons. The data set includes 21 features, consisting of e.g. different measurements of vocal fundamental frequencies, amplitude variations, variations in the fundamental frequencies, ratios of noise and tonal components, non-linear dynamical complexity measures and a signal fractal scaling exponent.

In [32], Little *et al.* report a correct classification rate on the data set of $91.8 \pm 2\%$, using a kernel SVM with a RBF kernel. The features of the Parkinson's data set show feature values in different orders of magnitude and therefore, they apply range scaling as feature pre-processing. Additionally they filter out 7 highly correlated features.

With the multiple kernel SVM we can deal with the different kinds of measurements by combining them into groups having similar orders of magnitude. For the full data set, this results to 7 subgroups of features each of which are dealt with a separate kernel in the multiple kernel setting. For the reduced data set with the features removed as suggested in [32], the remainder of features were equivalently clustered into 7 subgroups of similar range. Other than partitioning the data, no pre-processing was performed.

For bi-level learning, we used following settings: As loss function we used the quartic approximation, the precision of the lower level problem was set to 10^{-12} or a maximal number of 8000 iterations. Initial values for both the kernel parameters γ_p and the kernel scaling factors β_p were set via Matlab's `rand` function.

SVM version	H_{min}	Train: H	Train: CR	Test: H	Test: CR	Eval.
Multiple kernel SVM (full data set)	0.2992	0.0445	100%	0.1898	92.73%	391
Multiple kernel SVM (reduced data set)	0.2611	0.0265	100%	0.1308	92.73%	541
Multiple γ SVM (full data set)	0.4492	0.0029	100%	0.4360	87.27%	251
Bikernel SVM (full data set)	0.4495	0.0029	100%	0.4337	89.09%	46

TABLE 5.7: Summary of results for the full featured and the reduced Parkinson's data set using the multiple kernel SVM, the multiple γ SVM and the bikernel SVM. The table shows H_{min}, errors and classification rates (CR) for the training and test set.

In table 5.7 the results are summarized. We are able to reproduce the results of the paper, removing the features even improves the achieved classification results: The test classification rates are 92.73%. For comparison, we ran the multiple γ SVM and the bikernel SVM on the data set, resulting in 87.27% and 89.09% correct test classification rates respectively. The number of SVM evaluations needed are consistent with the numbers we have seen so far, with the multiple kernel SVM needing more evaluations than the other versions. The training error is exceptionally low and coincides with a 100% training classification rate.

To conclude the chapter, from all the experiments we have discussed do far, the multiple parameter SVMs are consistently outperforming the ones with one or two parameters only. Mostly, these results come at a higher number of total SVM evaluations but the numbers are not significant, especially when comparing the bi-level and the grid search

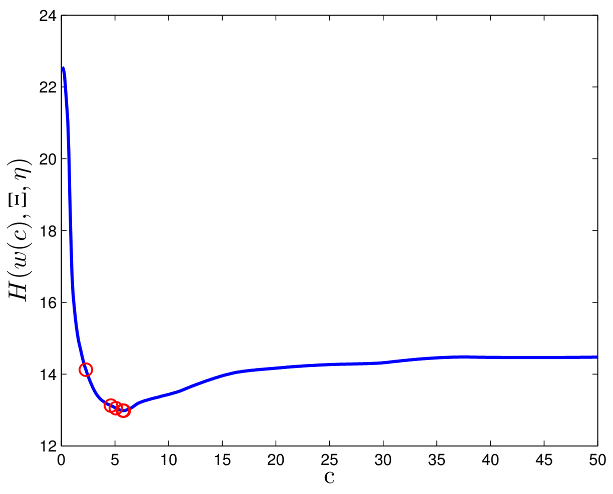


FIGURE 5.12: Linear bi-level SVM vs. grid search on the heart data set with alternative function $H(.)$

approaches. We gain expressive power when using highly parametrized models at a low cost and achieve better classification performances.

5.4 Considering an Alternative Function H

So far, we have assumed to use the mean squared error on the validation data set

$$H(w(c), \Xi, \eta) = \frac{1}{2L} \|\Xi w^T - \eta\|_2^2 \tag{5.1}$$

to judge the performance of the SVM. For the classification task however, we observe in tables 5.5, 5.6 or 5.7 that the resulting H_{min} obtained by bi-level learning is not always consistent with the resulting correct classification rates. When analysing what the MSE does, we find that the MSE calculates the squared distance to the actual class labels (or, otherwise put, to the margins of the classifier). Therefore, examples being farther away of the margins have a higher influence on the resulting error value than examples being closer, and also examples inside the margins even though actually on the correct side may contribute negatively to the MSE. The classification result, however, is the result of an example being on the positive side of the classifier or not, and therefore we came up with the idea to use an alternative function as error function, namely the squared Hinge loss:

$$H(w(c), \Xi, \eta) = \sum_{l=1}^{L} \max\left(0, 1 - \eta_l(\xi_l w^T)\right)^2 . \tag{5.2}$$

This function sets all correct examples outside of the margin to zero error, puts an error between 0 and 1 on the examples between margin and classifier, and a quadratically increasing error on wrong examples.

We run a small experiment using the linear bi-level SVM with the new higher level objective $H(.)$ on the Heart data set previously examined in section 5.3.1 to have something to compare with. We show the result of the linear bi-level SVM and grid search in figure 5.12. The error function is not as smooth as it was using the MSE (compare with figure 5.2), but still the test correct classification rate is with 85.71% almost the same as with the other error function (compare results in table 5.6). These intermediate results suggest that using the squared Hinge loss as an alternative higher level objective would merit further investigation.

Chapter 6

Image Classification Benchmarks

As the chapter title suggests, this chapter provides detailed information about the conducted experiments on image classification tasks. The aim of those experiments is to show the applicability of the bi-level SVM classifiers on medium-scale data sets. We do not expect state of the art results in the chosen image classification benchmarks, because the most potential for improving the performance of an image classification pipeline lies in the careful selection of pre-processing and feature selection steps tailored to the kind of input data, e.g. [54], and this is clearly not the focus of this work.

As a testing framework we used the VLFeat Library [46] which is an open-source toolbox for computer vision algorithms focusing on visual feature extraction and clustering methods and comes handy with a Matlab interface. For the experiments we were inspired by their sample programs for basic recognition. As pre-processing of the input images they use a bag of visual words (BOVW) model (see [29, 49]) as a means of feature extraction: Similar features in images form a ”word”; when extracting all relevant words from a bunch of images, a dictionary of words can be created. When counting the occurrences of specific words in images of the same category, one gets similar histograms of word occurrences, while they can be very different in distinct categories. These histograms are used as features for a classification algorithm such as the Support Vector Machine.

In the VLFeat sample, the pre-processing of the images for classification is based on so-called PHOW features, a variant of dense SIFT features extracted at several scales [33], k-means clustering for constructing the visual vocabulary and building the bag-of-visual-words (BOVW) model, computation of spatial histograms [50] and finally the construction of the explicit χ^2 kernel map such that a linear SVM classifier can be used [48] (see e.g. [52], [31] for references on computing explicit kernel maps). The reason why they use χ^2 kernel is that they show empirically better performance with histogram representations such as bag-of-words models than RBF kernels.

For the concept of explicit kernel maps it is important to recapitulate that kernel SVMs are nothing else than linear SVMs running in a sufficiently high feature space. For any kernel there exists a function $\Psi(x)$ (i.e. feature map) mapping the input data to an infinite dimensional feature space. The clue is that there exists in many cases a finite dimensional feature map $\hat{\Psi}$ which is a sufficiently good approximation of the kernel, i.e. the explicit kernel maps [48]. When we use our bi-level kernel SVM, the calculation of the explicit kernel map is left out.

The approach to apply our bi-level framework on image classification consists basically of the following steps:

(1) execute the pre-processing steps as explained above,

(2) partition the data into T=5 sets for cross-validation,

(3a) learn the optimal hyper-parameter c for the linear SVM with the bi-level approach or

(3b) learn the optimal c and γ for the bi-level kernel SVM

(4) retrain the model on the optimal hyper-parameters,

(5) evaluate the result on a test set.

In the derivations of the variants of the bi-level SVM in chapter 2 and 3 we used only RBF kernels, but we also implemented a χ^2 kernel to be able to compare with the VLFeat examples. However, we use an exponential χ^2 kernel (eq. 6.1), and for computing the feature maps they use an additive χ^2 kernel (eq. 6.2). Note that $x_{j,d}$ is the corresponding value at feature dimension d of example x_j, analogous for $x_{i,d}$.

$$k(x_j, x_i) = \exp\left(-\gamma \sum_{d=1}^{D} \frac{(x_{j,d} - x_{i,d})^2}{x_{j,d} + x_{i,d}}\right) \tag{6.1}$$

$$k(x_j, x_i) = \sum_{d=1}^{D} \frac{2x_{j,d}x_{i,d}}{x_{j,d} + x_{i,d}} \tag{6.2}$$

In the following section we discuss the experiments and results. We did experiments on classic image classification benchmarks such as Caltech101 [21] and the Graz02 database [34]. Finally we discuss several results on the famous MNIST handwritten digits database [30] including experiments applying the BOVW model and on raw data.

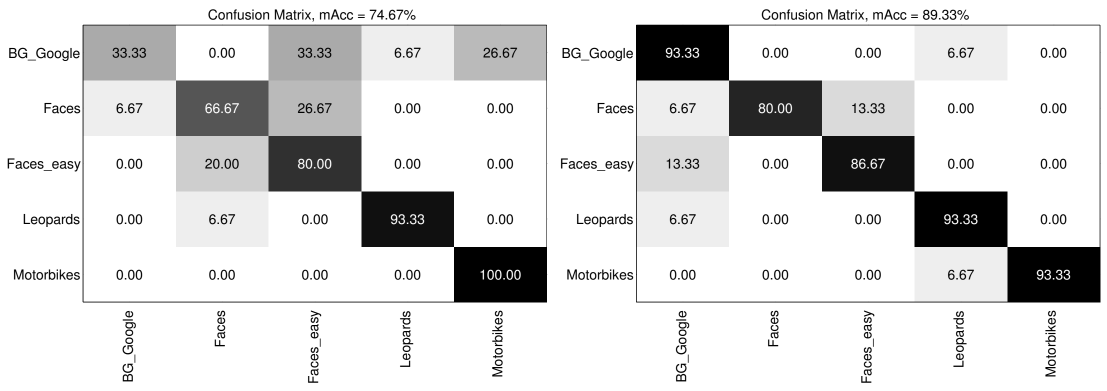
Caltech101 is a very famous image classification benchmark dataset and contains images of 101 object categories including one background category. Object classes are e.g. motorbike, cougar, ant, faces, piano etc.

The Graz02 database contains images of three categories (bike, car, person) and one background category (none). The natural images show high variability in each category, and a lot of clutter in the background.

6.1 Experiments with Caltech101

For the Caltech experiments, we used only a subset of the categories for bi-level learning. We perform experiments with the bikernel SVM and use a subset of 10 classes. A bottleneck of the bi-level kernel SVM are the memory requirements due to the kernel matrix size of $N \times N$ with N the number of training examples. For the image classification experiments the number of training examples scales with the number of classes because we learn a 1 vs. rest classifier for each class.

Caltech101 - Bikernel In this part, we did not calculate the explicit kernel map, but use the bi-level kernel SVM on the BOVW histogram directly. In figure 6.1 we compare the confusion matrices of the bi-level kernel SVM using either a RBF kernel or an exponential χ^2 kernel on a small subset of the Caltech data. The diagonal entries in the confusion matrix show the percentage of true positives over the number of the total true and false classified test examples for each class. The entries outside of the diagonal show the percentage of the false positives over the total number of test examples. We observe that the χ^2 outperforms the RBF kernel, as suggested by [52]. For the following experiments we use therefore the χ^2 kernel because it performs better than the RBF kernel on histograms/bag of word models.



(A) Confusion matrix, RBF kernel (B) Confusion matrix, χ^2 kernel

FIGURE 6.1: Comparison of classification results on Caltech5, RBF and χ^2 kernel

In the following we report results on the Caltech data set using 10 classes. We used 30 training/validation examples and 15 test examples, the vocabulary size for the BOVW model was set to 400. We compare the bikernel SVM with a grid search on the kernel SVM with $c = \{0.5, 1, 1.5, 2\}$ and $\gamma = \{0.5, 1, 1.5, 2\}$. The values used for grid search were chosen after inspecting the range of results of the bikernel SVM. After performing grid search, we picked the best performing hyper-parameter combination and reported the corresponding results.

In figures 6.2a and 6.2b the corresponding confusion matrices are shown. We observe that all classes but one perform better using the bikernel SVM. The concerned class "anchor" has particularly few available examples, only 42 in total. This leads to a reduced training/validation set on this class which may affect the performance of the bi-level approach.

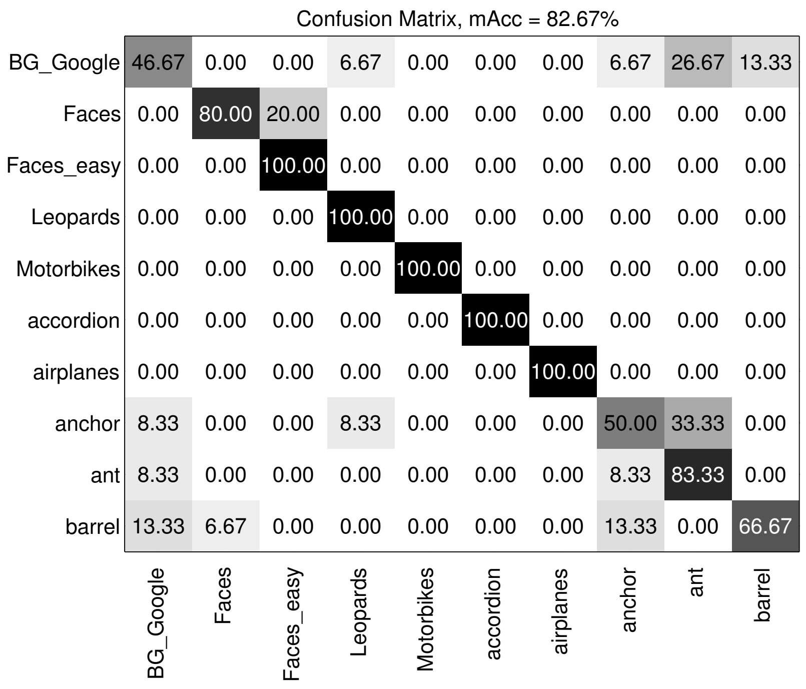
In general, it is important to take care that the training set is big enough when splitting it to a training and a validation set such that the validation error can be taken as a representative measure for bi-level learning.

6.2 Experiments with Graz02

In this section, results on the Graz02 data sets are reported. The pre-processing is done with the VLFeat Library as previously described. In both the bi-level linear and kernel SVM the number of training/validation images was set to 60, and the number of test examples was set to 30 for each class. The number of words for the BOVW model was set to 300.

Graz02 - Bilinear For learning the Graz02 data set with the linear SVM, the explicit kernel maps were computed. In the following, we compare the results for the bilinear SVM and for grid search using the linear SVM using the explicit χ^2 feature map. The precision of the SVM was set to 10^{-4} and the initial value for the regularization parameter was $c_0 = 10$. We compare the bilinear SVM to a very coarse grid search on values for $c \in \{1, 10, 100, 1000\}$ of which the best results were picked.

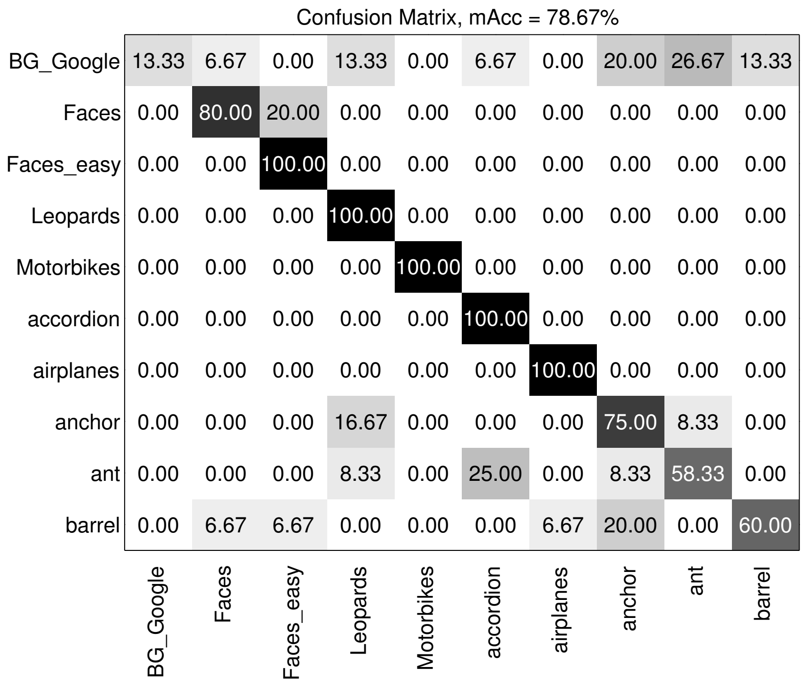
To compare the results we show the confusion matrices in figure 6.3. We observe that the bi-level SVM outperforms the linear SVM. The overall performance is increased by about 1%, the results for all categories are better or equal than grid search using the bi-level version of the SVM. Note that the category "none" is no real category in this sense, but rather a general background category (compare with Caltech's category "BG_Google"). It may contain all kinds of different backgrounds, so usually these categories cannot be clearly classified (cf. figure 6.2).



Confusion Matrix, mAcc = 82.67%

	BG_Google	Faces	Faces_easy	Leopards	Motorbikes	accordion	airplanes	anchor	ant	barrel
BG_Google	46.67	0.00	0.00	6.67	0.00	0.00	0.00	6.67	26.67	13.33
Faces	0.00	80.00	20.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Faces_easy	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Leopards	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00	0.00
Motorbikes	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00
accordion	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00
airplanes	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00
anchor	8.33	0.00	0.00	8.33	0.00	0.00	0.00	50.00	33.33	0.00
ant	8.33	0.00	0.00	0.00	0.00	0.00	0.00	8.33	83.33	0.00
barrel	13.33	6.67	0.00	0.00	0.00	0.00	0.00	13.33	0.00	66.67

(A) Confusion matrix, bi-level optimization with LBFGS-B



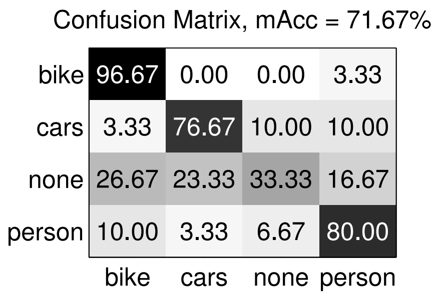
Confusion Matrix, mAcc = 78.67%

	BG_Google	Faces	Faces_easy	Leopards	Motorbikes	accordion	airplanes	anchor	ant	barrel
BG_Google	13.33	6.67	0.00	13.33	0.00	6.67	0.00	20.00	26.67	13.33
Faces	0.00	80.00	20.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Faces_easy	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Leopards	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00	0.00
Motorbikes	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00	0.00
accordion	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00	0.00
airplanes	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00	0.00	0.00
anchor	0.00	0.00	0.00	16.67	0.00	0.00	0.00	75.00	8.33	0.00
ant	0.00	0.00	0.00	8.33	0.00	25.00	0.00	8.33	58.33	0.00
barrel	0.00	6.67	6.67	0.00	0.00	0.00	6.67	20.00	0.00	60.00

(B) Confusion matrix, grid search using χ^2 kernel SVM

FIGURE 6.2: Comparison of classification results on Caltech10, using the LBFGS-B algorithm and grid search

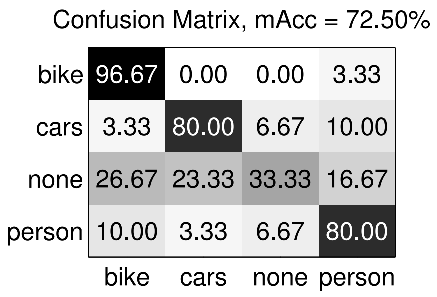
Graz02 - Bikernel In the following we compare the results for the bikernel SVM and a simple grid search on the kernel SVM using a χ^2 kernel, and show the results using the LBFGS-B algorithm. The calculation of the explicit feature map is left out, the other pre-processing steps are the same as before. The initial values for regularization and



Confusion Matrix, mAcc = 71.67%

	bike	cars	none	person
bike	96.67	0.00	0.00	3.33
cars	3.33	76.67	10.00	10.00
none	26.67	23.33	33.33	16.67
person	10.00	3.33	6.67	80.00

(A) Confusion matrix, result of linear grid search



Confusion Matrix, mAcc = 72.50%

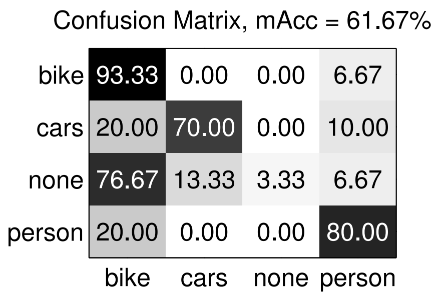
	bike	cars	none	person
bike	96.67	0.00	0.00	3.33
cars	3.33	80.00	6.67	10.00
none	26.67	23.33	33.33	16.67
person	10.00	3.33	6.67	80.00

(B) Confusion matrix, result of linear bi-level SVM (LBFGS-B)

FIGURE 6.3: Comparison of the linear and the bilinear SVM on the Graz02 data set

kernel parameters were $c = 0.5$ and $\gamma = 1$. Parameter values for grid search were chosen $c = \{1, 10, 100, 1000\}$ and $\gamma = \{0.5, 1, 1.5, 2\}$.

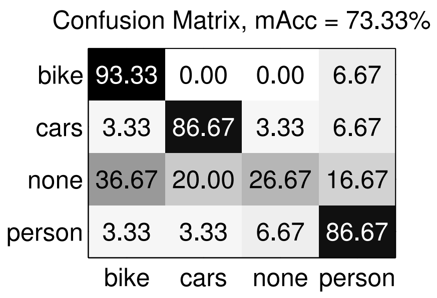
The results on all the categories are equal or better than the best result obtained with grid search. This comparison shows, that grid search on two parameters gets already a bit tricky, the difference in performance is bigger with the kernel SVM than with the linear SVM. After all, you can still make a good guess for a single parameter and obtain satisfactory results. The bikernel SVM also outperforms the bilinear SVM in the categories "cars" and "person". Note that by using the explicit kernel maps the difference in performance between linear and kernel SVM shouldn't be significant as argued in [48].



Confusion Matrix, mAcc = 61.67%

	bike	cars	none	person
bike	93.33	0.00	0.00	6.67
cars	20.00	70.00	0.00	10.00
none	76.67	13.33	3.33	6.67
person	20.00	0.00	0.00	80.00

(A) Confusion matrix, result of grid search using the kernel SVM



Confusion Matrix, mAcc = 73.33%

	bike	cars	none	person
bike	93.33	0.00	0.00	6.67
cars	3.33	86.67	3.33	6.67
none	36.67	20.00	26.67	16.67
person	3.33	3.33	6.67	86.67

(B) Confusion matrix, result of bikernel SVM (LBFGS-B)

FIGURE 6.4: Comparison of classification results on the Graz02 data, using the LBFGS-B algorithm and grid search

6.3 Experiments with MNIST

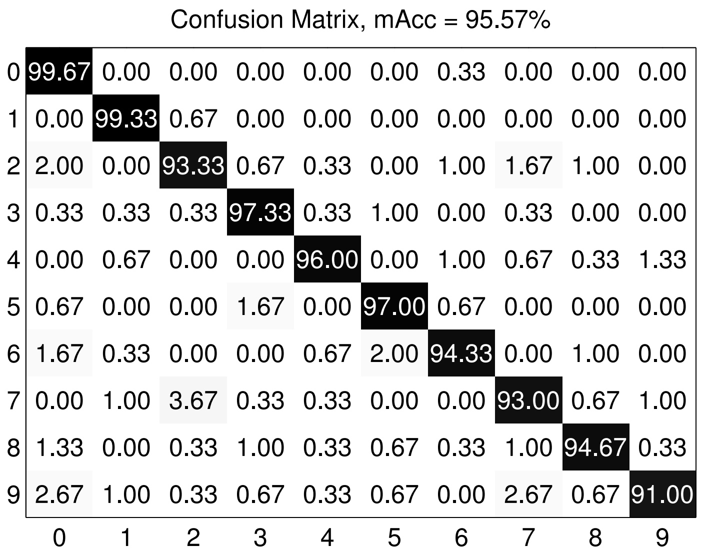
In this section we experimented with the probably most used classification benchmark data set - the MNIST handwritten digits database. First, we perform experiments on the raw data, and then we apply the BOVW model as pre-processing for the MNIST data set. One training example of MNIST is an image with 28×28 pixels which makes 784 feature dimensions. The aim is to classify the handwritten digits into categories from ”0” to ”9”.

All the experiments were conducted using the bikernel SVM with a χ^2 kernel. We used 300 training/validation examples as well as 300 test examples per class. The precision of the SVM was set to 10^{-4}, as solver we used the LBFGS-B algorithm. As initial values we used $c = 10$ and $\gamma = 1$. The raw data was used as-is. For the bag of visual words modelled data, we considered the handwritten digits as images and applied the previously described feature transformation to the data set. Before extracting the PHOW features we blew up the images to 480×480 pixels (we found empirically that this size yielded good results). The corresponding classification results are shown in form of confusion matrices in figure 6.5.

Both variants show a really good classification performance, with the BOW features outperforming the raw data which is not surprising. The results are quite satisfactory, considered that we did not use the full number of available examples of the MNIST data set. Using more training examples should improve the results further.

The results suggest that it might be promising to combine both feature sets using a multiple kernel SVM.

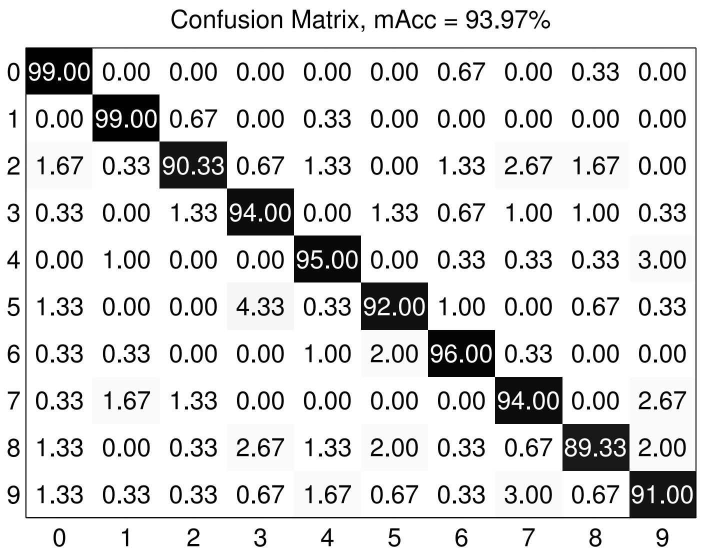
In the final Chapter we draw some conclusions and give an outlook to possible future work.



Confusion Matrix, mAcc = 95.57%

	0	1	2	3	4	5	6	7	8	9
0	99.67	0.00	0.00	0.00	0.00	0.00	0.33	0.00	0.00	0.00
1	0.00	99.33	0.67	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	2.00	0.00	93.33	0.67	0.33	0.00	1.00	1.67	1.00	0.00
3	0.33	0.33	0.33	97.33	0.33	1.00	0.00	0.33	0.00	0.00
4	0.00	0.67	0.00	0.00	96.00	0.00	1.00	0.67	0.33	1.33
5	0.67	0.00	0.00	1.67	0.00	97.00	0.67	0.00	0.00	0.00
6	1.67	0.33	0.00	0.00	0.67	2.00	94.33	0.00	1.00	0.00
7	0.00	1.00	3.67	0.33	0.33	0.00	0.00	93.00	0.67	1.00
8	1.33	0.00	0.33	1.00	0.33	0.67	0.33	1.00	94.67	0.33
9	2.67	1.00	0.33	0.67	0.33	0.67	0.00	2.67	0.67	91.00

(A) Confusion matrix, result of bikernel SVM (LBFGS-B) on BOVW modeled data



Confusion Matrix, mAcc = 93.97%

	0	1	2	3	4	5	6	7	8	9
0	99.00	0.00	0.00	0.00	0.00	0.00	0.67	0.00	0.33	0.00
1	0.00	99.00	0.67	0.00	0.33	0.00	0.00	0.00	0.00	0.00
2	1.67	0.33	90.33	0.67	1.33	0.00	1.33	2.67	1.67	0.00
3	0.33	0.00	1.33	94.00	0.00	1.33	0.67	1.00	1.00	0.33
4	0.00	1.00	0.00	0.00	95.00	0.00	0.33	0.33	0.33	3.00
5	1.33	0.00	0.00	4.33	0.33	92.00	1.00	0.00	0.67	0.33
6	0.33	0.33	0.00	0.00	1.00	2.00	96.00	0.33	0.00	0.00
7	0.33	1.67	1.33	0.00	0.00	0.00	0.00	94.00	0.00	2.67
8	1.33	0.00	0.33	2.67	1.33	2.00	0.33	0.67	89.33	2.00
9	1.33	0.33	0.33	0.67	1.67	0.67	0.33	3.00	0.67	91.00

(B) Confusion matrix, result of bikernel SVM (LBFGS-B) on raw data

FIGURE 6.5: Comparison of the bikernel SVM on two different feature sets of the MNIST data

Chapter 7

Conclusion and Outlook

In conclusion, this thesis shows the applicability of the bi-level approach on a particular machine learning algorithm, namely the Support Vector Machine. For our approach to work, we require that the energy function of the SVM (the lower level problem) as well as the validation error function (higher level problem) are differentiable which leads us to the presented approximations of the loss functions used for the lower level problem.

With the approach presented in this thesis, we are able to optimize as many parameters as there are dimensions in the training data and have a clear advantage over grid search methods when it comes to model selection. The benefit is greater when the model is complex and requires multiple parameters for a good description of the problem and/or if the classification performance is sensitive to the parameters.

In some cases, as we have seen in figures 5.4 and 5.5, there can be numerous reasonable parameter choices that yield a comparable classification performance up to a precision of 10^{-2}. Even the optimization algorithm gives us different answers depending on the initial values (due to numerical inaccuracies and the fact that we solve the inner problem only up to a certain precision), but we can still be certain that the parameters that our bi-level program suggests yield good results. However, it has to be taken care that there is enough training data such that when parts of the training data are used for validation purposes, the validation error is a reasonable estimate of the performance of the SVM.

Additionally, we are able to gain much insight in the model and also the data. When using a multiple parameter bi-level SVM, either a linear or a kernel version, we can draw conclusions about the importance of each feature dimension on the resulting classifier. In our model, multiple parameters are applied dimension-wise, so we can find out which dimensions contribute more to a meaningful representation of the data. This can be seen as weighting of the feature dimensions: The less meaningful dimensions will get smaller parameter values. When it turns out that some features have a near-zero weight we can skip the features as a means of performing feature selection. We can still get a good classification result and at the same time benefit from a reduced complexity of the model. On the other hand, if we observe that all parameters stay alike after optimization, perhaps the data does not need so many degrees of freedom and we can stick to a simpler model using e.g. only one kernel/regularization parameter.

Naturally, training a simple SVM classifier with fixed hyper-parameters is faster than solving a bi-level SVM. However, when working with standard classifiers, often much time is lost in finding ”good” parameters. Additionally, there are seldom restrictions on the training time of a machine learning algorithm, and classification using the trained classifier is fast in both variants.

Attention has to be paid on choosing the bounds on the variables, especially for the kernel SVM. Depending on the data and the γ parameter, the kernel value $k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$ makes sense or not. For this reason, we do not want to set the lower bounds for the parameter optimization too low, instead we set them for both LBFGS-B and RPROP algorithms to 0.01 or 0.1.

In the current Matlab implementation we face limitations of data set size and number of features respectively due to the restrictions of memory and maximal allowed matrix sizes. E.g., for the image classification datasets, the feature transformations reduce the number of feature dimensions of the natural images drastically, but still there may be 4000+ feature dimensions, and additionally the data set size can be very large. To tackle these problems, an implementation in another programming language or even a GPU implementation can be a remedy.

An additional research question can be to examine the influence of the size of the validation set on the final performance, and to investigate eventual over-fitting effects if the validation set is too small, even tough cross-validation is used. This is an important consideration since in bi-level optimization the error on the validation set is the objective function to be minimized. In this context, also the study of alternatives for the higher level objective function can be interesting.

Further investigations can be made in exploring methods to solve the inner problem: The VLFeat Library [46] offers two implementations of large-scale linear SVM solvers using stochastic gradient descent and stochastic dual coordinate ascent which show a very good performance. Stochastic gradient minimizes the expected loss on the training set, which has in general a very good performance, see e.g. [53, 11]. For sparse data, other variants of coordinate descent methods could also be considered to solve the SVM, see e.g. [24].

Appendix A

Additional Derivations for the Bi-level Kernel SVM

In the following we show some additional derivations for the gradients that are necessary for solving the bi-level kernel SVM. First, we show the derivations for the gradients given in equations 3.6 to 3.9.

Derivation $\frac{\partial H}{\partial \alpha}$, see equation 3.6

$$H = \frac{1}{2L}\langle \mathcal{K}\alpha - \eta, \mathcal{K}\alpha - \eta \rangle$$

$$dH = \frac{1}{2L}(\langle \mathcal{K}d\alpha, \mathcal{K}\alpha - \eta \rangle + \langle \mathcal{K}\alpha - \eta, \mathcal{K}d\alpha \rangle)$$

$$dH = \frac{1}{L}\langle \mathcal{K}d\alpha, \mathcal{K}\alpha - \eta \rangle = \frac{1}{L}(\mathcal{K}d\alpha)^T(\mathcal{K}\alpha - \eta) = \frac{1}{L}d\alpha^T\mathcal{K}^T(\mathcal{K}\alpha - \eta) = \frac{1}{L}\langle d\alpha, \mathcal{K}^T(\mathcal{K}\alpha - \eta)\rangle$$

$$\frac{\partial H}{\partial \alpha} = \frac{1}{L}\mathcal{K}^T(\mathcal{K}\alpha - \eta)$$

Derivation $\frac{\partial E}{\partial \alpha}$, see equation 3.9

$$E = \underbrace{\frac{c}{2}\alpha^T K\alpha}_{e_1} + \underbrace{\sum_{j=1}^{N} \ell(k_j\alpha, y_j)}_{e_2}$$

$$de_1 = \frac{c}{2}(d\alpha^T K\alpha + \alpha^T K d\alpha) =$$

$$= \frac{c}{2}(\langle d\alpha, K\alpha\rangle + \langle K^T\alpha, d\alpha\rangle) =$$

$$= \frac{c}{2}\langle d\alpha, K\alpha + K^T\alpha\rangle \overset{K=K^T}{=}$$

$$= \frac{c}{2}\langle d\alpha, 2K\alpha\rangle$$

$$\frac{\partial e_1}{\partial \alpha} = cK\alpha$$

$$e_2 = \sum_{j=1}^{N} \ell(k_j\alpha, y_j)$$

$$t_j \coloneqq k_j\alpha$$

$$e_2 = \sum_{j=1}^{N} \ell(t_j, y_j)$$

$$de_2 = \sum_{j=1}^{N} d\ell(t_j, y_j) = \sum_{j=1}^{N} \ell'(t_j, y_j)dt_j$$

$$dt_j = k_j d\alpha$$

$$de_2 = \sum_{j=1}^{N} \ell'(t_j, y_j)k_j d\alpha = \sum_{j=1}^{N} \langle \ell'(t_j, y_j)k_j^T, d\alpha\rangle =$$

$$de_2 = \langle \underbrace{\sum_{j=1}^{N} \ell'(t_j, y_j)k_j^T}_{\left(k_1^T k_2^T \ldots k_N^T\right)\underbrace{\begin{pmatrix}\ell'(t_1, y_1)\\ \vdots \\ \ell'(t_N, y_N)\end{pmatrix}}_{\ell'(t,y)}}, d\alpha\rangle = \langle K^T\ell'(t, y), d\alpha\rangle$$

$$\frac{\partial e_2}{\partial \alpha} = K^T\ell'(t, y) = K\ell'(t, y)$$

$$\frac{\partial E}{\partial \alpha} = cK\alpha + K\ell'(t, y)$$

Recap of logarithmic loss function Note that for simplicity, in our derivations we assume a logarithmic loss function. However, the results are applicable to all of the loss functions explained in section 2.2. Let us shortly recapitulate $\ell(t_j, y_j)$, assuming $t_j \coloneqq k_j\alpha$:

$$\ell(t_j, y_j) = \log(1 + e^{-t_j y_j})$$

$$\ell'(t_j, y_j) = \frac{-y_j}{e^{t_j y_j} + 1}$$

$$\ell''(t_j, y_j) = \frac{y_j^2 e^{t_j y_j}}{(e^{t_j y_j} + 1)^2} \overset{y_j^2=1}{=} \frac{e^{t_j y_j}}{(e^{t_j y_j} + 1)^2}$$

Derivation $\frac{\partial^2 E}{\partial \alpha^2}$, see equation 3.7

$$\frac{\partial E}{\partial \alpha} = \underbrace{cK\alpha}_{e_1} + \underbrace{K\ell'(t, y)}_{e_2}$$

$$de_1 = cKd\alpha$$

$$\frac{\partial e_1}{\partial \alpha} = cK^T = cK$$

$$de_2 = Kd\ell'(t, y)$$

Element wise view:

$$d\ell'(t_j, y_j) = \ell''(t_j, y_j)dt_j \overset{t_j=k_j\alpha}{=} \ell''(k_j\alpha, y_j)k_j d\alpha$$

$$de_2 = K \begin{pmatrix} \ell''(k_1\alpha, y_1)k_1 d\alpha \\ \ell''(k_2\alpha, y_2)k_2 d\alpha \\ \vdots \\ \ell''(k_N\alpha, y_N)k_N d\alpha \end{pmatrix} =$$

$$= K \begin{pmatrix} \ell''(k_1\alpha, y_1) & 0 & \dots & 0 \\ 0 & \ell''(k_2\alpha, y_2) & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \ell''(k_N\alpha, y_N) \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \\ \vdots \\ k_N \end{pmatrix} d\alpha =$$

$$= K \operatorname{diag}(\ell''(t, y))Kd\alpha$$

$$\frac{\partial e_2}{\partial \alpha} = (K \operatorname{diag}(\ell''(t, y))K)^T = K \operatorname{diag}(\ell''(t, y))K$$

$$\frac{\partial^2 E}{\partial \alpha^2} = cK + K \operatorname{diag}(\ell''(t, y))K$$

Derivation $\frac{\partial^2 E}{\partial \alpha \partial c}$, see equation 3.8

$$\frac{\partial E}{\partial \alpha} = \underbrace{cK\alpha + K\ell'(t, y)}_{e}$$

$$de = K\alpha dc$$

$$\frac{\partial e}{\partial c} = (K\alpha)^T = \alpha^T K$$

$$\frac{\partial^2 E}{\partial \alpha \partial c} = \alpha^T K$$

Derivation $\frac{\partial H}{\partial \gamma}$, see equation 3.12

$$H = \frac{1}{2L}\|\mathcal{K}\alpha - \eta\|_2^2 = \frac{1}{2L}\langle \mathcal{K}\alpha - \eta, \mathcal{K}\alpha - \eta\rangle$$

$$dH = \frac{1}{2L}(\langle d\mathcal{K}\alpha, \mathcal{K}\alpha - \eta\rangle + \langle \mathcal{K}\alpha - \eta, d\mathcal{K}\alpha\rangle)$$

$$dH = \frac{1}{L}\langle d\mathcal{K}\alpha, \mathcal{K}\alpha - \eta\rangle$$

$$d\mathcal{K} = \begin{pmatrix} \exp(-\gamma\|\xi_1 - x_1\|_2^2)(-\|\xi_1 - x_1\|_2^2)d\gamma & \exp(-\gamma\|\xi_1 - x_2\|_2^2)(-\|\xi_1 - x_2\|_2^2)d\gamma & \dots \\ \vdots & \ddots & \vdots \\ \exp(-\gamma\|\xi_L - x_1\|_2^2)(-\|\xi_L - x_1\|_2^2)d\gamma & \dots & \end{pmatrix}$$

$$\mathcal{K} = \begin{pmatrix} \exp(-\gamma\|\xi_1 - x_1\|_2^2) & \exp(-\gamma\|\xi_1 - x_2\|_2^2) & \dots \\ \vdots & \ddots & \vdots \\ \exp(-\gamma\|\xi_L - x_1\|_2^2) & \dots & \end{pmatrix}$$

$$\mathcal{K}_{in} = \begin{pmatrix} -\|\xi_1 - x_1\|_2^2 & -\|\xi_1 - x_2\|_2^2 & \dots \\ \vdots & \ddots & \vdots \\ -\|\xi_L - x_1\|_2^2 & \dots & \end{pmatrix}$$

$$d\mathcal{K} = (\mathcal{K} \odot \mathcal{K}_{in})d\gamma \overset{\mathcal{K}\odot\mathcal{K}_{in}=\bar{\mathcal{K}}}{=} \bar{\mathcal{K}}d\gamma$$

$$dH = \frac{1}{L}\langle \bar{\mathcal{K}}\alpha d\gamma, \mathcal{K}\alpha - \eta\rangle \overset{d\gamma\ldots\text{scalar}}{=} \frac{1}{L}\langle \bar{\mathcal{K}}\alpha, \mathcal{K}\alpha - \eta\rangle d\gamma$$

$$\frac{\partial H}{\partial \gamma} = \frac{1}{L}(\bar{\mathcal{K}}\alpha)^T(\mathcal{K}\alpha - \eta) = \frac{1}{L}\alpha^T\bar{\mathcal{K}}^T(\mathcal{K}\alpha - \eta)$$

Derivation $\frac{\partial^2 E}{\partial \alpha \partial \gamma}$, see equation 3.13

$$\frac{\partial E}{\partial \alpha} = \underbrace{cK\alpha}_{e_1} + \underbrace{K\ell'(t,y)}_{e_2}$$

$$de_1 = cdK\alpha$$

$$dK = (K \odot K_{in})d\gamma = \bar{K}d\gamma \text{ (compare with derivations of } d\mathcal{K})$$

$$de_1 = c\bar{K}\alpha d\gamma$$

$$\frac{\partial e_1}{\partial \gamma} = (c\bar{K}\alpha)^T = c\alpha^T\bar{K}$$

$$e_2 = \underbrace{dK}_{\checkmark}\,\ell'(t,y) + K\,\underbrace{d\ell'(t,y)}_{?}$$

$$d\ell'(t,y) = \ell''(t,y)dt$$

$$dt = dK\alpha$$

$$d\ell'(t,y) = \begin{pmatrix} \ell''(t_1,y_1)dk_1\alpha \\ \ell''(t_2,y_2)dk_2\alpha \\ \vdots \\ \ell''(t_N,y_N)dk_N\alpha \end{pmatrix} = \operatorname{diag}(\ell''(t,y)) \begin{pmatrix} dk_1 \\ dk_2 \\ \vdots \\ dk_N \end{pmatrix} \alpha = \operatorname{diag}(\ell''(t,y))dK\alpha$$

$$d\ell'(t,y) = \operatorname{diag}(\ell''(t,y))\bar{K}\alpha d\gamma$$

$$ge_2 = \bar{K}\ell'(t,y)d\gamma + K\operatorname{diag}(\ell''(t,y))\bar{K}\alpha d\gamma =$$

$$= (\bar{K}\ell'(t,y) + K\operatorname{diag}(\ell''(t,y))\bar{K}\alpha)d\gamma$$

$$\frac{\partial g_2}{\partial \gamma} = (\bar{K}\ell'(t,y) + K\operatorname{diag}(\ell''(t,y))\bar{K}\alpha)^T$$

$$\frac{\partial^2 E}{\partial \alpha \partial \gamma} = (c\bar{K}\alpha + \bar{K}\ell'(t,y) + K\operatorname{diag}(\ell''(t,y))\bar{K}\alpha)^T$$

Appendix B

Additional Derivations for the Bi-level Multiple Kernel SVM

In the following we show some additional derivations for the gradients that are necessary for solving the bi-level multiple kernel SVM. We show the derivations for the gradients given in equations 3.16 to 3.19.

Derivation $\frac{\partial H}{\partial \gamma_p}$, see equation 3.16

$$dH = \frac{1}{L}\langle d\mathcal{K}_\beta \alpha, \mathcal{K}_\beta \alpha - \eta \rangle$$

$$d\mathcal{K}_\beta = \begin{pmatrix} \beta_p \exp(-\gamma_p \|\xi_1 - x_1\|_2^2)(-\|\xi_1 - x_1\|_2^2) d\gamma_p & \beta_p \exp(-\gamma_p \|\xi_1 - x_2\|_2^2)(-\|\xi_1 - x_2\|_2^2) d\gamma_p & \dots \\ \vdots & \ddots & \vdots \\ \beta_p \exp(-\gamma_p \|\xi_L - x_1\|_2^2)(-\|\xi_L - x_1\|_2^2) d\gamma_p & \dots & \end{pmatrix}$$

$$\mathcal{K}_\beta^p = \begin{pmatrix} \exp(-\gamma_p \|\xi_1 - x_1\|_2^2) & \exp(-\gamma_p \|\xi_1 - x_2\|_2^2) & \dots \\ \vdots & \ddots & \vdots \\ \exp(-\gamma_p \|\xi_L - x_1\|_2^2) & \dots & \end{pmatrix} = \begin{pmatrix} \kappa_\beta^p(\xi_1, x_1) & \kappa_\beta^p(\xi_1, x_2) & \dots \\ \vdots & \ddots & \vdots \\ \kappa_\beta^p(\xi_L, x_1) & \dots & \end{pmatrix}$$

$$d\mathcal{K}_\beta = \beta_p \mathcal{K}_{in} \odot \mathcal{K}_\beta^p d\gamma_p \overset{\bar{\mathcal{K}}_\beta^p := \mathcal{K}_{in} \odot \mathcal{K}_\beta^p}{=} \beta_p \bar{\mathcal{K}}_\beta^p d\gamma_p$$

$$dH = \frac{1}{L}\langle \beta_p \bar{\mathcal{K}}_\beta^p \alpha, \mathcal{K}_\beta \alpha - \eta \rangle d\gamma_p$$

$$\frac{\partial H}{\partial \gamma_p} = \frac{1}{L} \alpha^T \beta_p \bar{\mathcal{K}}_\beta^{pT} (\mathcal{K}_\beta \alpha - \eta)$$

Derivation $\frac{\partial^2 E}{\partial\alpha\partial\gamma_p}$, see equation 3.17

$$\frac{\partial E}{\partial \alpha} = cK_\beta\alpha + K_\beta\ell'(\underbrace{K_\beta\alpha}_{t_\beta}, y) := e$$

$$de = cdK_\beta\alpha + dK_\beta\ell'(t_\beta, y) + K_\beta d\ell'(t_\beta, y) \tag{B.1}$$

Analogous to $d\mathcal{K}_\beta$,

$$dK_\beta = \beta_p K_{in} \odot K^p_\beta d\gamma_p \overset{\bar{K}^p_\beta = K_{in}\odot K^p_\beta}{=} \beta_p \bar{K}^p_\beta d\gamma_p$$

$$d\ell'(t_\beta, y) = \operatorname{diag}(\ell''(t_\beta, y))dt_\beta$$

$$dt_\beta = dK_\beta\alpha$$

$$de = c\beta_p\bar{K}^p_\beta\alpha d\gamma_p + \beta_p\bar{K}^p_\beta\ell'(t_\beta, y)d\gamma_p + K_\beta \operatorname{diag}(\ell''(t_\beta, y))\beta_p\bar{K}^p_\beta\alpha d\gamma_p$$

$$\frac{\partial^2 E}{\partial\alpha\partial\gamma_p} = (c\beta_p\bar{K}^p_\beta\alpha + \beta_p\bar{K}^p_\beta\ell'(t_\beta, y) + K_\beta \operatorname{diag}(\ell''(t_\beta, y))\beta_p\bar{K}^p_\beta\alpha)^T$$

Derivation $\frac{\partial H}{\partial\beta_p}$, see equation 3.18

$$dH = \frac{1}{L}\langle d\mathcal{K}_\beta\alpha, \mathcal{K}_\beta\alpha - \eta\rangle$$

$$d\mathcal{K}_\beta = \begin{pmatrix} \kappa^p_\beta(\xi_1, x_1)d\beta_p & \kappa^p_\beta(\xi_1, x_2)d\beta_p & \dots \\ \vdots & \ddots & \vdots \\ \kappa^p_\beta(\xi_L, x_1)d\beta_p & \dots & \end{pmatrix}$$

$$d\mathcal{K}_\beta = \mathcal{K}^p_\beta d\beta_p$$

$$dH = \frac{1}{L}\langle \mathcal{K}^p_\beta\alpha, \mathcal{K}_\beta\alpha - \eta\rangle d\beta_p$$

$$\frac{\partial H}{\partial\beta_p} = \frac{1}{L}\alpha^T\mathcal{K}^{pT}_\beta(\mathcal{K}_\beta\alpha - \eta)$$

Derivation $\frac{\partial^2 E}{\partial\alpha\beta_p}$, see equation 3.19 As a starting point we take equation B.1:

$$de = cdK_\beta\alpha + dK_\beta\ell'(t_\beta, y) + K_\beta d\ell'(t_\beta, y)$$

$$dK_\beta = K^p_\beta d\beta_p$$

$$d\ell'(t_\beta, y) = \mathrm{diag}(\ell''(t_\beta, y))dt_\beta$$

$$dt_\beta = dK_\beta\alpha$$

$$de = cK_\beta^p\alpha d\beta_p + K_\beta^p\ell'(t_\beta, y)d\beta_p + K_\beta\,\mathrm{diag}(\ell''(t_\beta, y))K_\beta^p\alpha d\beta_p$$

$$\frac{\partial^2 E}{\partial\alpha\beta_p} = (cK_\beta^p\alpha + K_\beta^p\ell'(t_\beta, y) + K_\beta\,\mathrm{diag}(\ell''(t_\beta, y))K_\beta^p\alpha)^T$$

Bibliography

[1] T. Andersson, G. Läthén, R. Lenz, and M. Borga. A fast optimization method for level set segmentation. In *Proceedings of the 16:th Scandinavian Conference on Image Analysis (SCIA)*, volume 5575 of *Lecture Notes in Computer Science*, pages 400–409, Oslo, Norway, June 2009. Springer.

[2] K. Bache and M. Lichman. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2013.

[3] Gükhan H. Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan. *Predicting Structured Data (Neural Information Processing)*. The MIT Press, 2007.

[4] Adrian Barbu. Training an active random field for real-time image denoising. *IEEE Transactions on Image Processing*, 18(11):2451–2462, 2009.

[5] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Img. Sci.*, 2(1):183–202, March 2009.

[6] Kristin P. Bennett, Jing Hu, Xiaoyun Ji, Gautam Kunapuli, and Jong-Shi Pang. Model selection via bilevel optimization. In *IJCNN*, pages 1922–1929. IEEE, 2006.

[7] Kristin P. Bennett, Gautam Kunapuli, Jing Hu, and Jong-Shi Pang. Bilevel optimization and machine learning. In Jacek M. Zurada, Gary G. Yen, and Jun Wang, editors, *WCCI*, volume 5050 of *Lecture Notes in Computer Science*, pages 25–47. Springer, 2008.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(1):281–305, February 2012.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[10] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[11] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer.

[12] Jerome Bracken and James T. McGill. Mathematical programs with optimization problems in the constraints. *Operations Research*, 21(1):37–44, 1973.

[13] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, September 1995.

[14] Olivier Chapelle. Training a support vector machine in the primal. *Neural Comput.*, 19(5):1155–1178, May 2007.

[15] Olivier Chapelle, École Polytechnique, and Nello Cristianini. Choosing multiple parameters for support vector machines. In *Machine Learning*, page 2002, 2002.

[16] Yunjin Chen, Thomas Pock, René Ranftl, and Horst Bischof. Revisiting loss-specific training of filter-based MRFs for image restoration. *CoRR*, abs/1401.4107, 2014.

[17] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[18] Justin Domke. Generic methods for optimization-based modeling. In Neil D. Lawrence and Mark Girolami, editors, *AISTATS*, volume 22 of *JMLR Proceedings*, pages 318–326. JMLR.org, 2012.

[19] Kaibo Duan, S.Sathiya Keerthi, and Aun Neow Poo. Evaluation of simple performance measures for tuning SVM hyperparameters. *Neurocomputing*, 51(0):41 – 59, 2003.

[20] F. Facchinei and J.S. Pang. *Finite-Dimensional Variational Inequalities and Complementarity Problems*. Number Bd. 1 in Finite-dimensional Variational Inequalities and Complementarity Problems. Springer, 2003.

[21] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Comput. Vis. Image Underst.*, 106(1):59–70, April 2007.

[22] Mehmet Gonen and Ethem Alpaydin. Multiple kernel learning algorithms. *J. Mach. Learn. Res.*, 12:2211–2268, July 2011.

[23] Tin Kam Ho and E.M. Kleinberg. Building projectable classifiers of arbitrary complexity. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, volume 2, pages 880–885 vol.2, Aug 1996.

[24] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 408–415, New York, NY, USA, 2008. ACM.

[25] Christian Igel and Michael Hüsken. Empirical evaluation of the improved rprop learning algorithm. *Neurocomputing*, 50:2003, 2003.

[26] Jeremy Jancsary, Sebastian Nowozin, and Carsten Rother. Loss-specific training of non-parametric image restoration models: A new state of the art. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part VII*, ECCV'12, pages 112–125, Berlin, Heidelberg, 2012. Springer-Verlag.

[27] Gautam Kunapuli, Kristin P. Bennett, Jing Hu, and Jong-Shi Pang. Classification model selection via bilevel programming. *Optimization Methods and Software*, 23(4):475–489, 2008.

[28] Karl Kunisch and Thomas Pock. A bilevel optimization approach for parameter learning in variational models. *SIAM J. Imaging Sciences*, 6(2):938–983, 2013.

[29] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178, 2006.

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

[31] Fuxin Li, Catalin Ionescu, and Cristian Sminchisescu. Random fourier approximations for skewed multiplicative histogram kernels. In *DAGM-Symposium*, volume 6376 of *Lecture Notes in Computer Science*, pages 262–271. Springer, 2010.

[32] Max A. Little, Patrick E. McSharry, Eric J. Hunter, Jennifer L. Spielman, and Lorraine O. Ramig. Suitability of dysphonia measurements for telemonitoring of parkinson's disease. *IEEE Trans. Biomed. Engineering*, 56(4):1015–1022, 2009.

[33] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[34] Andreas Opelt, Axel Pinz, Michael Fussenegger, and Peter Auer. Generic object recognition with boosting. *PAMI*, 28:2006, 2004.

[35] G. Peyré and J. Fadili. Learning analysis sparsity priors. In *Proc. of Sampta'11*, 2011.

[36] Martin Riedmiller. Rprop - description and implementation details. Technical Report, Universität Karlsruhe, 1994.

[37] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS*, pages 586–591, 1993.

[38] Kegan G. G. Samuel and Marshall F. Tappen. Learning optimized MAP estimates in continuously-valued MRF models. In *CVPR*, pages 477–484. IEEE, 2009.

[39] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[40] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

[41] B. Siddiquie, S.N. Vitaladevuni, and Larry S. Davis. Combining multiple kernels for efficient image classification. In *Applications of Computer Vision (WACV), 2009 Workshop on*, pages 1 – 8, 2009/12// 2009.

[42] Ankur Sinha. Evolutionary bilevel optimization. http://bilevel.org/, May 2014.

[43] Marshall F. Tappen, Ce Liu, Edward H. Adelson, and William T. Freeman. Learning gaussian conditional random fields for low-level vision. In *CVPR*. IEEE Computer Society, 2007.

[44] Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[45] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[46] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

[47] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman. Multiple kernels for object detection. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2009.

[48] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *Pattern Analysis and Machine Intellingence*, 34(3), 2011.

[49] Jun Yang, Yu-Gang Jiang, Alexander G. Hauptmann, and Chong-Wah Ngo. Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of the International Workshop on Workshop on Multimedia Information Retrieval*, MIR '07, pages 197–206, New York, NY, USA, 2007. ACM.

[50] Hongming Zhang, Wen Gao, Xilin Chen, and Debin Zhao. Object detection using spatial histogram features. *Image and Vision Computing*, 24(4):327 – 341, 2006.

[51] Jian Zhang, Rong Jin, Yiming Yang, and Alexander G. Hauptmann. Modified logistic regression: An approximation to SVM and its applications in large-scale text categorization. In Tom Fawcett and Nina Mishra, editors, *ICML*, pages 888–895. AAAI Press, 2003.

[52] Jianguo Zhang, Marcin Marszałek, Svetlana Lazebnik, and Cordelia Schmid. Local features and kernels for classification of texture and object categories: a comprehensive study. *International Journal of Computer Vision*, 73(2):213–238, jun 2007.

[53] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 116–, New York, NY, USA, 2004. ACM.

[54] Xi Zhou, Kai Yu, Tong Zhang, and Thomas S. Huang. Image classification using super-vector coding of local image descriptors. In *Computer Vision – ECCV 2010*, volume 6315 of *Lecture Notes in Computer Science*, pages 141–154. Springer Berlin Heidelberg, 2010.

[55] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, December 1997.