Optimization Algorithms for Kernel Methods

Vojtěch Franc

xfrancv@cmp.felk.cvut.cz

CTU–CMP–2005–22

July 29, 2005

Available at
ftp://cmp.felk.cvut.cz/pub/cmp/articles/franc/Franc-PhD.pdf

Thesis Advisor: Prof. Ing. Václav Hlaváč, CSc.

The research presented in this thesis was supported by the Ministry of Education, Youth and Sports of the Czech Republic under the grant no. MSM6840770013 and the European Commission under project IST-004176 COSPAL and by the Czech Science Foundation under project GACR 102/03/0440 and by the Austrian Ministry of Education under project CONEX GZ 45.535 and by the EU INTAS project PRINCESS 04-77-7347.

Research Reports of CMP, Czech Technical University in Prague, No. 22, 2005

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Optimization Algorithms for Kernel Methods

A dissertation presented to the Faculty of the Electrical Engineering of the Czech Technical University in Prague in partial fulfillment of the requirements for the Ph.D. degree in Study Programme No. P2612 – Electrotechnics and Informatics, branch No. 3902V035 – Artificial Intelligence and Biocybernetics, by

Vojtěch Franc

July 29, 2005

Thesis Advisor

Prof. Ing. Václav Hlaváč, CSc.

Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo Náměstí 13, 121 35 Prague 2, Czech Republic
Fax: +420 224 357 385, phone: +420 224 357 465
`http://cmp.felk.cvut.cz/`

Abstract

Kernel methods are learning systems which employ embedding of the input data in high-dimensional feature spaces. The embedding is implemented via kernel functions which act as a dot product in Reproducing Kernel Hilbert Spaces (RKHS). Learning algorithms which require only the canonical dot product of data represented in finite dimensional linear spaces can be generalized to perform in RKHS. This generalization, called a kernel trick, is implemented by substituting a kernel function for the dot products. This technique is useful whenever the kernel functions represent similarity between data better or more efficiently than the canonical dot product.

This thesis concentrates mainly on the Support Vector Machines (SVM) learning of classifiers which is a particular example of kernel methods. Learning of the SVM classifier aims to minimize the empirical error while the complexity of the classification rule is controlled at the same time. The learning is expressed as a specific convex Quadratic Programming (QP) task. A design of QP solvers is a challenging problem for two reasons at least: (i) a large training data are often available for learning which give rise to large QP tasks with many variables and (ii) the solvers should be fast because a lot of instances of SVM classifiers must be learned during the model selection stage. The first part of the thesis concentrates on the QP solvers based on known algorithms for the Minimal Norm Problem (MNP) and the Nearest Point Problem (NPP). The main contributions of the thesis involve: (i) known algorithms for MNP and NPP are described in a common framework, (ii) the algorithms are generalized to solve more complex QP tasks and convergence proofs in a finite number of iterations are given and (iii) a novel and faster QP solver is proposed. The proposed QP solvers are simple to implement and they were successfully tested on problems with hundred thousands variables.

A novel greedy algorithm for approximation of the training data embedded in the RKHS is proposed in the second part of the thesis. The method is called Greedy Kernel Principal Component Analysis (Greedy KPCA). Similarly to the ordinary Kernel Principal Component Analysis (KPCA), the aim is to minimize the squared reconstruction error of the approximated data. In contrast to the ordinary KPCA, the Greedy KPCA also aims to find a simple model in which the data are described. The proposed method is suitable for reduction of computational and memory requirements of the kernel methods. It can be also applied for reduction of complexity of functions learned by the kernel methods. It was experimentally verified that the method can reduce computational complexity of the Kernel Least Squares Regression and it can speed up evaluation of the SVM classifier.

A basic SVM learning is formulated for the binary classifiers. There exists a multiclass formulation which, however, leads to a considerable more complex QP task compared to the binary case. A novel method which allows to transform the learning of the multiclass SVM to the singleclass SVM classifier is proposed in the third part of the thesis. The transformation is based on a simplification of the original problem and employing the Kesler's construction. The entire transformation is performed solely by a specially designed kernel function. As a result, any solver for a singleclass SVM problem can be readily used to solve the multiclass problem. The proposed method was successfully applied to learning of the Optical Character Recognition systems for a commercial application.

The proposed methods were incorporated to the Statistical Pattern Recognition (STPR) toolbox `http://cmp.felk.cvut.cz/~xfrancv/stprtool` written in Matlab. A substantial part of the toolbox was designed and implemented by the author of the thesis. The toolbox contains an ensemble of pattern recognition techniques, e.g., methods for learning the linear discriminant functions, feature extraction, density estimation and clustering, Support Vector Machines, various kernel methods, etc.

Resumé

Jádrové metody jsou učící se systémy reprezentující vstupní data v příznakovém prostoru vysoké dimenze pomocí jádrových funkcí. Příznakový prostor je v tomto případě Hilbertův prostor s reprodukčními jádry (RKHS). Učící se algoritmy pracující pouze se skalárními součiny dat representovaných v lineárním prostoru s konečnou dimenzí lze zobecnit tak, aby pracovaly v RKHS. Toto zobecnění, známé jako "jádrový trik", se provede nahrazením skalárních součinů zvolenou jádrovou funkcí. Tato technika je vhodná v případech, kdy jádrové funkce vyjadřují podobnost mezi daty lépe nebo efektivněji než kanonický skalární součin.

Tato disertační práce je zaměřena na metodu učení klasifikátorů zvanou Support Vector Machines (SVM), jenž je typickým příkladem jádrových metod. Při učení SVM klasifikátoru je cílem minimalizovat empirickou chybu a současně udržovat nízkou složitost klasifikátoru. Problém učení je vyjádřen jako konvexní optimalizační problém kvadratického programování (QP). Návrh optimalizátorů pro učení SVM je těžký problém hlavně ze dvou důvodu: (i) typicky je třeba učit z rozsáhlých trénovacích dat, což vede na QP s mnoha proměnnými a (ii) optimalizace musí být rychlá, protože je třeba učit mnoho SVM klasifikátorů během fáze výběru modelu. První část disertační práce je věnovaná návrhu QP optimalizátorů, které jsou založeny na známých algoritmech pro řešení tzv. Minimal Norm Problem (MNP) a Nearest Point Problem (NPP) z výpočetní geometrie. Hlavními přínosy disertační práce jsou: (i) známé algoritmy řešící MNP a NPP jsou popsány ve společném rámci, (ii) tyto algoritmy jsou zobecněny pro řešení složitějšího QP problému a je dokázána jejich konvergence v konečném počtu kroků a (iii) je navržen nový rychlejší algoritmus. Navržené algoritmy jsou jednoduché a byly úspěšně testovaný na velkých problémech se sto tisíci proměnnými.

Ve druhé části disertační práce je navržen hladový algoritmus pro aproximaci trénovacích dat v RKHS. Metoda je nazvána Greedy Kernel Principal Component Analysis (Greedy KPCA). Stejně jako u klasická Kernel Principal Component Analysis (KPCA) je i zde cílem minimalizovat rekonstrukční chybu aproximovaných dat. Na rozdíl od KPCA, se Greedy KPCA snaží navíc popsat data jednoduchým modelem. Navržená metoda je vhodná pro snížení paměťové a výpočetní náročnosti jádrových metod a současně ke zjednodušení analytického popisu funkcí, které se jádrovými metodami učí. Experimentálně bylo ukázáno, že lze navrženou metodu použít ke snížení výpočetní náročnosti regresní metody Kernel Least Squares a k urychlení SVM klasifikátoru.

Základní verze SVM učení je navržena pouze pro případ binárních klasifikátorů. Formulace SVM učení klasifikátorů do více tříd existuje, ale vede na problém QP podstatně těžší v porovnání k binárnímu případu. Ve třetí části disertační práce je navržena transformace, která umožňuje převést problém SVM učení vícetřídního klasifikátoru na problém učení klasifikátoru do jedné třídy. Transformace je založena na zjednodušní původního problému a použití Keslerovy konstrukce. Celá transformace je provedena pouze použitím speciálně navržené jádrové funkce. To znamená, že jakákoliv metoda pro učení jednotřídního SVM klasifikátoru může být okamžitě použita pro učení klasifikátoru do více tříd. Navržená metoda byla úspěšně použita pro návrh komerčního systému pro rozpoznávání znaků.

Všechny navržené metody byly začleněny do Statistical Pattern Recognition Toolboxu `http://cmp.felk.cvut.cz/~xfrancv/stprtool` pro Matlab. Podstatná část toolboxu byla navržena a implementována autorem této disertační práce. Toolbox obsahuje soubor nástrojů pro rozpoznávání jako například metody pro učení lineárních diskriminačních funkcí, metody pro odhad hustot pravděpodobnosti a shlukování, Support Vector Machines, jádrové metody a jiné.

Acknowledgement

I am indebted to my supervisor Professor Václav Hlaváč for leading me and giving me an opportunity to study at the Center For Machine Perception. He showed me how to conduct research and help me with writing and researching of this thesis. I wish also to thank Professor Michail Ivanovič Schlesinger whose work was a source of many ideas and largely formed my view of pattern recognition. My special thanks go to Professor Mirko Navara for careful reading and commenting on the thesis. His suggestions and comments had a substantial impact on the final form of the thesis. Last but not least I would like to thank all colleagues from CMP who made me possible to work in an inspiring and a nice atmosphere.

The research presented in this thesis was supported by the Ministry of Education, Youth and Sports of the Czech Republic under the grant No. MSM6840770013 and the European Commission under project IST-004176 COSPAL and by the Czech Science Foundation under project GACR 102/03/0440 and by the Austrian Ministry of Education under project CONEX GZ 45.535 and by the EU INTAS project PRINCESS 04-77-7347.

Contents

Notation

Upper-case bold letters denote matrices, for instance X. Vectors are implicitly column vectors. Vectors are denoted by lower-case bold italic letters. For instance, $\mathrm{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m]$ is a matrix which contains m column vectors $\boldsymbol{x}_i$, $i = 1, \ldots, m$. The i-th entry of the vector $\boldsymbol{x}_j$ is denoted as $[\boldsymbol{x}_j]_i$. For non-indexed vectors, e.g., the vector $\boldsymbol{x}$, the notation $\boldsymbol{x} = [x_1, \ldots, x_n]^T$ is also used to refer to the vector entries. Entry at the i-the row and j-th column of the matrix X is denoted by $[\mathrm{X}]_{i,j}$.

Symbols	
$\mathbb{N}$	the set of natural numbers
$\mathbb{R}$	the set of real numbers
$\mathcal{X}$	input space (space of observable states)
$\mathcal{Y}$	output space (space of hidden states)
$\mathcal{D}$	set of decisions
$\mathcal{H}$	feature space
f	decision function (or discriminant function) $f\colon \mathcal{X} \to \mathcal{D}$
q	classification rule $q\colon \mathcal{X} \to \mathcal{Y}$
$\langle \boldsymbol{x}, \boldsymbol{x}' \rangle$	dot product between $\boldsymbol{x}$ and $\boldsymbol{x}'$
$k(x, x')$	kernel function
$p(x, y)$	probability distribution function
$\lVert \cdot \rVert$	Euclidean norm, $\lVert \boldsymbol{x} \rVert = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}$
$\boldsymbol{\mu}$	mean vector
S	scatter matrix $\mathrm{S} = \sum_{i=1}^{m} (\boldsymbol{x}_i - \mu)(\boldsymbol{x}_i - \mu)^T$
$\mathcal{T}_{\mathcal{X}}$	unlabeled training set $\mathcal{T}_{\mathcal{X}} = \{x, \ldots, x_m\}$
$\mathcal{T}_{\mathcal{X}\mathcal{Y}}$	labeled training set $\mathcal{T}_{\mathcal{X}\mathcal{Y}} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
n	dimension of input space
m	number of training examples
$\delta(i, j)$	Kronecker delta $\delta(i, j) = 1$ for $i = j$ and $\delta(i, j) = 0$ otherwise
$\binom{m}{l}$	binomial coefficient
E	identity matrix

Abbreviations	
AB	AdaBoost
BSVM	Bounded formulation of Support Vector Machines
DAGs	Direct Acyclic Graphs
ERM	Empirical Risk Minimization
GMNP	Generalized Minimal Norm Problem
GNPP	Generalized Nearest Point Problem
IMDM	Improved Mitchell-Demyanov-Malozemov algorithm
KFD	Kernel Fisher Discriminant
KPCA	Kernel Principal Component Analysis
MDM	Mitchell-Demyanov-Malozemov algorithm
MNP	Minimal Norm Problem
NPP	Nearest Point Problem
PCA	Principal Component Analysis
QP	Quadratic Programming
RKHS	Reproducing Kernel Hilbert Space
RSDE	Reduced Set Density Estimator
SMO	Sequential Minimal Optimizer
SRM	Structural Risk Minimization
SVDD	Support Vector Data Description
SVM	Support Vector Machines
VC	Vapnik-Chervonenkis (dimension)
OCR	Optical Character Recognition

1 Introduction

1.1 Learning theory and kernel methods

A basic task of *learning theory* is to estimate a functional dependency between states of the analyzed object given a finite set of examples. The analyzed object is described by its input (observable) state $x \in \mathcal{X}$ and output (hidden) state $y \in \mathcal{Y}$. The space $\mathcal{X}$ is referred to as the *input space* and the space $\mathcal{Y}$ as the *output space*. The relation between the elements of $\mathcal{X}$ and $\mathcal{Y}$ is assumed to be given by a probability distribution $p(x, y)$ over the set $\mathcal{X} \times \mathcal{Y}$. The distribution $p(x, y)$ is unknown but a training set $\mathcal{T}_{\mathcal{XY}} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ of examples drawn from the distribution is provided. The goal is to learn a function $f: \mathcal{X} \rightarrow \mathcal{D}$ which makes a decision about the object based on the observable input state $x \in \mathcal{X}$. The symbol $\mathcal{D}$ denotes the set of *possible decisions*. The space $\mathcal{F}$ of all admissible functions f is referred to as the *hypothesis space*. Let $V: \mathcal{Y} \times \mathcal{D} \rightarrow \mathbb{R}$ be a *loss function* which penalizes a decision $f(x)$ when the true output was y. A common approach is to transform the learning of function f into an optimization problem of the risk minimization. The *expected risk* (also *Bayesian risk*) to minimize is defined by

$$R[f] = \int_{\mathcal{X} \times \mathcal{Y}} V(y, f(x))\, p(x, y)\, \mathrm{d}x\, \mathrm{d}y\,, \tag{1.1}$$

i.e., it is the mathematical expectation of value of the loss function $V(y, f(x))$ with respect to the distribution $p(x, y)$. Learning of f is expressed as a problem of selecting $f^* \in \mathcal{F}$ which minimizes the expected risk $R[f]$.

A favorable case in which the distribution $p(x, y)$ is completely known permits to use well established *Bayesian theory* [44]. The Bayesian theory allows to infer the optimal function f^*, also called the *Bayesian strategy*, which minimizes the expected risk $R[f]$. In practice, however, the knowledge available is very often confined to the training set of examples $\mathcal{T}_{\mathcal{XY}}$. If the training set is sufficiently rich and a priori knowledge about the distribution is known then the distribution $p(x, y)$ can be replaced by its estimate $\hat{p}(x, y)$. However, the estimation $\hat{p}(x, y)$ is itself a difficult task and the inferred function is no more optimal.

The learning of the decision function f directly without taking a detour over the estimation of the probability distribution is a subject of the learning theory [7, 46, 53, 54] (also called *Vapnik-Chervonenkis theory*). A straightforward approach is to replace the expected risk with a good approximation which can be evaluated without the unknown distribution. A natural approximation of the expected risk is the *empirical risk*

$$R_{emp}[f] = \frac{1}{m} \sum_{i=1}^{m} V(y_i, f(x_i))\,. \tag{1.2}$$

Learning of f based on the minimization of the empirical risk $R_{emp}[f]$ over the hypothesis space $\mathcal{F}$ is called the *empirical risk minimization* (ERM) induction principle. The empirical risk can be efficiently minimized by known algorithms for some loss functions

V and function spaces $\mathcal{F}$, e.g., Perceptron algorithm, back-propagation, etc. However, the important result of the learning theory states that the minimization of the empirical risk $R_{emp}[f]$ over general hypothesis space $\mathcal{F}$ does not imply desired minimization of the expected risk $R[f]$ regardless how large the training set is. If the hypothesis space is exaggeratedly rich then arbitrary small empirical risk can be achieved but the expected risk may be still high. This effect is called *over-fitting*. The difference between the empirical and expected risk is known as the problem of generalization.

The statistical learning theory provides bounds on the expected error

$$R[f] \leq R_{emp}[f] + R_{str}\left(m, \frac{1}{h}, \frac{1}{\delta}\right) , \tag{1.3}$$

which holds true with a probability at least $1-\delta$ where $\delta \in (0,1)$. The $R_{str}(m, \frac{1}{h}, \frac{1}{\delta})$ is called the *structural risk* which bounds the difference between the expected risk and the empirical risk. The structural risk is a function of the size m of the training set and a capacity h which measures the size of the hypothesis space $\mathcal{F}$. The best-known capacity measure is the Vapnik-Chervonenkis (VC) dimension. Higher value of the VC dimension means higher capacity of the hypothesis space and consequently higher risk of over-fitting. This implies that aside from the minimization of the empirical risk the capacity of the hypothesis space must be controlled. The *structural risk minimization* (SRM) induction principle implements this idea. First, a set of hypotheses spaces $\mathcal{F}_1$, $\mathcal{F}_2$, ..., $\mathcal{F}_p$ with respective capacities $h_1 < h_2 < \ldots < h_p$ is selected. Second, the minimizers f_i of the empirical risk $R_{emp}[f]$ with respect to the hypothesis spaces $\mathcal{F}_i$ is found for all $i = 1, \ldots, p$. Finally, the function f' with the minimal bound (1.3) is selected out of f_1, f_2, ..., f_p.

The *regularized risk minimization* is another theoretical framework used to analyze the learning algorithms [9, 42]. The learning can be seen as the function approximation problem from a finite sample set. The approximation problem from small sample sets can be ill-posed and thus the approximation can lead to numerical instabilities. A standard way to solve this problem in the approximation theory is to use the regularization approach [52]. The regularization approach applied to the learning problem leads to the regularized risk minimization. The learning of f is thus formulated as minimization of the regularized risk functional

$$R_{reg}[f] = R_{emp}[f] + \lambda\Omega[f] , \tag{1.4}$$

where $\lambda > 0$ is called the regularization constant and $\Omega[f]$ is the added regularization term. The first term guarantees that the empirical risk is minimized. The second, regularization, term is selected such that complex and non-smooth functions $f \in \mathcal{F}$ are penalized. The constant λ controls a trade-off between regularization and the empirical risk. The added regularization term restricts the set of functions $f \in \mathcal{F}$ from which the resulting f is learned similarly to the SRM principle. The regularized risk minimization is closely related to the SRM principle [9].

This thesis concentrates on *kernel methods*, especially, on the *support vector machines* (SVM) learning of classifiers [1, 3, 5, 7, 46, 53, 54]. These methods can be formalized both in the framework of the structural risk or in the regularized risk minimization. Besides the theoretical justification, kernel methods are worth studying for their good results in practical applications[1], e.g., hand-written character recognition [33], face detection [39],

[1]An up-to-date list of SVM applications is being maintained at `http://www.clopinet.com/isabelle/Projects/SVM/applist.html` by Isabelle Guyon.

text categorization [27], etc. The kernel methods denote approaches which use kernel functions to represent similarities between instances of the input space $\mathcal{X}$. The use of kernel functions defines both the measure between inputs and it also specifies the hypothesis space of the learned functions.

The *kernel functions* (also called Mercer kernels or positive definite kernels) are symmetric positive definite functions defined on $\mathcal{X} \times \mathcal{X}$ domain which can be *regarded as a dot product* in the Reproducing Kernel Hilbert Space (RKHS). The embedding of input data into the linear space with the dot products allows to employ many linear algorithms for learning. *Linear algorithms* in question are such learning methods which assume that the input domain $\mathcal{X}$ is a finite-dimensional linear space $\mathcal{X} \subseteq \mathbb{R}^n$ and the similarities between input data are represented as a canonical dot product. The reformulation of linear algorithms in the RKHS, done simply by substituting the kernel functions, leads to their generalizations able to learn a broader class of functions.

An important aspect of the learning methods is the algorithmical solvability. In particular, the SVM learning of classifiers is expressed as a specific *quadratic programming* (QP) task which has to be solved to obtain a desired classifier. The QP task associated with SVM learning is a convex optimization problem with linear constraints. Although such optimization tasks have been well studied, this particular QP task is still a challenging problem. The difficulty stems especially from the size of the matrix which defines the quadratic term. The matrix scales quadratically with the number of training data. The large QP tasks cause problems both for the memory and the long computational time requirements.

This thesis focuses on the design of the optimization methods which make the learning tasks feasible even if huge training data are to be processed. The results, however, are not connected solely to the SVM learning but they can be found useful in other optimization problems occurring in the machine learning and pattern recognition as will be shown.

1.2 Thesis road map

Chapter 1 aims to describe a background and problems arising in the kernel based machine learning. Namely, attention is focused on the *Support Vector Machines* (SVM) for classification. The main idea of using the kernel functions to extend linear learning methods is outlined. Different formulations of classifier learning based on SVM's are introduced with focus on the optimization point of view. The *Kernel Principal Component Analysis* is described. Also some basic tools of the optimization theory relevant to the content of the thesis are summarized.

Chapter 2 summarizes the goals of the thesis. The goals are (i) to design the *quadratic programming* (QP) solvers which can handle large tasks arising in the SVM learning, (ii) to design a method which allows to control complexity of functions produced by kernel methods and (iii) to facilitate optimization problem associated with the learning of the multiclass SVM classifier.

Chapter 3 presents the state-of-the-art and the literature related to the problems dealt with in the thesis. The topics include (i) the QP solvers for SVM design, (ii) the idea of the sparse matrix approximation and (iii) the work relevant to the multiclass SVM learning and its modified formulations.

Chapter 4 proposes a new analysis of algorithms for optimization of special instances of the QP task. Namely, the *Generalized Minimal Norm Problem* (GMNP) and *Generalized Nearest Point Problem* (GNPP) are instances of the QP task in question. The algorithms solving the original MNP and NPP are generalized, described in a common framework and their convergence is proven. A novel QP solver is derived. The applications of the proposed QP solvers are listed and their experimental evaluation is given.

Chapter 5 proposes a new method named Greedy Kernel Principal Component Analysis (Greedy KPCA). Applications of the greedy KPCA for controlling complexity of the SVM classifier and the Regularized Kernel Least Squares is described with experimental evaluation.

Chapter 6 proposes a new method which facilitates optimization of the QP task associated to the learning of multiclass SVM classifiers. The method is based on the transformation of the multiclass to the singleclass problem well known from the analysis of linear discriminant functions. It is shown that the same idea can be applied for the learning of the multiclass SVM classifier. An experimental evaluation on benchmark and real data is given as well.

Chapter 7 gives a summary of contributions proposed in the thesis and described in Chapter 4, Chapter 5 and Chapter 6.

Chapter 8 presents directions of the future research.

1.3 Kernel methods

Learning methods can be classified according to the type of functions they learn and according to the input data (representation of the input state). Linear methods constitute one class of approaches to learning. The input state of the *object is represented as a vector* (referred to as the feature vector) in a finite-dimensional linear space. It is assumed that the data term of the learned function is linear in the parameters and the feature vector. Moreover, a broad class of linear learning methods requires only the information which is contained in the dot products between the feature vectors. This type of linear learning methods is described in Section 1.3.1.

Linear methods can be simply extended to learn a broader class of functions which are non-linear with respect to the input feature vector. This technique is based on the non-linear mapping of the input vectors to a new (straightened) feature space. This method is also referred to as the *feature space straightening* [44]. The learned function is non-linear in the original feature space but is linear in the new space so that linear learning methods can be applied. Section 1.3.2 describes the feature space straightening in more detail.

Kernel methods constitute another class of learning methods which extends linear methods. The kernel methods assume that *similarities between input data can be completely represented by kernel functions*. The kernel functions are the dot products in some high dimensional spaces. This allows to apply the linear method which uses only data in terms of dot products in the high dimensional spaces. This is implemented simply by replacing the dot products with the kernel function. Although the idea of using kernel functions is very similar to the feature space straightening mentioned above, it has several advantages.

First, this approach is in some cases computationally much more efficient than the explicit non-linear data mapping, e.g., the polynomial data mapping. Second, the linear methods can be used to learn a much broader class of functions. For example, there are kernel functions with dot products in infinite dimensional function spaces and thus the explicit non-linear data mapping cannot be used. Third, the kernel functions can be defined on an arbitrary input set and not only over linear finite dimensional vector space. For instance, there exist kernels which represent similarities between two images, sentences and or graphs. These are the cases in which the representation of input data as vectors in a finite dimensional space could be rather artificial. Some important results concerning the kernel functions and their use for machine learning are given in Section 1.3.3.

1.3.1 Linear methods

Linear methods assume that the observable input state $x \in \mathcal{X}$ is represented as a vector $\boldsymbol{x} = \Phi(x)$ in a finite-dimensional linear space $\mathcal{H} \subseteq \mathbb{R}^n$. The space $\mathcal{H}$ is referred to as the *feature space* and $\boldsymbol{x} \in \mathcal{H}$ is the *feature vector* associated with the input $x \in \mathcal{X}$. The symbol $\Phi\colon \mathcal{X} \to \mathcal{H}$ denotes the feature map. The function f to be learned is a linear real-valued function $f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ given by parameter vector $\boldsymbol{w} \in \mathcal{H}$ and scalar $b \in \mathbb{R}$.

An example of the linear method is the Perceptron algorithm. Let a training set $\mathcal{T}_{\mathcal{X}\mathcal{Y}} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_1)\} \in (\mathbb{R}^n \times \{+1,-1\})^m$ of m examples of observations and corresponding hidden states be given. The observations are represented as n-dimensional feature vectors and the hidden states can attain only two values $\{+1, -1\}$. The aim is to learn a linear discriminant function $f(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{w} \rangle + b$ its sign can be used to classify observations to classes (hidden states) such that

$$y = \operatorname{sgn}(f(\boldsymbol{x})) = \operatorname{sgn}\left(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b\right) = \begin{cases} +1 & \text{for} \quad f(\boldsymbol{x}) \geq 0\,, \\ -1 & \text{for} \quad f(\boldsymbol{x}) < 0\,. \end{cases} \tag{1.5}$$

The linear classifier which classifies all training examples correctly is sought. Assuming that such classifier exists the training set is called *linearly separable*. The Perceptron algorithm is an iterative procedure which finds such a linear classifier. It is proven that the Perceptron halts in a finite number of iterations provided the training set is linearly separable. The Perceptron algorithm is the following:

Algorithm 1: Perceptron

1. Initialization. Set $\boldsymbol{w}^{(0)} = 0$ and $b^{(0)} = 0$.
2. Find any pair $(\boldsymbol{x}_k, y_k) \in \mathcal{T}_{\mathcal{X}\mathcal{Y}}$ which is misclassified, i.e.,

$$y_k(\langle \boldsymbol{w}^{(t)}, \boldsymbol{x}_k \rangle + b^{(t)}) \leq 0\,.$$

   If no such pair exists than halt with solution $(\boldsymbol{w}^{(t)}, b^{(t)})$, otherwise go to Step 3.

3. Update solution

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + y_k \boldsymbol{x}_k\,, \quad \text{and} \quad b^{(t+1)} = b^{(t)} + y_k\,.$$

   Continue to Step 2.

It can be observed that both the evaluation of the linear classifier (1.5) and its learning by the Perceptron Algorithm 1 works with feature vector representation of the input data. In many cases, however, it is sufficient to use only the information contained in the dot products between the feature vectors which represent the input data. The dot product of two feature vectors $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^n$ is defined as

$$\langle \boldsymbol{x}, \boldsymbol{x}' \rangle = \sum_{i=1}^{n} [\boldsymbol{x}]_i \, [\boldsymbol{x}']_i \, . \tag{1.6}$$

This can be easily shown for the above mentioned example of the Perceptron Algorithm 1. The key observation is that the vector $\boldsymbol{w}$ can be always expressed as a linear combination of the training feature vectors, i.e.,

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

where $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m]^T \in \mathbb{R}^m$ is a weight vector. The mentioned fact follows directly from the update formula in Step 3 of the Perceptron Algorithm 1. Therefore the discriminant function of the linear classifier can be expressed equivalently as

$$f(\boldsymbol{x}) = \left\langle \sum_{i=1}^{m} \alpha_i \boldsymbol{x}_i, \boldsymbol{x} \right\rangle + b = \sum_{i=1}^{m} \alpha_i \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle + b \, . \tag{1.7}$$

The discriminant function (1.7) is thus defined by the pair $(\boldsymbol{\alpha}, b) \in (\mathbb{R}^m \times \mathbb{R})$. The Perceptron Algorithm 1 can be readily rewritten to update the vector $\boldsymbol{\alpha}$ instead of the vector $\boldsymbol{w}$. This reformulation is called *the dual representation of the Perceptron algorithm* and it looks as follows:

Algorithm 2: Dual Perceptron

1. Initialization. Set $\boldsymbol{\alpha}^{(0)} = 0$ and $b^{(0)} = 0$.

2. Find an index k of any pair $(\boldsymbol{x}_k, y_k) \in \mathcal{T}_{\mathcal{XY}}$ which is misclassified, i.e.,

$$y_k \left(\sum_{i=1}^{m} [\boldsymbol{\alpha}^{(t)}]_i \langle \boldsymbol{x}_i, \boldsymbol{x}_k \rangle + b^{(t)} \right) \leq 0 \, .$$

   If no such pair exists than halt with solution $(\boldsymbol{\alpha}^{(t)}, b^{(t)})$, otherwise go to Step 3.

3. Update solution

$$[\boldsymbol{\alpha}^{(t+1)}]_i = \begin{cases} [\boldsymbol{\alpha}^{(t)}]_i + 1 & \text{for} \quad i = k \, , \\ [\boldsymbol{\alpha}^{(t)}]_i & \text{for} \quad i \neq k \, , \end{cases} \, , \quad \text{and} \quad b^{(t+1)} = b^{(t)} + y_k \, .$$

   Continue to Step 2.

It can be seen that both the evaluation of the linear classifier (1.7) and its learning by the dual representation of Perceptron Algorithm 2 requires the input data in terms of the dot products only. The explicit representation of the input data as feature vectors is not used at all.

There are many linear learning methods which can be expressed in the form using the dot products of input data only. The Representer Theorem 1.1 introduced bellow describes a broad class of such learning methods designed not only for binary classifiers. Well-known examples are the linear Support Vector Machines (SVM), the Fisher Linear Discriminant (FLD), the Principal Component Analysis (PCA), etc.

The linear methods are often simple but the linear function does not have to fit well to a given feature representation of data. If this situation occurs then one can change the representation of the input data and used the same linear learning method. The representation of the input data can be changed in two ways:

(i) The feature vector representation of the input observations is changed. It means that the linear learning method is provided with different training vectors $\boldsymbol{x}' = \Phi'(x)$ obtained from a different feature map $\Phi' \colon \mathcal{X} \to \mathcal{H}'$.

(ii) The dot product, i.e., the similarity measure of the input observations, is changed. It means that the original dot products $\langle \Phi(x), \Phi(x') \rangle$ used in the linear learning method are replaced with a different dot product generally denoted as the *kernel (or positive definite) functions* $k(x, x')$.

The extension of the linear method by using a different feature map is related to so called *feature space straightening* described in Section 1.3.2. The thesis deals with the method in which the dot products are replaced by general kernel functions. A definition and basic properties of kernel functions are described in Section 1.3.3.

1.3.2 Feature space straightening

The feature space straightening is a technique which extends the linear learning algorithms to produce non-linear decision functions. It is assumed that the input space $\mathcal{X} \subseteq \mathbb{R}^n$ is n-dimensional linear space. Let the input vectors $\{x_1, \dots, x_m\}$ be mapped to their images $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}$ in d-dimensional feature space $\mathcal{H} \subseteq \mathbb{R}^d$ by feature map Φ defined as

$$\Phi(x) = [\phi_1(x), \dots, \phi_d(x)]^T \,. \tag{1.8}$$

The feature map Φ is composed of some non-linear functions $\phi_i \colon \mathbb{R}^n \to \mathbb{R}$, $i = 1, \dots, d$. Usually, the dimension d of the feature space $\mathcal{H}$ is much higher than the dimension n of the input space $\mathcal{X}$.

An arbitrary linear method applied on the training data $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}$ (it can be both labelled or unlabelled) produces a linear function in the feature space $\mathcal{H}$. However, the function can be seen as the non-linear function in the input space $\mathcal{X}$. It is important that the function is still linear in its parameters and thus the linear learning method can be applied. For instance, the discriminant function of the linear classifier (1.5) can be expressed as a function of the vectors of input space $\mathcal{X}$ as

$$f(x) = \langle \boldsymbol{w}, \Phi(x) \rangle + b = \sum_{i=1}^{d} \phi_i(x)[\boldsymbol{w}]_i + b \,. \tag{1.9}$$

The non-linearity is controlled by the type of the used feature map (1.8). The above-mentioned technique is called the non-linear data mapping or the feature space straightening [44].

1.3.3 Kernel functions

This section briefly describes kernel functions and associated feature space $\mathcal{H}$ in which the kernel functions act as the dot products. The introduced definitions and theorem are adopted from [46, 56].

Let $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel function which measures similarity between two input patterns $x, x' \in \mathcal{X}$. The similarities between a finite set of input patterns $\{x_1, \ldots, x_m\}$ can be represented as the kernel matrix (also called Gram matrix).

Definition 1.1 (Kernel matrix) *Given a function $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and patterns $\{x_1, \ldots, x_m\} \in \mathcal{X}^m$, the real matrix $\mathrm{K} \in \mathbb{R}^{m\times m}$ with elements*

$$[\mathrm{K}]_{i,j} = k(x_i, x_j)\,,$$

is called the kernel matrix with respect to $\{x_1, \ldots, x_m\}$.

The kernel functions (also called Mercer kernels, positive definite kernels) are defined as follows:

Definition 1.2 (Positive definite kernel) *Let $\mathcal{X}$ be a non-empty set. A function k on $\mathcal{X}\times\mathcal{X}$ is called a kernel if for all $m \in \mathbb{N}$ and all $\{x_1, \ldots, x_m\} \in \mathcal{X}^m$ the corresponding kernel matrix $\mathrm{K} \in \mathbb{R}^{m\times m}$ is a symmetric and positive semi-definite, i.e., for all $\boldsymbol{\alpha} \in \mathbb{R}^m$ the inequality*

$$\langle \boldsymbol{\alpha}, \mathrm{K}\,\boldsymbol{\alpha} \rangle \geq 0$$

holds.

Some examples of broadly used kernel functions are listed in Table 1.1.

When using kernel functions, the input examples $\{x_1, \ldots, x_m\} \in \mathcal{X}^m$ are assumed to be represented as a set of functions $k_x\colon \mathcal{X} \to \mathbb{R}$ such that $k_x(x') = k(x, x')$, $\forall x' \in \mathcal{X}$. The pattern $x \in \mathcal{X}$ serves here as an index of the function k_x. Let $\mathcal{H}_0$ be the space formed by all finite linear combinations of functions k_x, $x \in \mathcal{X}$. It means that $\mathcal{H}_0$ is the linear span of functions k_x and, consequently, each $f \in \mathcal{H}_0$ can be expressed as

$$f(x) = \sum_{i=1}^{m} \alpha_i k_{x_i}(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x)\,, \quad \forall x \in \mathcal{X}\,,$$

where $\{x_1, \ldots, x_m\} \in \mathcal{X}^m$ and $\boldsymbol{\alpha} = \mathbb{R}^m$. Let $g \in \mathcal{H}_0$ be a function defined as $g(x) = \sum_{i=1}^{m'} \beta_i k(x'_i, x)$. The dot product between $g, f \in \mathcal{H}_0$ is defined as

$$\langle f, g \rangle = \sum_{i=1}^{m} \sum_{j=1}^{m'} k(x_i, x'_j)\,. \tag{1.10}$$

It can be shown that (1.10) is well defined, i.e., it really is a dot product. The mathematical concept used to describe the feature space $\mathcal{H}$ induced by kernel functions is called the Reproducing Kernel Hilbert Space and is defined as follows:

Kernel Function	Name
$k(\boldsymbol{x},\boldsymbol{x}') = \langle \boldsymbol{x},\boldsymbol{x}'\rangle$	Linear kernel
$k(\boldsymbol{x},\boldsymbol{x}') = \exp(-\frac{\\|\boldsymbol{x}-\boldsymbol{x}'\\|^2}{2\sigma^2})$	Gaussian kernel
$k(\boldsymbol{x},\boldsymbol{x}') = (\langle \boldsymbol{x},\boldsymbol{x}'\rangle + 1)^d$	Polynomial of degree d
$k(\boldsymbol{x},\boldsymbol{x}') = \tanh(\langle \boldsymbol{x},\boldsymbol{x}'\rangle - \theta)$	Multi Layer Perceptron

Table 1.1: Examples of kernel functions for input data from $\boldsymbol{x},\boldsymbol{x}' \in \mathbb{R}^n$.

Definition 1.3 (Reproducing Kernel Hilbert Space (RKHS)) *The RKHS is the closure of space $\mathcal{H}_0$ with respect to the norm induced by the dot product (1.10), $\|\cdot\| = \sqrt{\langle\cdot,\cdot\rangle}$.*

Henceforth, $\mathcal{H}$ will be used to denote the RKHS if not stated otherwise.

The representer theorem mentioned below describes a broad class of learning methods which allow to represent input data in the RKHS via kernel functions. It is assumed that the labeled training set $\mathcal{T}_{\mathcal{XY}} = \{(x_1,y_1),\dots,(x_m,y_m)\}$ is given and the similarities between patterns are represented by a kernel function $k\colon \mathcal{X}\times\mathcal{X} \to \mathbb{R}$. The goal is to learn a real valued function $f\colon \mathcal{X} \to \mathbb{R}$ which estimates well the hidden state $y \in \mathcal{Y} \subseteq \mathbb{R}$ from the input pattern x. A single estimate of hidden state y with the learned $f(x)$ is assessed by the loss function $V\colon \mathbb{R}^2 \to \mathbb{R}$. The function space $\mathcal{F}$ is assumed to be formed by a real-valued function $f = g + h$, where $g \in \mathcal{H}$ and $h \in \mathrm{Span}(\{\psi_1,\dots,\psi_M\})$. The functions $\psi_p\colon \mathcal{X} \to \mathbb{R}$, $p = 1,\dots,M$, have the property that the matrix $\mathrm{T} \in \mathbb{R}^{m\times M}$, $[\mathrm{T}]_{i,p} = \psi_p(x_i)$ is of rank M.

Theorem 1.1 (Representer theorem) *Each minimizer $f = g + h$, $g \in \mathrm{Span}(\{\psi_1,\dots,\psi_M\})$, $h \in \mathcal{H}$ of the regularized risk functional*

$$R_{reg}[f] = \frac{1}{m}\sum_{i=1}^{m} V(y_i, f(x_i)) + \lambda\|h\|^2 \,,$$

has a representation in the form

$$f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x) + \sum_{p=1}^{M} b_p \psi_p(x) \,,$$

where $\boldsymbol{\alpha} = [\alpha_1,\dots,\alpha_m]^T \in \mathbb{R}^m$ and $\boldsymbol{b} = [b_1,\dots,b_M]^T \in \mathbb{R}^M$.

The Representer Theorem 1.1 can be applied to describe solution of the Support Vectors Machines and kernel PCA which are introduced bellow. In these cases, $M = 1$ and $\psi_1(x) = 1$ is a constant function.

1.4 Support Vector Machines

The following sections describe formulations of learning tasks for binary, multiclass and singleclass SVM classifiers. The SVM's are studied here from the optimization point of view, i.e., how to solve the associated quadratic programming (QP) tasks the formulation of which is described in details. The background regarding the theoretical justification and the generalization bounds is omitted with reference to the relevant literature [7, 46, 53, 54]. The key point used in the formulation of the SVM learning is the transformation between the primal and dual optimization tasks. The relevant part of the optimization theory concerning this problem is introduced in Section 1.6.

1.4.1 Binary Support Vector Machines

The SVMs were originally designed for learning the binary classifier from the labeled training set $\mathcal{T}_{\mathcal{X}\mathcal{Y}} = \{(x_1, y_1), \ldots, (x_m, y_m)\} \in (\mathcal{X} \times \mathcal{Y})^m$. Let $I = \{1, \ldots, m\}$ be a set of indices defined for convenient notation. The output state can attain a binary value from $\mathcal{Y} = \{+1, -1\}$. The input states are assumed to be represented in the feature space $\mathcal{H}$ via the map $\Phi\colon \mathcal{X} \to \mathcal{H}$. The images of input examples in the feature space $\mathcal{H}$ are denoted as $\boldsymbol{x}_i = \Phi(x_i)$. The function $f(x) = \langle \Phi(x), \boldsymbol{w} \rangle + b$ to be learned is the discriminant function the sign of which is used to classify input patterns. The output state is estimated as $y = \operatorname{sgn}(f(x))$. The aim is to design the classifier which minimizes the probability of misclassification. This corresponds to minimization of expected risk $R[f]$ (c.f. (1.1)) with the 0/1-loss function

$$V(y, f(x)) = \begin{cases} 0 & \text{for} \quad y = \operatorname{sgn}(f(x))\,, \\ 1 & \text{for} \quad y \neq \operatorname{sgn}(f(x))\,. \end{cases} \tag{1.11}$$

The minimization of the expected risk is in the SVM learning replaced by the regularized risk minimization [9, 42]

$$(\boldsymbol{w}^*, b^*) = \operatorname*{argmin}_{\boldsymbol{w} \in \mathcal{H}, b \in \mathbb{R}} \left(\frac{1}{m} \sum_{i \in \mathcal{I}} V(y_i, f(x_i)) + \lambda \|\boldsymbol{w}\|^2 \right) . \tag{1.12}$$

However, the *regularized risk minimization* task (1.12) for the 0/1-risk loss function is known to be *intractable in polynomial time*. Thus the SVM's use approximations to the 0/1-loss function which are described below.

The task becomes tractable for the *separable case*, i.e., when the solution with zero empirical risk exists. In the separable case, the hard margin loss function can be used. The hard margin loss is defined as

$$V(y, f(x)) = (\max\{0, 1 - yf(x)\})^\infty = \begin{cases} 0 & \text{if} \quad yf(x) \geq 1\,, \\ \infty & \text{otherwise}\,. \end{cases} \tag{1.13}$$

In practice, however, the *non-separable case* is more common. In the non-separable case, the 0/1-loss function can be replaced by a linear and quadratic approximations. The linear approximation is referred to as the L_1-soft margin loss and is defined as

$$V(y, f(x)) = \max\{0, 1 - yf(x)\} = \begin{cases} 0 & \text{if} \quad yf(x) \geq 1\,, \\ 1 - yf(x) & \text{otherwise}\,. \end{cases} \tag{1.14}$$

The quadratic approximation is referred to as the L_2-soft margin loss function defined as

$$V(y, f(x)) = (\max\{0, 1 - yf(x)\})^2 = \begin{cases} 0 & \text{if} \quad yf(x) \geq 1\,, \\ (1 - yf(x))^2 & \text{otherwise}\,. \end{cases} \tag{1.15}$$

Figure 1.4.1 shows the desired 0/1-loss function compared to its approximation by the L_1-soft and L_2-soft margin loss functions. It is seen that the L_1-soft margin is a better approximation for bigger errors $yf(x) < 0$ while the L_2-soft margin is better for small errors $0 < yf(x) < 1$.

The regularized risk minimization task (1.12) with the hard, L_1-soft, and L_2-soft margin loss functions lead to the quadratic programming (QP) tasks over a convex feasible set.

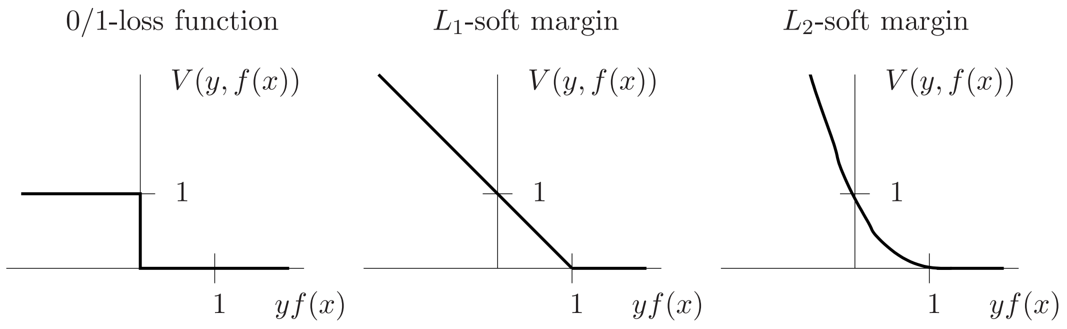


Figure 1.1: The desired 0/1-loss function and its approximation by the L_1-soft and L_2-soft margin loss functions.

This task has only the global optima and can be well algorithmically solved. The QP tasks in question are introduced below.

The linear function to be learned can be seen as the hyperplane $f(x) = \langle \Phi(x), \boldsymbol{w} \rangle + b = 0$ in the feature space separating the positive examples $y_i = +1$ from the negative ones $y_i = -1$. The learning task involves both the minimization of the empirical and the regularization term. The empirical term corresponds to the number of the misclassified patterns when the 0/1-loss function is used. In fact, the linear and quadratic approximations allow to minimize the upper bound on the number of misclassifications. The regularization term restricts the class of hyperplanes to those which have a small normal vector $\|\boldsymbol{w}\|$.

In the separable case, the minimization of the norm $\|\boldsymbol{w}\|$ has a clear geometrical interpretation. It corresponds to the maximization of the margin which is the distance between the hyperplane and the closest training vector defined as

$$\rho(\boldsymbol{w}, b) = \min_{i \in \mathcal{I}} y_i \frac{\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b}{\|\boldsymbol{w}\|} .$$

The hyperplane $f(x) = \langle \boldsymbol{w}^*, \Phi(x) \rangle + b^* = 0$, which separates the positive from the negative examples and has the maximal margin $\rho(\boldsymbol{w}^*, b^*)$, is referred to as the optimal separating hyperplane, i.e.,

$$(\boldsymbol{w}^*, b^*) = \operatorname*{argmax}_{\boldsymbol{w} \in \mathcal{H}, b \in \mathbb{R}} \rho(\boldsymbol{w}, b) .$$

The concept of the optimal separating hyperplane can be generalized for non-separable data. The learning of the generalized optimal separating hyperplane is expressed as the following QP task

$$(\boldsymbol{w}^*, b^*) = \operatorname*{argmin}_{\boldsymbol{w}, b, \boldsymbol{\xi}} \left(\frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i \in \mathcal{I}} (\xi_i)^p \right) , \tag{1.16}$$

subject to

$$\begin{aligned} y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) &\geq 1 - \xi_i , \quad i \in \mathcal{I} , \\ \xi_i &\geq 0 , \qquad\quad i \in \mathcal{I} . \end{aligned}$$

According to the conventions the regularization constant $C \in \mathbb{R}^+$ was used instead of λ in the formulation (1.12). The relation between these constants is given by $C = 2/(m\lambda)$. The *slack variables* $\boldsymbol{\xi} = [\xi_1, \ldots, \xi_m]^T$ were used to handle the non-separable case. The constant

p determines the type of used loss function: (i) hard margin loss function $p = \infty$, (ii) L_1-soft margin loss function $p = 1$ and (iii) L_2-soft margin loss function $p = 2$. Notice that the hard margin case ($p = \infty$), i.e., when the optimal separating hyperplane is sought, is equivalent to the formulation (1.16) without all terms containing slack variables ξ_i removed.

The task (1.16), also referred to as the *primal formulation*, can be transformed to its dual formulation. The solution of the primal formulation can be analytically computed from the solution of the dual task. The *dual formulation* is used for two reasons: (i) it is more convenient for optimization due to a simpler set of linear constraints and, mainly, (ii) the data vectors appear in terms of the dot products only. The transformation from the primal to dual formulation is described in Section 1.6. The corresponding dual formulations are introduced below without derivation which can be found for instance in [3, 7, 46].

First, the hard margin case ($p = \infty$) can be transformed to the dual formulation

$$\boldsymbol{\alpha}^* = \operatorname*{argmax}_{\boldsymbol{\alpha}} \left(\sum_{i\in\mathcal{I}} \alpha_i - \frac{1}{2} \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{I}} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \right) , \tag{1.17}$$

subject to

$$\sum_{i\in\mathcal{I}} \alpha_i y_i = 0 \,, \quad \text{and} \quad \alpha_i \geq 0 \,, i \in \mathcal{I} \,.$$

The solution of the dual task (1.17) determines the solution of the primal task (1.16) such that

$$\boldsymbol{w} = \sum_{i\in\mathcal{I}} y_i \alpha_i \boldsymbol{x}_i \,, \quad \text{and} \quad b = y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \,, \quad \text{for any} \quad 0 < \alpha_i \,.$$

Second, the L_1-soft margin case ($p = 1$) can be transformed to the dual formulation

$$\boldsymbol{\alpha}^* = \operatorname*{argmax}_{\boldsymbol{\alpha}} \left(\sum_{i\in\mathcal{I}} \alpha_i - \frac{1}{2} \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{I}} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \right) , \tag{1.18}$$

subject to

$$\sum_{i\in\mathcal{I}} \alpha_i y_i = 0 \,, \quad \text{and} \quad C \geq \alpha_i \geq 0 \,, i \in \mathcal{I} \,.$$

The solution of the dual task (1.17) determines the solution of the primal task (1.16) such that

$$\boldsymbol{w} = \sum_{i\in\mathcal{I}} y_i \alpha_i \boldsymbol{x}_i \,, \quad \text{and} \quad b = y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \,, \quad \text{for any} \quad 0 < \alpha_i < C \,.$$

Third, the L_2-soft margin case ($p = 2$) can be transformed to the dual formulation

$$\boldsymbol{\alpha}^* = \operatorname*{argmax}_{\boldsymbol{\alpha}} \left(\sum_{i\in\mathcal{I}} \alpha_i - \frac{1}{2} \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{I}} \alpha_i \alpha_j y_i y_j \left(\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \frac{\delta(i,j)}{2C} \right) \right) , \tag{1.19}$$

subject to

$$\sum_{i\in\mathcal{I}} \alpha_i y_i = 0 \,, \quad \text{and} \quad \alpha_i \geq 0 \,, i \in \mathcal{I} \,.$$

The solution of the dual task (1.17) determines the solution of the primal task (1.16) such that

$$\boldsymbol{w} = \sum_{i \in \mathcal{I}} y_i \alpha_i \boldsymbol{x}_i \;, \quad \text{and} \quad b = y_i - \frac{\alpha_i}{2C} - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \;, \quad \text{for any} \quad \alpha_i > 0 \;.$$

The solution vector $\boldsymbol{w}$ is expressed in all the formulations as a linear combination of the training vectors. The training vectors $\boldsymbol{x}_i$ corresponding to the non-zero multipliers $\alpha_i > 0$ are called the *support vectors* and they are sufficient to determine the solution vector and the resulting function. Let $\mathcal{I}_\emptyset = \{i \in \mathcal{I} \colon \alpha_i > 0\}$ contain the indices of the support vectors. The important property of the SVM is that usually the support vectors form a small part of the whole training set. Due to this property, the solution yielded by the SVM is denoted as sparse. Geometrically, the support vectors $\boldsymbol{x}_i$, $i \in \mathcal{I}_\emptyset$ are the training examples which are closest to the hyperplane in the separable case. The learned function can be expressed as

$$f(x) = \sum_{i \in \mathcal{I}_\emptyset} y_i \alpha_i \langle \Phi(x), \boldsymbol{x}_i \rangle + b \;.$$

It is seen that both the learning and the evaluation of $f(x)$ can be done in terms of the dot products only. Therefore replacing all dot products $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ with an arbitrary kernel function $k(x_i, x_j)$ allows to learn functions f from the corresponding RKHS.

1.4.2 Multiclass Support Vector Machines

In the multiclass case, the set of output states $\mathcal{Y} = \{1, 2, \dots, M\}$ attains more than two values $M > 2$. The task is to learn a classification rule which estimates the output state $y \in \mathcal{Y}$ from the input state $x \in \mathcal{X}$ when a training set of examples $\mathcal{T}_{\mathcal{X}\mathcal{Y}} = \{(x_1, y_1), \dots, (x_m, y_m)\} \in (\mathcal{X} \times \mathcal{Y})^m$ is given. The classification rule $q \colon \mathcal{X} \to \mathcal{Y}$ is composed of discriminant functions $f_y(x) = \langle \boldsymbol{w}_y, \Phi(x) \rangle + b_y$, $y \in \mathcal{Y}$, which are here assumed to be linear in the feature space $\mathcal{H}$. The resulting classification rule is then

$$q(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} f_y(x) \;. \tag{1.20}$$

There are two main approaches to design the multiclass SVM classifier of the form (1.20). First, the multiclass SVM formulation and, second, the decomposition based methods. The decomposition based methods are mentioned at the end of this section. In this thesis, the attention is mainly focused on the the multiclass SVM formulation described below.

The multiclass SVM [54, 57] is formulated analogically to the binary case. In contrast to the decomposition-based approaches described below, the parameters of discriminant functions $f_y(x) = \langle \boldsymbol{w}_y, \Phi(x) \rangle + b_y$, $y \in \mathcal{Y}$ are learned all at once. The *primal formulation* of the $(p = 1, 2)$-norm soft margin multiclass SVM task reads

$$(\boldsymbol{w}^*, b^*, \boldsymbol{\xi}) = \operatorname*{argmin}_{\boldsymbol{w}, b, \boldsymbol{\xi}} \left(\frac{1}{2} \sum_{y \in \mathcal{Y}} ||\boldsymbol{w}_y||^2 + C \sum_{i \in \mathcal{I}} \sum_{y \in \mathcal{Y} \setminus \{y_i\}} (\xi_i^y)^p \right) \;, \tag{1.21}$$

subject to

$$\begin{aligned} \langle \boldsymbol{w}_{y_i}, \boldsymbol{x}_i \rangle + b_{y_i} - (\langle \boldsymbol{w}_y, \boldsymbol{x}_i \rangle + b_y) &\geq 1 - \xi_i^y \;, & i \in \mathcal{I}, y \in \mathcal{Y} \setminus \{y_i\} \;, \\ \xi_i^y &\geq 0 \;, & i \in \mathcal{I}, y \in \mathcal{Y} \setminus \{y_i\} \;. \end{aligned}$$

The multiclass formulation (1.21) can be again seen as regularized risk minimization. The sum of squared norms of the parameter vectors corresponds to the regularization term. The empirical risk with 0/1-loss function is approximated by the set of linear inequalities and the associated slack variables. The satisfaction of the linear inequalities implies zero empirical risk. In the non-separable case, the sum of non-zero slack variables approximates the number of misclassified training examples. A disadvantage is that one misclassification of one example can be counted several times to the empirical risk term, i.e., the sum of slack variables is untight upper bound on the number of misclassifications. This disadvantage was resolved by another reformulation of the task proposed by Crammer and Singer [6]. However, they assume the discriminant functions without a bias. In the literature, the hard margin ($p = \infty$) and L_1-soft margin ($p = 1$) formulation were introduced only. However, the L_2-soft ($p = 2$) margin formulation can be defined in analogy to the binary case as well.

The *dual formulation* of the L_1-soft margin SVM task (1.21) reads

$$\boldsymbol{\alpha}^* = \operatorname*{argmax}_{\boldsymbol{\alpha}} \left(\sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{I}} \left(\frac{1}{2}\delta(y_i, j) S_i S_j - \sum_{y\in\mathcal{Y}} \alpha_i^y \alpha_j^{y_i} + \frac{1}{2} \sum_{y\in\mathcal{Y}} \alpha_i^y \alpha_j^y \right) \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - 2 \sum_{i\in\mathcal{I}} \sum_{y\in\mathcal{Y}} \alpha_i^y \right), \tag{1.22}$$

subject to

$$\begin{array}{rcll} \sum_{i\in\mathcal{I}} \alpha_i^y & = & \sum_{i\in\mathcal{I}} \delta(y_i, y) S_i\,, & y \in \mathcal{Y}\,, \\ 0 & \leq & \alpha_i^y \leq C\,, & i \in \mathcal{I}\,,\ y \in \mathcal{Y}\,, \\ 0 & = & \alpha_i^{y_i}\,, & i \in \mathcal{I}\,, \\ S_i & = & \sum_{y\in\mathcal{Y}} \alpha_i^y\,, & i \in \mathcal{I}\,. \end{array} \tag{1.23}$$

The discriminant functions of the classification rule are determined as

$$f_y(\boldsymbol{x}) = \sum_{i\in\mathcal{I}} (\delta(y, i) S_i - \alpha_i^y) \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + b_y\,, \qquad y \in \mathcal{Y}\,,$$

where biases b_y, $y \in \mathcal{Y}$ can be determined from the Karush-Kuhn-Tucker (KKT) optimality conditions (c.f. Section 1.6 for more details about KKT conditions).

It is apparent that the dual task (1.22) is considerably harder compared to the dual task of the binary case (1.19). The complexity stems from the complicated set of linear constraints (1.23). Hence the optimizers for this problem are inevitably more complex which makes this approach less feasible in practice.

Next, the most often used technique of design of the multiclass rule from SVM classifiers is the decomposition based-approach. In this case, the multiclass rule is decomposed into a sequence of binary rules which are learned by a standard binary SVM. The *one-against-rest decomposition* is most frequently used. The discriminant function f_y is learned on a modified training set of examples $\mathcal{T}^y_{\mathcal{XY}} = \{(x_1, y'_1), \ldots, (x_l, y'_m)\}$ with modified class labels

$$y'_i = \begin{cases} +1 & \text{for} \quad y_i = y\,, \\ -1 & \text{for} \quad y_i \neq y\,. \end{cases}$$

It means that each single binary classifier f_y, $y \in \mathcal{Y}$ is learned to distinguish the class y from the rest $\mathcal{Y}\setminus\{y\}$. In many practical problems, the resulting classifier (1.20) composed of the binary ones performs well. There exist other decomposition approaches which, however, use different strategies than (1.20) combining binary rules. The well known approaches are, for instance, the *one-against-one decomposition* [24] and *directed acyclic graphs* (DAGs) [40].

A detailed experimental-based comparison of both the multiclass SVM formulations and the decomposition approaches is given in [25]. The authors claim that (i) there is no approach which would significantly outperform the others in terms of classification error, (ii) the learning time required by the decomposition based approaches (especially one-against-one and DAGs) is shorter than the multiclass SVM formulations and (iii) the multiclass SVM formulations yield simpler classification rules (i.e., with less number of support vectors) and thus they require shorter testing times. These observations are in accordance with the experimental results performed in this thesis, c.f. Chapter 6.

1.4.3 Singleclass Support Vector Machines

The SVM approach is also used for the problem of the novelty detection [45, 46, 51] (also called outlier detection or quantile estimation). In this case, the training examples $\{x_1, \ldots, x_m\} \in \mathcal{X}^m$ are assumed to be generated by an underlying unknown distribution p. The task is to learn a function able to decide whether a new incoming input x was generated from the same distribution $p(x)$ or not. The inputs are again assumed to be represented in a feature space $\mathcal{H}$ via the feature map $\Phi\colon \mathcal{X} \to \mathcal{H}$. In the SVM framework, this task was transformed to the problem of learning function f which separates the training set or its portion from the rest of the feature space. Two similar formulations of this learning task exist: (i) the examples are separated by the hyperplane passing through the origin and maximizing distance from the examples and (ii) the examples are described with the minimal enclosing ball. The first formulation is defined in analogy to the binary SVM classifier and it is described below. The second formulation is referred to as the *Support Vector Data Description* (SVDD) and it is mentioned in Section 4.9.

In contrast to the binary SVM, the singleclass SVM and SVDD for outlier description do not possess such solid theoretical background. However, the tasks themselves are meaningful for other reasons at least. For instance, the radius of the minimal ball is a quantity used to bound the generalization error of the SVM classifier. Next, it will be also shown that learning of the multiclass SVM can be transformed to the singleclass SVM problem.

The singleclass separating hyperplane is formulated as follows. In the separable case, the optimal singleclass separating hyperplane $f(x) = \langle \Phi(x), \boldsymbol{w} \rangle = 0$ separates the training examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ represented in the feature space from the origin while margin $\rho(\boldsymbol{w})$ is maximized. The margin is again the distance of the closest training example to the hyperplane defined as

$$\rho(\boldsymbol{w}) = \min_{i \in \mathcal{I}} \frac{\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle}{\|\boldsymbol{w}\|} \,.$$

The optimal singleclass hyperplane is given by the vector

$$\boldsymbol{w}^* = \operatorname*{argmax}_{\boldsymbol{w} \in \mathcal{H}} \rho(\boldsymbol{w}) \,.$$

In analogy to the binary case, the concept of the optimal singleclass separating hyperplane can be generalized to the non-separable case as well. The generalized optimal singleclass separating hyperplane is defined as the solution of the following QP task

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w},\boldsymbol{\xi}} \left(\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i\in\mathcal{I}}(\xi_i)^p \right) , \tag{1.24}$$

subject to

$$\langle \boldsymbol{x}_i, \boldsymbol{w}\rangle \geq 1 - \xi_i \,, \quad \text{for} \quad i \in \mathcal{I} \,.$$

The slack variables $\boldsymbol{\xi} = [\xi_1, \ldots, \xi_m]^T \in \mathbb{R}^m$ are introduced to allow for the patterns not separable from the origin. The scalar $C \in \mathbb{R}^+$ is the regularization constant. The size of the regularization constant can be used to control the portion of the examples separated by the hyperplane. The formulation (1.24) covers the hard margin ($p = \infty$), L_1-soft margin ($p = 1$) and L_2-soft margin ($p = 2$) penalization of the overlapping patterns. The optimal hyperplane for the separable case is found by solving (1.24) with $p = \infty$. This is equivalent to removing all terms which contain the slack variables ξ_i from the formulation of the task (1.24). Notice, that there exists a similar formulation which has an additional term added to the objective function [45]. The added term allows to set up an upper bound on the number of outliers in the training set.

It is convenient to express the primal task (1.24) in its dual formulation. First, the hard margin case ($p = \infty$) leads to the dual formulation

$$\boldsymbol{\alpha}^* = \operatorname*{argmax}_{\boldsymbol{\alpha}} \left(\sum_{i\in\mathcal{I}} \alpha_i - \frac{1}{2}\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{I}} \alpha_i\alpha_j\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle \right) , \tag{1.25}$$

subject to

$$\alpha_i \geq 0 \,, \quad i \in \mathcal{I} \,.$$

Second, the L_1-soft margin case ($p = 1$) leads to the dual formulation

$$\boldsymbol{\alpha}^* = \operatorname*{argmax}_{\boldsymbol{\alpha}} \left(\sum_{i\in\mathcal{I}} \alpha_i - \frac{1}{2}\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{I}} \alpha_i\alpha_j\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle \right) , \tag{1.26}$$

subject to

$$C \geq \alpha_i \geq 0 \,, \quad i \in \mathcal{I} \,.$$

Third, the L_2-soft margin case ($p = 2$) leads to the dual formulation

$$\boldsymbol{\alpha}^* = \operatorname*{argmax}_{\boldsymbol{\alpha}} \left(\sum_{i\in\mathcal{I}} \alpha_i - \frac{1}{2}\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{I}} \alpha_i\alpha_j \left(\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle + \frac{\delta(i,j)}{2C} \right) \right) , \tag{1.27}$$

subject to

$$\alpha_i \geq 0 \,, \quad i \in \mathcal{I} \,.$$

In all cases, the resulting linear function is determined as

$$f(x) = \sum_{i\in\mathcal{I}_\emptyset} \alpha_i\langle \boldsymbol{x}_i, \Phi(x)\rangle \,,$$

where the subset $\mathcal{I}_\emptyset = \{i\colon \alpha_i > 0\}$ contains indices of the support vectors. The rule $f(x) > b$ is used to approximate the unknown inequality $p(x) > b'$.

It can be again seen that both the learning and the evaluation of the learned function requires data in terms of the the dot products only. Thus the patterns can be embedded to the RKHS via the kernel function substituted for the dot products.

1.5 Kernel Principal Component Analysis

The *Kernel Principal Component Analysis* [47, 46] (KPCA) is an extension of the ordinary Principal Component Analysis (PCA). The KPCA is formulated in the RKHS via the kernel functions. The main idea is again to start with the linear version and to show that only the dot products of the input data are needed.

The ordinary linear PCA is a broadly used technique for compression, denoising, unsupervised feature extraction, etc. The goal is to find a linear orthonormal mapping

$$\boldsymbol{y} = \mathrm{W}\boldsymbol{x} + \boldsymbol{b}\,, \tag{1.28}$$

from the input space $\boldsymbol{x} \in \mathbb{R}^n$ onto a lower dimensional space $\boldsymbol{y} \in \mathbb{R}^d$, $d < n$, which allows to reconstruct training data $\mathrm{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m] \in \mathbb{R}^{n\times m}$ with the minimal quadratic error. The representation of training data in the matrix X is used as it brings a more convenient notation.

Let $\tilde{\mathrm{X}} = [\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_m] \in \mathbb{R}^{n\times m}$ be a matrix of vectors reconstructed by an inverse linear transform from the images $\mathrm{Y} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m] \in \mathbb{R}^{d\times m}$ computed by (1.28). The quadratic reconstruction error is defined as follows

$$\varepsilon_{MS} = \frac{1}{m}\sum_{i=1}^{m} \|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|^2\,. \tag{1.29}$$

The well-known solution to the problem is based on computing eigenvectors of the scatter matrix defined as

$$\mathrm{S} = \sum_{i=1}^{m} (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T\,.$$

The sample mean vector is

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}_i\,.$$

There is a need to solve the eigenvalue decomposition problem: Find a real number $\lambda > 0$ and a vector $\boldsymbol{w} \neq 0$ so that

$$\lambda \boldsymbol{w} = \mathrm{S}\boldsymbol{w}\,.$$

Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$ be the d highest eigenvalues and $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d$, their corresponding eigenvectors. Then $\mathrm{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_d]^T$ is the optimal transformation matrix for which the criterion (1.29) is minimal. The optimal bias vector $\boldsymbol{b}$ is equal to $\boldsymbol{b} = -\mathrm{W}\boldsymbol{\mu}$, so that $\boldsymbol{y} = \mathrm{W}(\boldsymbol{x} - \boldsymbol{\mu})$. Thanks to the linearity of the transformation (1.29) the reconstructed vector $\tilde{\boldsymbol{x}}$ can be explicitly computed from its image $\boldsymbol{y}$ by the inverse transformation

$$\tilde{\boldsymbol{x}} = \mathrm{W}^T\boldsymbol{y} + \boldsymbol{\mu}\,.$$

The KPCA aims to approximate the training data embedded to the RKHS associated with the selected kernel function. The kernel extension is possible as the linear PCA can be expressed in terms of dot products as shown below.

Let $\hat{\mathrm{X}} = \mathrm{X} - \mathrm{XM}$ denote the centered training data where $\mathrm{M} \in \mathbb{R}^{m\times m}$ is a matrix with all entries equal to $1/m$. The dot product matrix of centered data is

$$\begin{aligned}\hat{\mathrm{X}}^T\hat{\mathrm{X}} &= (\mathrm{X} - \mathrm{XM})^T(\mathrm{X} - \mathrm{XM}) \\ &= \mathrm{X}^T\mathrm{X} - \mathrm{M}^T\mathrm{X}^T\mathrm{X} - \mathrm{X}^T\mathrm{XM} + \mathrm{M}^T\mathrm{X}^T\mathrm{XM}\,.\end{aligned} \tag{1.30}$$

The key idea used in KPCA is the correspondence of eigenvectors and eigenvalues of the dot product matrix $\hat{\mathrm{X}}^T\hat{\mathrm{X}}$ and the scatter matrix $\hat{\mathrm{X}}\hat{\mathrm{X}}^T$. The eigenvalue decomposition of the dot product matrix $\hat{\mathrm{X}}^T\hat{\mathrm{X}}$ is

$$\hat{\mathrm{X}}^T\hat{\mathrm{X}}\mathrm{U} = \mathrm{U}\Lambda\,, \tag{1.31}$$

where $\mathrm{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_d] \in \mathbb{R}^{m\times d}$ is an orthonormal matrix of d eigenvectors and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \in \mathbb{R}^{d\times d}$ is a diagonal matrix of corresponding eigenvalues sorted in descending order, i.e., $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$. Multiplying both sides of (1.31) by $\hat{\mathrm{X}}$ yields

$$(\hat{\mathrm{X}}\hat{\mathrm{X}}^T)(\hat{\mathrm{X}}\mathrm{U}) = (\hat{\mathrm{X}}\mathrm{U})\Lambda\,,$$

which means that Λ are also eigenvalues of the scatter matrix $\hat{\mathrm{X}}\hat{\mathrm{X}}^T$ and $\hat{\mathrm{X}}\mathrm{U}$ is a matrix of the corresponding eigenvectors. However, the eigenvectors $\hat{\mathrm{X}}\mathrm{U}$ are not orthonormal since

$$(\hat{\mathrm{X}}\mathrm{U})^T(\hat{\mathrm{X}}\mathrm{U}) = \mathrm{U}^T\hat{\mathrm{X}}^T\hat{\mathrm{X}}\mathrm{U} = \mathrm{U}^T\mathrm{U}\Lambda = \Lambda\,,$$

which follows from (1.31). Finally, the orthonormal eigenvectors $\mathrm{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_l]$ of the scatter matrix $\hat{\mathrm{X}}\hat{\mathrm{X}}^T$ can be computed as

$$\mathrm{V} = \hat{\mathrm{X}}\mathrm{U}\Lambda^{-\frac{1}{2}} = \hat{\mathrm{X}}\mathrm{B}\,, \tag{1.32}$$

where $\Lambda^{-\frac{1}{2}} = \mathrm{diag}(\frac{1}{\sqrt{\lambda_1}}, \ldots, \frac{1}{\sqrt{\lambda_l}})$ is a diagonal matrix and $\mathrm{B} = \mathrm{U}\Lambda^{-\frac{1}{2}}$. It is apparent from the equation (1.32) that the eigenvectors are linear combinations of the training data. The linear PCA projection of a vector $\boldsymbol{x}$ on the eigenvectors V is

$$\begin{aligned} \boldsymbol{y} &= \mathrm{V}^T(\boldsymbol{x} - \boldsymbol{\mu}) \\ &= (\mathrm{XB} - \mathrm{XMB})^T(\boldsymbol{x} - \mathrm{X}\boldsymbol{m}) \\ &= (\mathrm{B} - \mathrm{MB})^T\mathrm{X}^T\boldsymbol{x} - \mathrm{B}^T\mathrm{X}^T\mathrm{X}\boldsymbol{m} + \mathrm{B}^T\mathrm{M}^T\mathrm{X}^T\mathrm{X}\boldsymbol{m}\,. \end{aligned} \tag{1.33}$$

The equation (1.33) was obtained by substituting (1.32), $\hat{\mathrm{X}} = \mathrm{X} - \mathrm{XM}$ and using $\boldsymbol{\mu} = \mathrm{X}\boldsymbol{m}$, where $\boldsymbol{m} \in \mathbb{R}^m$ has all entries equal to $1/m$.

It has been shown that the centering of data (1.30), eigenvector decomposition (1.32) as well as the linear projection (1.33) require the data in terms of dot products only. Therefore the kernel version of PCA can be simply derived by substituting kernels for these dot products. The training data $\mathcal{T}_{\mathcal{X}} = \{x_1, \ldots, x_m\}$ (they do not have to be vectors) are embedded to the RKHS with the selected kernel function k. The KPCA algorithm is introduced below:

Algorithm 3: Kernel Principal Component Analysis (KPCA)

1. Compute kernel matrix $\mathrm{K} \in \mathbb{R}^{m\times m}$ of the training data $\mathcal{T}_{\mathcal{X}} = \{x_1,\dots,x_m\} \in \mathcal{X}^m$ such that $[\mathrm{K}]_{i,j} = k(x_i,x_j)$, $i=1,\dots,m$, $j=1,\dots,m$.

2. Compute centered kernel matrix
$$\hat{\mathrm{K}} = \mathrm{K} - \mathrm{M}^T\mathrm{K} - \mathrm{K}\mathrm{M} + \mathrm{M}^T\mathrm{K}\mathrm{M}\,.$$

3. Solve eigenvalue decomposition problem, i.e., find matrix $\mathrm{U} \in \mathbb{R}^{m\times m}$ with orthonormal columns and a non-zero diagonal matrix $\Lambda \in \mathbb{R}^{m\times m}$ which satisfy
$$\hat{\mathrm{K}}\mathrm{U} = \mathrm{U}\Lambda\,.$$

4. Compose matrix $\Lambda = \mathrm{diag}(\lambda_1,\dots,\lambda_d)$, $\lambda_1 \geq \dots \geq \lambda_d$ and $\mathrm{U} = [\boldsymbol{u}_1,\dots,\boldsymbol{u}_d]$ containing the d highest eigenvalues and eigenvector respectively. Compute
$$\mathrm{B} = \mathrm{U}\Lambda^{-\frac{1}{2}}\,.$$

5. Compute coefficients of the kernel projection
$$\begin{aligned} \mathrm{A} &= (\mathrm{B} - \mathrm{M}\mathrm{B})^T\,, \\ \boldsymbol{\theta} &= \mathrm{B}^T\mathrm{M}^T\mathrm{K}\boldsymbol{m} - \mathrm{B}^T\mathrm{K}\boldsymbol{m}\,. \end{aligned}$$

The Kernel PCA Algorithm 3 produces a matrix $\mathrm{A} \in \mathbb{R}^{d\times m}$ and a vector $\boldsymbol{\theta} \in \mathbb{R}^d$ which are coefficients of the non-linear kernel projection

$$\boldsymbol{y} = \mathrm{A}\boldsymbol{k}(x) + \boldsymbol{\theta}\,, \tag{1.34}$$

where $\boldsymbol{k}(x) = [k(x,x_1),\dots,k(x,x_m)]^T \in \mathbb{R}^m$. The vector $\boldsymbol{k}(x)$ is referred to as the *empirical feature map* of the input $x \in \mathcal{X}$.

1.6 Used results of the optimization theory

This section contains the basic results of the optimization theory relevant to the learning problems dealt with in the thesis. These results were adopted mainly from [2, 10]. In particular, the transformation between the primal and dual formulations of the optimization tasks and their relation is of the interest. Next, the Karush-Kuhn-Tucker (KKT) conditions which specify the necessary conditions for the optimal solution of a constraint optimization problem are defined. Finally, the specific properties of the convex quadratic programming (QP) tasks are pointed out.

Let the optimization task be defined as

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}\in\mathcal{D}} f(\boldsymbol{\theta}) \tag{1.35}$$

subject to

$$\begin{array}{rcll} g_i(\boldsymbol{\theta}) & \leq & 0\,, & i = 1,\dots,l\,,\\ h_i(\boldsymbol{\theta}) & = & 0\,, & i = 1,\dots,m\,. \end{array}$$

The symbol f stands for the real-valued objective function, g_i, $i = 1,\dots,l$, h_j, $j = 1,\dots,m$, are equality and inequality constrains defined on the domain $\mathcal{D} \subseteq \mathbb{R}^n$. The vector of parameters to be optimized is denoted by $\boldsymbol{\theta}$. The task (1.35) is referred to as the *primal optimization task*. The Lagrangian function of the task (1.35) is defined as

$$L(\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta}) = f(\boldsymbol{\theta}) + \sum_{i=1}^{l} \alpha_i g_i(\boldsymbol{\theta}) + \sum_{j=1}^{m} \beta_i h_i(\boldsymbol{\theta})\,,$$

where $\boldsymbol{\alpha} = [\alpha_1,\dots,\alpha_l]^T \in \mathbb{R}^l$ and $\boldsymbol{\beta} = [\beta_1,\dots,\beta_m]^T \in \mathbb{R}^m$ are Lagrangian multipliers associated with the equality and inequality constrains respectively. The *dual optimization task* is defined as

$$(\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*) = \operatorname*{argmax}_{\boldsymbol{\alpha},\boldsymbol{\beta}} q(\boldsymbol{\alpha},\boldsymbol{\beta}) \tag{1.36}$$

subject to

$$\alpha_i \geq 0\,, \qquad i = 1,\dots,l\,,$$

where

$$q(\boldsymbol{\alpha},\boldsymbol{\beta}) = \min_{\boldsymbol{\theta}\in\mathcal{D}} L(\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta})$$

is the dual objective function. In some cases, the analytical form of the dual function $q(\boldsymbol{\alpha},\boldsymbol{\beta})$ can be derived, e.g, for linear or quadratic programming problems. To this end, the solution $\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}\in\mathcal{D}} L(\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta})$ is found by setting

$$\frac{\partial L(\boldsymbol{\theta},\boldsymbol{\alpha},\boldsymbol{\beta})}{\partial \boldsymbol{\theta}} = 0\,, \tag{1.37}$$

and solving for $\boldsymbol{\theta}$. The solution $\boldsymbol{\theta}^*$ is substituted to the Lagrangian so that $q(\boldsymbol{\alpha},\boldsymbol{\beta}) = L(\boldsymbol{\theta}^*,\boldsymbol{\alpha},\boldsymbol{\beta})$.

The difference between the primal objective function $f(\boldsymbol{\theta})$ and the dual objective function $q(\boldsymbol{\alpha},\boldsymbol{\beta})$ is called the *duality gap*. There are two useful theorems concerning the duality gap.

Theorem 1.2 (Weak duality) *Let $\boldsymbol{\theta}^*$ be the solution of the primal task (1.35) and $(\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*)$ be the solution of the dual task (1.36). Then $f(\boldsymbol{\theta}^*) \geq q(\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*)$.*

Theorem 1.2 shows that the dual objective function can be used as the lower bound on the primal objective function. The following theorem says that in some specific cases the duality gap vanishes at the optimal solution.

Theorem 1.3 (Strong duality) *If the domain $\mathcal{D} \subseteq \mathbb{R}^n$ of the primal task (1.35) is convex, the objective function $f(\boldsymbol{\theta})$ is convex and the constraints are affine, i.e.,*

$$\begin{array}{rcll} g_i(\boldsymbol{\theta}) & \leq & \langle \boldsymbol{a}_i,\boldsymbol{\theta}\rangle + b_i\,, & i = 1,\dots,l\,,\\ h_i(\boldsymbol{\theta}) & = & \langle \boldsymbol{c}_i,\boldsymbol{\theta}\rangle + d_i\,, & i = 1,\dots,m\,, \end{array}$$

where $\boldsymbol{a}_i$, $\boldsymbol{c}_i$ are some vectors and b_i, d_i some scalars, then the duality gap vanishes, i.e., $f(\boldsymbol{\theta}^*) = q(\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*)$ in the optimal solution $\boldsymbol{\theta}^*$ and $(\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*)$.

The task is a convex optimization task if the objective function is convex and the constraints define a convex region. Convex optimization problems are known to have only global optima. The Karush-Kuhn-Tucker (KKT) conditions in general define necessary conditions for the optimal solution of the constrained optimization task. For the convex optimization task the KKT conditions are also sufficient.

Theorem 1.4 (Karush-Kuhn-Tucker) *Let (1.35) be a given task with convex domain $\mathcal{D} \subseteq \mathbb{R}^n$ with f differentiable and with convex and affine constraints. Then a necessary and sufficient condition for a point $\boldsymbol{\theta}^*$ to be an optimum is the existence of $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ such that*

$$
\begin{aligned}
\frac{\partial L(\boldsymbol{\theta}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{\theta}} &= 0\,, \\
\frac{\partial L(\boldsymbol{\theta}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} &= 0\,, \\
\alpha_i^* g_i(\boldsymbol{\theta}^*) &= 0\,, \qquad i = 1, \dots, l\,, \\
g_i(\boldsymbol{\theta}^*) &\leq 0\,, \qquad i = 1, \dots, l\,, \\
\alpha_i^* &\geq 0\,, \qquad i = 1, \dots, l\,.
\end{aligned}
$$

Although the convex optimization task has only global optima, they do not have to be necessarily unique. If the non-unique solution occurs then the solution at one optimal solution is continuously deformable into the other optimal solution, in such a way that all intermediate points are also solutions:

Theorem 1.5 *Let $\boldsymbol{\theta}_1^*$ and $\boldsymbol{\theta}_2^*$ be two points at which the convex objective function f attains its minimum. There exists a path $\boldsymbol{\theta}^* = (1-\tau)\boldsymbol{\theta}_1^* + \tau\boldsymbol{\theta}_2^*$, $\tau \in [0,1]$, such that θ^* are optimal solutions.*

The quadratic programming (QP) tasks dealt with in the thesis are the convex optimization problems. The matrix of the quadratic term is positive semidefinite and the constraints are composed of linear equalities and inequalities. The duality gap of these QP tasks is zero, i.e., the values of the primal and dual criterion coincide in the optimal solution. The solution of the primal formulation can be analytically computed from the solution of the dual formulation. The KKT conditions are necessary and sufficient for the optimal solution.

2 Motivation and goals

The goals of the thesis were motivated by problems encountered during a design of classifiers based on the Support Vector Machines (SVM) used for practical applications.

- *The first aim is to design such quadratic programming (QP) solver which can handle large tasks, is reasonably fast, and is not exaggeratedly difficult for implementation.*

  The SVM learning is transformed to a special instance of the QP task. There is a need to have an efficient QP solver able to optimize large problems quickly. The large QP tasks originate from large data sets available for common applications. The requirement on the fast optimization emerges from the need to learn many SVM classifiers during the model selection stage. Even though the problem of designing an efficient QP solver for SVM has been intensively studied there is still a room for further improvements.

- *The second aim is to design a method to control complexity of a resulting classifier.*

  The decision function learned by the SVM has a form of a weighted sum of nonlinear functions centered in a subset of training data called support vectors. The complexity of the rule is proportional to the number of the support vectors. A large number of support vectors implies a long time required for evaluation of the classification rule. A fast evaluation is a strong requirement in many practical applications. Even though the support vectors usually form a small part of the training data, their number can be still large. There is a need to control the complexity of the resulting rule explicitly.

- *The third aim is to ease the optimization problem associated with learning of the multiclass SVM classifier.*

  The basic SVM learning of classifiers is formulated for the binary case only. There exists a multiclass formulation which, however, leads to a QP task considerably more complex for optimization as compared to the binary case. The complexity stems from the complicated set of linear constraints of the QP task. This is the main obstacle of design of efficient QP solvers contrary to the binary case for which many optimizers exist. Any simplification of the QP task associated to the learning problem of the multiclass SVM classifier is thus desirable.

3 State of the art

3.1 Quadratic Programming solvers for SVM learning

This sections outlines approaches to solve the specific quadratic programming (QP) tasks arising in SVM learning. The QP tasks corresponding to the learning of binary SVM classifiers were introduced in Section 1.4.1. The QP task associated with the binary SVM classifier with L_1-soft margin reads

$$\boldsymbol{\alpha}^* = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^m} Q(\boldsymbol{\alpha}) = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^m} \left(\frac{1}{2}\langle \boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha}\rangle - \langle \boldsymbol{e}, \boldsymbol{\alpha}\rangle \right) , \tag{3.1}$$

subject to

$$\begin{array}{rcl} \langle \boldsymbol{\alpha}, \boldsymbol{y}\rangle & = & 0\,, \\ \boldsymbol{\alpha} & \geq & 0\,, \\ \boldsymbol{\alpha} & \leq & C\boldsymbol{e}\,, \end{array}$$

where $\mathrm{H} \in \mathbb{R}^{m\times m}$ is a symmetric positive definite matrix, $\boldsymbol{e} \in \mathbb{R}^m$ is a vector of ones, $y = [y_1, \dots, y_m]^T \in \mathbb{R}^m$ is a vector of labels $y_i \in \{+1, -1\}$, and $C \in \mathbb{R}^+$ is a constant. The QP task corresponding to learning of the binary SVM with L_2-soft margin reads

$$\boldsymbol{\alpha}^* = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^m} Q'(\boldsymbol{\alpha}) = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^m} \left(\frac{1}{2}\langle \boldsymbol{\alpha}, \mathrm{H}'\boldsymbol{\alpha}\rangle - \langle \boldsymbol{e}, \boldsymbol{\alpha}\rangle \right) , \tag{3.2}$$

subject to

$$\begin{array}{rcl} \langle \boldsymbol{\alpha}, \boldsymbol{y}\rangle & = & 0\,, \\ \boldsymbol{\alpha} & \geq & 0\,, \end{array}$$

where $\mathrm{H}' \in \mathbb{R}^{m\times m}$ is a symmetric positive definite matrix. The matrices H and H′ are related by the equation $\mathrm{H}' = \mathrm{H} + \mathrm{E}/(2C)$, where $\mathrm{E} \in \mathbb{R}^{m\times m}$ is the identity matrix. Therefore both tasks coincide in the hard margin case in which $C = \infty$. It is seen that the QP task for the L_2-soft margin is slightly simpler then the L_1-soft margin case because it does not contain the upper bound constraint $\boldsymbol{\alpha} \leq C\boldsymbol{e}$.

The QP tasks (3.1) and (3.2) are convex optimization tasks. The Karush-Kuhn-Tucker (KKT) conditions (c.f. Section 1.6) are necessary and sufficient for these problems. The QP tasks are solved by iterative algorithms building a sequence of solutions $\boldsymbol{\alpha}^{(1)}, \boldsymbol{\alpha}^{(2)}, \dots, \boldsymbol{\alpha}^{(t)}$ which converges to the optimal solution $\boldsymbol{\alpha}^*$. A relaxed version of the KKT conditions is usually used as the stopping conditions which indicate that current solution $\boldsymbol{\alpha}^{(t)}$ is close to the optimal solution $\boldsymbol{\alpha}^*$. Another option is to use the duality gap, i.e., the difference between the upper bound and the lower bound of the optimal value $Q(\boldsymbol{\alpha}^*)$ (or $Q'(\boldsymbol{\alpha}^*)$) to define the stopping conditions. Both the KKT conditions and the duality gap can be efficiently evaluated for these QP tasks [7, 30, 34].

Common optimization methods for solving general QP tasks are not applicable to the problems in question. The main obstacle is the size of the Hessian matrix H (or H′) which grows quadratically with the number of training data. General techniques require expensive operations with the Hessian matrix which can be hardly stored in the memory for common problems with thousands of training data.

The efficient QP solvers used in SVM learning exploit all special properties of the tasks (3.1) and (3.2) which are the following:

- The constraints are particularly simple, i.e., box constraints and a single linear equality constraint are present only.
- The optimal solution is sparse, i.e., only a portion of the variables $\boldsymbol{\alpha}^*$ is non-zero.
- The Hessian matrix is large, thus the access to its entries should be minimized.

Most QP solvers were developed for the SVM problem with L_1-soft margin. These solvers are based on decomposition strategies which are summarized in Section 3.2. Recently, it was shown that the SVM problem with L_2-soft margin is equivalent to the Nearest Point Problem (NPP) which can be solved by simple iterative algorithms. This idea and known algorithms are described in Section 3.3.

3.2 Decomposition algorithms

This section summarizes the decomposition strategies commonly applied to solve the QP task (3.1) associated with the SVM learning problem and L_1-soft margin. The idea is to decompose the original QP task into a sequence of smaller QP tasks which can be already efficiently solved. The algorithms are of an iterative nature, i.e., they build a sequence of solutions $\boldsymbol{\alpha}^{(1)}$, $\boldsymbol{\alpha}^{(2)}$, …, $\boldsymbol{\alpha}^{(t)}$. In each iteration, the variables $\boldsymbol{\alpha}^{(t)} \in \mathbb{R}^m$ are split into a working set and a fixed set. Let $\mathcal{I}_W$ denote the indices of the working set and $\mathcal{I}_F$ be the indices of the fixed set such that $\mathcal{I}_W \cup \mathcal{I}_F = \{1,\dots,m\}$ and $\mathcal{I}_W \cap \mathcal{I}_F = \emptyset$. Let $\boldsymbol{\alpha}$, H, $\boldsymbol{y}$ and $\boldsymbol{e}$ from the definition of the QP task (3.1) be rearranged so that

$$\boldsymbol{\alpha} = \begin{bmatrix} \boldsymbol{\alpha}_W \\ \boldsymbol{\alpha}_F \end{bmatrix}, \quad \mathrm{H} = \begin{bmatrix} \mathrm{H}_{WW} & \mathrm{H}_{WF} \\ \mathrm{H}_{FW} & \mathrm{H}_{FF} \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_W \\ \boldsymbol{y}_F \end{bmatrix}, \quad \boldsymbol{e} = \begin{bmatrix} \boldsymbol{e}_W \\ \boldsymbol{e}_F \end{bmatrix}.$$

The objective function $Q(\boldsymbol{\alpha})$ of the QP task (3.1) can be equivalently expressed as a function

$$Q(\boldsymbol{\alpha}_W, \boldsymbol{\alpha}_F) = -\langle \boldsymbol{\alpha}_W, (\boldsymbol{e}_W - \mathrm{H}_{WF}\boldsymbol{\alpha}_F)\rangle + \frac{1}{2}\langle \boldsymbol{\alpha}_W, \mathrm{H}_{WW}\boldsymbol{\alpha}_W\rangle + \frac{1}{2}\langle \boldsymbol{\alpha}_F, \mathrm{H}_{FF}\boldsymbol{\alpha}_F\rangle - \langle \boldsymbol{\alpha}_F, \boldsymbol{e}_F\rangle .$$

The constraint can be also decomposed into the working and fixed part such that

$$\begin{array}{rcl} \langle \boldsymbol{\alpha}_W, \boldsymbol{y}_W\rangle + \langle \boldsymbol{\alpha}_F, \boldsymbol{y}_F\rangle & = & 1 , \\ \boldsymbol{\alpha} & \geq & 0 , \\ \boldsymbol{\alpha} & \leq & C\boldsymbol{e} . \end{array} \tag{3.3}$$

Let $\mathcal{A}_W(\boldsymbol{\alpha}_F)$ denote a feasible set of solutions $\boldsymbol{\alpha}_W$ when $\boldsymbol{\alpha}_F$ is fixed, i.e., $\boldsymbol{\alpha}_W \in \mathcal{A}_W(\boldsymbol{\alpha}_F)$ satisfies the constraints (3.3). Let the vector $\boldsymbol{\alpha}^{(t)} = [\boldsymbol{\alpha}_W^{(t)}; \boldsymbol{\alpha}_F^{(t)}]$ denote the solution in the t-th iteration. The general decomposition algorithm works as follows:

Algorithm 4: General decomposition algorithm

```
1. Initialization. Select $\alpha^{(1)} = [\boldsymbol{\alpha}_W^{(1)}; \boldsymbol{\alpha}_F^{(1)}]$ satisfying constraints (3.3).
2. Repeat until stopping condition is satisfied:
    a) Select variables for working set $\mathcal{I}_W$ and fixed set $\mathcal{I}_F = \{1, \ldots, m\} \setminus \mathcal{I}_W$.
    b) Set $\boldsymbol{\alpha}_F^{(t+1)} = \boldsymbol{\alpha}_F^{(t)}$ and optimize with respect to the working set
        $\boldsymbol{\alpha}_W^{(t+1)} = \operatorname*{argmin}_{\boldsymbol{\alpha}_W \in \mathcal{A}_W(\boldsymbol{\alpha}_F^{(t)})} Q(\boldsymbol{\alpha}_W, \boldsymbol{\alpha}_F^{(t)})$ .   (3.4)
```

It is seen that the size of the QP task (3.4) depends on the number $q < m$ of selected working variables $\boldsymbol{\alpha}_W$. It is also obvious that the optimization in Step 2(b) cannot increase the value of the criterion, i.e., $Q(\boldsymbol{\alpha}^{(t)}) - Q(\boldsymbol{\alpha}^{(t+1)}) = \Delta^{(t+1)} \geq 0$ always holds. The intention is, of course, to select such working set $\mathcal{I}_W$ that the improvement $\Delta^{(t+1)}$ will be maximized while the size q of the set $\mathcal{I}_W$ is sufficiently small. Provided that the method for selection of the working set satisfies some elementary conditions, the decomposition algorithm ultimately converges to the optimal solution. The convergence proofs of various modifications of the decomposition algorithm can be found in [28, 34, 38].

Particular decomposition algorithms used in SVM learning differ in the size of the working set, the selection method used in the Step 2(a) and the QP solver employed in the Step 2(b) of Algorithm 4. Important representatives of the general decomposition algorithm are the following:

Chunking methods proposed by Vapnik [55]. The working set contains all variables which were non-zero at the last iteration plus some portion of variables which were zero and violated the KKT conditions. The algorithm stops when the working set contains all non-zero variables (support vectors) and the fixed variables do not violate the KKT conditions. This method is simple to implement but becomes uneconomical when the number of support vectors is large.

Decomposition algorithms with fixed size of working set proposed by Osuna [38]. The working set size is fixed. In each iteration, a part of variables from the working set is replaced by previously fixed variables which violate the KKT conditions. This idea is used for instance in SVM^{light} by Joachims [26]. SVM^{light} is the state-of-the-art SVM learning package.

Sequential Minimal Optimizer (SMO) by Platt [41]. The SMO is an extreme case of the general decomposition algorithm in which the working set contains just two variables. This brings the advantage of solving the QP task of the Step 2(b) analytically. The original paper [41] contains a set of heuristics how to select the two variables for optimization. The selection method was further improved by Keerthi et al. [30]. The SMO approach has become very popular due to its implementational simplicity and a fast convergence. It is implemented for instance in the popular *LIBSVM* software package by Chang and Lin [4].

3.3 Nearest point algorithms

This section describes works in which algorithms designed for specific tasks in computational geometry were employed in SVM learning. The tasks in question are the *nearest point problem* (NPP) and *minimal norm problem* (MNP). The NPP is equivalent to the learning of the linear binary SVM classifier with hard margin and the MNP is equivalent to the learning of the linear singleclass SVM classifier with hard margin. There is a couple of simple iterative algorithms solving the NPP and the MNP which can be, therefore, readily used as solvers for learning of the binary and linear SVM classifiers. Moreover, these algorithms can be simply modified to use data in terms of dot products only. This modification allows for two extensions. First, kernel functions can be applied and thus a broad class of functions can be learned. Second, the algorithms can be used also for SVM problems with L_2-soft margin. This is true because the problem with L_2-soft margin can be transformed to the hard margin case via a particular kernel function (c.f. Section 1.4.1).

The idea of learning the separating hyperplane in the linear case and for separable data by the use of a particular algorithm for the NPP was proposed by Kozinec [32]. A couple of variants of the Kozinec algorithm applied for the design of linear classifiers are described in the book by Schlesinger and Hlaváč [44]. The use of NPP algorithms explicitly for the learning of binary SVM classifiers was proposed in works by Keerthi et al. [29] and Kowalczyk [31]. Keerthi et al. combined several known algorithms for NPP to a new one. Kowalczyk derived his own algorithm for the NPP. Both the works show the extension with the kernel functions and using the L_2-soft margin. They do not explicitly deal with the solvers for the singleclass SVM problem.

The rest of Section 3.3 is organized as follows. Section 3.3.1 describes MNP and NPP and their connection to the SVM learning problems. Two variants of the Kozinec algorithm for MNP and NPP are described in Section 3.3.2. The work of Keerthi et al. is described in Section 3.3.3. The work of Kowalczyk is mentioned in Section 3.3.4.

3.3.1 Nearest point and minimal norm problems

This section describes NPP and MNP and their relation to the SVM learning. The key point is the fact that the optimal separating hyperplane can be constructed from the nearest points of the convex hulls of the training examples represented by two finite vector sets. A similar relation can be stated for the singleclass optimal hyperplane and the minimal norm point from a convex hull. This can be easily understood from a geometrical interpretation, but exact proofs exist as well. The proof of equivalence between the MNP and the singleclass optimal hyperplane can be found in [44]. The proof for the binary case is in [31]. A new simpler proof for both the cases is also given in Section 4.9.1.

The NPP problem is defined as follows. Let $\mathcal{T}_{\mathcal{XY}} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_1)\}$ be a finite set of vectors in $\mathbb{R}^n$ which are endowed with the binary labels $\{+1, -1\}$. The input vectors are assumed to be divided into the positive and negative class according to the labels. Let the set $\mathcal{I}_+ = \{y_i : y_i = +1\}$ contain indices of the vectors from the positive class and the set $\mathcal{I}_- = \{y_i : y_i = -1\}$ the indices of the vectors from the the negative class, respectively. Let further $\tilde{\mathcal{X}}_+ = \{\boldsymbol{x}_i : i \in \mathcal{I}_+\}$ and $\tilde{\mathcal{X}}_- = \{\boldsymbol{x}_i : i \in \mathcal{I}_-\}$ be introduced for convenience of notation. The NPP is defined as the following optimization task

$$(\boldsymbol{w}_+^*, \boldsymbol{w}_-^*) = \operatorname*{argmin}_{\boldsymbol{w}_+ \in \overline{\mathcal{X}}_+, \boldsymbol{w}_- \in \overline{\mathcal{X}}_-} \|\boldsymbol{w}_+ - \boldsymbol{w}_-\|^2 \,, \tag{3.5}$$

where $\overline{\mathcal{X}}_+$ and $\overline{\mathcal{X}}_-$ are the convex hulls of the vectors of the positive and the negative class, respectively. The convex hulls are defined as

$$\begin{aligned} \overline{\mathcal{X}}_+ &= \left\{ \boldsymbol{x} \in \mathbb{R}^n \colon \boldsymbol{x} = \sum_{i \in \mathcal{I}_+} \alpha_i \boldsymbol{x}_i \,, \sum_{i \in \mathcal{I}_+} \alpha_i = 1 \,, \alpha_i \geq 0 \,, i \in \mathcal{I}_+ \right\} \,, \\ \overline{\mathcal{X}}_- &= \left\{ \boldsymbol{x} \in \mathbb{R}^n \colon \boldsymbol{x} = \sum_{i \in \mathcal{I}_-} \alpha_i \boldsymbol{x}_i \,, \sum_{i \in \mathcal{I}_-} \alpha_i = 1 \,, \alpha_i \geq 0 \,, i \in \mathcal{I}_- \right\} \,. \end{aligned}$$

The vectors $\boldsymbol{w}_+^*$ and $\boldsymbol{w}_-^*$ are the nearest vectors from the two convex hulls $\overline{\mathcal{X}}_+$ and $\overline{\mathcal{X}}_-$. The optimal separating hyperplane $\langle \boldsymbol{w}^*, \boldsymbol{x} \rangle + b^* = 0$ can be constructed from the vectors $\boldsymbol{w}_+^*$ and $\boldsymbol{w}_-^*$ such that

$$\boldsymbol{w}^* = \boldsymbol{w}_+^* - \boldsymbol{w}_-^* \quad \text{and} \quad b^* = \frac{1}{2} \left(\|\boldsymbol{w}_-^*\|^2 - \|\boldsymbol{w}_+^*\|^2 \right) \,. \tag{3.6}$$

More precisely, the hyperplane $\langle \boldsymbol{w}^*, \boldsymbol{x} \rangle + b^* = 0$ given by (3.6) is the optimal one, but it is not in the canonical form $\langle \boldsymbol{w}_c^*, \boldsymbol{x} \rangle + b_c^* = 0$ as defined by (1.16) (the canonical form means that $y(\langle \boldsymbol{w}_c^*, \boldsymbol{x} \rangle + b_c^*) = 1$ for the closest vectors $\boldsymbol{x}$ with label y from the training set). These two hyperplanes differ only in a scale factor. It is straightforward to show that

$$\boldsymbol{w}_c^* = 2 \frac{\boldsymbol{w}^*}{\langle \boldsymbol{w}^*, \boldsymbol{w}^* \rangle} \quad \text{and} \quad b_c^* = 2 \frac{b^*}{\langle \boldsymbol{w}^*, \boldsymbol{w}^* \rangle} \,.$$

A similar relation can be established between the singleclass optimal hyperplane and the MNP. The singleclass optimal hyperplane $\langle \boldsymbol{w}_c^*, \boldsymbol{x} \rangle = 0$ defined by (1.24) can be computed from the minimal norm point of the convex hull of the training vectors. Let $\tilde{\mathcal{X}} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ be the set of vectors in $\mathbb{R}^n$ and $\mathcal{I} = \{1, \ldots, m\}$ be the set of indices. The MNP is defined as

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w} \in \overline{\mathcal{X}}} \|\boldsymbol{w}\|^2 \,, \tag{3.7}$$

where

$$\overline{\mathcal{X}} = \left\{ \boldsymbol{x} \in \mathbb{R}^n \colon \boldsymbol{x} = \sum_{i \in \mathcal{I}} \alpha_i \boldsymbol{x}_i \,, \sum_{i \in \mathcal{I}} \alpha_i = 1 \,, \alpha_i \geq 0 \,, i \in \mathcal{I} \right\} \,,$$

is the convex hull of $\tilde{\mathcal{X}}$. The vector $\boldsymbol{w}^*$ given by (3.7) defines the optimal singleclass hyperplane but its norm differs from the norm of the canonical solution $\boldsymbol{w}_c^*$ defined by (1.24). These vectors are related by

$$\boldsymbol{w}_c = \frac{\boldsymbol{w}^*}{\langle \boldsymbol{w}^*, \boldsymbol{w}^* \rangle} \,.$$

It is easy to see that both the NPP and MNP are special instances of the QP task. Namely, the NPP and MNP can be equivalently expressed as the task

$$\boldsymbol{\alpha}^* = \operatorname*{argmin}_{\boldsymbol{\alpha} \in \mathcal{A}} \langle \boldsymbol{\alpha}, \mathrm{H} \boldsymbol{\alpha} \rangle \,,$$

where the matrix $\mathrm{H} \in \mathbb{R}^{m \times m}$ is symmetric positive definite and $\mathcal{A}$ stands for the feasible set. In the case of MNP, the feasible set $\mathcal{A}$ and matrix H are defined as

$$\mathcal{A} = \left\{ \boldsymbol{\alpha} \in \mathbb{R}^m \colon \sum_{i \in \mathcal{I}} \alpha_i = 1 \,, \alpha_i \geq 0 \right\} \quad \text{and} \quad [\mathrm{H}]_{i,j} = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \,, i \in \mathcal{I} \,, j \in \mathcal{I} \,.$$

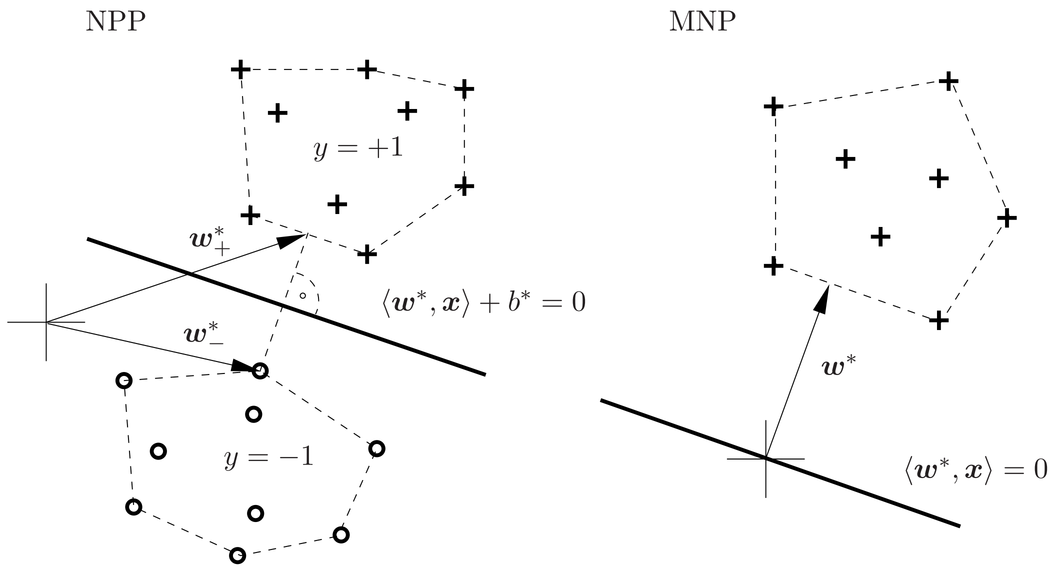

NPP
MNP
$y=+1$
$y=-1$
$\boldsymbol{w}_+^*$
$\boldsymbol{w}_-^*$
$\langle \boldsymbol{w}^*, \boldsymbol{x}\rangle + b^* = 0$
$\boldsymbol{w}^*$
$\langle \boldsymbol{w}^*, \boldsymbol{x}\rangle = 0$

Figure 3.1: Illustration of the nearest point problem (NPP) and the minimal norm problem. The linear classifiers constructed from the NPP and MNP are denote by solid thick line.

In the case of NPP

$$\mathcal{A} = \left\{ \boldsymbol{\alpha} \in \mathbb{R}^m \colon \sum_{i \in \mathcal{I}_+} \alpha_i = \sum_{i \in \mathcal{I}_-} \alpha_i = 1 \,, \alpha_i \geq 0 \,, i \in \mathcal{I}_+ \cup \mathcal{I}_- \right\} ,$$

and

$$[\mathrm{H}]_{i,j} = \left\{ \begin{array}{rll} \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle & \text{if} & y_i = y_j \,, \\ -\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle & \text{if} & y_i \neq y_j \,, \end{array} \right. \qquad i,j \in \mathcal{I}_+ \cup \mathcal{I}_- \,.$$

The NPP and MNP in relation to the linear SVM classifiers are illustrated in Figure 3.1.

Iterative algorithms which find an approximate solution of the NPP and the MNP tasks are described below. The aim is to give an idea how the algorithms work whereas their detailed analysis is a subject of Chapter 4.

3.3.2 Kozinec algorithm

This section adopts results mainly from the book by Schlesinger and Hlaváč [44]. It describes the connection between the MNP, NPP and the optimal separating hyperplanes. The book contains a proof which shows that the singleclass optimal hyperplane can be determined from the solution of the MNP. Several variants of the Kozinec algorithm [32] solving the MNP and the NPP are analyzed as well. However, only the linear case and separable training examples are assumed.

The Kozinec Algorithm 5 for MNP is an iterative procedure which aims to approximate the minimal norm point $\boldsymbol{w}^*$ defined by (3.7). The algorithm starts to iterate from an arbitrary vector $\boldsymbol{w}^{(0)}$ which belongs to $\overline{\mathcal{X}}$. In each step, the vector $\boldsymbol{x}^{(t)}$ is found having the minimal distance from the hyperplane $\langle \boldsymbol{w}^{(t)}, \boldsymbol{x} \rangle = 0$. A new solution vector $\boldsymbol{w}^{(t+1)}$ is found as the minimal norm vector which lies on the line segment between $\boldsymbol{x}^{(t)}$ and $\boldsymbol{w}^{(t)}$. The computation of the vector with the minimal norm lying on the line segment has an analytical solution given by (3.10) below. It is easy to show that the norm of the vector $\boldsymbol{w}^{(t)}$ monotonically decreases. This procedure builds a sequence $\boldsymbol{w}^{(0)}, \boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(t)}$, which converges to the optimal solution $\boldsymbol{w}^*$. The algorithm works until it gets sufficiently close to the optimal solution which is indicated by the stopping condition (3.9) below. The satisfaction of the condition (3.9) implies that the inequality

$$\underbrace{\min_{\boldsymbol{x}\in\tilde{\mathcal{X}}} \frac{\langle \boldsymbol{w}^*, \boldsymbol{x}\rangle}{\|\boldsymbol{w}^*\|}}_{\rho(\boldsymbol{w}^*)} - \underbrace{\min_{\boldsymbol{x}\in\tilde{\mathcal{X}}} \frac{\langle \boldsymbol{w}, \boldsymbol{x}\rangle}{\|\boldsymbol{w}\|}}_{\rho(\boldsymbol{w})} \le \varepsilon \,, \tag{3.8}$$

holds. This means that the margin $\rho(\boldsymbol{w})$ (c.f. Section 1.4.3) of the found hyperplane differs from the optimal margin $\rho(\boldsymbol{w}^*)$ at most by a prescribed ε. Any hyperplane $\langle \boldsymbol{w}, \boldsymbol{x}\rangle = 0$ given by $\boldsymbol{w}$ which satisfies condition (3.8) is named the ε-optimal hyperplane [44].

Algorithm 5: Kozinec Algorithm for MNP

1. Initialization. Set $\boldsymbol{w}^{(0)}$ to any vector from $\overline{\mathcal{X}}$.

2. Check the stopping condition

$$\|\boldsymbol{w}^{(t)}\| - \frac{\langle \boldsymbol{w}^{(t)}, \boldsymbol{x}^{(t)}\rangle}{\|\boldsymbol{w}^{(t)}\|} \le \varepsilon \,, \quad \text{where} \quad \boldsymbol{x}^{(t)} = \operatorname*{argmin}_{\boldsymbol{x}\in\tilde{\mathcal{X}}} \langle \boldsymbol{x}, \boldsymbol{w}^{(t)}\rangle \,. \tag{3.9}$$

   If it holds then exit the algorithm with the solution $\boldsymbol{w}^{(t)}$. Otherwise continue to Step 3.

3. Update

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)}(1-\tau) + \tau \boldsymbol{x}^{(t)} \,, \quad \text{where} \quad \tau = \min\left(1, \frac{\langle \boldsymbol{w}^{(t)}, \boldsymbol{w}^{(t)} - \boldsymbol{x}^{(t)}\rangle}{\|\boldsymbol{w}^{(t)} - \boldsymbol{x}^{(t)}\|^2}\right) \,, \tag{3.10}$$

   and continue to Step 2.

The Kozinec Algorithm 6 can be also used to solve the NPP in the same fashion as the MNP. In this case, however, two vectors $\boldsymbol{w}_+^{(t)} \in \overline{\mathcal{X}}_+$ and $\boldsymbol{w}_-^{(t)} \in \overline{\mathcal{X}}_-$ are sought simultaneously. In each step, the point $\boldsymbol{x}^{(t)}$ closest to the hyperplane

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}, \boldsymbol{x}\rangle + \frac{1}{2}\left(\|\boldsymbol{w}_-^{(t)}\|^2 - \|\boldsymbol{w}_+^{(t)}\|^2\right) = 0 \,,$$

is used for an update. If the vector $\boldsymbol{x}^{(t)}$ is from the set $\tilde{\mathcal{X}}_+$ then the new vector $\boldsymbol{w}_+^{(t+1)}$ is computed as point on the line segment between vectors $\boldsymbol{w}_+^{(t)}$ and $\boldsymbol{x}^{(t)}$ which is the closest to the $\boldsymbol{w}_-^{(t)}$. The new vector $\boldsymbol{w}_-^{(t+1)}$ is set to $\boldsymbol{w}_-^{(t)}$. In the case in which $\boldsymbol{x}^{(t)} \in \tilde{\mathcal{X}}_-$ it is proceeded vice versa, i.e., the $\boldsymbol{w}_+^{(t)}$ is not changed and $\boldsymbol{w}_-^{(t)}$ is updated. It can be shown again that the norm of the vector $\boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}$ monotonically decreases and the sequence

$(\boldsymbol{w}_+^{(0)}, \boldsymbol{w}_-^{(0)})$, $(\boldsymbol{w}_+^{(1)}, \boldsymbol{w}_-^{(1)})$,..., $(\boldsymbol{w}_+^{(t)}, \boldsymbol{w}_-^{(t)})$ converges to the optimal vectors $(\boldsymbol{w}_+^*, \boldsymbol{w}_-^*)$. The algorithm works until the stopping condition

$$\|\boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}\| - \min_{\boldsymbol{x} \in \tilde{\mathcal{X}}_+ \cup \tilde{\mathcal{X}}_-} g(\boldsymbol{x}) \leq \frac{\varepsilon}{2}\,, \tag{3.11}$$

is satisfied, where $g(\boldsymbol{x})$ is given by (3.12). The satisfaction of the stopping condition (3.11) implies that the margin $\rho(\boldsymbol{w}, b)$ of the found hyperplane differs from the optimal one $\rho(\boldsymbol{w}^*, b^*)$ by prescribed ε at most.

Algorithm 6: Kozinec Algorithm for NPP

1. Initialization. Set $\boldsymbol{w}_+^{(0)}$ to any vector from $\overline{\mathcal{X}}_+$ and $\boldsymbol{w}_-^{(0)}$ to any vector from $\overline{\mathcal{X}}_-$.

2. Find the vector $\boldsymbol{x}^{(t)}$ closest to the current hyperplane such that $\boldsymbol{x}^{(t)} = \operatorname*{argmin}_{\boldsymbol{x} \in \tilde{\mathcal{X}}_+ \cup \tilde{\mathcal{X}}_-} g(\boldsymbol{x})$ and

$$g(\boldsymbol{x}) = \begin{cases} \dfrac{\langle \boldsymbol{x} - \boldsymbol{w}_-^{(t)}, \boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)} \rangle}{\|\boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}\|} & \text{for} \quad \boldsymbol{x} \in \tilde{\mathcal{X}}_+\,, \\ \dfrac{\langle \boldsymbol{x} - \boldsymbol{w}_+^{(t)}, \boldsymbol{w}_-^{(t)} - \boldsymbol{w}_+^{(t)} \rangle}{\|\boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}\|} & \text{for} \quad \boldsymbol{x} \in \tilde{\mathcal{X}}_-\,. \end{cases} \tag{3.12}$$

   If the stopping condition $\|\boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}\| - g(\boldsymbol{x}^{(t)}) \leq \frac{\varepsilon}{2}$ holds then exit the algorithm with the solution $\boldsymbol{w} = \boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}$ and $b = \frac{1}{2}(\|\boldsymbol{w}_-^{(t)}\|^2 - \|\boldsymbol{w}_+^{(t)}\|^2)$. Otherwise go to Step 3.

3. If $\boldsymbol{x}^{(t)} \in \tilde{\mathcal{X}}_+$ then set $\boldsymbol{w}_-^{(t+1)} = \boldsymbol{w}_-^{(t)}$ and update

$$\boldsymbol{w}_+^{(t+1)} = \boldsymbol{w}_+^{(t)}(1-\tau) + \tau \boldsymbol{x}^{(t)}\,, \quad \text{where} \quad \tau = \min\left(1, \frac{\langle \boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}, \boldsymbol{w}_+^{(t)} - \boldsymbol{x}^{(t)} \rangle}{\|\boldsymbol{w}_+^{(t)} - \boldsymbol{x}^{(t)}\|^2}\right)\,.$$

   Otherwise if $\boldsymbol{x}^{(t)} \in \tilde{\mathcal{X}}_-$ then set $\boldsymbol{w}_+^{(t+1)} = \boldsymbol{w}_+^{(t)}$ and update

$$\boldsymbol{w}_-^{(t+1)} = \boldsymbol{w}_-^{(t)}(1-\tau) + \tau \boldsymbol{x}^{(t)}\,, \quad \text{where} \quad \tau = \min\left(1, \frac{\langle \boldsymbol{w}_-^{(t)} - \boldsymbol{w}_+^{(t)}, \boldsymbol{w}_-^{(t)} - \boldsymbol{x}^{(t)} \rangle}{\|\boldsymbol{w}_-^{(t)} - \boldsymbol{x}^{(t)}\|^2}\right)\,.$$

   Continue to Step 2.

3.3.3 Keerthi algorithm

Keerthi et al. [29] proposed to use the NPP algorithms to learn the SVM classifier with L_2-soft margin. They used the relation between the hard margin and L_2-soft margin formulation. This relation was described in Section 1.4.1. Keerthi et al. also employed the kernel functions to learn the non-linear SVM classifiers because they showed that all computations with data can be expressed in terms of dot products. The Keerthi algorithm[1] combines ideas of two methods proposed earlier to solve the NPP. Namely, it is based on the algorithms proposed by Gilbert [20] and Mitchell et al. [36]. Gilbert algorithm is essentially the same as the Kozinec algorithm described in Section 3.3.2. The algorithm by Mitchell et al. and the Keerthi algorithm, which combines both methods, are described below.

The Mitchell-Demyanov-Malozemov (MDM) Algorithm 7 is designed to solve the MNP. The MDM algorithm uses inherently the representation of the solution as a convex combination of the input vectors, i.e.,

$$\boldsymbol{w} = \sum_{i \in \mathcal{I}} \alpha_i \boldsymbol{x}_i \ .$$

The weight vector $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_m]^T \in \mathbb{R}^m$ can attain the value from $\mathcal{A} = \{\boldsymbol{\alpha} \in \mathbb{R}^m \colon \sum_{i \in \mathcal{I}} \alpha_i = 1, \alpha_i \geq 0 \ , i \in \mathcal{I}\}$ to guarantee that the vector $\boldsymbol{w}$ lies in the convex hull $\overline{\mathcal{X}}$. Let $\boldsymbol{w}^* = \sum_{i \in \mathcal{I}} \alpha_i^* \boldsymbol{x}_i$ be the optimal solution given by (3.7). It can be shown based on the KKT conditions that the equality $\langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle = \langle \boldsymbol{w}^*, \boldsymbol{w}^* \rangle$ holds true for all support vectors $\boldsymbol{x}_i \in \{\boldsymbol{x}_j \colon \alpha_j^* > 0\}$. For any other $\boldsymbol{w} = \sum_{i \in \mathcal{I}} \alpha_i \boldsymbol{x}_i \neq \boldsymbol{w}^*$, may it be very close to the optimal $\boldsymbol{w}^*$, there could be vectors $\boldsymbol{x}_i$ with corresponding weights $\alpha_i > 0$ such that

$$\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \gg \langle \boldsymbol{w}, \boldsymbol{w} \rangle \ .$$

It is important both for the efficient (sparse) representation and performance of the algorithm to exclude such vectors from the representation of $\boldsymbol{w}$. It can be further shown that the inequality

$$\kappa(\boldsymbol{w}) = \max_{i \in \{j \in \mathcal{I} \colon \alpha_j > 0\}} \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - \min_{i \in \mathcal{I}} \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq 0 \ ,$$

holds true for all $\boldsymbol{w}$ and the equality occurs only for the optimal $\boldsymbol{w}^*$. The MDM algorithm aims to decrease the quantity $\kappa(\boldsymbol{w})$ in each iteration. The MDM algorithm builds a sequence of vectors $\boldsymbol{w}^{(0)}$, $\boldsymbol{w}^{(1)}$, …, $\boldsymbol{w}^{(t)}$, which converges to the optimal $\boldsymbol{w}^*$. Let indices u and v be defined by (3.13). Then $\kappa(\boldsymbol{w}^{(t)}) = \langle \boldsymbol{w}^{(t)}, \boldsymbol{x}_v \rangle - \langle \boldsymbol{w}^{(t)}, \boldsymbol{x}_u \rangle$. The update rule decreases the weight $[\boldsymbol{\alpha}^{(t)}]_v$ and, at the same time, increases the weight $[\boldsymbol{\alpha}^{(t)}]_u$. It is easy to show that the MDM algorithm monotonically decreases the norm of the vector $\boldsymbol{w}^{(t)}$.

Algorithm 7: MDM Algorithm for MNP

1. Initialization. Select any $\boldsymbol{\alpha}^{(0)}$ such that $\sum_{i \in \mathcal{I}} [\boldsymbol{\alpha}^{(0)}]_i = 1$, $[\boldsymbol{\alpha}^{(0)}]_i \geq 0$, $i \in \mathcal{I}$.

2. Repeat until stopping condition is satisfied.

   a) Find indices

$$u \in \operatorname*{argmin}_{i \in \mathcal{I}} \langle \boldsymbol{w}^{(t)}, \boldsymbol{x}_i \rangle \ , \quad \text{and} \quad v \in \operatorname*{argmax}_{i \in \{j \in \mathcal{I} \colon [\boldsymbol{\alpha}^{(t)}]_j > 0\}} \langle \boldsymbol{w}^{(t)}, \boldsymbol{x}_i \rangle \ , \qquad (3.13)$$

[1]The authors named their method the "nearest point algorithm (NPA)". However, all the algorithms of this sections solve NPP and thus they deserve the same name. Due to this reason the name Keerthi algorithm is used here to distinguish this specific approach described in paper [29].

where $\boldsymbol{w}^{(t)} = \sum\limits_{i\in\mathcal{I}} \boldsymbol{x}_i[\boldsymbol{\alpha}^{(t)}]_i$. Then create vector

$$[\boldsymbol{\beta}^{(t)}]_i = \begin{cases} [\boldsymbol{\alpha}^{(t)}]_u + [\boldsymbol{\alpha}^{(t)}]_v & \text{if} \quad i = u\,, \\ 0 & \text{if} \quad i = v\,, \\ [\boldsymbol{\alpha}^{(t)}]_i & \text{otherwise}\,. \end{cases}$$

b) Update $\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1-\tau) + \boldsymbol{\beta}^{(t)}\tau$, where

$$\tau = \min\left(1, \frac{\langle \boldsymbol{w}^{(t)}, \boldsymbol{w}^{(t)} - \boldsymbol{x}^{(t)}\rangle}{\|\boldsymbol{w}^{(t)} - \boldsymbol{x}^{(t)}\|^2}\right)\,,$$

where $\boldsymbol{x}^{(t)} = \sum\limits_{i\in\mathcal{I}} [\boldsymbol{\beta}^{(t)}]_i \boldsymbol{x}_i$.

Both the Gilbert algorithm (alias Kozinec algorithm) and MDM algorithm can be seen as follows. They approximate the convex hull $\overline{\mathcal{X}}$ by a line segment between the current solution $\boldsymbol{w}^{(t)} \in \overline{\mathcal{X}}$ and the vector $\boldsymbol{x}^{(t)} \in \overline{\mathcal{X}}$ which is selected by different rules. A new solution vector $\boldsymbol{w}^{(t+1)}$ is computed as the minimal norm vector over the line segment which approximates the whole convex hull. The advantage of this approach is that the solution of the minimal norm problem for the line segment has an analytical form. Another advantage is that the selection of the vector $\boldsymbol{x}^{(t)}$ which guarantees decrease of the norm $\boldsymbol{w}^{(t)}$ is simple (see rules of the Kozinec and MDM algorithm).

Keerthi et al. proposed to extend this idea so that a triangle is used to approximate the convex hull instead of the line segment. Finding the minimal norm vector over the triangle has also an analytical solution though a little bit more complex. A question arises how to construct the three vertices of the triangle which approximates the convex hull well. Keerthi et al. tried many combinations and they proposed to use the triangle $(\boldsymbol{w}^{(t)}, \boldsymbol{x}^{(t)}, \boldsymbol{z}^{(t)})$ where $\boldsymbol{x}^{(t)}$ is constructed by the rule of Gilbert (Kozinec) algorithm and $\boldsymbol{z}^{(t)}$ by the rule of MDM algorithm. Algorithm 8 describes the version for the MNP. The analytical form of the subtask (3.14) is omitted as it will be analyzed in details in Chapter 4.

Algorithm 8: Keerthi Algorithm For MNP

1. Initialization. Select any $\boldsymbol{\alpha}^{(0)}$ such that $\sum\limits_{i\in\mathcal{I}} [\boldsymbol{\alpha}^{(0)}]_i = 1$, $[\boldsymbol{\alpha}^{(0)}]_i \geq 0$, $i \in \mathcal{I}$ holds.

2. Repeat until stopping condition is satisfied.

   a) Find indices

$$u \in \operatorname*{argmin}_{\boldsymbol{x}\in\tilde{\mathcal{X}}} \langle \boldsymbol{w}^{(t)}, \boldsymbol{x}\rangle\,, \quad \text{and} \quad v \in \operatorname*{argmax}_{\boldsymbol{x}\in\{\boldsymbol{x}_i : [\boldsymbol{\alpha}^{(t)}]_i > 0\}} \langle \boldsymbol{w}^{(t)}, \boldsymbol{x}\rangle\,,$$

   where $\boldsymbol{w}^{(t)} = \sum\limits_{i\in\mathcal{I}} \boldsymbol{x}_i[\boldsymbol{\alpha}^{(t)}]_i$. Create vectors

$$[\boldsymbol{\beta}]_i = \begin{cases} 1 & \text{if} \quad i = u\,, \\ 0 & \text{otherwise}\,, \end{cases} \qquad [\boldsymbol{\gamma}]_i = \begin{cases} [\boldsymbol{\alpha}^{(t)}]_u + [\boldsymbol{\alpha}^{(t)}]_v & \text{if} \quad i = u\,, \\ 0 & \text{if} \quad i = v\,, \\ [\boldsymbol{\alpha}^{(t)}]_i & \text{otherwise.} \end{cases}$$

b) Update $\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1-\tau-\omega) + \boldsymbol{\beta}^{(t)}\tau + \omega\boldsymbol{\gamma}^{(t)}$ such that

$$(\tau,\omega) = \operatorname*{argmin}_{\tau'\in[0,1],\omega'\in[0,1]} \left\| \boldsymbol{w}^{(t)}(1-\tau'-\omega') + \boldsymbol{x}^{(t)}\tau' + \boldsymbol{z}^{(t)}\omega' \right\| , \tag{3.14}$$

where $\boldsymbol{x}^{(t)} = \sum\limits_{i\in\mathcal{I}} [\boldsymbol{\beta}^{(t)}]_i \boldsymbol{x}_i$ and $\boldsymbol{z}^{(t)} = \sum\limits_{i\in\mathcal{I}} [\boldsymbol{\gamma}^{(t)}]_i \boldsymbol{x}_i$.

The Keerthi algorithm for the NPP is derived in the same fashion as the Kozinec Algorithm 6. It means that two vectors $\boldsymbol{w}_+^{(t)}$ and $\boldsymbol{w}_-^{(t)}$ are being updated. In each iteration, one of these vectors is fixed and the second is updated. The algorithm iterates until the vectors $\boldsymbol{w}_+^{(t)}$ and $\boldsymbol{w}_-^{(t)}$ get sufficiently close to the optimal solution $\boldsymbol{w}_+^*$ and $\boldsymbol{w}_-^*$. Keerthi et al. proposed to use the following stopping condition

$$\min_{\boldsymbol{x}\in\tilde{\mathcal{X}}_+} \langle \boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}, \boldsymbol{x}\rangle + \min_{\boldsymbol{x}\in\tilde{\mathcal{X}}_-} \langle \boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}, \boldsymbol{x}\rangle \geq (1-\overline{\varepsilon}) \|\boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}\|^2 . \tag{3.15}$$

The satisfaction of the stopping condition (3.15) implies that the inequality $\rho(\boldsymbol{w}^*, b^*) \leq \rho(\boldsymbol{w}, b)(1-\overline{\varepsilon})$ holds, where $\rho(\boldsymbol{w}^*, b^*)$ is the optimal margin, $\rho(\boldsymbol{w}, b)$ is the margin of the found hyperplane and $\overline{\varepsilon}$ is a prescribed constant.

3.3.4 Kowalczyk algorithm

Independently, Kowalczyk [31] proposed to learn the SVM binary classifier with the L_2-soft margin using the NPP algorithm. However, he derived a different algorithm to solve the NPP which is described below[2]. The Kowalczyk Algorithm 9 for NPP uses a line approximation of the convex hull. The line is constructed between the current solution $\boldsymbol{w}^{(t)} = \boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}$ and a vector $\boldsymbol{x}^{(t)}$ which is selected out of candidates $\boldsymbol{x}_i^{(t)}$, $i \in \mathcal{I}_+ \cup \mathcal{I}_-$. The candidates are constructed by two rules (3.16) and (3.17). The vector $\boldsymbol{x}^{(t)}$ is selected out of the candidates such that the optimization over the line segment between the current solution $\boldsymbol{w}^{(t)}$ and $\boldsymbol{x}^{(t)}$ yields the biggest decrease of $\|\boldsymbol{w}_-^{(t)} - \boldsymbol{w}_+^{(t)}\|$. It can be simply shown that the algorithm monotonically decreases the norm of the vector $\boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}$ and the sequence $(\boldsymbol{w}_+^{(0)}, \boldsymbol{w}_-^{(0)})$, $(\boldsymbol{w}_+^{(1)}, \boldsymbol{w}_-^{(1)})$, …, $(\boldsymbol{w}_+^{(t)}, \boldsymbol{w}_-^{(t)})$ converges to the optimal solution $(\boldsymbol{w}_+^*, \boldsymbol{w}_-^*)$. The algorithm works until it gets sufficiently close to the optimal solution which is indicated by the stopping conditions. The stopping condition used by Kowalczyk is essentially the same as the condition (3.15) used by the Keerthi algorithm.

Algorithm 9: Kowalczyk Algorithm For NPP

1. Initialization. Select any $\boldsymbol{\alpha}^{(0)}$ such that $\sum\limits_{i\in\mathcal{I}_+} [\boldsymbol{\alpha}^{(0)}]_i = 1$, $\sum\limits_{i\in\mathcal{I}_-} [\boldsymbol{\alpha}^{(0)}]_i = 1$ and $[\boldsymbol{\alpha}^{(0)}]_i \geq 0$, $i \in \mathcal{I}_+ \cup \mathcal{I}_-$.

2. Repeat until stopping condition is satisfied.

[2]Kowalczyk originally named his algorithm the maximal margin Perceptron. The name Kowalczyk algorithm is used here instead to have consistent notation which allows to clearly distinguish the algorithm according to the name of the author.

a) Construct candidate vectors $\boldsymbol{\beta}_i^{(t)}$, $i \in \mathcal{I}_+ \cup \mathcal{I}_-$. The vectors $\boldsymbol{\beta}_i^{(t)}$, $i \in \mathcal{I}_+$, are constructed (i) by the rule

$$[\boldsymbol{\beta}_i^{(t)}]_j = \begin{cases} 1 & \text{if} \quad j = i \wedge j \in \mathcal{I}_+ \,, \\ 0 & \text{if} \quad j \neq i \wedge j \in \mathcal{I}_+ \,, \\ [\boldsymbol{\alpha}^{(t)}]_j & \text{if} \quad j \in \mathcal{I}_- \,, \end{cases} \tag{3.16}$$

if $\langle \boldsymbol{x}_i, \boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)} \rangle < \langle \boldsymbol{w}_+^{(t)}, \boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)} \rangle$ or (ii) by the rule

$$[\boldsymbol{\beta}_i^{(t)}]_j = \begin{cases} 0 & \text{if} \quad j = i \wedge j \in \mathcal{I}_+ \,, \\ \frac{[\boldsymbol{\alpha}^{(t)}]_j}{1-[\boldsymbol{\alpha}^{(t)}]_i} & \text{if} \quad j \neq i \wedge j \in \mathcal{I}_+ \,, \\ [\boldsymbol{\alpha}^{(t)}]_j & \text{if} \quad j \in \mathcal{I}_- \,, \end{cases} \tag{3.17}$$

if $\langle \boldsymbol{x}_i, \boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)} \rangle > \langle \boldsymbol{w}_+^{(t)}, \boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)} \rangle$ and $[\boldsymbol{\alpha}^{(t)}]_i < 1$. Otherwise $\Delta_i^{(t)} = 0$. The vectors $\boldsymbol{\beta}_i^{(t)}$, $i \in \mathcal{I}_-$, are constructed by the same rules except for the sets $\mathcal{I}_+$ and $\mathcal{I}_-$ interchanged. The "if conditions" are changed to $\langle \boldsymbol{x}_i, \boldsymbol{w}_-^{(t)} - \boldsymbol{w}_+^{(t)} \rangle \gtrless \langle \boldsymbol{w}_-^{(t)}, \boldsymbol{w}_-^{(t)} - \boldsymbol{w}_+^{(t)} \rangle$.
For all candidate vectors compute possible improvement

$$\Delta_i^{(t)} = \|\boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)}\|^2 - \min_{\tau \in [0,1]} \|(\boldsymbol{w}_+^{(t)} - \boldsymbol{w}_-^{(t)})(1-\tau) + \gamma \boldsymbol{x}_i^{(t)}\|^2 \,, \tag{3.18}$$

where $\boldsymbol{w}_+^{(t)} = \sum\limits_{j \in \mathcal{I}_+} [\boldsymbol{\alpha}^{(t)}]_j \boldsymbol{x}_j$, $\boldsymbol{w}_-^{(t)} = \sum\limits_{j \in \mathcal{I}_-} [\boldsymbol{\alpha}^{(t)}]_j \boldsymbol{x}_j$, $\boldsymbol{x}_i^{(t)} = \sum\limits_{j \in \mathcal{I}_+} [\boldsymbol{\beta}_i^{(t)}]_j \boldsymbol{x}_j - \sum\limits_{j \in \mathcal{I}_-} [\boldsymbol{\beta}_i^{(t)}]_j \boldsymbol{x}_j$.
Finally set

$$r \in \operatorname*{argmax}_{i \in \mathcal{I}_+ \cup \mathcal{I}_-} \Delta_i^{(t)} \,.$$

b) Update

$$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1-\tau) + \tau \boldsymbol{\beta}_r^{(t)} \,,$$

where

$$\tau = \min\left(1, \frac{\langle \boldsymbol{w}^{(t)}, \boldsymbol{w}^{(t)} - \boldsymbol{x}_r^{(t)} \rangle}{\|\boldsymbol{w}^{(t)} - \boldsymbol{x}_r^{(t)}\|^2}\right) \,.$$

3.4 Sparse matrix approximation

This section outlines the idea of sparse greedy matrix approximation which was proposed by Smola and Schölkopf [46, 48]. Let $\mathrm{K} \in \mathbb{R}^{m \times m}$ be a kernel matrix which represents the training examples $\{x_1, \ldots, x_m\}$ in the RKHS given by a selected kernel function $k \colon \mathcal{X} \times \mathcal{X} \to \mathcal{H}$, i.e., $[\mathrm{K}]_{i,j} = k(x_i, x_j)$. The size of the kernel matrix K scales quadratically with the number m of training examples. A large kernel matrix can cause problems due to memory and computational requirements. A possible solution to this problem is to find a matrix $\tilde{\mathrm{K}} \in \mathbb{R}^{m \times m}$ which would approximate the original kernel matrix K well. Moreover, the approximated matrix $\tilde{\mathrm{K}}$ is required to be represented in a more compressed form than the original K. The computation of the approximated kernel matrix was posed by Smola and Schölkopf as the following optimization task

$$\mathcal{J}^* = \operatorname*{argmin}_{\mathcal{J} \subset \mathcal{I}} \varepsilon_K(\mathcal{J}) = \operatorname*{argmin}_{\mathcal{J} \subset \mathcal{I}} \|\mathrm{K} - \tilde{\mathrm{K}}\|_F^2 \,, \tag{3.19}$$

where the approximated matrix $\tilde{\mathrm{K}}$ is defined by

$$[\tilde{\mathrm{K}}]_{:,i} = \sum_{j \in \mathcal{J}} [\mathrm{T}]_{j,i} [\mathrm{K}]_{:,j} \,.$$

The symbol $[\tilde{\mathrm{K}}]_{:,i}$ is the i-th column vector of the matrix K, $\| \cdot \|_F^2$ is Frobenius matrix norm, $\mathrm{T} \in \mathbb{R}^{l \times m}$ denotes a matrix of coefficients and $\mathcal{J} \subset \mathcal{I}$ is a subset of indices of all matrix columns $\mathcal{I} = \{1, \ldots, m\}$. The approximation error $\varepsilon_K(\mathcal{J})$ is in fact a function of both the selected columns $\mathcal{J}$ and the coefficient matrix T. However, it can be shown that for fixed set $\mathcal{J}$ the optimal coefficient matrix T can be analytically computed because the problem (3.19) becomes an unconstrained convex QP task. For this reason, T is not included in the definition of the error because it can be uniquely determined by $\mathcal{J}$.

The approximation of the kernel matrix based on the task (3.19) brings two advantages: (i) the learning of kernel based classifiers simplifies as the kernel matrix has more compact representation and (ii) the learned function simplifies as well because it will be represented in a linear span of a smaller number of functions. Therefore there is a hope that the learning and the evaluation stage becomes faster.

The idea is to select a subset $\mathcal{J}$ of columns of the kernel matrix K and express all remaining columns as linear combinations. Such columns $\mathcal{J}$ should be selected that the approximation error $\varepsilon_K(\mathcal{J})$ is minimized. The number l of selected columns $\mathcal{J}$ can be determined explicitly or implicitly by using such l which yields a desired limit on the approximation error. To guarantee the optimal subset $\mathcal{J}^* \subset \mathcal{I}$ which contains l columns selected out of m one has to try $\binom{m}{l}$ possibilities which is an intractable combinatorial problem.

Smola and Schölkopf proposed to use a greedy procedure to select the subset $\mathcal{J}$. The basic idea is to start from the empty set $\mathcal{J} = \{\emptyset\}$ and iteratively add one column vector such that the criterion (3.19) is stepwise decreased. In each iteration, the r-th column is sought which maximally decreases the criterion $\varepsilon_K(\mathcal{J} \cup \{r\})$. However, an implementation of this idea is computationally very expensive as a selection of the best column requires at least $O(m^3)$ operations. Therefore Smola and Schölkopf proposed a probabilistic speed up. A subset $\mathcal{M} \subset \mathcal{I} \setminus \mathcal{J}$ of p column vectors is randomly drawn. The best vector to be added is sought in this subset $\mathcal{M}$, thus only p candidates are checked instead of m. Even if the probabilistic speed up is used, the computational complexity of the algorithm scales with $O(m^2)$ which can be still too expensive. Therefore Smola and Schölkopf also proposed an alternative approach based on an approximation of the training examples represented as functions in the RKHS instead of columns of the matrix K. This approach is described below.

Let $\Phi(x)(\cdot) = k(x, \cdot)$ denote a function from the feature space $\mathcal{H}$ to which the input $x \in \mathcal{X}$ is mapped. The input patterns represented in the feature space $\mathcal{H}$ form a set of functions $\{\Phi(x_1), \ldots, \Phi(x_m)\}$. The aim is to find approximated functions $\{\tilde{\Phi}(x_1), \ldots, \tilde{\Phi}(x_m)\}$ such that the squared error is minimized, i.e., the problem is posed as the following optimization task

$$\mathcal{J}^* = \operatorname*{argmin}_{\mathcal{J} \subset \mathcal{I}} \varepsilon_\Phi(\mathcal{J}) = \operatorname*{argmin}_{\mathcal{J} \subset \mathcal{I}} \sum_{i \in \mathcal{I}} \|\Phi(x_i) - \tilde{\Phi}(x_i)\|^2 \,, \tag{3.20}$$

where approximated functions are determined as

$$\tilde{\Phi}(x_i) = \sum_{j \in \mathcal{J}} [\mathrm{T}]_{j,i} \Phi(x_j) \,.$$

It is important that (3.20) can be expressed in terms of dot products and thus the kernel functions can be employed instead of working with functions $\Phi(x_i)$ explicitly. The optimal coefficient matrix $\mathrm{T} \in \mathbb{R}^{l \times m}$ can be again computed analytically when $\mathcal{J}$ is fixed. It can be simply shown that the later task (3.20) is an approximation of the former task (3.19). More precisely, the trace of the matrix $\mathrm{K} - \tilde{\mathrm{K}}$ equals the objective function $\varepsilon_\Phi(\mathcal{J})$ of task (3.20). The problem is to select the optimal subset $\mathcal{J}^*$ in a reasonable computational time. Smola and Schölkopf proposed the same greedy approach to optimize (3.20). The procedure is outlined in Algorithm (10).

Algorithm 10: Sparse Matrix Approximation

```
1. Initialization. Set $\mathcal{J} = \{\emptyset\}$.
2. Repeat until $\varepsilon_\Phi(\mathcal{J}) \leq \varepsilon$:
    a) Draw random subset $\mathcal{M}$ from $\mathcal{I} \setminus \mathcal{J}$.
    b) Select the index $r = \operatorname*{argmax}_{r \in \mathcal{M}} \varepsilon_\Phi(\mathcal{J} \cup \{r\})$ and update $\mathcal{J} = \mathcal{J} \cup \{r\}$.
```

3.5 Modified multiclass SVM formulation

The problem of learning the multiclass SVM classifier was described in Section 1.4.2. Namely, the decomposition based approaches and the formulation of the multiclass SVM problem were mentioned. The use of decomposition approaches is computationally more effective compared to the multiclass SVM formulation. There was an attempt to derive an approximated multiclass SVM problem which is more convenient for optimization. This approach is described here.

The original multiclass SVM learning problem is expressed as a specific QP task (1.21) and its dual formulation (1.22) which contains data in terms of dot products. There are many QP solvers developed for the binary SVM problem which were mentioned in Section 3.1. The same holds for the singleclass SVM problem which is even simpler than the binary class problem. In contrast, the linear constraints of the multiclass SVM problem (1.22) are considerably more complicated. This fact is the main obstacle to design efficient algorithms which can deal with large problems. For instance, Weston and Watkins [57], who proposed the multiclass SVM formulation performed, its experimental evaluation on small training sets only (the maximal number of training examples was about 500).

Hsu and Lin [25] proposed to modify the objective function (1.21) of the multiclass SVM formulation by adding the term $\frac{1}{2}\sum_{y \in \mathcal{Y}} b_y^2$ (b_y is the bias of the y-th linear discriminant function). This modification causes constraints of the dual task becomes simpler compared to the original one. The modified objective function is

$$(\boldsymbol{w}_y^*, y \in \mathcal{Y}, b_y^*, y \in \mathcal{Y}, \boldsymbol{\xi}) = \operatorname*{argmin}_{\substack{\boldsymbol{w}_y, y\in\mathcal{Y} \\ b_y, y\in\mathcal{Y} \\ \boldsymbol{\xi}}} \left(\frac{1}{2} \sum_{y\in\mathcal{Y}} (\|\boldsymbol{w}_y\|^2 + b_y^2) + C \sum_{i\in\mathcal{I}} \sum_{y\in\mathcal{Y}\setminus y_i} \xi_i^y \right) , \qquad (3.21)$$

while the linear constraints remain the same

$$\begin{array}{rcll} \langle \boldsymbol{w}_{y_i}, \boldsymbol{x}_i \rangle + b_{y_i} - (\langle \boldsymbol{w}_y, \boldsymbol{x}_i \rangle + b_y) & \geq & 1 - \xi_i^y \,, & i \in \mathcal{I}, y \in \mathcal{Y} \setminus y_i \,, \\ \xi_i^y & \geq & 0 \,, & i \in \mathcal{I}, y \in \mathcal{Y} \setminus y_i \,. \end{array}$$

The authors named the modified problem (3.21) as “bounded formulation”, abbreviated BSVM. The dual expression of the primal formulation (3.21) becomes

$$\boldsymbol{\alpha}^* = \operatorname*{argmax}_{\boldsymbol{\alpha}} \left(\sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{I}} \left(\frac{1}{2}\delta(y_i, j) S_i S_j - \sum_{y\in\mathcal{Y}} \alpha_i^y \alpha_j^{y_i} + \frac{1}{2} \sum_{y\in\mathcal{Y}} \alpha_i^y \alpha_j^y \right) (\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + 1) - 2 \sum_{i\in\mathcal{I}} \sum_{y\in\mathcal{Y}} \alpha_i^y \right) , \qquad (3.22)$$

subject to

$$\begin{array}{rcll} 0 & \leq & \alpha_i^y \leq C \,, & i \in \mathcal{I} \,, y \in \mathcal{Y} \,, \\ 0 & = & \alpha_i^{y_i} \,, & i \in \mathcal{I} \,, \\ S_i & = & \sum\limits_{y\in\mathcal{Y}} \alpha_i^y \,, & i \in \mathcal{I} \,. \end{array} \qquad (3.23)$$

Comparing the constrains (1.23) of the original dual multiclass SVM task with the constrains (3.23) of the BSVM dual formulation shows that a set of M linear constraints disappeared in the later case. Hsu and Lin [25] designed a decomposition method able to solve the dual of the BSVM formulation efficiently. The authors also experimentally compared the multiclass BSVM formulation to other approaches.

The addition of the term b^2 to the definition of the binary SVM was previously used in [19, 35] which also simplifies the binary task. A theoretical analysis of a connection between the original and modified binary task is given in [35]. Namely, it was shown that the optimal solutions of the original and the modified task are usually very close.

4 Quadratic Programming solvers

This chapter presents a contribution of the thesis which is related to the problem of solving the Quadratic Programming (QP) task. Namely, the QP solvers designed for the *Generalized Minimal Norm Problem* (GMNP) and the *Generalized Nearest Point Problem* (GNPP) are in question. The GMNP and GNPP arise from the MNP and NPP after adding a linear term into the quadratic criterion and assuming that the Hessian of the criterion is an arbitrary symmetric positive definite matrix. In other words, the original problems are recovered after the linear term is removed and the Hessian equals to the product of two identical matrices. A summary of known algorithms for solving the MNP and the NPP was given in Section 3.3. Even though the algorithms are different it will be shown that they can be described in a common framework. Moreover, the algorithms will be extended to solve the generalized formulations, i.e., the GMNP and the GNPP. A novel and faster algorithm for optimization of the GMNP and the GNPP will be proposed. The performance of algorithms will be compared on synthetical and real problems. The NPP algorithms were originally used for learning of the binary SVM classifier with L_2-soft margin. The generalized formulations, however, match more optimization problems from machine learning. A list of some applications of the proposed QP solvers will be presented.

4.1 Generalized minimal norm and nearest point problems

Let a quadratic objective function

$$Q(\boldsymbol{\alpha}) = \frac{1}{2}\langle\boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha}\rangle + \langle\boldsymbol{c}, \boldsymbol{\alpha}\rangle \,, \tag{4.1}$$

be determined by a vector $\boldsymbol{c} \in \mathbb{R}^m$ and a symmetric positive definite matrix $\mathrm{H} \in \mathbb{R}^{m\times m}$. Let $\mathcal{A} \subseteq \mathbb{R}^m$ be a convex set of feasible solutions $\boldsymbol{\alpha} \in \mathcal{A}$. The goal is to solve the following task

$$\boldsymbol{\alpha}^* = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathcal{A}} \left(\frac{1}{2}\langle\boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha}\rangle + \langle\boldsymbol{c}, \boldsymbol{\alpha}\rangle\right) \,. \tag{4.2}$$

Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be non-empty disjoint sets of indices such that $\mathcal{I}_1 \cup \mathcal{I}_2 = \mathcal{I} = \{1, 2, \dots, m\}$. Let vectors $\boldsymbol{e}, \boldsymbol{e}_1, \boldsymbol{e}_2 \in \mathbb{R}^m$ be defined as follows

$$[\boldsymbol{e}]_i = 1\,, i \in \mathcal{I}\,, \quad [\boldsymbol{e}_1]_i = \left\{\begin{array}{lll} 1 & \text{for} & i \in \mathcal{I}_1 \\ 0 & \text{for} & i \in \mathcal{I}_2 \end{array}\right. \,, \quad [\boldsymbol{e}_2]_i = \left\{\begin{array}{lll} 0 & \text{for} & i \in \mathcal{I}_1 \\ 1 & \text{for} & i \in \mathcal{I}_2 \end{array}\right. \,.$$

The optimization problems (4.2) with two distinct feasible sets $\mathcal{A}$ are assumed. In the first case, the *Generalized Minimal Norm Problem* (GMNP) is the optimization problem (4.2) with the feasible set $\mathcal{A}$ determined by

$$\mathcal{A} = \{\boldsymbol{\alpha} \in \mathbb{R}^m \colon \langle\boldsymbol{\alpha}, \boldsymbol{e}\rangle = 1, \boldsymbol{\alpha} \geq 0\} \,. \tag{4.3}$$

In the second case, the *Generalized Nearest Point Problem* (GNPP) is the optimization problem (4.2) with the feasible set $\mathcal{A}$ determined by

$$\mathcal{A} = \{\boldsymbol{\alpha} \in \mathbb{R}^m \colon \langle\boldsymbol{\alpha}, \boldsymbol{e}_1\rangle = 1, \langle\boldsymbol{\alpha}, \boldsymbol{e}_2\rangle = 1\,, \boldsymbol{\alpha} \geq 0\} \,, \tag{4.4}$$

where vectors $\boldsymbol{e}, \boldsymbol{e}_1, \boldsymbol{e}_2 \in \mathbb{R}^m$ are defined as follows

$$[\boldsymbol{e}]_i = 1\,, i \in \mathcal{I}\,, \quad [\boldsymbol{e}_1]_i = \left\{ \begin{array}{lll} 1 & \text{for} & i \in \mathcal{I}_1 \\ 0 & \text{for} & i \in \mathcal{I}_2 \end{array} \right. \,, \quad [\boldsymbol{e}_2]_i = \left\{ \begin{array}{lll} 0 & \text{for} & i \in \mathcal{I}_1 \\ 1 & \text{for} & i \in \mathcal{I}_2 \end{array} \right. .$$

The GMNP and GNPP are convex optimization problems as both the objective functions (4.1) and the feasible sets (4.3) and (4.4), respectively, are convex.

4.2 Sequential algorithm

The optimization problem (4.2) with the feasible set $\mathcal{A}$ can be transformed to a sequence of auxiliary optimization problems with the same objective function (4.1) but with much simpler auxiliary feasible sets $\mathcal{A}^{(0)}$, $\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(t)}$. Solving the problem (4.2) with respect to the auxiliary feasible sets yields a sequence of solutions $\boldsymbol{\alpha}^{(0)}, \boldsymbol{\alpha}^{(1)}, \dots, \boldsymbol{\alpha}^{(t)}$. The sequence of feasible sets $\mathcal{A}^{(0)}$, $\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(t)}$ is assumed to be constructed such that (i) the sequence $Q(\boldsymbol{\alpha}^{(0)}) > Q(\boldsymbol{\alpha}^{(1)}) > \dots > Q(\boldsymbol{\alpha}^{(t)})$ converges to the optimal solution $Q(\boldsymbol{\alpha}^*)$ and (ii) the auxiliary problems can be solved efficiently. The sequential algorithm solving the QP task which implements the idea mentioned above is summarized by Algorithm 11.

Algorithm 11: A Sequential Optimization Algorithm

1. Initialization. Select $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$.
2. Repeat until stopping condition is satisfied:
   a) Select a feasible set $\mathcal{A}^{(t+1)}$ such that
   $$Q(\boldsymbol{\alpha}^{(t)}) > \min_{\boldsymbol{\alpha} \in \mathcal{A}^{(t+1)}} Q(\boldsymbol{\alpha})\,. \tag{4.5}$$
   b) Solve the auxiliary task
   $$\boldsymbol{\alpha}^{(t+1)} = \operatorname*{argmin}_{\boldsymbol{\alpha} \in \mathcal{A}^{(t+1)}} Q(\boldsymbol{\alpha})\,. \tag{4.6}$$

Two cases of simple auxiliary feasible sets $\mathcal{A}^{(t+1)}$ will be assumed:

- A line segment
  $$\mathcal{A}_L^{(t+1)} = \{\boldsymbol{\alpha} \in \mathbb{R}^m \colon \boldsymbol{\alpha} = (1-\tau)\boldsymbol{\alpha}^{(t)} + \tau\boldsymbol{\beta}^{(t)}, 0 \le \tau \le 1\}\,,$$
  where $\boldsymbol{\alpha}^{(t)}$ is the current solution and $\boldsymbol{\beta}^{(t)} \in \mathcal{A}$ is selected such that the set $\mathcal{A}_L^{(t+1)}$ satisfies the condition (4.5).
- A triangle
  $$\mathcal{A}_T^{(t+1)} = \{\boldsymbol{\alpha} \in \mathbb{R}^m \colon \boldsymbol{\alpha} = \boldsymbol{\alpha}^{(t)} + \tau(\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}) + \omega(\boldsymbol{\gamma}^{(t)} - \boldsymbol{\alpha}^{(t)}), 0 \le \tau, 0 \le \omega, \tau + \omega \le 1\},$$
  where $\boldsymbol{\alpha}^{(t)}$ is the current solution and $\boldsymbol{\beta}^{(t)} \in \mathcal{A}$, $\boldsymbol{\gamma}^{(t)} \in \mathcal{A}$ are selected such that the set $\mathcal{A}_T^{(t+1)}$ satisfies the condition (4.5).

The optimization task (4.6) has an analytical solution for the case that the feasible set is a line segment or a triangle. Moreover, there exist simple rules to construct the vectors $\boldsymbol{\beta}^{(t)}$, $\boldsymbol{\gamma}^{(t)}$ which guarantee that condition (4.5) is satisfied.

The next sections describe particular algorithms to solve the QP task which are special instances of the sequential Algorithm 11. The algorithms differ in the used auxiliary feasible set, i.e., whether the line segment or the triangle approximation is used. Another difference lies in the rule to construct the vectors $\boldsymbol{\beta}^{(t)}$ and $\boldsymbol{\gamma}^{(t)}$. On the other hand, the optimization problem (4.6) over auxiliary feasible set has the same analytical solution for all algorithms. The analytical solution of task (4.6) for a line segment is derived in Section 4.4 and for a triangle in Section 4.5.

Algorithm (11) monotonically decreases the objective function $Q(\boldsymbol{\alpha})$ until it gets sufficiently close to the optimal value $Q(\boldsymbol{\alpha}^*)$, which is indicated by the used stopping condition. The stopping conditions are derived in Section 4.3.

Particular algorithms for the GMNP are described in Section 4.6. Section 4.7 describes algorithms for the GNPP. The convergence proofs are introduced in sections describing the corresponding algorithm.

4.3 Stopping conditions

There is a need to stop the algorithm when it gets sufficiently close to the optimum. Two reasonable stopping conditions are assumed:

1. ε-optimal solution. The algorithm stops if the inequality
$$Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}^*) \leq \varepsilon \,, \tag{4.7}$$
holds for a prescribed $\varepsilon > 0$.

2. Scale invariant ε-optimal solution. The algorithm stops if the inequality
$$Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}^*) \leq \varepsilon |Q(\boldsymbol{\alpha})| \,, \tag{4.8}$$
holds for a prescribed $\varepsilon > 0$.

The stopping conditions (4.7) and (4.8) can be evaluated despite the unknown optimal value $Q(\boldsymbol{\alpha}^*)$ because a lower bound $Q_{LB}(\boldsymbol{\alpha})$ can be used instead. Let the inequality $Q(\boldsymbol{\alpha}^*) \geq Q_{LB}(\boldsymbol{\alpha})$ hold. Then the satisfaction of the condition
$$Q(\boldsymbol{\alpha}) - Q_{LB}(\boldsymbol{\alpha}) \leq \varepsilon \,,$$
implies that condition (4.7) holds as well. Similarly, if the condition
$$Q(\boldsymbol{\alpha}) - Q_{LB}(\boldsymbol{\alpha}) \leq \varepsilon |Q(\boldsymbol{\alpha})| \,,$$
holds then (4.8) is also satisfied.

The current value of the objective function $Q(\boldsymbol{\alpha})$ can be used as an upper bound of the optimal value $Q(\boldsymbol{\alpha}^*)$. The lower bound $Q_{LB}(\boldsymbol{\alpha})$ has to be determined such that the following inequality holds
$$Q(\boldsymbol{\alpha}) \geq Q(\boldsymbol{\alpha}^*) \geq Q_{LB}(\boldsymbol{\alpha}) \,, \quad \forall \boldsymbol{\alpha} \in \mathcal{A} \,.$$

The equality occurs if $\boldsymbol{\alpha}$ is the solution of the task (4.2). The computation of the lower bound $Q_{LB}(\boldsymbol{\alpha})$ depends on the feasible set used. The lower bounds below are derived separately for the feasible set (4.3) and (4.4), respectively.

In the case of the GMNP with the feasible set $\mathcal{A}$ defined by (4.3), the following lower bound $Q_{LB}(\boldsymbol{\alpha})$ can be used

$$Q_{LB}(\boldsymbol{\alpha}) = \min_{i\in\mathcal{I}} [\mathrm{H}\boldsymbol{\alpha} + \boldsymbol{c}]_i - \frac{1}{2}\langle \boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha}\rangle \, .$$

The inequality $Q(\boldsymbol{\alpha}^*) \geq Q_{LB}(\boldsymbol{\alpha})$, $\forall \boldsymbol{\alpha} \in \mathcal{A}$ is proven in the following derivation. Let the vector $\nabla Q(\boldsymbol{\alpha}^*)$ be the gradient of the quadratic function Q evaluated at the vector $\boldsymbol{\alpha}^*$. It follows from the convexity of the function $Q(\boldsymbol{\alpha})$ that

$$\begin{aligned} Q(\boldsymbol{\alpha}^*) + \langle (\boldsymbol{\alpha} - \boldsymbol{\alpha}^*), \nabla Q(\boldsymbol{\alpha}^*)\rangle &\leq Q(\boldsymbol{\alpha}) \, , \\ \frac{1}{2}\langle \boldsymbol{\alpha}^*, \mathrm{H}\boldsymbol{\alpha}^*\rangle + \langle \boldsymbol{\alpha}^*, \boldsymbol{c}\rangle + \langle (\boldsymbol{\alpha} - \boldsymbol{\alpha}^*), (\mathrm{H}\boldsymbol{\alpha}^* + \boldsymbol{c})\rangle &\leq \frac{1}{2}\langle \boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha}\rangle + \langle \boldsymbol{\alpha}, \boldsymbol{c}\rangle \, , \end{aligned}$$

which can be further arranged to

$$\begin{aligned} \langle \boldsymbol{\alpha}^*, (\mathrm{H}\boldsymbol{\alpha} + \boldsymbol{c})\rangle - \frac{1}{2}\langle \boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha}\rangle &\leq \frac{1}{2}\langle \boldsymbol{\alpha}^*, \mathrm{H}\boldsymbol{\alpha}^*\rangle + \langle \boldsymbol{\alpha}^*, \boldsymbol{c}\rangle \, , \\ \min_{i\in\mathcal{I}}[\mathrm{H}\boldsymbol{\alpha} + \boldsymbol{c}]_i - \frac{1}{2}\langle \boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha}\rangle &\leq \frac{1}{2}\langle \boldsymbol{\alpha}^*, \mathrm{H}\boldsymbol{\alpha}^*\rangle + \langle \boldsymbol{\alpha}^*, \boldsymbol{c}\rangle \, . \end{aligned}$$

The last inequality holds true as the convex combination of real numbers cannot be less than their minimum. It remains to show that the equality $Q_{LB}(\boldsymbol{\alpha}^*) = Q(\boldsymbol{\alpha}^*)$ occurs in the optimum. The Lagrangian of the optimization problem (4.2) and the feasible set (4.3) reads

$$L(\boldsymbol{\alpha}, \lambda, \boldsymbol{\mu}) = \frac{1}{2}\langle \boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha}\rangle + \langle \boldsymbol{c}, \boldsymbol{\alpha}\rangle + \lambda(\langle \boldsymbol{\alpha}, \boldsymbol{e}\rangle - 1) - \langle \boldsymbol{\alpha}, \boldsymbol{\mu}\rangle \, ,$$

and the corresponding KKT conditions (c.f. Section 1.6) are defined as

$$\begin{aligned} \frac{\partial L(\boldsymbol{\alpha}, \lambda, \boldsymbol{\mu})}{\partial \boldsymbol{\alpha}} = \mathrm{H}\boldsymbol{\alpha} + \boldsymbol{c} + \lambda \boldsymbol{e} - \boldsymbol{\mu} &= 0 \, , && (4.9) \\ \langle \boldsymbol{\alpha}, \boldsymbol{e}\rangle &= 1 \, , && (4.10) \\ \boldsymbol{\alpha} &\geq 0 \, , && (4.11) \\ \langle \boldsymbol{\alpha}, \boldsymbol{\mu}\rangle &= 0 \, , && (4.12) \\ \boldsymbol{\mu} &\geq 0 \, . && (4.13) \end{aligned}$$

From conditions (4.9), (4.11), (4.12), and (4.13) it follows that

$$\begin{aligned} [\mathrm{H}\boldsymbol{\alpha} + \boldsymbol{c}]_i + \lambda &\geq 0 \, , \qquad i \in \mathcal{I} \, , \\ [\mathrm{H}\boldsymbol{\alpha} + \boldsymbol{c}]_i + \lambda &= 0 \, , \qquad i \in \mathcal{I}_\emptyset = \{\alpha_i \in \mathcal{I} \colon \alpha_i > 0\} \, . \end{aligned} \qquad (4.14)$$

Multiplying (4.9) by $\boldsymbol{\alpha}$ and using (4.10), (4.12) yields

$$\langle \boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha}\rangle + \langle \boldsymbol{\alpha}, \boldsymbol{c}\rangle + \lambda = 0 \, . \qquad (4.15)$$

Combination of (4.14) and (4.15) gives

$$\langle \boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha}\rangle + \langle \boldsymbol{\alpha}, \boldsymbol{c}\rangle = \min_{i\in\mathcal{I}}[\mathrm{H}\boldsymbol{\alpha} + \boldsymbol{c}]_i \, ,$$

which proves the equality $Q_{LB}(\boldsymbol{\alpha}^*) = Q(\boldsymbol{\alpha}^*)$.

In the case of the GNPP with the feasible set $\mathcal{A}$ defined by (4.4), the lower bound Q_{LB} reads

$$Q_{LB}(\boldsymbol{\alpha}) = \min_{i\in\mathcal{I}_1}[\mathrm{H}\boldsymbol{\alpha}+\boldsymbol{c}]_i + \min_{i\in\mathcal{I}_2}[\mathrm{H}\boldsymbol{\alpha}+\boldsymbol{c}]_i - \frac{1}{2}\langle\boldsymbol{\alpha},\mathrm{H}\boldsymbol{\alpha}\rangle \,.$$

The derivation of the inequality $Q(\boldsymbol{\alpha}^*) \geq Q_{LB}(\boldsymbol{\alpha})$, $\boldsymbol{\alpha}\in\mathcal{A}$, is similar to the previous case and thus is omitted. The equality $Q_{LB}(\boldsymbol{\alpha}^*) = Q(\boldsymbol{\alpha}^*)$ occurs in the optimum which can be shown as follows. The Lagrangian of the optimization problem (4.2) and the feasible set (4.4) reads

$$L(\boldsymbol{\alpha},\lambda_1,\lambda_2,\boldsymbol{\mu}) = \frac{1}{2}\langle\boldsymbol{\alpha},\mathrm{H}\boldsymbol{\alpha}\rangle + \langle\boldsymbol{c},\boldsymbol{\alpha}\rangle + \lambda_1(\langle\boldsymbol{\alpha},\boldsymbol{e}_1\rangle - 1) + \lambda_2(\langle\boldsymbol{\alpha},\boldsymbol{e}_2\rangle - 1) - \langle\boldsymbol{\alpha},\boldsymbol{\mu}\rangle \,.$$

The corresponding KKT conditions are defined as

$$\begin{aligned}
\frac{\partial L(\boldsymbol{\alpha},\lambda_1,\lambda_2,\boldsymbol{\mu})}{\partial\boldsymbol{\alpha}} = \mathrm{H}\boldsymbol{\alpha}+\boldsymbol{c}+\lambda_1\boldsymbol{e}_1+\lambda_2\boldsymbol{e}_2-\boldsymbol{\mu} &= 0\,, && (4.16)\\
\langle\boldsymbol{\alpha},\boldsymbol{e}_1\rangle &= 1\,, && (4.17)\\
\langle\boldsymbol{\alpha},\boldsymbol{e}_2\rangle &= 1\,, && (4.18)\\
\boldsymbol{\alpha} &\geq 0\,, && (4.19)\\
\langle\boldsymbol{\alpha},\boldsymbol{\mu}\rangle &= 0\,, && (4.20)\\
\boldsymbol{\mu} &\geq 0\,. && (4.21)
\end{aligned}$$

From conditions (4.16), (4.19), (4.20), and (4.21) follows that

$$\begin{array}{rcll}
[\mathrm{H}\boldsymbol{\alpha}+\boldsymbol{c}]_i+\lambda_1 & \geq & 0\,, & i\in\mathcal{I}_1\,,\\
[\mathrm{H}\boldsymbol{\alpha}+\boldsymbol{c}]_i+\lambda_2 & \geq & 0\,, & i\in\mathcal{I}_2\,,\\
[\mathrm{H}\boldsymbol{\alpha}+\boldsymbol{c}]_i+\lambda_1 & = & 0\,, & i\in\mathcal{I}_1\cap\mathcal{I}_\emptyset\,,\\
[\mathrm{H}\boldsymbol{\alpha}+\boldsymbol{c}]_i+\lambda_2 & = & 0\,, & i\in\mathcal{I}_2\cap\mathcal{I}_\emptyset\,,
\end{array} \qquad (4.22)$$

where the set $\mathcal{I}_\emptyset = \{\alpha_i\in\mathcal{I}\colon \alpha_i > 0\}$. Multiplying (4.16) by $\boldsymbol{\alpha}$ and using (4.17), (4.18), (4.20) yields

$$\langle\boldsymbol{\alpha},\mathrm{H}\boldsymbol{\alpha}\rangle + \langle\boldsymbol{\alpha},\boldsymbol{c}\rangle + \lambda_1 + \lambda_2 = 0\,. \qquad (4.23)$$

Combination of (4.22) and (4.23) gives

$$\langle\boldsymbol{\alpha},\mathrm{H}\boldsymbol{\alpha}\rangle + \langle\boldsymbol{c},\boldsymbol{\alpha}\rangle = \min_{i\in\mathcal{I}_1}[\mathrm{H}\boldsymbol{\alpha}+\boldsymbol{c}]_i + \min_{i\in\mathcal{I}_2}[\mathrm{H}\boldsymbol{\alpha}+\boldsymbol{c}]_i\,,$$

which proves the equality $Q_{LB}(\boldsymbol{\alpha}^*) = Q(\boldsymbol{\alpha}^*)$.

4.4 Solution for a line segment

Let the feasible set $\mathcal{A}_L^{(t+1)}$ be a line segment between vectors $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\beta}^{(t)}$ so that

$$\mathcal{A}_L^{(t+1)} = \{\boldsymbol{\alpha}\in\mathbb{R}^m\colon \boldsymbol{\alpha} = \boldsymbol{\alpha}^{(t)}(1-\tau)+\tau\boldsymbol{\beta}^{(t)}, 0\leq\tau\leq 1\}\,. \qquad (4.24)$$

The quadratic objective function (4.1) defined over the line segment $\mathcal{A}_L^{(t+1)}$ reads

$$\begin{aligned}
Q_L^{(t+1)}(\tau) &= Q\left(\boldsymbol{\alpha}^{(t)}(1-\tau)+\tau\boldsymbol{\beta}^{(t)}\right)\\
&= \frac{1}{2}(1-\tau)^2\langle\boldsymbol{\alpha}^{(t)},\mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \tau(1-\tau)\langle\boldsymbol{\beta}^{(t)},\mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle \qquad (4.25)\\
&\quad+\frac{1}{2}\tau^2\langle\boldsymbol{\beta}^{(t)},\mathrm{H}\boldsymbol{\beta}^{(t)}\rangle + (1-\tau)\langle\boldsymbol{c},\boldsymbol{\alpha}^{(t)}\rangle + \tau\langle\boldsymbol{c},\boldsymbol{\beta}^{(t)}\rangle\,.
\end{aligned}$$

The objective function is now parametrized by a single τ. The solution of the QP task (4.2) with respect to the feasible set $\mathcal{A}_L^{(t+1)}$ can be written as

$$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1-\tau^*) + \tau^*\boldsymbol{\beta}^{(t)} \,, \tag{4.26}$$

where τ^* is the solution of the following task

$$\tau^* = \operatorname*{argmin}_{0\leq\tau\leq 1} Q_L^{(t+1)}(\tau) \,.$$

It is obvious that $Q_L^{(t+1)}(0) = Q(\boldsymbol{\alpha}^{(t)})$ and $Q_L^{(t+1)}(1) = Q(\boldsymbol{\beta}^{(t)})$. The derivative of $Q_L^{(t+1)}(\tau)$ with respect to τ reads

$$\frac{\partial Q_L^{(t+1)}(\tau)}{\partial\tau} = (\tau-1)\langle\boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + (1-2\tau)\langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \tau\langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle + \langle\boldsymbol{c}, (\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)})\rangle \,. \tag{4.27}$$

The derivative (4.27) at zero equals to

$$\left.\frac{\partial Q_L^{(t+1)}(\tau)}{\partial\tau}\right|_{\tau=0} = \langle(\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle \,. \tag{4.28}$$

If the vector $\boldsymbol{\beta}^{(t)}$ is selected such that the derivative (4.28) is negative, i.e., the following inequality holds

$$\langle(\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle < 0 \,, \tag{4.29}$$

then it implies an improvement in the optimized criterion so that

$$\min_{0\leq\tau\leq 1} Q_L^{(t+1)}(\tau) < Q(\boldsymbol{\alpha}^{(t)}) \,.$$

The optimal τ^* which minimizes the objective $Q_L^{(t+1)}(\tau)$ can be found by setting the derivative (4.27) to zero and solving for τ. This yields

$$\tau^* = \min\left(1, \frac{\langle(\boldsymbol{\alpha}^{(t)} - \boldsymbol{\beta}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle}{\langle\boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle - 2\langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle}\right) \,, \tag{4.30}$$

where the minimum $\min(1,\cdot)$ guarantees that the solution does not leave the feasible set $\mathcal{A}_L^{(t+1)}$.

The above derivation can be used to solve the auxiliary subtasks (4.6) of Algorithm 11. The new solution $\boldsymbol{\alpha}^{(t+1)}$ computed by (4.30) and (4.26) guarantees that $\boldsymbol{\alpha}^{(t+1)}$ minimizes the objective function $Q(\boldsymbol{\alpha})$ over the line segment $\mathcal{A}_L^{(t+1)}$. The condition (4.29) implies that the optimization over the line segment $\mathcal{A}_L^{(t+1)}$ to an improvement

$$\Delta^{(t+1)} = Q(\boldsymbol{\alpha}^{(t)}) - Q(\boldsymbol{\alpha}^{(t+1)}) = Q(\boldsymbol{\alpha}^{(t)}) - \min_{0\leq\tau\leq 1} Q_L^{(t+1)}(\tau) > 0 \,.$$

From (4.25) and (4.30) it follows that for $\tau < 1$ the improvement $\Delta^{(t+1)}$ equals

$$\Delta^{(t+1)} = \frac{\langle(\boldsymbol{\alpha}^{(t)} - \boldsymbol{\beta}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle^2}{2(\langle\boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle - 2\langle\boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle + \langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle)} \,, \tag{4.31}$$

and for $\tau = 1$ it is

$$\begin{aligned}\Delta^{(t+1)} &= \frac{1}{2}\langle\boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \langle\boldsymbol{c}, \boldsymbol{\alpha}^{(t)}\rangle - \left(\frac{1}{2}\langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle + \langle\boldsymbol{c}, \boldsymbol{\beta}^{(t)}\rangle\right) \\ &= Q(\boldsymbol{\alpha}^{(t)}) - Q(\boldsymbol{\beta}^{(t)}) \,. \end{aligned} \tag{4.32}$$

4.5 Solution for a triangle

Let the feasible set $\mathcal{A}_T^{(t+1)}$ be a triangle given by the vectors $\boldsymbol{\alpha}^{(t)}$, $\boldsymbol{\beta}^{(t)}$ and $\boldsymbol{\gamma}^{(t)}$ so that

$$\mathcal{A}_T^{(t+1)} = \{\boldsymbol{\alpha} \in \mathbb{R}^m \colon \boldsymbol{\alpha} = \boldsymbol{\alpha}^{(t)} + \tau(\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}) + \omega(\boldsymbol{\gamma}^{(t)} - \boldsymbol{\alpha}^{(t)}),\ 0 \le \tau, 0 \le \omega, \tau + \omega \le 1\} \,. \tag{4.33}$$

The quadratic objective function (4.1) defined over the triangle $\mathcal{A}_T^{(t+1)}$ reads

$$\begin{aligned} Q_T^{(t+1)}(\tau,\omega) &= Q(\boldsymbol{\alpha}^{(t)} + \tau(\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}) + \omega(\boldsymbol{\gamma}^{(t)} - \boldsymbol{\alpha}^{(t)})) \\ &= \frac{1}{2}(1-\tau-\omega)^2\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \tau(1-\omega-\tau)\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle \\ &\quad + \omega(1-\omega-\tau)\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\gamma}^{(t)}\rangle + \frac{1}{2}\tau^2\langle \boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle + \tau\omega\langle \boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\gamma}^{(t)}\rangle \\ &\quad + \frac{1}{2}\omega^2\langle \boldsymbol{\gamma}^{(t)}, \mathrm{H}\boldsymbol{\gamma}^{(t)}\rangle + (1-\tau-\omega)\langle \boldsymbol{c}, \boldsymbol{\alpha}^{(t)}\rangle + \tau\langle \boldsymbol{c}, \boldsymbol{\beta}^{(t)}\rangle + \omega\langle \boldsymbol{c}, \boldsymbol{\gamma}^{(t)}\rangle \,. \end{aligned}$$

The objective function is now parametrized by two variables τ and ω. The solution of the task (4.2) with respect to the feasible set $\mathcal{A}_T^{(t+1)}$ can be written as

$$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)} + \tau^*(\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}) + \omega^*(\boldsymbol{\gamma}^{(t)} - \boldsymbol{\alpha}^{(t)}) \,, \tag{4.34}$$

where τ^* and ω^* are the solutions of the following task

$$(\tau^*, \omega^*) = \operatorname*{argmin}_{\substack{\tau \ge 0 \\ \omega \ge 0 \\ \tau+\omega \le 1}} Q_T^{(t+1)}(\tau,\omega) \,. \tag{4.35}$$

It is obvious that $Q_T^{(t+1)}(0,0) = Q(\boldsymbol{\alpha}^{(t)})$, $Q_T^{(t+1)}(1,0) = Q(\boldsymbol{\beta}^{(t)})$ and $Q_T^{(t+1)}(0,1) = Q(\boldsymbol{\gamma}^{(t)})$. The derivatives of $Q_T^{(t+1)}(\tau,\omega)$ with respect to τ and ω, respectively, yield

$$\begin{aligned} \frac{\partial Q_T^{(t+1)}(\tau,\omega)}{\partial \tau} &= (\omega+\tau-1)\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + (1-\omega-2\tau)\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle \\ &\quad -\omega\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\gamma}^{(t)}\rangle + \tau\langle \boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle + \omega\langle \boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\gamma}^{(t)}\rangle \\ &\quad -\langle \boldsymbol{c}, \boldsymbol{\alpha}^{(t)}\rangle + \langle \boldsymbol{c}, \boldsymbol{\beta}^{(t)}\rangle \,. \end{aligned} \tag{4.36}$$

$$\begin{aligned} \frac{\partial Q_T^{(t+1)}(\tau,\omega)}{\partial \omega} &= (\omega+\tau-1)\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle - \tau\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle \\ &\quad (1-2\omega-\tau)\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\gamma}^{(t)}\rangle + \tau\langle \boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\gamma}^{(t)}\rangle + \omega\langle \boldsymbol{\gamma}^{(t)}, \mathrm{H}\boldsymbol{\gamma}^{(t)}\rangle \\ &\quad -\langle \boldsymbol{c}, \boldsymbol{\alpha}^{(t)}\rangle + \langle \boldsymbol{c}, \boldsymbol{\gamma}^{(t)}\rangle \,. \end{aligned} \tag{4.37}$$

If the vectors $\boldsymbol{\beta}^{(t)} \in \mathcal{A}$ and $\boldsymbol{\gamma}^{(t)} \in \mathcal{A}$ are selected such that the derivatives (4.36) and (4.37) at zero are negative, i.e.,

$$\begin{aligned} \left.\frac{\partial Q_T^{(t+1)}(\tau,\omega)}{\partial \tau}\right|_{\substack{\tau=0\\ \omega=0}} &= \langle(\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle < 0 \\ \left.\frac{\partial Q_T^{(t+1)}(\tau,\omega)}{\partial \omega}\right|_{\substack{\tau=0\\ \omega=0}} &= \langle(\boldsymbol{\gamma}^{(t)} - \boldsymbol{\alpha}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle < 0 \,, \end{aligned}$$

then the optimization subtask (4.35) leads to the improvement, i.e.,

$$Q(\boldsymbol{\alpha}^{(t+1)}) = \min_{\substack{\tau \geq 0 \\ \omega \geq 0 \\ \tau+\omega \leq 1}} Q_T^{(t+1)}(\tau,\omega) < Q(\boldsymbol{\alpha}^{(t)}) \,.$$

The vector $\boldsymbol{\alpha}^{(t+1)}$ which minimizes the objective function $Q(\boldsymbol{\alpha})$ over the set $\mathcal{A}_T^{(t+1)}$ can be computed analytically by solving the unconstrained problem

$$(\tau,\omega) = \operatorname*{argmin}_{\tau',\omega'} Q_T^{(t+1)}(\tau',\omega') \,. \tag{4.38}$$

If the conditions

$$\tau \geq 0 \,, \omega \geq 0 \quad \text{and} \quad \tau + \omega \leq 1 \,,$$

hold then the $\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)} + \tau(\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}) + \omega(\boldsymbol{\gamma}^{(t)} - \boldsymbol{\alpha}^{(t)})$. This occurs if the optimal $\alpha^{(t+1)}$ lies inside the triangle $\mathcal{A}_T$. In the opposite case, $\alpha^{(t+1)}$ lies on one of the three edges of the triangle $\mathcal{A}_T$. If this occurs then the optimal $\boldsymbol{\alpha}^{(t+1)}$ is computed by optimizing over the line segments $(\boldsymbol{\alpha}^{(t)},\boldsymbol{\beta}^{(t)})$, $(\boldsymbol{\alpha}^{(t)},\boldsymbol{\gamma}^{(t)})$ and $(\boldsymbol{\beta}^{(t)},\boldsymbol{\gamma}^{(t)})$. The solution which yields the biggest improvement is taken. The optimization over a line segment was described in Section 4.4.

The analytical solution of the unconstrained problem (4.38) can be found by setting the derivatives (4.36) and (4.36) to zero and solving for τ and ω which yields

$$\tau = \frac{a_3 a_4 - a_1 a_5}{a_1^2 - a_2 a_4} \qquad \text{and} \qquad \omega = \frac{a_2 a_5 - a_3 a_1}{a_1^2 - a_2 a_4} \,, \tag{4.39}$$

where

$$\begin{aligned}
a_1 &= \langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)} \rangle - \langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)} \rangle - \langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\gamma}^{(t)} \rangle + \langle \boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\gamma}^{(t)} \rangle \\
a_2 &= \langle (\boldsymbol{\alpha}^{(t)} - \boldsymbol{\beta}^{(t)}), \mathrm{H}(\boldsymbol{\alpha}^{(t)} - \boldsymbol{\beta}^{(t)}) \rangle \\
a_3 &= \langle (\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}) \rangle \\
a_4 &= \langle (\boldsymbol{\alpha}^{(t)} - \boldsymbol{\gamma}^{(t)}), \mathrm{H}(\boldsymbol{\alpha}^{(t)} - \boldsymbol{\gamma}^{(t)}) \rangle \\
a_5 &= \langle (\boldsymbol{\gamma}^{(t)} - \boldsymbol{\alpha}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}) \rangle \,.
\end{aligned}$$

Algorithm 12 summarizes how to optimize the quadratic criterion $Q(\boldsymbol{\alpha})$ over the triangle $\mathcal{A}_T^{(t+1)}$ given by vectors $(\boldsymbol{\alpha}^{(t)},\boldsymbol{\beta}^{(t)},\boldsymbol{\gamma}^{(t)})$.

Algorithm 12: Optimization of $Q(\boldsymbol{\alpha})$ over triangle $\mathcal{A}_T^{(t+1)}$

1. Solve the unconstrained problem

   $$(\tau,\omega) = \operatorname*{argmin}_{\tau',\omega'} Q_T^{(t+1)}(\tau',\omega') \,,$$

   using (4.39). If $\tau \geq 0$ and $\omega \geq 0$ and $1 - \tau - \omega > 0$ then the solution vector reads

   $$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)} + \tau(\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}) + \omega(\boldsymbol{\gamma}^{(t)} - \boldsymbol{\alpha}^{(t)}) \,,$$

   otherwise continue with Step 2.

2. Optimize $Q(\boldsymbol{\alpha})$ over the edges of the triangle $\mathcal{A}_T^{(t+1)}$:

   a) Compute vectors $\boldsymbol{\alpha}_1^{(t+1)}$, $\boldsymbol{\alpha}_2^{(t+1)}$ and $\boldsymbol{\alpha}_3^{(t+1)}$ which minimize the function $Q(\boldsymbol{\alpha})$ over the line segments between vectors $(\boldsymbol{\alpha}^{(t)}, \boldsymbol{\beta}^{(t)})$, $(\boldsymbol{\alpha}^{(t)}, \boldsymbol{\gamma}^{(t)})$ and $(\boldsymbol{\beta}^{(t)}, \boldsymbol{\gamma}^{(t)})$. The vector $\boldsymbol{\alpha}_i^{(t+1)}$ is computed for given pair $(\boldsymbol{a}_i, \boldsymbol{b}_i)$ as:

$$\boldsymbol{\alpha}_i^{(t+1)} = \begin{cases} \boldsymbol{a}_i & \text{for} \quad \tau_i \leq 0\,, \\ \boldsymbol{b}_i & \text{for} \quad \tau_i \geq 1\,, \\ \boldsymbol{a}_i(1-\tau_i) + \tau_i \boldsymbol{b}_i & \text{otherwise}\,, \end{cases}$$

   where

$$\tau_i = \frac{\langle (\boldsymbol{a}_i - \boldsymbol{b}_i), (\mathrm{H}\boldsymbol{a}_i + \boldsymbol{c}) \rangle}{\langle \boldsymbol{a}_i, \mathrm{H}\boldsymbol{a}_i \rangle - 2\langle \boldsymbol{b}_i, \mathrm{H}\boldsymbol{a}_i \rangle + \langle \boldsymbol{b}_i, \mathrm{H}\boldsymbol{b}_i \rangle}\,.$$

   b) Select the vector with the minimal value of Q, i.e.,

$$\boldsymbol{\alpha}^{(t+1)} = \operatorname*{argmin}_{i \in \{1,2,3\}} Q(\boldsymbol{\alpha}_i^{(t+1)})\,.$$

4.6 Algorithms for the generalized minimal norm problem

The GMNP to solve is

$$\boldsymbol{\alpha}^* = \operatorname*{argmin}_{\boldsymbol{\alpha} \in \mathcal{A}} \left(\frac{1}{2} \langle \boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha} \rangle + \langle \boldsymbol{c}, \boldsymbol{\alpha} \rangle \right)\,, \tag{4.40}$$

where the feasible set is defined by

$$\mathcal{A} = \{ \boldsymbol{\alpha} \in \mathbb{R}^m \colon \langle \boldsymbol{\alpha}, \boldsymbol{e} \rangle = 1, \boldsymbol{\alpha} \geq 0 \}\,. \tag{4.41}$$

The algorithms introduced bellow are special cases of the general framework described by Algorithm 11. Particular algorithms differ in the way how the auxiliary feasible set $\mathcal{A}^{(t+1)}$ is constructed in Step 2(a) of Algorithm 11. The optimization over $\mathcal{A}^{(t+1)}$ has for all cases an analytical solution because the line segment or the triangle approximation is used in all algorithms.

The algorithms introduced below are extensions of the original algorithms for the MNP and the NPP. Namely, the algorithm proposed by Kozinec [32], Mitchell et al. [36], Kowalczyk [31] and Keerthi et al. [29] will be extended. The original variants are recovered after $\mathrm{H} = \mathrm{X}\mathrm{X}^T$, $\boldsymbol{c} = 0$ is substituted and the vector $\boldsymbol{w} = \mathrm{X}\boldsymbol{\alpha}$ is updated instead of the vector $\boldsymbol{\alpha}$. A novel proposed method, named Improved Mitchell-Demyanov-Malozemov algorithm, combines ideas of the work by Mitchell et al. [36] and Kowalczyk [31].

4.6.1 Kozinec algorithm for GMNP

The Kozinec Algorithm 13 approximates the feasible set $\mathcal{A}$ by a line segment $\mathcal{A}_L^{(t+1)}$. The line segment $\mathcal{A}_L^{(t+1)}$ is constructed between the current solution $\boldsymbol{\alpha}^{(t)}$ and a vector

$\boldsymbol{\beta}^{(t)} \in \mathcal{A}$. The vector $\boldsymbol{\beta}^{(t)} \in \mathcal{A}$ is defined such that the derivative of the function Q_L at zero is minimal, i.e.,

$$\begin{aligned}\left.\frac{\partial Q_L(\tau)}{\partial \tau}\right|_{\tau=0} &= \langle(\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle \\ &= \min_{\boldsymbol{\beta}\in\mathcal{A}} \langle(\boldsymbol{\beta} - \boldsymbol{\alpha}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle \\ &= \min_{i\in\mathcal{I}} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i - \langle\boldsymbol{\alpha}^{(t)}, (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle < 0\,.\end{aligned} \tag{4.42}$$

Notice that the task (4.42) is a special instance of the linear programming its solution can be simply found. The negative derivative implies that the optimization over the line segment $\mathcal{A}_L^{(t+1)}$ leads to the improvement in the optimized criterion Q.

Algorithm 13: Kozinec Algorithm for GMNP

1. Initialization. Set $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$.
2. Repeat until stopping condition is satisfied:

   a) Construct vector $\boldsymbol{\beta}^{(t)} \in \mathcal{A}$ such that

   $$[\boldsymbol{\beta}^{(t)}]_i = \begin{cases} 1 & \text{for} \quad i = u\,, \\ 0 & \text{for} \quad i \neq u\,, \end{cases} \qquad \forall i \in \mathcal{I}\,, \tag{4.43}$$

   where

   $$u \in \operatorname*{argmin}_{i\in\mathcal{I}} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i\,. \tag{4.44}$$

   b) Update

   $$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1-\tau) + \tau\boldsymbol{\beta}^{(t)}\,,$$

   where

   $$\tau = \min\left(1, \frac{\langle(\boldsymbol{\alpha}^{(t)} - \boldsymbol{\beta}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle}{\langle\boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle - 2\langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle}\right)\,.$$

The Kozinec Algorithm 13 stops after a finite number of iterations if the ε-optimality condition (4.7) is applied. The upper bound on the maximal number of iterations t_{max} can be derived using the following quantities: the initial value of the criterion $Q(\boldsymbol{\alpha}^{(0)})$, the value of the optimal solution $Q(\boldsymbol{\alpha}^*)$, the number of variables $m \in \mathbb{N}$ and the *diameter* $D \in \mathbb{R}^+$ of the minimal ball enclosing column vectors of the matrix $\mathrm{R} = [\boldsymbol{r}_1, \ldots, \boldsymbol{r}_m]$, where $\mathrm{H} = \mathrm{R}^T\mathrm{R}$. The existence of matrix $\mathrm{R} \in \mathbb{R}^{m\times m}$ follows from the positive definiteness and symmetry of the input matrix H, because any such matrix can be decomposed as $\mathrm{H} = \mathrm{R}^T\mathrm{R}$. Thus the diameter D is the solution of the following task

$$\begin{aligned} D^2 &= \max_{\boldsymbol{\alpha}\in\mathcal{A},\boldsymbol{\beta}\in\mathcal{A}} \left(\langle\boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha}\rangle - 2\langle\boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\beta}\rangle + \langle\boldsymbol{\beta}, \mathrm{H}\boldsymbol{\beta}\rangle\right) \\ &= \max_{\boldsymbol{\alpha}\in\mathcal{A},\boldsymbol{\beta}\in\mathcal{A}} \left(\langle\mathrm{R}\boldsymbol{\alpha}, \mathrm{R}\boldsymbol{\alpha}\rangle - 2\langle\mathrm{R}\boldsymbol{\alpha}, \mathrm{R}\boldsymbol{\beta}\rangle + \langle\mathrm{R}\boldsymbol{\beta}, \mathrm{R}\boldsymbol{\beta}\rangle\right) \\ &= \max_{\boldsymbol{\alpha}\in\mathcal{A},\boldsymbol{\beta}\in\mathcal{A}} \|\mathrm{R}\boldsymbol{\alpha} - \mathrm{R}\boldsymbol{\beta}\|^2 > 0\,. \end{aligned} \tag{4.45}$$

Theorem 4.1 *Kozinec Algorithm 13 started from an arbitrary vector $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$ returns the vector $\boldsymbol{\alpha}$ satisfying for any $\varepsilon > 0$ the ε-optimality condition*

$$Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}^*) \leq \varepsilon \,, \tag{4.46}$$

after at most $t_{max} < \infty$ iterations, where

$$t_{max} = \frac{2D^2}{\varepsilon^2}(Q(\boldsymbol{\alpha}^{(0)}) - Q(\boldsymbol{\alpha}^*)) + m \,. \tag{4.47}$$

PROOF: The proof is based on showing that in each iteration the improvement $\Delta^{(t+1)} = Q(\boldsymbol{\alpha}^{(t)}) - Q(\boldsymbol{\alpha}^{(t+1)})$ of the optimized criterion Q is bounded by

$$\Delta^{(t+1)} \geq \frac{\varepsilon^2}{2D^2} > 0 \,, \tag{4.48}$$

except for at most m iterations. The violation of the stopping condition (4.46) implies (c.f. Section 4.3) that

$$\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)} \rangle + \langle \boldsymbol{\alpha}^{(t)}, \boldsymbol{c} \rangle - \min_{i \in \mathcal{I}} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i > \varepsilon \,. \tag{4.49}$$

The improvement $\Delta^{(t+1)}$ is, for general $\boldsymbol{\beta}^{(t)}$, given by (4.31) if $\tau < 1$. Substituting the vector $\boldsymbol{\beta}^{(t)}$ constructed by rule (4.43) to formula (4.31) yields

$$\Delta^{(t+1)} = \frac{(\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)} \rangle + \langle \boldsymbol{\alpha}^{(t)}, \boldsymbol{c} \rangle - [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_u)^2}{2(\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)} \rangle - 2[\mathrm{H}\boldsymbol{\alpha}^{(\boldsymbol{t})}]_u + [\mathrm{H}]_{u,u})} \,, \tag{4.50}$$

where $u = \mathrm{argmin}_{i \in \mathcal{I}} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i$. The inequality (4.49) can be used to bound the numerator of (4.50). Similarly, the squared diameter D^2 can be used as an upper bound on the denominator of (4.50) which follows from (4.45). The combination of (4.49), (4.45) and (4.50) gives directly the bound (4.48) on the minimal improvement for the case $\tau < 1$.

The improvement $\Delta^{(t+1)}$ for the case $\tau = 1$ cannot be generally bounded. However, this case can occur at most m times. This implies from three facts: (i) if $\tau = 1$ then $\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\beta}^{(t)}$, (ii) there are only m distinct vectors $\boldsymbol{\beta}^{(t)}$ which can be constructed by (4.43) and (iii) the sequence $\boldsymbol{\alpha}^{(0)}$, $\boldsymbol{\alpha}^{(1)}$, ..., $\boldsymbol{\alpha}^{(t)}$ cannot contain two equal vectors as the inequalities $Q(\boldsymbol{\alpha}^{(0)}) > Q(\boldsymbol{\alpha}^{(1)}) > \ldots > Q(\boldsymbol{\alpha}^{(t)})$ hold.

Using the bound (4.48) and the fact that the improvement of at most m iterations cannot be bounded yields the inequality

$$Q(\boldsymbol{\alpha}^*) \leq Q(\boldsymbol{\alpha}^{(t)}) \leq Q(\boldsymbol{\alpha}^{(0)}) - \frac{\varepsilon^2}{2D^2}(t - m) \,,$$

which can be further rearranged to

$$t \leq \frac{2D^2}{\varepsilon^2}(Q(\boldsymbol{\alpha}^{(0)}) - Q(\boldsymbol{\alpha}^*)) + m \,,$$

which ends the proof.

■

Notice that the term m can be excluded from the bound (4.47) if the Kozinec algorithm starts from the vector

$$[\boldsymbol{\alpha}^{(0)}]_i = \begin{cases} 1 & \text{for} \quad i = u \,, \\ 0 & \text{for} \quad i \neq u \,, \end{cases} \tag{4.51}$$

where $u \in \mathrm{argmin}_{i \in \mathcal{I}}([\mathrm{H}]_{i,i} + [\boldsymbol{c}]_i)$. The vector (4.51) is a good candidate for an initial solution made in Step 1 of Kozinec Algorithm 13.

4.6.2 Kowalczyk algorithm for GMNP

The Kowalczyk Algorithm 14 approximates the feasible set $\mathcal{A}$ by a line segment $\mathcal{A}_L^{(t+1)}$. The algorithm builds in each step a set of candidate line segments $\mathcal{A}_{L(i)}^{(t+1)}$, $i \in \mathcal{I}$, constructed between the current solution $\boldsymbol{\alpha}^{(t)}$ and the vectors $\boldsymbol{\beta}_i^{(t)}$, $i \in \mathcal{I}$. The vector $\boldsymbol{\beta}_i^{(t)}$ for given $i \in \mathcal{I}$ is constructed by one of two different rules. The first rule is the same as the rule used in the Kozinec Algorithm 13, however, the second rule can be seen as its inversion. Consequently, the line segment $\mathcal{A}_{L(r)}^{(t+1)}$ which leads to the biggest improvement is used for approximation of the feasible set $\mathcal{A}$. The Kowalczyk algorithm for the GMNP reads:

Algorithm 14: Kowalczyk Algorithm for GMNP

1. Initialization. Set $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$.
2. Repeat until stopping condition is satisfied:
   a) For all $i \in \mathcal{I}$ compute an improvement

$$\Delta_i^{(t+1)} = Q(\boldsymbol{\alpha}^{(t)}) - \min_{\boldsymbol{\alpha} \in \mathcal{A}_{L(i)}^{(t+1)}} Q(\boldsymbol{\alpha}) \,, \tag{4.52}$$

   using formulas (4.30), (4.31) and (4.32). The line segments $\mathcal{A}_{L(i)}^{(t+1)}$, $i \in \mathcal{I}$, are given by the vectors $\boldsymbol{\beta}_i^{(t)}$, $i \in \mathcal{I}$ defined as follows. If $[\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i < \langle \boldsymbol{\alpha}^{(t)}, (\mathrm{H}\boldsymbol{\alpha}^{(t)} + c\rangle$ then

$$[\boldsymbol{\beta}_i^{(t)}]_j = \begin{cases} 1 & \text{for} \quad j = i \,, \\ 0 & \text{for} \quad j \neq i \,, \end{cases} \qquad \forall j \in \mathcal{I} \,. \tag{4.53}$$

   If $[\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i > \langle \boldsymbol{\alpha}^{(t)}, (\mathrm{H}\boldsymbol{\alpha}^{(t)} + c)\rangle$ and $[\boldsymbol{\alpha}^{(t)}]_i < 1$ then the vector $\boldsymbol{\beta}^{(t)}$ is defined by

$$[\boldsymbol{\beta}_i^{(t)}]_j = \begin{cases} 0 & \text{for} \quad j = i \,, \\ \frac{[\boldsymbol{\alpha}^{(t)}]_j}{1 - [\boldsymbol{\alpha}^{(t)}]_i} & \text{for} \quad j \neq i \,, \end{cases} \qquad \forall j \in \mathcal{I} \,. \tag{4.54}$$

   For $[\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i > \langle \boldsymbol{\alpha}^{(t)}, (\mathrm{H}\boldsymbol{\alpha}^{(t)} + c)\rangle$ and $[\boldsymbol{\alpha}^{(t)}]_i = 1$ set $\Delta_i^{(t+1)} = 0$. Finally choose

$$r \in \operatorname*{argmax}_{i \in \mathcal{I}} \Delta_i^{(t+1)} \,.$$

   b) Update

$$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1 - \tau) + \tau \boldsymbol{\beta}_r^{(t)} \,,$$

   where

$$\tau = \min\left(1, \frac{\langle (\boldsymbol{\alpha}^{(t)} - \boldsymbol{\beta}_r^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle}{\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle - 2\langle \boldsymbol{\beta}_r^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \langle \boldsymbol{\beta}_r^{(t)}, \mathrm{H}\boldsymbol{\beta}_r^{(t)}\rangle}\right) \,.$$

There are two distinct rules (4.53) and (4.54) used to construct the line segment $\mathcal{A}_{L(i)}^{(t+1)}$ for given $i \in \mathcal{I}$. It can be shown that just one of these two rules can lead to an improvement. Let $Q_{L(i)}$ be the criterion Q parametrized over the line segment $\mathcal{A}_{L(i)}^{(t+1)}$ constructed by rule (4.53). Substituting (4.53) to (4.27) yields the derivative of $Q_{L(i)}$ at zero which reads

$$\left.\frac{\partial Q_{L(i)}(\tau)}{\partial \tau}\right|_{\tau=0} = [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i - \langle \boldsymbol{\alpha}^{(t)}, (\mathrm{H}\boldsymbol{\alpha}^{(t)} + c)\rangle \,. \tag{4.55}$$

Similarly for the rule (4.54) the derivative reads

$$\left.\frac{\partial Q_{L(i)}(\tau)}{\partial \tau}\right|_{\tau=0} = \frac{[\boldsymbol{\alpha}^{(t)}]_i}{1-[\boldsymbol{\alpha}^{(t)}]_i}\Big(\langle \boldsymbol{\alpha}^{(t)}, (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle - [\mathrm{H}\boldsymbol{\alpha}^{(t)} + c]_i\Big) \,. \tag{4.56}$$

The negative sign of the derivative of $Q_{L(i)}$ implies an improvement in the optimized criterion. Comparison between (4.55) and (4.56) shows that the sign of $[\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i - \langle \boldsymbol{\alpha}^{(t)}, (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle$ determines which rule is to be applied. In the case that the derivative (4.55) is positive and $[\boldsymbol{\alpha}^{(t)}]_i = 1$ then no improvement can be achieved.

Kowalczyk Algorithm 14 stops after a finite number of iterations if the ε-optimality condition (4.7) is applied.

Theorem 4.2 *Kowalczyk Algorithm 14 started from an arbitrary vector $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$ returns the vector $\boldsymbol{\alpha}$ satisfying for any $\varepsilon > 0$ the ε-optimality condition*

$$Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}^*) \leq \varepsilon \,, \tag{4.57}$$

after at most $t_{max} < \infty$ iterations, where

$$t_{max} = \frac{2D^2}{\varepsilon^2}(Q(\boldsymbol{\alpha}^{(0)}) - Q(\boldsymbol{\alpha}^*)) + m \,. \tag{4.58}$$

PROOF: Kowalczyk Algorithm 14 in each iterations selects the line segment $\mathcal{A}_{L(r)}^{(t+1)}$ out of candidates $\mathcal{A}_{L(i)}^{(t+1)}$, $i \in \mathcal{I}$, which leads to the biggest improvement. The set of candidates contains also the line segment which is used by the Kozinec Algorithm 13. This implies that the bound on the maximal number of iterations derived for Kozinec Algorithm 13 holds also for Kowalczyk Algorithm 14. ■

4.6.3 Mitchell-Demyanov-Malozemov algorithm for GMNP

The Mitchell-Demyanov-Malozemov (MDM) algorithm reads:

Algorithm 15: MDM Algorithm for GMNP

1. Initialization. Set $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$.

2. Repeat until stopping condition is satisfied:

a) Construct vector $\boldsymbol{\beta}^{(t)} \in \mathcal{A}$ such that

$$[\boldsymbol{\beta}^{(t)}]_i = \begin{cases} [\boldsymbol{\alpha}^{(t)}]_u + [\boldsymbol{\alpha}^{(t)}]_v & \text{for} \quad i = u\,, \\ 0 & \text{for} \quad i = v\,, \\ [\boldsymbol{\alpha}^{(t)}]_i & \text{for} \quad i \neq u \wedge i \neq v, \end{cases} \qquad \forall i \in \mathcal{I}\,, \tag{4.59}$$

where

$$u \in \operatorname*{argmin}_{i\in\mathcal{I}} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i\,, \quad \text{and} \quad v \in \operatorname*{argmax}_{i\in\mathcal{I}_\emptyset} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i\,. \tag{4.60}$$

b) Update

$$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1-\tau) + \tau\boldsymbol{\beta}^{(t)}\,,$$

where

$$\tau = \min\left(1, \frac{\langle(\boldsymbol{\alpha}^{(t)} - \boldsymbol{\beta}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle}{\langle\boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle - 2\langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle}\right)\,.$$

The MDM Algorithm 15 approximates the feasible set $\mathcal{A}$ by a line segment $\mathcal{A}_L^{(t+1)}$ constructed between the current solution $\boldsymbol{\alpha}^{(t)}$ and a vector $\boldsymbol{\beta}^{(t)} \in \mathcal{A}$. The vector $\boldsymbol{\beta}^{(t)}$ equals to the current solution $\boldsymbol{\alpha}^{(t)}$ except for two entries u and v. The indices u and v are determined to maximize the quantity

$$\kappa(u,v) = [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_v - [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_u\,. \tag{4.61}$$

This means that

$$(u,v) \in \operatorname*{argmax}_{(u',u')\in\mathcal{I}\times\mathcal{I}} \kappa(u',v')\,, \tag{4.62}$$

which has the solution given by (4.60). It will be shown below that $\kappa(u,v)$ approximates value of the improvement $\Delta^{(t+1)} = Q(\boldsymbol{\alpha}^{(t)}) - Q(\boldsymbol{\alpha}^{(t+1)})$ which should be maximized. Notice that the index v must be from $\mathcal{I}_\emptyset = \{i \in \mathcal{I}\colon [\boldsymbol{\alpha}^{(t)}]_i > 0\}$ otherwise $\boldsymbol{\beta}^{(t)}$ would equal to $\boldsymbol{\alpha}^{(t)}$. Further, it is easy to see that $\kappa(u,v) > 0$ except for $\boldsymbol{\alpha}^{(\boldsymbol{t})}$ equal to the optimum which follows from the KKT conditions (4.9). The value of the derivative of Q_L (c.f. (4.28)) for $\boldsymbol{\beta}^{(t)}$ constructed by (4.59) reads

$$\left.\frac{\partial Q_L(\tau)}{\partial\tau}\right|_{\tau=0} = [\boldsymbol{\alpha}^{(t)}]_v\left([\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_u - [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_v\right) < 0\,. \tag{4.63}$$

The negative sign of the derivative (4.63) follows from $\kappa(u,v) > 0$ and thus the optimization over the line segment must lead to an improvement in the optimized criterion.

MDM Algorithm 15 stops after a finite number of iterations if the ε-optimality condition (4.7) is applied:

Theorem 4.3 *MDM Algorithm 15 started from an arbitrary vector $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$ returns the vector $\boldsymbol{\alpha}$ satisfying for any $\varepsilon > 0$ the ε-optimality condition*

$$Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}^*) \leq \varepsilon\,, \tag{4.64}$$

after at most $t_{max} < \infty$ iterations, where

$$t_{max} = \frac{2D^2(m-1)}{\varepsilon^2}(Q(\boldsymbol{\alpha}^{(0)}) - Q(\boldsymbol{\alpha}^*))\,. \tag{4.65}$$

PROOF: The proof is based on showing that in each iteration the improvement $\Delta^{(t+1)} = Q(\boldsymbol{\alpha}^{(t)}) - Q(\boldsymbol{\alpha}^{(t+1)})$ is bounded by

$$\Delta^{(t+1)} \geq \frac{\varepsilon^2}{2D^2} > 0 \,, \tag{4.66}$$

except for the case when $\tau = 1$ which can, however, happen at most $(m-1)$ times in a line. The violation of the stopping condition (4.64) implies (c.f. Section 4.3) that

$$\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)} \rangle + \langle \boldsymbol{\alpha}^{(t)}, \boldsymbol{c} \rangle - \min_{i \in \mathcal{I}} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i > \varepsilon \,, \tag{4.67}$$

The improvement $\Delta^{(t+1)}$ of the optimization over the line segment is given by (4.31) for $\tau < 1$. Substituting the vector $\boldsymbol{\beta}^{(t)}$ constructed by the rule (4.59) to formula (4.31) yields the value of the improvement equal to

$$\Delta^{(t+1)} = \frac{([\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_v - [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_u)^2}{2([\mathrm{H}]_{u,u} - 2[\mathrm{H}]_{u,v} + [\mathrm{H}]_{v,v})} \,, \tag{4.68}$$

where $u = \max_{i \in \mathcal{I}_\emptyset} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i$ and $u = \min_{i \in \mathcal{I}} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i$. The inequality (4.67) can be used to derive an upper bound on the numerator of (4.68) because

$$\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)} \rangle + \langle \boldsymbol{c}, \boldsymbol{\alpha}^{(t)} \rangle \leq \max_{i \in \mathcal{I}_\emptyset} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i \,,$$

holds. Similarly, the squared diameter D^2 can used as an upper bound on the denominator of (4.68) which follows from (4.45). The combination of (4.67), (4.45) and (4.68) gives directly the bound (4.66) on the minimal guaranteed improvement for the case $\tau < 1$.

The minimal improvement bound (4.66) holds for the case $\tau < 1$. For the case $\tau = 1$, a simple bound cannot be derived. However, it can be shown that the case $\tau = 1$ cannot occur more than $(m-1)$ times in a line. This follows from the fact that an application of rule (4.59) for $\tau = 1$ sets the v-th entry of $\boldsymbol{\alpha}^{(t)}$ to zero which, however, can happen at most $(m-1)$ times, otherwise $\boldsymbol{\alpha}^{(t+1)}$ would not lie in the feasible set $\mathcal{A}$. Using this reasoning and the bound (4.66) yields the inequality

$$Q(\boldsymbol{\alpha}^*) \leq Q(\boldsymbol{\alpha}^{(t)}) \leq Q(\boldsymbol{\alpha}^{(0)}) - \frac{\varepsilon^2}{2D^2} \left(\frac{t}{m-1} \right) \,,$$

which can be further rearranged to

$$t \leq \frac{2D^2(m-1)}{\varepsilon^2} (Q(\boldsymbol{\alpha}^{(0)}) - Q(\boldsymbol{\alpha}^*)) \,,$$

which ends the proof.

■

4.6.4 Improved Mitchell-Demyanov-Malozemov algorithm for GMNP

This section describes a novel method proposed in this thesis which is based on the MDM Algorithm 15. The proposed improvement concerns the rule for construction of the vector

$\boldsymbol{\beta}^{(t)}$ in Step 2(a) of the MDM algorithm. The original rule constructs the vector $\boldsymbol{\beta}^{(t)}$ such that it equals to the current solution $\boldsymbol{\alpha}^{(t)}$ except for two entries u and v, i.e.,

$$[\boldsymbol{\beta}^{(t)}]_i = \begin{cases} [\boldsymbol{\alpha}^{(t)}]_u + [\boldsymbol{\alpha}^{(t)}]_v & \text{for} \quad i = u\,, \\ 0 & \text{for} \quad i = v\,, \\ [\boldsymbol{\alpha}^{(t)}]_i & \text{for} \quad i \neq u \wedge i \neq v, \end{cases} \qquad \forall i \in \mathcal{I}\,. \tag{4.69}$$

If proper u and v are used then the optimization over the line segment between $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\beta}^{(t)}$ leads to an improvement $\Delta^{(t+1)}(u,v) = Q(\boldsymbol{\alpha}^{(t)}) - Q(\boldsymbol{\alpha}^{(t+1)})$. The exact value of the improvement can be derived by substituting (4.69) to (4.31) which for $\tau < 1$ gives

$$\Delta^{(t+1)}(u,v) = \frac{([\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_v - [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_u)^2}{2([\mathrm{H}]_{u,u} - 2[\mathrm{H}]_{u,v} + [\mathrm{H}]_{v,v})}\,, \tag{4.70}$$

and substituting (4.69) to (4.32) which for $\tau = 1$ gives

$$\Delta^{(t+1)}(u,v) = [\boldsymbol{\alpha}^{(t)}]_v([\mathrm{H}\boldsymbol{\alpha}^{(t)}+\boldsymbol{c}]_v-[\mathrm{H}\boldsymbol{\alpha}^{(t)}+\boldsymbol{c}]_u)-\frac{1}{2}[\boldsymbol{\alpha}^{(t)}]_v^2([\mathrm{H}]_{u,u}-2[\mathrm{H}]_{u,v}+[\mathrm{H}]_{v,v})\,. \tag{4.71}$$

From (4.63) it follows that u and v must be selected such that $[\boldsymbol{\alpha}^{(t)}]_v > 0$ and the inequality

$$\kappa(u,v) = [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_v - [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_u > 0\,,$$

holds to guarantee a non-zero improvement. The computation of τ for $\boldsymbol{\beta}^{(t)}$ given by (4.69) simplifies to

$$\tau = \frac{[\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_v - [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_u}{[\boldsymbol{\alpha}^{(t)}]_v([\mathrm{H}]_{u,u} - 2[\mathrm{H}]_{u,v} + [\mathrm{H}]_{v,v})}\,. \tag{4.72}$$

The quantity $\kappa(u,v)$ can be seen as an approximation of the exact formulas (4.70) and (4.71) for the improvement. The MDM algorithm selects u and v so that $\kappa(u,v)$ is maximized.

A novel method proposed here is based on searching for the entries u and v which maximize the exact value of the improvement instead of its approximation like in the original MDM algorithm. The computation of the improvement for given u and v requires evaluation of the (4.72) and, based on the value of τ, (4.70) or (4.71) is used. To select the optimal pair (u,v) one has to try $\frac{d(d+1)}{2}$ combinations where d is the number of non-zero entries of the current solution $\boldsymbol{\alpha}^{(t)}$. Moreover, the search for the optimal (u,v) would require to access d columns of the matrix H which would be too expensive (this point will be discussed in Section 4.8). To overcome this difficulty, the following strategy of selecting (u,v) is proposed. The index u is computed by the same rule as used in the MDM algorithm, i.e.,

$$u = \operatorname*{argmin}_{i \in \mathcal{I}} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i\,.$$

For given u the index v is computed such that

$$v = \operatorname*{argmax}_{i \in \mathcal{I}_V} \Delta^{(t+1)}(u,i)\,,$$

where $\mathcal{I}_V = \{i \in \mathcal{I} \colon [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i > [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_u \wedge [\boldsymbol{\alpha}^{(t)}]_i > 0\}$ is a set of admissible indices for v for which the improvement can be greater than zero. A similar strategy would be to fix v and search for the optimal u or to apply both these searches together. All these three combinations were experimentally tested and the proposed strategy required on average the minimal access to the matrix H. The proposed strategy requires neglectable computational augment compared to the original MDM algorithm and it requires no extra access to the matrix H which was the main requirement. Algorithm 16 summarizes the proposed improvement.

Algorithm 16: Improved MDM Algorithm for GMNP

1. Initialization. Set $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$.
2. Repeat until stopping condition is satisfied:
    a) Construct vector $\boldsymbol{\beta}^{(t)} \in \mathcal{A}$ such that

$$[\boldsymbol{\beta}^{(t)}]_i = \begin{cases} [\boldsymbol{\alpha}^{(t)}]_u + [\boldsymbol{\alpha}^{(t)}]_v & \text{for} \quad i = u\,, \\ 0 & \text{for} \quad i = v\,, \\ [\boldsymbol{\alpha}^{(t)}]_i & \text{for} \quad i \neq u \wedge i \neq v, \end{cases} \qquad \forall i \in \mathcal{I}\,,$$

    where

$$u \in \operatorname*{argmin}_{i \in \mathcal{I}} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i\,, \quad \text{and} \quad v \in \operatorname*{argmax}_{i \in \mathcal{I}_V} \Delta^{(t+1)}(u,i)\,.$$

    The improvement $\Delta^{(t+1)}(u,i)$ is computed using formulas (4.72), (4.70) and (4.71).

    b) Update

$$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1-\tau) + \tau\boldsymbol{\beta}^{(t)}\,,$$

    where

$$\tau = \min\left(1, \frac{\langle(\boldsymbol{\alpha}^{(t)} - \boldsymbol{\beta}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle}{\langle\boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle - 2\langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle}\right)\,.$$

The convergence Theorem 4.3 introduced for the original MDM algorithm holds obviously for the improved version as well.

4.6.5 Keerthi algorithm for GMNP

The Keerthi Algorithm 15 approximates the feasible set $\mathcal{A}$ by a triangle $\mathcal{A}_T^{(t+1)}$ defined by (4.33). The triangle $\mathcal{A}_T^{(t+1)}$ is constructed between the current solution $\boldsymbol{\alpha}^{(t)}$ and the vectors $\boldsymbol{\beta}^{(t)}$, $\boldsymbol{\gamma}^{(t)} \in \mathcal{A}$. The vector $\boldsymbol{\beta}^{(t)}$ is defined by the rule (4.43) used in the Kozinec algorithm and the vector $\boldsymbol{\gamma}^{(t)}$ by the rule (4.59) used in the MDM algorithm. It was shown in Section 4.6.1 and Section 4.6.3 that the optimization over line segments between vectors ($\boldsymbol{\alpha}^{(t)}$, $\boldsymbol{\beta}^{(t)}$) and ($\boldsymbol{\alpha}^{(t)}$, $\boldsymbol{\gamma}^{(t)}$), respectively, leads to an improvement in the optimized criterion Q. It is obvious that the optimization over the triangle $\mathcal{A}_T^{(t+1)}$ must lead to an improvement as well because the line segments are its subsets. The optimization of Q over the triangle $\mathcal{A}_T^{(t+1)}$ can be computed analytically by Algorithm 12 as described in Section 4.5.

Algorithm 17: Keerthi Algorithm for GMNP

```
1. Initialization. Set $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$.
2. Repeat until stopping condition is satisfied:
   a) Construct vector $\boldsymbol{\beta}^{(t)} \in \mathcal{A}$ by rule (4.43) and vector $\boldsymbol{\gamma}^{(t)} \in \mathcal{A}$ by rule (4.59)
      which define a triangle (4.33).
   b) Solve the subtask
      $$\boldsymbol{\alpha}^{(t+1)} = \operatorname*{argmin}_{\boldsymbol{\alpha} \in \mathcal{A}_T^{(t+1)}} Q(\boldsymbol{\alpha}),$$
      using Algorithm 12.
```

Keerthi Algorithm 17 stops after a finite number of iterations if the ε-optimality condition (4.7) is applied:

Theorem 4.4 *Keerthi Algorithm 17 started from an arbitrary vector $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$ returns the vector $\boldsymbol{\alpha}$ satisfying for any $\varepsilon > 0$ the ε-optimality condition*

$$Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}^*) \leq \varepsilon \,, \tag{4.73}$$

after at most $t_{max} < \infty$ iterations, where

$$t_{max} = \frac{2D^2}{\varepsilon^2}(Q(\boldsymbol{\alpha}^{(0)}) - Q(\boldsymbol{\alpha}^*)) + m \,. \tag{4.74}$$

PROOF: Keerthi Algorithm 17 in each iteration optimizes the criterion Q over a triangle $\boldsymbol{\alpha}^{(t)}$, $\boldsymbol{\beta}^{(t)}$ and $\boldsymbol{\gamma}^{(t)}$. Kozinec Algorithm 13 optimizes only over the line segment between vector $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\beta}^{(t)}$. Because the line segment is a subset of the triangle then the improvement of Keerthi Algorithm 17 cannot be smaller than that of Kozinec Algorithm 13 its improvement can be bounded from bellow by (4.50). This implies that the bound on the maximal number of iterations derived for Kozinec Algorithm 13 holds also for Keerthi Algorithm 17. ■

4.7 Algorithms for the generalized nearest point problem

The GNPP to solve is

$$\boldsymbol{\alpha}^* = \operatorname*{argmin}_{\boldsymbol{\alpha} \in \mathcal{A}} \left(\frac{1}{2}\langle \boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha} \rangle + \langle \boldsymbol{c}, \boldsymbol{\alpha} \rangle \right) , \tag{4.75}$$

where the feasible set is defined by

$$\mathcal{A} = \{ \boldsymbol{\alpha} \in \mathbb{R}^m \colon \langle \boldsymbol{\alpha}, \boldsymbol{e}_1 \rangle = 1, \langle \boldsymbol{\alpha}, \boldsymbol{e}_2 \rangle = 1 \,, \boldsymbol{\alpha} \geq 0 \} \,. \tag{4.76}$$

It is convenient to introduce the following notation

$$\boldsymbol{\alpha}^{(t)} = \begin{bmatrix} \boldsymbol{\alpha}_1^{(t)} \\ \boldsymbol{\alpha}_2^{(t)} \end{bmatrix} , \quad \boldsymbol{\beta}^{(t)} = \begin{bmatrix} \boldsymbol{\beta}_1^{(t)} \\ \boldsymbol{\beta}_2^{(t)} \end{bmatrix} , \quad \mathrm{H} = \begin{bmatrix} \mathrm{H}_{11} & \mathrm{H}_{12} \\ \mathrm{H}_{21} & \mathrm{H}_{22} \end{bmatrix} , \quad \boldsymbol{c} = \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{bmatrix} ,$$

where the vectors $\boldsymbol{\alpha}_1^{(t)}$ and $\boldsymbol{\alpha}_2^{(t)}$ contain entries of the vector $\boldsymbol{\alpha}^{(t)}$ with indices from $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively. Using this notation allows to rewrite the objective function $Q(\boldsymbol{\alpha}^{(t)})$ as

$$\begin{aligned} Q(\boldsymbol{\alpha}^{(t)}) &= \frac{1}{2}\Big(\langle \boldsymbol{\alpha}_1^{(t)}, \mathrm{H}_{11}\boldsymbol{\alpha}_1^{(t)}\rangle + 2\langle \boldsymbol{\alpha}_1^{(t)}, \mathrm{H}_{12}\boldsymbol{\alpha}_2^{(t)}\rangle + \langle \boldsymbol{\alpha}_2^{(t)}, \mathrm{H}_{22}\boldsymbol{\alpha}_2^{(t)}\rangle\Big) \\ &\quad + \langle \boldsymbol{\alpha}_1^{(t)}, \boldsymbol{c}_1\rangle + \langle \boldsymbol{\alpha}_2^{(t)}, \boldsymbol{c}_2\rangle \,. \end{aligned}$$

Further, let $\mathcal{A}_1^{(t+1)} = \{\boldsymbol{\alpha} \in \mathcal{A} \colon [\boldsymbol{\alpha}]_i = [\boldsymbol{\alpha}^{(t)}]_i, i \in \mathcal{I}_2\}$ denote the set of vectors from the feasible set $\mathcal{A}$ which have the entries $\mathcal{I}_2$ fixed to corresponding entries of the current solution $\boldsymbol{\alpha}_2^{(t)}$. Similarly, let $\mathcal{A}_2^{(t+1)} = \{\boldsymbol{\alpha} \in \mathcal{A} \colon [\boldsymbol{\alpha}]_i = [\boldsymbol{\alpha}^{(t)}]_i, i \in \mathcal{I}_1\}$ be vectors with the entries $\mathcal{I}_1$ fixed.

The algorithms to solve the GNPP are of the same face as those solving the GMNP problem (c.f. Section 4.6). Algorithms for the GNPP require a modification of the rules used to construct the auxiliary feasible set. The rules are modified in such a way that the vector $\boldsymbol{\beta}^{(t)}$ which determines the auxiliary feasible set belongs either to $\mathcal{A}_1^{(t+1)}$ or to $\mathcal{A}_2^{(t+1)}$. For instance, let the auxiliary set be a line segment between the vectors $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\beta}^{(t)}$. A rule used in an algorithm for GMNP is applied to construct the vector $\boldsymbol{\beta}_1^{(t)}$ and to set the vector $\boldsymbol{\beta}_2^{(t)}$ to the current solution $\boldsymbol{\alpha}_2^{(t)}$ or it proceeds vice versa.

4.7.1 Kozinec algorithm for GNPP

The Kozinec Algorithm 18 for the GNPP uses again the line segment approximation constructed between the current solution vector $\boldsymbol{\alpha}^{(t)}$ and the vector $\boldsymbol{\beta}^{(t)}$. The vector $\boldsymbol{\beta}^{(t)}$ is constructed to minimize the derivative of Q_L assuming that $\boldsymbol{\beta}^{(t)}$ belongs either to the set $\mathcal{A}_1^{(t+1)}$ or to $\mathcal{A}_2^{(t+1)}$. The derivative of Q_L at zero is for general $\boldsymbol{\beta}^{(t)}$ given by (4.28). In the case when $\boldsymbol{\beta}^{(t)} \in \mathcal{A}_1^{(t+1)}$, the minimal value of the derivative (4.28) equals

$$\begin{aligned} \mathrm{d}Q_1 &= \min_{\boldsymbol{\beta}^{(t)} \in \mathcal{A}_1^{(t+1)}} \langle (\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle \\ &= \min_{i \in \mathcal{I}_1} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i - \langle \boldsymbol{\alpha}_1^{(t)}, (\mathrm{H}_{11}\boldsymbol{\alpha}_1^{(t)} + \mathrm{H}_{12}\boldsymbol{\alpha}_2^{(t)} + \boldsymbol{c}_1)\rangle \,, \end{aligned} \tag{4.77}$$

and the vector $\boldsymbol{\beta}^{(t)} \in \mathcal{A}_1^{(t+1)}$ is constructed by (4.79). Similarly, for the vector $\boldsymbol{\beta}^{(t)} \in \mathcal{A}_2^{(t+1)}$ the minimal value of the derivative (4.28) equals to

$$\begin{aligned} \mathrm{d}Q_2 &= \min_{\boldsymbol{\beta}^{(t)} \in \mathcal{A}_2^{(t+1)}} \langle (\boldsymbol{\beta}^{(t)} - \boldsymbol{\alpha}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle \\ &= \min_{i \in \mathcal{I}_2} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i - \langle \boldsymbol{\alpha}_2^{(t)}, (\mathrm{H}_{21}\boldsymbol{\alpha}_1^{(t)} + \mathrm{H}_{22}\boldsymbol{\alpha}_2^{(t)} + \boldsymbol{c}_2)\rangle \,, \end{aligned} \tag{4.78}$$

and the vector $\boldsymbol{\beta}^{(t)} \in \mathcal{A}_2^{(t+1)}$ is constructed by the rule (4.80). It can be simply verified that both the derivatives (4.77) and (4.78) are negative unless $\boldsymbol{\alpha}^{(\boldsymbol{t})}$ is at the optimum. Thus optimization over the corresponding line segments leads to an improvement of the optimized criterion.

Algorithm 18: Kozinec Algorithm for GNPP

1. Initialization. Set $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$.

2. Repeat until stopping condition is satisfied:

   a) Compute derivatives $\mathrm{d}Q_1$ using (4.77) and $\mathrm{d}Q_2$ using (4.78). If $\mathrm{d}Q_1 \le \mathrm{d}Q_2$ then construct the vector $\boldsymbol{\beta}^{(t)}$ by the rule

$$[\boldsymbol{\beta}^{(t)}]_i = \begin{cases} 1 & \text{for} \quad i = u_1 \wedge i \in \mathcal{I}_1 \,, \\ 0 & \text{for} \quad i \ne u_1 \wedge i \in \mathcal{I}_1 \,, \\ [\boldsymbol{\alpha}^{(t)}]_i & \text{for} \quad i \in \mathcal{I}_2 \,, \end{cases} \tag{4.79}$$

   where $u_1 \in \operatorname*{argmin}_{i \in \mathcal{I}_1} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i$. Otherwise, if $\mathrm{d}Q_1 > \mathrm{d}Q_2$, use the rule

$$[\boldsymbol{\beta}^{(t)}]_i = \begin{cases} 1 & \text{for} \quad i = u_2 \wedge i \in \mathcal{I}_2 \,, \\ 0 & \text{for} \quad i \ne u_2 \wedge i \in \mathcal{I}_2 \,, \\ [\boldsymbol{\alpha}^{(t)}]_i & \text{for} \quad i \in \mathcal{I}_1 \,, \end{cases} \tag{4.80}$$

   where $u_2 \in \operatorname*{argmin}_{i \in \mathcal{I}_2} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i$.

   b) Update

$$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1-\tau) + \tau\boldsymbol{\beta}^{(t)} \,,$$

   where

$$\tau = \min\left(1, \frac{\langle(\boldsymbol{\alpha}^{(t)} - \boldsymbol{\beta}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle}{\langle\boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle - 2\langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle}\right) \,.$$

Kozinec Algorithm 18 stops after a finite number of iterations if the ε-optimality condition (4.7) is applied. The upper bound on the maximal number of iterations requires a definition of the diameter D of the matrix H with respect to the constraints (4.76). This is defined as

$$D^2 = \max\left(D_1^2, D_2^2\right) \,, \tag{4.81}$$

where

$$\begin{aligned} D_1^2 &= \operatorname*{argmax}_{\boldsymbol{\alpha}_1 \in \mathcal{A}_1, \boldsymbol{\beta}_1 \in \mathcal{A}_1} \left(\langle\boldsymbol{\alpha}_1, \mathrm{H}_{11}\boldsymbol{\alpha}_1\rangle - 2\langle\boldsymbol{\alpha}_1, \mathrm{H}_{11}\boldsymbol{\beta}_1\rangle + \langle\boldsymbol{\beta}_1, \mathrm{H}_{11}\boldsymbol{\beta}_1\rangle\right) > 0 \,, \\ D_2^2 &= \operatorname*{argmax}_{\boldsymbol{\alpha}_2 \in \mathcal{A}_2, \boldsymbol{\beta}_2 \in \mathcal{A}_2} \left(\langle\boldsymbol{\alpha}_2, \mathrm{H}_{22}\boldsymbol{\alpha}_2\rangle - 2\langle\boldsymbol{\alpha}_2, \mathrm{H}_{22}\boldsymbol{\beta}_2\rangle + \langle\boldsymbol{\beta}_2, \mathrm{H}_{22}\boldsymbol{\beta}_2\rangle\right) > 0 \,. \end{aligned}$$

The sets $\mathcal{A}_1$ and $\mathcal{A}_2$ are defined so that for any $\boldsymbol{\alpha}_1 \in \mathcal{A}_1$ and $\boldsymbol{\alpha} \in \mathcal{A}_2$ the vector $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_1; \boldsymbol{\alpha}_2]$ belongs to $\mathcal{A}$. The positivity of D_1 and D_2 follows from the positive definiteness of the matrix H.

Theorem 4.5 *Kozinec Algorithm 18 started from an arbitrary vector $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$ returns the vector $\boldsymbol{\alpha}$ satisfying for any $\varepsilon > 0$ the ε-optimality condition*

$$Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}^*) \le \varepsilon \,, \tag{4.82}$$

after at most $t_{max} < \infty$ iterations, where

$$t_{max} = \frac{8D^2}{\varepsilon^2}(Q(\boldsymbol{\alpha}^{(0)}) - Q(\boldsymbol{\alpha}^*)) + 2m \,. \tag{4.83}$$

PROOF: The proof is very like the proof of Theorem 4.1 for the GMNP. The only difference is the bound on the minimal improvement

$$\Delta^{(t+1)} \geq \frac{\varepsilon^2}{8D^2} > 0 \,, \tag{4.84}$$

which is derived as follows. The violation of the stopping condition (4.82) implies (c.f. Section 4.3) that the inequality

$$\langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)} \rangle + \langle \boldsymbol{\alpha}^{(t)}, \boldsymbol{c} \rangle - \min_{i \in \mathcal{I}_1} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i - \min_{i \in \mathcal{I}_2} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i > \varepsilon \tag{4.85}$$

holds. The improvement of the optimization over a line segment is given by (4.31) for $\tau < 1$. Substituting the vector $\boldsymbol{\beta}^{(t)}$ constructed by rule (4.79) to formula (4.31) yields the value of the improvement equal to

$$\Delta^{(t+1)} = \frac{([\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_{u_1} - \langle \boldsymbol{\alpha}_1^{(t)}, (\mathrm{H}_{11}\boldsymbol{\alpha}_1^{(t)} + \mathrm{H}_{12}\boldsymbol{\alpha}_2^{(t)} + \boldsymbol{c}_1) \rangle)^2}{2(\langle \boldsymbol{\alpha}_1^{(t)}, \mathrm{H}_{11}\boldsymbol{\alpha}_1^{(t)} \rangle - 2[\mathrm{H}_{11}\boldsymbol{\alpha}_1^{(t)}]_{u_1} + [\mathrm{H}_{11}]_{u_1,u_1})} \,, \quad \text{for} \quad \tau < 1 \,, \tag{4.86}$$

where $u_1 = \min_{i \in \mathcal{I}_1} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i$. The numerator of (4.86) equals to squared value of the derivative $\mathrm{d}Q_1$ (c.f. (4.77)) which can be bounded using inequality (4.85). Condition (4.85) can be rewritten as

$$\begin{aligned} &\underbrace{\langle \boldsymbol{\alpha}_1^{(t)}, (\mathrm{H}_{11}\boldsymbol{\alpha}_1^{(t)} + \mathrm{H}_{12}\boldsymbol{\alpha}_2^{(t)} + \boldsymbol{c}_1) \rangle - \min_{i \in \mathcal{I}_1} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i}_{-\mathrm{d}Q_1} \\ + &\underbrace{\langle \boldsymbol{\alpha}_2^{(t)}, (\mathrm{H}_{21}\boldsymbol{\alpha}_1^{(t)} + \mathrm{H}_{22}\boldsymbol{\alpha}_2^{(t)} + \boldsymbol{c}_2) \rangle - \min_{i \in \mathcal{I}_2} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i}_{-\mathrm{d}Q_2} \quad > \quad \varepsilon \,. \end{aligned} \tag{4.87}$$

The rule (4.79) is applied if $\mathrm{d}Q_1 < \mathrm{d}Q_2$ which allows to write

$$-\mathrm{d}Q_1 = \langle \boldsymbol{\alpha}_1^{(t)}, (\mathrm{H}_{11}\boldsymbol{\alpha}_1^{(t)} + \mathrm{H}_{12}\boldsymbol{\alpha}_2^{(t)} + \boldsymbol{c}_1) \rangle - \min_{i \in \mathcal{I}_1} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i \geq \frac{\varepsilon}{2} \,. \tag{4.88}$$

Next, the squared diameter D^2 can be used as an upper bound on the denominator of (4.86) which follows from the definition (4.81). The combination of (4.88), (4.81) and (4.86) gives directly the bound (4.84) on the minimal improvement for the case $\tau < 0$ and the rule (4.79). The same bound can be derived for the case $\tau < 0$ and the rule (4.80).

The case $\tau = 1$ can occur at most $2m$ times using the same reasoning as in the proof of Theorem 4.1. Using the bound (4.84) and the fact that the improvement of $2m$ iterations cannot be bounded yields the inequality

$$Q(\boldsymbol{\alpha}^*) \leq Q(\boldsymbol{\alpha}^{(t)}) \leq Q(\boldsymbol{\alpha}^{(0)}) - \frac{\varepsilon^2}{8D^2}(t - 2m) \,,$$

which can be further rearranged to

$$t \leq \frac{8D^2}{\varepsilon^2}(Q(\boldsymbol{\alpha}^{(0)}) - Q(\boldsymbol{\alpha}^*)) + 2m \,,$$

which ends the proof.

■

4.7.2 Kowalczyk algorithm for GNPP

Kowalczyk Algorithm 19 for the GNPP uses the line segment approximation constructed between the current solution $\boldsymbol{\alpha}^{(t)}$ and the vector $\boldsymbol{\beta}^{(t)}$. First, a set of candidate vectors $\boldsymbol{\beta}_i^{(t)}$, $i \in \mathcal{I}_1 \cup \mathcal{I}_2$ is constructed using rules (4.90) and (4.91). Second, the vector $\boldsymbol{\beta}_i^{(t)}$ which yields the biggest improvement $\Delta_i^{(t+1)}$ is applied. The candidate vector $\boldsymbol{\beta}_i^{(t)}$ is constructed such that it belongs either to the set $\mathcal{A}_1^{(t+1)}$ if $i \in \mathcal{I}_1$ or to $\mathcal{A}_2^{(t+1)}$ if $i \in \mathcal{I}_2$. It can be simply shown that for given $i \in \mathcal{I}_1 \cup \mathcal{I}_2$ just one of the rules (4.90) and (4.91) leads to an improvement. To decide which rule improves the criterion, the sign of the derivative (4.28) is used. The exact value of the improvement is computed for given $\boldsymbol{\beta}_i^{(t)}$ analytically using formulas (4.30), (4.31), and (4.32).

Algorithm 19: Kowalczyk Algorithm for GNPP

1. Initialization. Set $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$.

2. Repeat until stopping condition is satisfied:

   a) For all $i \in \mathcal{I}$ compute an improvement

$$\Delta_i^{(t+1)} = Q(\boldsymbol{\alpha}^{(t)}) - \min_{\boldsymbol{\alpha} \in \mathcal{A}_{L(i)}^{(t+1)}} Q(\boldsymbol{\alpha}) \,, \tag{4.89}$$

   where $\mathcal{A}_{L(i)}^{(t+1)} = \{\boldsymbol{\alpha} \in \mathbb{R}^m \colon \boldsymbol{\alpha} = \boldsymbol{\alpha}^{(t)}(1-\tau) + \tau\boldsymbol{\beta}_i^{(t)}, \tau \in [0,1]\}$. The vectors $\boldsymbol{\beta}_i^{(t)}$, $i \in \mathcal{I}_1$ are constructed (i) by the rule

$$[\boldsymbol{\beta}_i^{(t)}]_j = \left\{ \begin{array}{cll} 1 & \text{for} & j = i \wedge j \in \mathcal{I}_1 \,, \\ 0 & \text{for} & j \neq i \wedge j \in \mathcal{I}_1 \,, \\ [\boldsymbol{\alpha}^{(t)}]_j & \text{for} & j \in \mathcal{I}_2 \,, \end{array} \right. \tag{4.90}$$

   if $[\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i < \langle \boldsymbol{\alpha}_1^{(t)}, (\mathrm{H}_{11}\boldsymbol{\alpha}_1^{(t)} + \mathrm{H}_{12}\boldsymbol{\alpha}_2^{(t)} + \boldsymbol{c}_1\rangle$ or (ii) by the rule

$$[\boldsymbol{\beta}_i^{(t)}]_j = \left\{ \begin{array}{cll} 0 & \text{for} & j = i \wedge i \in \mathcal{I}_1 \,, \\ \frac{[\boldsymbol{\alpha}^{(t)}]_j}{1-[\boldsymbol{\alpha}^{(t)}]_i} & \text{for} & j \neq i \wedge i \in \mathcal{I}_1 \,, \\ [\boldsymbol{\alpha}^{(t)}]_j & \text{for} & j \in \mathcal{I}_2 \,, \end{array} \right. \tag{4.91}$$

   if $[\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i > \langle \boldsymbol{\alpha}_1^{(t)}, (\mathrm{H}_{11}\boldsymbol{\alpha}_1^{(t)} + \mathrm{H}_{12}\boldsymbol{\alpha}_2^{(t)} + \boldsymbol{c}_1\rangle$ and $[\boldsymbol{\alpha}^{(t)}]_i < 1$. Otherwise set $\Delta_i^{(t+1)} = 0$.

   The vectors $\boldsymbol{\beta}_i^{(t)}$, $i \in \mathcal{I}_2$ are constructed by the same rules (4.90) and (4.91) except for the sets $\mathcal{I}_1$ and $\mathcal{I}_2$ being interchanged. The "if conditions" are changed to $[\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i \underset{>}{\overset{<}{}} \langle \boldsymbol{\alpha}_2^{(t)}, (\mathrm{H}_{21}\boldsymbol{\alpha}_1^{(t)} + \mathrm{H}_{22}\boldsymbol{\alpha}_2^{(t)} + \boldsymbol{c}_2\rangle$. Finally select

$$r \in \operatorname*{argmax}_{i \in \mathcal{I}} \Delta_i^{(t+1)} \,.$$

b) Update

$$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1-\tau) + \tau\boldsymbol{\beta}_r^{(t)} ,$$

where

$$\tau = \min\left(1, \frac{\langle(\boldsymbol{\alpha}^{(t)} - \boldsymbol{\beta}_r^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle}{\langle\boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle - 2\langle\boldsymbol{\beta}_r^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \langle\boldsymbol{\beta}_r^{(t)}, \mathrm{H}\boldsymbol{\beta}_r^{(t)}\rangle}\right) .$$

Kowalczyk Algorithm 19 stops after a finite number of iterations if the ε-optimality condition (4.7) is applied which is stated by the following Theorem 4.6.

Theorem 4.6 *Kowalczyk Algorithm 19 started from an arbitrary vector $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$ returns the vector $\boldsymbol{\alpha}$ satisfying for any $\varepsilon > 0$ the ε-optimality condition*

$$Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}^*) \leq \varepsilon , ,$$

after at most $t_{max} < \infty$ iterations, where

$$t_{max} = \frac{8D^2}{\varepsilon^2}(Q(\boldsymbol{\alpha}^{(0)}) - Q(\boldsymbol{\alpha}^*)) + 2m .$$

PROOF: The reasoning used in the proof of the Theorem 4.2 is applicable also for the Theorem 4.6. ■

4.7.3 Mitchell-Demyanov-Malozemov algorithm for GNPP

MDM Algorithm 20 for the GNPP uses again the line segment approximation constructed between the current solution vector $\boldsymbol{\alpha}^{(t)}$ and the vector $\boldsymbol{\beta}^{(t)}$ constructed by the rule (4.94). The vector $\boldsymbol{\beta}^{(t)}$ equals to the current solution $\boldsymbol{\alpha}^{(t)}$ except for two entries u and v. Both the indices u and v must belong either to the set $\mathcal{I}_1$ or to $\mathcal{I}_2$ in order to ensure that the vector $\boldsymbol{\beta}^{(t)}$ is from $\mathcal{A}$. The indices are determined to maximize the quantity $\kappa(u,v)$ (c.f. (4.61)), i.e,

$$\begin{aligned}(u,v) &\in \operatorname{argmax}\left(\max_{u'\in\mathcal{I}_1, v'\in\mathcal{I}_1} \kappa(u',v'), \max_{u'\in\mathcal{I}_2, v'\in\mathcal{I}_2} \kappa(u',v')\right) \\ &= \operatorname*{argmax}_{(u',v')\in\{(u_1,v_1),(u_2,v_2)\}} \kappa(u',v') ,\end{aligned} \tag{4.92}$$

where

$$\begin{aligned} u_1 &\in \operatorname*{argmin}_{i\in\mathcal{I}_1}[\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i , \quad v_1 \in \operatorname*{argmax}_{i\in\mathcal{I}_1\cap\mathcal{I}_\emptyset}[\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i , \\ u_2 &\in \operatorname*{argmin}_{i\in\mathcal{I}_2}[\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i , \quad v_2 \in \operatorname*{argmax}_{i\in\mathcal{I}_2\cap\mathcal{I}_\emptyset}[\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i .\end{aligned}$$

It can be seen that $\kappa(u,v)$ is positive unless $\boldsymbol{\alpha}^{(t)}$ is already in the optimum. The vector $\boldsymbol{\beta}^{(t)}$ constructed by (4.94) substituted to the derivative (4.28) yields

$$\left.\frac{\partial Q_L(\tau)}{\partial\tau}\right|_{\tau=0} = [\boldsymbol{\alpha}^{(t)}]_v([\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_u - [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_v) < 0 . \tag{4.93}$$

The negative sign follows from $\kappa(u,v) > 0$ thus the optimization over the line segment must lead to an improvement of the optimized criterion.

Algorithm 20: MDM Algorithm for GNPP

```
1. Initialization. Set $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$.
2. Repeat until stopping condition is satisfied:
    a) Construct vector $\boldsymbol{\beta}^{(t)} \in \mathcal{A}$
       $$[\boldsymbol{\beta}^{(t)}]_i = \begin{cases} [\boldsymbol{\alpha}^{(t)}]_u + [\boldsymbol{\alpha}^{(t)}]_v & \text{for} \quad i = u\,, \\ 0 & \text{for} \quad i = v\,, \\ [\boldsymbol{\alpha}^{(t)}]_i & \text{for} \quad i \neq u \wedge i \neq v, \end{cases} \qquad (4.94)$$
       where the indices $u$ and $v$ are given by (4.92).
    b) Update
       $$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1-\tau) + \tau\boldsymbol{\beta}^{(t)}\,,$$
       where
       $$\tau = \min\left(1, \frac{\langle(\boldsymbol{\alpha}^{(t)} - \boldsymbol{\beta}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle}{\langle\boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle - 2\langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle}\right)\,.$$
```

MDM Algorithm 20 stops after a finite number of iterations if the ε-optimality condition (4.7) is applied which is stated by the following Theorem 4.7.

Theorem 4.7 *MDM Algorithm 20 started from an arbitrary vector $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$ returns the vector $\boldsymbol{\alpha}$ satisfying for any $\varepsilon > 0$ the ε-optimality condition*

$$Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}^*) \leq \varepsilon\,, \tag{4.95}$$

after at most $t_{max} < \infty$ iterations, where

$$t_{max} = \frac{8D^2(m-2)}{\varepsilon^2}(Q(\boldsymbol{\alpha}^{(0)}) - Q(\boldsymbol{\alpha}^*))\,. \tag{4.96}$$

PROOF: The proof is very like the proof of Theorem 4.3 for the GMNP. The only difference is the bound on the minimal improvement

$$\Delta^{(t+1)} \geq \frac{\varepsilon^2}{8D^2} > 0\,, \tag{4.97}$$

which is derived as follows. The improvement for the case $\tau < 1$ is derived by substituting the vector $\boldsymbol{\beta}^{(t)}$ constructed by the rule (4.94) to formula (4.31) which yields

$$\Delta^{(t+1)} = \frac{([\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_v - [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_u)^2}{2([\mathrm{H}]_{u,u} - 2[\mathrm{H}]_{u,v} + [\mathrm{H}]_{v,v})}\,. \tag{4.98}$$

The formula (4.98) holds regardless to which index set ($\mathcal{I}_1$ or $\mathcal{I}_2$) the pair (u, v) belongs to. The violation of condition (4.95) implies (c.f. (4.87))

$$-\mathrm{d}Q_1 - \mathrm{d}Q_2 \geq \varepsilon\,. \tag{4.99}$$

From (4.61) it follows that $-\mathrm{d}Q_1 \leq \kappa(u_1, v_1)$ and $-\mathrm{d}Q_2 \leq \kappa(u_2, v_2)$ which substituted to (4.99), yields

$$\kappa(u_1, v_1) + \kappa(u_2, v_2) \geq \varepsilon \, . \tag{4.100}$$

Thus for (u, v) given by (4.92) the inequality $\kappa(u, v) \geq \frac{\varepsilon}{2}$ holds. Using (4.100) as a bound on the numerator of (4.98) and (4.81) as a bound on the denominator yields (4.97).

The bound (4.97) on minimal improvement holds for the case $\tau < 1$. The improvement for $\tau = 1$ cannot be generally bounded. But the case $\tau = 1$ cannot occur more than $(m-2)$ times in a line as it would lead to a vector $\boldsymbol{\alpha}^{(t)}$ having all entries zero. This would mean that $\boldsymbol{\alpha}^{(t)}$ is not from $\mathcal{A}$. Therefore

$$Q(\boldsymbol{\alpha}^*) \leq Q(\boldsymbol{\alpha}^{(t)}) \leq Q(\boldsymbol{\alpha}^{(0)}) - \frac{\varepsilon^2}{8D^2}\left(\frac{t}{m-2}\right) ,$$

which can be further rearranged to

$$t \leq \frac{8D(m-2)}{\varepsilon^2}(Q(\boldsymbol{\alpha}^{(0)}) - Q(\boldsymbol{\alpha}^*)) ,$$

which ends the proof. ■

4.7.4 Improved Mitchell-Demyanov-Malozemov algorithm for GNPP

The improved version of the MDM Algorithm 4.7.3 for the GNPP employs the same idea as introduced for the GMNP variant (c.f. Section 4.6.4). That means, first the indices u_1 and u_2 are computed as

$$\begin{aligned} u_1 &\in \operatorname*{argmin}_{i \in \mathcal{I}_1} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i \, , \\ u_2 &\in \operatorname*{argmin}_{i \in \mathcal{I}_2} [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i \, . \end{aligned} \tag{4.101}$$

Second, the optimal v_1 and v_2 are sought so that

$$\begin{aligned} v_1 &\in \operatorname*{argmin}_{i \in \mathcal{I}_{V1}} \Delta^{(t+1)}(u_1, i) \, , \\ v_2 &\in \operatorname*{argmin}_{i \in \mathcal{I}_{V2}} \Delta^{(t+1)}(u_2, i) \, , \end{aligned} \tag{4.102}$$

where

$$\begin{aligned} \mathcal{I}_{V1} &= \{i \in \mathcal{I}_1 \colon [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i > [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_{u_1} \wedge [\boldsymbol{\alpha}^{(t)}]_i > 0\} \, , \\ \mathcal{I}_{V2} &= \{i \in \mathcal{I}_2 \colon [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_i > [\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c}]_{u_2} \wedge [\boldsymbol{\alpha}^{(t)}]_i > 0\} \, , \end{aligned}$$

are sets of admissible indices. Finally, the pair of indices (u_1, v_1) or (u_2, v_2) which yields the bigger improvement is used to construct the vector $\boldsymbol{\beta}^{(t)}$. The formula for the improvement $\Delta^{(t+1)}(u, v)$ can be derived using the same way as in the the GMNP case. The improvement is computed by the formula (4.70) if $\tau < 1$ and (4.71) if $\tau = 1$. The proposed method is summarized in Algorithm 21.

Algorithm 21: Improved MDM Algorithm for GNPP

1. Initialization. Set $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$.
2. Repeat until stopping condition is satisfied:

   a) First, the pairs (u_1, v_1) and (u_2, v_2) are computed by (4.101) and (4.102). Second, the pair which yields the bigger improvement $\Delta^{(t+1)}(u, v)$ is taken for (u, v) and the vector $\boldsymbol{\beta}^{(t)} \in \mathcal{A}$ is constructed as

   $$[\boldsymbol{\beta}^{(t)}]_i = \begin{cases} [\boldsymbol{\alpha}^{(t)}]_u + [\boldsymbol{\alpha}^{(t)}]_v & \text{for} \quad i = u\,, \\ 0 & \text{for} \quad i = v\,, \\ [\boldsymbol{\alpha}^{(t)}]_i & \text{for} \quad i \neq u \wedge i \neq v. \end{cases}$$

   b) Update

   $$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1 - \tau) + \tau \boldsymbol{\beta}^{(t)}\,,$$

   where

   $$\tau = \min\left(1, \frac{\langle(\boldsymbol{\alpha}^{(t)} - \boldsymbol{\beta}^{(t)}), (\mathrm{H}\boldsymbol{\alpha}^{(t)} + \boldsymbol{c})\rangle}{\langle\boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle - 2\langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)}\rangle + \langle\boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\beta}^{(t)}\rangle}\right)\,.$$

The convergence Theorem 4.7 introduced for the original MDM algorithm holds obviously for the improved version as well.

4.7.5 Keerthi algorithm for GNPP

The Keerthi Algorithm 22 for the GNPP uses the triangle approximation of the feasible set. The triangle is constructed between the current solution and the vectors $\boldsymbol{\beta}^{(t)}$ and $\boldsymbol{\gamma}^{(t)}$. First, the indices u and v are determined by (4.92) as defined in the MDM Algorithm 20. Second, the vector $\boldsymbol{\beta}^{(t)}$ is constructed using the rules (4.79), (4.80) of the Kozinec Algorithm 18 and the $\boldsymbol{\gamma}^{(t)}$ is constructed using the rule (4.94) of the MDM Algorithm 20.

Algorithm 22: Keerthi Algorithm for GNPP

1. Initialization. Set $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$.
2. Repeat until stopping condition is satisfied:

   a) Determine indices u and v using (4.92). If u and v belong to $\mathcal{I}_1$ then construct the vector $\boldsymbol{\beta}^{(t)}$ using the rule (4.79). Otherwise, if u and v belong to $\mathcal{I}_2$ then construct the vector $\boldsymbol{\beta}^{(t)}$ using the rule (4.80). The vector $\boldsymbol{\gamma}^{(t)}$ is in both the cases constructed using the rule (4.94).

b) Solve the subtask

$$\boldsymbol{\alpha}^{(t+1)} = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathcal{A}_T^{(t+1)}} Q(\boldsymbol{\alpha}) ,$$

using Algorithm 12

Keerthi Algorithm 22 converges to the ε-optimal solution given by the condition (4.7) in finite number of iterations as stated in the following theorem.

Theorem 4.8 *Keerthi Algorithm 22 started from an arbitrary vector $\boldsymbol{\alpha}^{(0)} \in \mathcal{A}$ returns the vector $\boldsymbol{\alpha}$ satisfying for any $\varepsilon > 0$ the ε-optimality condition*

$$Q(\boldsymbol{\alpha}) - Q(\boldsymbol{\alpha}^*) \leq \varepsilon ,$$

after at most $t_{max} < \infty$ iterations, where

$$t_{max} = \frac{8D^2}{\varepsilon^2}(Q(\boldsymbol{\alpha}^{(0)}) - Q(\boldsymbol{\alpha}^*)) + 2m .$$

PROOF: The reasoning used in the proof of the Theorem 4.4 applicable also for the Theorem 4.8. ■

4.8 Efficient implementation

The main requirement on the developed QP solvers is the ability to deal with large problems, i.e., problems where the matrix H in the definition of the QP task is large and cannot be stored in the memory. The aim is to minimize access of the QP solver to the entries of the matrix H. It is seen that the algorithms described in Sections 4.7 and 4.6 require at most two columns of the matrix H in each iteration. Moreover, in many cases only a small subset of the columns is requested by the algorithms, i.e., those which corresponds to the non-zero entries $\mathcal{I}_\emptyset = \{i \in \mathcal{I} \colon [\alpha]_i > 0\}$ of the vector $\boldsymbol{\alpha}^{(t)}$. This allows to use a cache for the most often requested columns without the need to store the whole matrix H. Namely, the *First In First Out* (FIFO) turned out to be suitable in the experiments described below.

The efficient implementation of the algorithms lies in maintaining a cache of key variables. In the case of the GMNP problem, these variables are

$$\delta_\alpha^{(t)} = \langle \boldsymbol{\alpha}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)} \rangle , \quad \boldsymbol{h}_\alpha^{(t)} = \mathrm{H}\boldsymbol{\alpha}^{(t)} , \quad \delta_\beta^{(t)} = \langle \boldsymbol{\beta}^{(t)}, \mathrm{H}\boldsymbol{\alpha}^{(t)} \rangle , \tag{4.103}$$

where $\delta_\alpha^{(t)} \in \mathbb{R}$ is a scalar, $\boldsymbol{h}_\alpha^{(t)} \in \mathbb{R}^m$ is a vector and $\delta_\beta^{(t)} \in \mathbb{R}$ is a scalar. Having the variables (4.103), the number of computations in each iteration of any QP solvers scales with $O(m)$, i.e., it is linear with respect to the number of variables m.

Next, it will be shown how to compute the variables (4.103) efficiently for the Kozinec Algorithm 13. The vector $\boldsymbol{\alpha}^{(t)}$ is arbitrary, whereas the vector $\boldsymbol{\beta}^{(t)}$ has always the following structure

$$[\boldsymbol{\beta}^{(t)}]_i = \left\{ \begin{array}{lll} 0 & \text{for} & i \neq u , \\ 1 & \text{for} & i = u , \end{array} \right. \tag{4.104}$$

i.e., all entries are zeros except for the u-th entry which is equal to one. The update of $\boldsymbol{\alpha}^{(t)}$ is in each step computed as

$$\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}(1-\tau) + \tau\boldsymbol{\beta}^{(t)} \,. \tag{4.105}$$

The variables $\delta_\alpha^{(t+1)}$, $\boldsymbol{h}_\alpha^{(t+1)}$ can be determined from the old values $\delta_\alpha^{(t)}$, $\boldsymbol{h}_\alpha^{(t)}$ as

$$\begin{aligned} \delta_\alpha^{(t+1)} &= \delta_\alpha^{(t)}\tau^2 - 2\tau(1-\tau)[\boldsymbol{h}_\alpha^{(t)}]_u + \tau^2[\mathrm{H}]_{u,u} \\ \boldsymbol{h}_\alpha^{(t+1)} &= \boldsymbol{h}_\alpha^{(t)}(1-\tau) + \tau[\mathrm{H}]_{:,u} \,, \\ \delta_\beta^{(t)} &= [\boldsymbol{h}_\alpha^{(t)}]_u \,, \end{aligned} \tag{4.106}$$

which follows directly substituting (4.104) and (4.105) to (4.103). The symbol $[\mathrm{H}]_{:,u}$ denotes the u-th column vector of the matrix H. It is seen from (4.106) that the key variables (4.103) do not have to be computed from a scratch which would require $O(m^2)$ operations but they can be updated by (4.106) which requires only $O(m)$ computations. Moreover, the update in each iteration requires just one column vector $[\mathrm{H}]_{:,u}$ instead of the whole matrix H.

The updating of the key variables (4.103) for the remaining algorithms is derived similarly and thus omitted here. The same idea is also applied to the GNPP. In this case, however, there are more key variables to be cached. Namely, the updates must be used for

$$\begin{aligned} &\delta_{11}^{(t)} = \langle\boldsymbol{\alpha}_1^{(t)}, \mathrm{H}_{11}\boldsymbol{\alpha}_1^{(t)}\rangle \,, \quad \delta_{12}^{(t)} = \langle\boldsymbol{\alpha}_1^{(t)}, \mathrm{H}_{12}\boldsymbol{\alpha}_2^{(t)}\rangle \,, \quad \delta_{22}^{(t)} = \langle\boldsymbol{\alpha}_2^{(t)}, \mathrm{H}_{22}\boldsymbol{\alpha}_2^{(t)}\rangle \,, \\ &\boldsymbol{h}_{11}^{(t)} = \mathrm{H}_{11}\boldsymbol{\alpha}_1^{(t)} \,, \quad \boldsymbol{h}_{21}^{(t)} = \mathrm{H}_{21}\boldsymbol{\alpha}_1^{(t)} \,, \quad \boldsymbol{h}_{12}^{(t)} = \mathrm{H}_{12}\boldsymbol{\alpha}_2^{(t)} \,, \quad \boldsymbol{h}_{22}^{(t)} = \mathrm{H}_{22}\boldsymbol{\alpha}_2^{(t)} \,. \end{aligned}$$

The number of computations required for updating the key variables is in all cases $O(m)$ regardless of algorithms for the GMNP or GNPP.

The requirements to access the matrix H differs according to a given algorithm: The Kozinec and Kowalczyk algorithms require a single column, in contrast to the MDM and Keerthi algorithms which require two columns in each iteration. The proposed improved MDM algorithm requires also two columns thanks to the simple selection rule used.

4.9 Applications of proposed QP solvers

This section contains a list of several problems used in machine learning and pattern recognition which lead to the QP task solvable by the algorithms proposed in this chapter. It should be remarked that the original algorithms for MNP and NPP have been applied only for design of the singleclass and binary SVM with L_2-soft margin. Therefore the application of the proposed generalized QP solvers on the mentioned problems is new. Section 4.9.1 summarizes the application of the QP solvers for learning of the single, binary, and multiclass SVM classifiers. The application for computation of the Reduced Set Density Estimator (RSDE) is described in Section 4.9.3, computation of the minimal enclosing ball and Support Vector Data Description (SVDD) is described in Section 4.9.2.

Other SVM-based learning problems for classification and regression which match the GNPP are introduced in [22].

4.9.1 Support Vector Machines for classification

The equivalence between the NPP and the learning of the binary SVM classifier with the hard and L_2-soft margin was described in Section 3.3.1. The equivalence between the

MNP and the singleclass SVM with the hard and L_2-soft margin was mentioned as well. This equivalence allows to use the QP solvers proposed in this chapter for the learning of the singleclass and binary SVM classifiers. Moreover, it will be shown in Chapter 6 how to use the QP solvers for the singleclass SVM classifier to learn the multiclass SVM classifier.

The rest of this section contains a novel proof of the mentioned equivalence between the NPP and the QP task associated to the binary SVM classifier and the equivalence between the MNP and the QP task associated to the singleclass SVM classifier. The singleclass case will be analyzed first as it is simpler.

The learning of the singleclass SVM classifier with hard and L_2-soft margin loss functions corresponds to the following QP task (c.f. Section 1.4.3)

$$\boldsymbol{\alpha}^* = \operatorname*{argmin}_{\boldsymbol{\alpha}} \left(\frac{1}{2}\langle \boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha}\rangle - \langle \boldsymbol{e}, \boldsymbol{\alpha}\rangle \right) , \tag{4.107}$$

subject to

$$\boldsymbol{\alpha} \geq 0 .$$

The MNP reads

$$\boldsymbol{\beta}^* = \operatorname*{argmin}_{\boldsymbol{\beta}} \frac{1}{2}\langle \boldsymbol{\beta}, \mathrm{H}\boldsymbol{\beta}\rangle , \tag{4.108}$$

subject to

$$\langle \boldsymbol{\beta}, \boldsymbol{e}\rangle = 1 , \quad \text{and} \quad \boldsymbol{\beta} \geq 0 .$$

Task (4.108) is a special instance of the GMNP which can be optimized by the QP solvers proposed in this chapter. The aim is to show that the solution $\boldsymbol{\alpha}^*$ of task (4.107) can be readily computed from the solution $\boldsymbol{\beta}^*$ of task (4.108). The exact relation between these two solutions is stated by the following theorem.

Theorem 4.9 *Let $\boldsymbol{\beta}^*$ be the optimal solution of the MNP (4.108). Then the vector*

$$\boldsymbol{\alpha}^* = \frac{\boldsymbol{\beta}^*}{\langle \boldsymbol{\beta}^*, \mathrm{H}\boldsymbol{\beta}^*\rangle} ,$$

is the optimal solution of the QP task (4.107) associated to the singleclass SVM classifier with the hard and L_2-soft margin loss functions.

PROOF: Let $r > 0$ be a constant and the following QP task be defined

$$\boldsymbol{\gamma}^* = \operatorname*{argmin}_{\boldsymbol{\gamma}} \frac{1}{2}\langle \boldsymbol{\gamma}, \mathrm{H}\boldsymbol{\gamma}\rangle - r = \operatorname*{argmin}_{\boldsymbol{\gamma}} \frac{1}{2}\langle \boldsymbol{\gamma}, \mathrm{H}\boldsymbol{\gamma}\rangle , \tag{4.109}$$

subject to

$$\langle \boldsymbol{\gamma}, \boldsymbol{e}\rangle = r , \quad \text{and} \quad \boldsymbol{\gamma} \geq 0 .$$

It is easy to see that the solution $\boldsymbol{\gamma}^*$ of the task (4.109) coincides with the solution $\boldsymbol{\alpha}^*$ of the task (4.107) if the equality $r = \langle \boldsymbol{\alpha}^*, \boldsymbol{e}\rangle$ holds. Next, it will be shown that for arbitrary $r > 0$ the equality

$$\boldsymbol{\gamma}^* = r\boldsymbol{\beta}^* , \tag{4.110}$$

holds, where $\boldsymbol{\gamma}^*$ is the solution of the task (4.109) and $\boldsymbol{\beta}^*$ is the solution of the MNP task (4.108). The equality (4.110) follows directly from the KKT conditions for both the tasks (4.108) and (4.109). The KKT conditions are necessary and sufficient conditions for the optimal solution for the convex QP tasks. The Lagrange function for the task (4.108) reads

$$L(\boldsymbol{\beta},\boldsymbol{\mu},\lambda) = \frac{1}{2}\langle\boldsymbol{\beta},\mathrm{H}\boldsymbol{\beta}\rangle + \lambda(\langle\boldsymbol{\beta},\boldsymbol{e}\rangle - 1) - \langle\boldsymbol{\mu},\boldsymbol{\beta}\rangle \,,$$

and the corresponding KKT conditions are

$$\begin{aligned}
\frac{\partial L(\boldsymbol{\beta},\boldsymbol{\mu},\lambda)}{\partial\boldsymbol{\beta}} = \mathrm{H}\boldsymbol{\beta} + \lambda\boldsymbol{e} - \boldsymbol{\mu} &= 0\,,\\
\langle\boldsymbol{\beta},\boldsymbol{\mu}\rangle &= 0\,,\\
\langle\boldsymbol{\beta},\boldsymbol{e}\rangle &= 1\,,\\
\boldsymbol{\mu} &\geq 0\,,\\
\boldsymbol{\beta} &\geq 0\,.
\end{aligned} \tag{4.111}$$

Let $(\boldsymbol{\beta},\boldsymbol{\mu},\lambda)$ be any triplet which satisfies the KKT conditions (4.111). Then β is the optimal solution of the task (4.108). The Lagrange function for the task (4.109) reads

$$G(\boldsymbol{\gamma},\boldsymbol{\omega},\sigma) = \frac{1}{2}\langle\boldsymbol{\gamma},\mathrm{H}\boldsymbol{\gamma}\rangle + \sigma(\langle\boldsymbol{\gamma},\boldsymbol{e}\rangle - r) - \langle\boldsymbol{\omega},\boldsymbol{\gamma}\rangle \,,$$

and the corresponding KKT conditions are

$$\begin{aligned}
\frac{\partial G(\boldsymbol{\gamma},\boldsymbol{\omega},\sigma)}{\partial\boldsymbol{\gamma}} = \mathrm{H}\boldsymbol{\gamma} + \sigma\boldsymbol{e} - \boldsymbol{\omega} &= 0\,,\\
\langle\boldsymbol{\gamma},\boldsymbol{\omega}\rangle &= 0\,,\\
\langle\boldsymbol{\gamma},\boldsymbol{e}\rangle &= r\,,\\
\boldsymbol{\omega} &\geq 0\,,\\
\boldsymbol{\gamma} &\geq 0\,.
\end{aligned} \tag{4.112}$$

Let $(\boldsymbol{\gamma},\boldsymbol{\omega},\sigma)$ be any triplet which satisfies the KKT conditions (4.112). Then $\boldsymbol{\gamma}$ is the optimal solution of the task (4.109). It is enough to show that the the solution $(\boldsymbol{\gamma},\boldsymbol{\omega},\sigma)$ of the KKT conditions (4.112) can be directly computed from the solution $(\boldsymbol{\beta},\boldsymbol{\mu},\lambda)$ of the KKT conditions (4.111). Indeed, it can be simply verified that the relation between the variables $(\boldsymbol{\beta},\boldsymbol{\mu},\lambda)$ and $(\boldsymbol{\gamma},\boldsymbol{\omega},\sigma)$ is given by the following equalities

$$\boldsymbol{\gamma} = r\boldsymbol{\beta}\,, \quad \boldsymbol{\omega} = r\boldsymbol{\mu}\,, \quad \sigma = r\lambda\,.$$

Thus equality (4.110) has been proved. It remains to determine r^* such that $\boldsymbol{\alpha}^* = r^*\boldsymbol{\beta}^*$ solves the task (4.107). The optimal r^* can be found by solving the task

$$r^* = \operatorname*{argmin}_r \left(\frac{1}{2}\langle\boldsymbol{\beta}^*,\mathrm{H}\boldsymbol{\beta}^*\rangle r^2 - \langle\boldsymbol{\beta}^*,\boldsymbol{e}\rangle r\right) = \frac{\langle\boldsymbol{\beta}^*,\boldsymbol{e}\rangle}{\langle\boldsymbol{\beta}^*,\mathrm{H}\boldsymbol{\beta}^*\rangle} = \frac{1}{\langle\boldsymbol{\beta}^*,\mathrm{H}\boldsymbol{\beta}^*\rangle}\,,$$

which ends the proof. ■

The same relation can be proven for the NPP and the QP task associated to the binary SVM with hard and L_2-soft margin. The QP task associated to the learning of the binary SVM classifier with the hard and L_2-soft margin reads (c.f. Section 1.4.1)

$$\boldsymbol{\alpha}^* = \operatorname*{argmin}_{\boldsymbol{\alpha}} \left(\frac{1}{2}\langle\boldsymbol{\alpha},\mathrm{H}\boldsymbol{\alpha}\rangle - \langle\boldsymbol{\alpha},\boldsymbol{e}\rangle\right)\,, \tag{4.113}$$

subject to

$$\langle \boldsymbol{\alpha}, \boldsymbol{e}_1 \rangle = \langle \boldsymbol{\alpha}, \boldsymbol{e}_2 \rangle \,, \quad \text{and} \quad \boldsymbol{\alpha} \geq 0 \,.$$

The NPP reads

$$\boldsymbol{\beta}^* = \operatorname*{argmin}_{\boldsymbol{\beta}} \frac{1}{2} \langle \boldsymbol{\beta}, \mathrm{H}\boldsymbol{\beta} \rangle \,, \tag{4.114}$$

subject to

$$\langle \boldsymbol{\beta}, \boldsymbol{e}_1 \rangle = 1 \,, \quad \langle \boldsymbol{\beta}, \boldsymbol{e}_2 \rangle = 1 \,, \quad \boldsymbol{\beta} \geq 0 \,.$$

Theorem 4.10 *Let $\boldsymbol{\beta}^*$ be the optimal solution of the NPP (4.114). Then the vector*

$$\boldsymbol{\alpha}^* = \frac{\boldsymbol{\beta}^*}{\langle \boldsymbol{\beta}^*, \mathrm{H}\boldsymbol{\beta}^* \rangle} \,,$$

is the optimal solution of the QP task (4.113) associated to the singleclass SVM classifier with the hard and L_2-soft margin loss functions.

PROOF: The proof is very much like the proof for the singleclass case. The KKT conditions for the binary case slightly differ as they contain one extra term corresponding to one extra linear constraint. The rest of the proof remains completely the same and thus it is omitted. ■

4.9.2 Minimal enclosing ball

The problem of computing the minimal enclosing ball can be expressed as the QP task solvable by the proposed QP solvers. The radius of the minimal enclosing ball is a quantity required in evaluation of generalization bounds for SVM classifiers (for more details on the bounds refer to [7, 46, 54]). A computation of the minimal enclosing ball is also a headstone of the support vector data description (SVDD) [51].

Let training set $\mathcal{T}_{\mathcal{X}} = \{x_1, \ldots, x_m\} \in \mathcal{X}^m$ of examples be given. The examples are assumed to be represented in the feature space $\mathcal{H}$ via the kernel function $k \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Let $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\} \in \mathcal{H}^m$ be the image of $\mathcal{T}_{\mathcal{X}}$ in the feature space $\mathcal{H}$. The minimal enclosing ball is defined as the ball around the vectors $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ which has the minimal volume. The SVDD describes data distribution by the minimal enclosing ball but it also assumes that a portion of data can be outliers, i.e., there can be vectors lying outside the ball. The outliers are penalized by linear or quadratic loss function. This leads to the soft margin formulation of the minimal enclosing ball

$$(R^*, \boldsymbol{\xi}^*, \boldsymbol{\mu}^*) = \operatorname*{argmin}_{R, \boldsymbol{\mu}, \boldsymbol{\xi}} \left(R^2 + C \sum_{i \in \mathcal{I}} (\xi_i)^p \right) \,, \tag{4.115}$$

subject to

$$\begin{array}{rcll} \|\boldsymbol{x}_i - \boldsymbol{\mu}\|^2 & \leq & R^2 + \xi_i \,, & i \in \mathcal{I} \,, \\ \xi_i & \geq & 0 \,, & i \in \mathcal{I} \,, \end{array}$$

where $\boldsymbol{\mu}$ is the center and R is the radius of the minimal ball. $C \in \mathbb{R}^+$ is the regularization constant. Task (4.115) covers three formulations: (i) hard margin case with $p = \infty$ (no outliers are allowed), (ii) L_1-norm soft margin with $p = 1$ and (iii) L_2-soft margin case with $p = 2$. The hard margin formulation $p = \infty$ is equivalent to the removing all terms from (4.115) which contain the slack variables ξ_i, $i \in \mathcal{I}$. It is again more convenient to solve the dual formulation of the QP task (4.115). The dual formulation can be obtain using the standard procedure described in Section 1.6. The dual formulations for the hard margin and the L_2-soft margin case are introduced below. These cases can be solved by the proposed algorithms in contrast to the L_1-soft margin formulation.

First, the hard margin minimal ball leads to the dual task

$$\boldsymbol{\alpha}^* = \operatorname*{argmax}_{\boldsymbol{\alpha}} \left(\sum_{i \in \mathcal{I}} \alpha_i \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{I}} \alpha_i \alpha_j \langle \boldsymbol{x}_i \cdot, \boldsymbol{x}_j \rangle \right) ,$$

subject to

$$\sum_{i \in \mathcal{I}} \alpha_i = 1 \,, \quad \text{and} \quad \alpha_i \geq 0 \,, \qquad i \in \mathcal{I} \,.$$

The optimal variables of the primal task can be computed as

$$\boldsymbol{\mu} = \sum_{i \in \mathcal{I}_\emptyset} \alpha_i \boldsymbol{x}_i \,, \quad \text{and} \quad R^2 = \|\boldsymbol{x}_i - \boldsymbol{\mu}\|^2 \quad \text{for any} \quad i \in \mathcal{I}_\emptyset \,,$$

where the set $\mathcal{I}_\emptyset = \{i \in \mathcal{I} \colon \alpha_i > 0\}$ contains indices of the support vectors. The computation of R follows from the KKT conditions.

Second, the L_2-norm soft margin minimal ball leads to the dual task

$$\boldsymbol{\alpha}^* = \operatorname*{argmax}_{\boldsymbol{\alpha}} \left(\sum_{i \in \mathcal{I}} \alpha_i \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{I}} \alpha_i \alpha_j \left(\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \frac{1}{4C} \delta(i,j) \right) \right) ,$$

subject to

$$\sum_{i \in \mathcal{I}} \alpha_i = 1 \,, \quad \text{and} \quad \alpha_i \geq 0 \,, \qquad i \in \mathcal{I} \,.$$

The optimal variables of the primal task can be computed as

$$\boldsymbol{\mu} = \sum_{i \in \mathcal{I}_\emptyset} \alpha_i \boldsymbol{x}_i \,, \quad \text{and} \quad R^2 = \|\boldsymbol{x}_i - \boldsymbol{\mu}\|^2 - \frac{\alpha_i}{2C} \quad \text{for any} \quad i \in \mathcal{I}_\emptyset \,.$$

The radius R can be again from the KKT conditions. The distance between the center $\boldsymbol{\mu}$ and the vector $\Phi(x)$ can be computed by

$$f(x) = \|\Phi(x) - \boldsymbol{\mu}\|^2 = \langle \Phi(x), \Phi(x) \rangle - 2 \sum_{i \in \mathcal{I}_\emptyset} \alpha_i \langle \boldsymbol{x}_i, \Phi(x) \rangle + \sum_{i \in \mathcal{I}_\emptyset} \sum_{j \in \mathcal{I}_\emptyset} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle ,$$

To determine whether the incoming vector $\Phi(x)$ belongs to the ball, it is sufficient to evaluate the inequality $f(x) \leq R^2$.

It is again seen that both the learning and evaluation of the distance requires data in terms of the dot products only. Thus the patterns can be embedded into the RKHS via the kernel function substituted for the dot products. The dual formulations exactly match the QP task (4.2) with feasible set (4.3) and the proposed QP solvers can be readily applied.

4.9.3 Reduced Set Density Estimator

The Reduced Set Density Estimator (RSDE) [21] is a kernel-based density estimator which employs a small percentage of the training data sample. Let a finite data sample $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\} \in \mathbb{R}^{n \times m}$ be drawn from the underlying unknown probability density $p(\boldsymbol{x})$ defined on $\mathbb{R}^n$. It is assumed that the density can be described by a weighted sum of Gaussian kernels (it is also called Parzen model), i.e.,

$$\begin{aligned} \tilde{p}(\boldsymbol{x};\boldsymbol{\alpha}) &= \sum_{i\in\mathcal{I}} \alpha_i N(\boldsymbol{x};\boldsymbol{x}_i,\sigma) \\ &= \sum_{i\in\mathcal{I}} \alpha_i \frac{1}{(2\pi)^{\frac{n}{2}}\sigma^n} \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}_i\|^2}{2\sigma^2}\right) . \end{aligned} \tag{4.116}$$

The symbol $N(\boldsymbol{x};\boldsymbol{x}_i,\sigma)$ stands for the Gaussian kernel with the mean vector $\boldsymbol{x}_i$ and the standard deviation $\sigma \in \mathbb{R}^+$ which is assumed to be known. The vector of real multipliers $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m]^m \in \mathbb{R}^m$ must be from $\mathcal{A} = \{\boldsymbol{\alpha} \in \mathbb{R}^n \colon \sum_{i\in\mathcal{I}} \alpha_i = 1\,, \alpha_i \geq 0\}$ to guarantee that $\tilde{p}(\boldsymbol{x};\boldsymbol{\alpha})$ is a probability density. The RSDE aims to estimate the vector of multipliers such that

$$\begin{aligned} \boldsymbol{\alpha}^* &= \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathcal{A}} \int_{\mathbb{R}^n} \|p(\boldsymbol{x}) - \tilde{p}(\boldsymbol{x};\boldsymbol{\alpha})\|^2 \mathrm{d}\boldsymbol{x} \\ &= \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathcal{A}} \left(\int_{\mathbb{R}^n} \tilde{p}(\boldsymbol{x};\boldsymbol{\alpha})^2 \mathrm{d}\boldsymbol{x} - 2\int_{\mathbb{R}^n} p(\boldsymbol{x})\tilde{p}(\boldsymbol{x};\boldsymbol{\alpha})\mathrm{d}\boldsymbol{x} \right) , \end{aligned} \tag{4.117}$$

i.e., the the integrated square error between the true and the estimated density is minimized. The first term of (4.117) can be computed exactly after substituting (4.116) which yields

$$\begin{aligned} \int_{\mathbb{R}^n} \tilde{p}(\boldsymbol{x};\boldsymbol{\alpha})^2 \mathrm{d}\boldsymbol{x} &= \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{I}} \alpha_i\alpha_j \int_{\mathbb{R}^n} N(\boldsymbol{x};\boldsymbol{x}_i,\sigma)N(\boldsymbol{x};\boldsymbol{x}_j,\sigma)\mathrm{d}\boldsymbol{x} \\ &= \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{I}} \alpha_i\alpha_j N(\boldsymbol{x}_i;\boldsymbol{x}_j,\sqrt{2}\sigma) . \end{aligned} \tag{4.118}$$

The solution of the integral can be found in [21]. The second term of (4.117) contains the expectation of the value of the $-2\tilde{p}(\boldsymbol{x};\boldsymbol{\alpha})$ with respect to the true unknown density $p(\boldsymbol{x})$. This expectation can be replaced by its estimate computed over the training sample

$$\begin{aligned} \int_{\mathbb{R}^n} p(\boldsymbol{x})\tilde{p}(\boldsymbol{x};\boldsymbol{\alpha})\mathrm{d}\boldsymbol{x} &= \sum_{i\in\mathcal{I}} \alpha_i \int_{\mathbb{R}^n} p(\boldsymbol{x})N(\boldsymbol{x};\boldsymbol{x}_i,\sigma)\mathrm{d}\boldsymbol{x} \\ &\approx \sum_{i\in\mathcal{I}} \alpha_i \frac{1}{m} \sum_{j\in\mathcal{I}} N(\boldsymbol{x}_i;\boldsymbol{x}_j,\sigma) . \end{aligned} \tag{4.119}$$

Substituting (4.118) and (4.119) to the (4.117) gives rise to the following QP task

$$\boldsymbol{\alpha}^* = \operatorname*{argmin}_{\boldsymbol{\alpha}} \left(\langle \boldsymbol{\alpha}, \mathrm{H}\boldsymbol{\alpha} \rangle - \langle \boldsymbol{\alpha}, \boldsymbol{c} \rangle \right) , \tag{4.120}$$

subject to

$$\sum_{i\in\mathcal{I}} \alpha_i = 1\,, \quad \text{and} \quad \alpha_i \geq 0\,, i \in \mathcal{I} .$$

The elements of matrix $\mathrm{H} \in \mathbb{R}^{m\times m}$ and vector $\boldsymbol{c} \in \mathbb{R}^m$ are defined as

$$[\mathrm{H}]_{i,j} = N(\boldsymbol{x}_i;\boldsymbol{x}_j,\sqrt{2}\sigma)\,, i\in\mathcal{I}, j\in\mathcal{I} \quad \text{and} \quad [\boldsymbol{c}]_i = \frac{1}{m}\sum_{j\in\mathcal{I}} N(\boldsymbol{x}_i;\boldsymbol{x}_j,\sigma)\,, i\in\mathcal{I}\,.$$

It is seen that the QP task (4.120) associated with the RSDE method exactly matches the problem solved by the proposed optimizers.

4.10 Experiments

Experiments described in this section aim to compare the QP solvers proposed for the GMNP and the GNPP. The QP tasks arising in the design of the multiclass and binary SVM classifiers are used for testing.

Section 4.10.1 describes an experiment in which the QP solvers for the GMNP problem are benchmarked on the learning problems associated to the multiclass BSVM formulation with L_2-soft margin. Standard benchmark data and data from one real application are used in the experiments. The BSVM formulation and the data are described in details in Chapter 6 devoted to the transformation of the multiclass BSVM formulation to a suitable QP task.

The proposed QP solvers are appropriate for the SVM formulations in which the L_2-soft margin approximation of the loss function is used. However, the L_1-soft margin has been used more often especially in the learning of binary SVM classifiers. Section 4.10.2 presents comparison of the binary SVM classifiers with the L_1-soft and L_2-soft margins using standard benchmark data.

Section 4.10.3 contains experimental comparison of the QP solvers for the GNPP. The QP task arising in the design of the binary SVM classifier with L_2-soft margin is used for testing.

4.10.1 Comparison of QP Solvers on Multiclass BSVM L_2 Problem

The aim of the experiment described in this section is to compare the QP solves for the MNP on several real tasks. The QP task associated to the learning the multiclass BSVM with L_2-soft margin was selected for comparison. The transformation of the learning problem to the MNP is a subject of Chapter 6. It is shown that the matrix H which defines the quadratic term equals

$$\mathrm{H} = \mathrm{K} + \frac{1}{2C}\mathrm{E}\,,$$

where K is the kernel matrix, E is the identity matrix and $C \in \mathbb{R}^+$ is a regularization constant. The size $m(M-1)$ of the matrix H is given by the number m of training examples and the number M of classes. The regularization constant C is a free parameter to be tuned. The value of C controls similarity of the matrix H to the diagonal matrix $\frac{1}{2C}\mathrm{E}$ and thus it changes smoothly the geometry of the QP task. It is known that high values of C imply low values of the optimal value $Q(\boldsymbol{\alpha}^*)$ for which the QP task becomes difficult to optimize. Therefore the constant C is used to smoothly change the complexity of the QP task which allows for better comparison of the QP solvers. This approach of changing the complexity of the QP task was adopted from the work of Kowalczyk [31].

From practical point of view, solving a range of the QP tasks corresponding to different constants C has to be dealt with in the model selection procedure of the SVM classifiers.

The data sets with the multiclass classification problems were adopted from the experiments described in Section 6.2 where the multiclass BSVM classifiers are benchmarked. Table 4.1 summarizes the datasets used and gives the size (number of variables) of the corresponding QP task to be solved. More details about the datasets can be found in Section 6.2.

Problem	# training data m	# class M	size of H $m(M-1)$
Iris	150	3	300
Wine	178	3	356
Glass	214	6	1,070
Vehicle	846	4	2,538
Vowel	528	11	5,280
Segment	2,310	7	13,860
OCR	49,030	32	1,519,930

Table 4.1: Summary of the multiclass BSVM L_2 learning problems used to benchmark the QP solvers. The number of training data m, the number of classes M and the corresponding size of matrix H defining the QP task is listed.

The kernel parameters which yield the best classification performance were used here. For each data set there are 15 different QP tasks with changed difficulty controlled by the regularization constant $C = \{2^{-2}, 2^{-1}, \ldots, 2^{12}\}$. All the QP solvers were applied to the same QP problems. To compare their performance, the following statistics were measured: (i) the number of columns of the matrix H required by the QP solver, (ii) the number of iterations, (iii) the number of non-zero entries of the solution vector $\boldsymbol{\alpha}$ and (iv) required CPU time in seconds on common PC with AMD 2800 MHz processor. The measured statistics were plotted with respect to the value $Q(\boldsymbol{\alpha})$. The value of $Q(\boldsymbol{\alpha})$ is an upper bound on the optimal value $Q(\boldsymbol{\alpha}^*)$ controlled by the constant C.

The algorithms were implemented in C language and linked to Matlab 6 which was used as an experimental environment. To speed up the algorithms, a simple "First In First Out" (FIFO) memory cache for 2000 columns of the kernel matrix was used (this issue was discussed in Section 4.8). The scale invariant ε-optimality stopping condition (4.8) with $\varepsilon = 0.001$ was used in all the experiments.

The algorithms tested were the Kowalczyk, Mitchell-Demyanov-Malozemov (MDM), improved Mitchell-Demyanov-Malozemov (IMDM) and the Keerthi algorithm. The Kozinec algorithm was found to be incomparably slow for these tasks. Therefore the Kozinec algorithm was applied only on the first two problems their size is reasonably small.

The measured statistics (i), (ii) and (iv) are strongly correlated. Namely, the number of requested columns of H equals to the number of iterations in the case of the Kozinec and the Kowalczyk algorithm. It is doubled in the case of the MDM, IMDM and the Keerthi algorithm. The computational complexity proportional to the CPU time is mainly influenced by the number of the columns of H used during the optimization. On the other hand, the number of non-zero entries of the solution vector $\boldsymbol{\alpha}$ is equal for most algorithms due to the high relative precision ($\varepsilon = 0.001$) of the solution found.

The results for the Iris and Wine data sets are depicted in Figure 4.1. It is seen that the performance of the Kozinec algorithm is inferior compared to the others especially for small values of $Q(\boldsymbol{\alpha})$. The required computational time increases much faster with decreasing value of $Q(\boldsymbol{\alpha})$ compared to the other algorithms. Also, the number of non-zero elements of the solution vector $\boldsymbol{\alpha}$ is higher. This supports the common sense as the Kozinec algorithm does not have means to force the variables to zero due to its update rule. This is probably the main reason why it is slow.

The performance of the MDM, IMDM, Keerthi and the Kowalczyk algorithms were evaluated on other problems with results reported in Figures 4.2, 4.3 and 4.4. The IMDM algorithm turned up to perform best in all the experiments in terms of the CPU time, the number of required columns of H, as well as the number of iterations. The advantage of the IMDM algorithm over the others is especially apparent for small values of $Q(\boldsymbol{\alpha})$. The performance of the MDM, Keerthi and the Kowalczyk algorithm is on average very similar. The Kowalczyk algorithm is slightly better than the MDM and the Keerthi algorithm in terms of the number of required columns of H which is consistent with the results by Kowalczyk [31].

4.10.2 Comparison of binary SVM with L_1-soft and L_2-soft margin

The QP solvers analyzed in this thesis are suitable for optimization of various SVM formulations with the L_2-soft margin loss function. The aim of the experiment described in this section is to compare the binary SVM with L_2-soft margin to commonly used SVM with L_1-soft margin and to other state-of-the-art methods for binary classification. The data sets from the Intelligent Data Analysis (IDA) benchmark repository [1] and the experimental protocol described in [37] were used for benchmarking.

The IDA repository consists of 13 artificial and real-world binary problems classification collected from UCI, DELVE and STATLOG benchmark repositories (for more details refer to [37]). For each data set there are 100 random partitionings into training and testing part (except for Image and Splice data sets, where it is 20). Basic characteristics of the datasets are listed in Table 4.2.

The experimental protocol adopted is the following. The training parts of the first 5 realizations are used for model selection. The model is determined by the kernel and the regularization constant C which must be tuned. In this experiment, the Gaussian kernel (c.f. Table 1.1) was used and its width σ was tuned. The parameter space is two-dimensional, $(\sigma, C) \in \mathbb{R}^2$. The optimal parameters were obtained by discretizing the parameter space and selecting the values which gave the smallest classification error estimated by the 5-fold cross validation. Having the parameters selected, the classifiers are learned on all 100 realizations of the training set. The evaluation is computed on the corresponding testing sets. Thus the reported statistics are mean values and standard deviation computed over the 100 realizations.

The repository contains simulation results for a broad range of methods learning the binary classifiers. For instance it involves results for RBF-Networks (RBFNet), AdaBoost (AB), and Kernel Fisher Discriminant (KFD). These results were adopted for comparison. In this thesis, the evaluation of SVM with L_1-soft margin (SVM L_1) and L_2-soft margin (SVM L_2) was computed. The methods are compared in terms of a classification accuracy.

[1]The IDA repository can be downloaded from `http://ida.first.fraunhofer.de/projects/bench/`.

The summary of the results is given in Table 4.3. The best performance can be observed for the KFD and SVM L_1, L_2 classifiers.

It can be seen that both the L_1-soft and L_2-soft margin formulations are comparable in terms of the classification error. Further experiment aimed to compare both the SVM formulations in terms of the number of support vectors which determines the classification rule. The results obtained on the IDA problems are summarized in Table 4.4. It can be seen that the L_2-soft margin formulation yields on average 40% support vectors more than the L_1-soft margin. For completeness, the free parameters which yield the best performance are listed in Table 4.4.

Problem	#features	#training data	#testing data	#realizations
Banana	2	400	4900	100
Breast	9	200	77	100
Diabetes	8	468	300	100
German	20	700	300	100
Heart	13	170	100	100
Image	18	1300	1010	20
Ringnorm	20	400	7000	100
Flare	9	666	400	100
Splice	60	1000	2175	20
Thyroid	5	140	75	100
Titanic	3	150	2051	100
Twonorm	20	400	7000	100
Waveform	21	400	4600	100

Table 4.2: Basic characteristics of the problems of the IDA repository. The table contains name of the problem, dimension of the feature vector, number of training and testing examples and the number of realizations of the training and testing part.

4.10.3 Comparison of QP solvers on binary SVM L_2 problem

The aim of the experiment described in this section is to compare the QP solvers for the GNPP. The QP tasks arising in the learning of the binary SVM classifier with L_2-soft margin were used. The datasets of the IDA repository were selected for testing (c.f. Section 4.10.2 for more details). The performance of the QP solvers applied to the GNPP very much resembles the performance on the GMNP. Therefore, only the results on the Banana, German and Image data sets were reported. For each data set, the kernel which yields the smallest classification error (see Section 4.10.2) was used. The regularization constant was changed in range $C \in \{1, 10, 50, 100, 500, 1000\}$ to control the complexity of the QP task similarly to the experiment of Section 4.10.1.

For each data set there are 6 different QP tasks corresponding to various values of the regularization constant C. All the QP solvers were applied to the same QP problems. The following statistics were measured: (i) the number of columns of the matrix H requested by the algorithm and (ii) the number of non-zero entries of the solution vector $\boldsymbol{\alpha}$. The

Problem	RBFNet	AB	KFD	SVM L_1	SVM L_2
Banana	10.8 ± 0.6	12.3 ± 0.7	10.8 ± 0.5	10.4 ± 0.4	10.5 ± 0.5
Breast	27.6 ± 4.7	30.4 ± 4.7	25.8 ± 4.6	26.1 ± 4.9	26.0 ± 4.5
Diabetes	24.3 ± 1.9	26.5 ± 2.3	23.2 ± 1.6	23.2 ± 1.7	23.1 ± 1.8
German	24.7 ± 2.4	27.5 ± 2.5	23.7 ± 2.2	23.7 ± 2.2	23.4 ± 2.3
Heart	17.6 ± 3.3	20.3 ± 3.4	16.1 ± 3.4	15.7 ± 3.3	16.0 ± 3.1
Image	3.3 ± 0.6	2.7 ± 0.7	4.8 ± 0.6	3.0 ± 0.5	3.1 ± 0.6
Ringnorm	1.7 ± 0.2	1.9 ± 0.3	1.5 ± 0.1	1.6 ± 0.1	1.7 ± 0.1
Flare	34.4 ± 2.0	35.7 ± 1.8	33.2 ± 1.7	32.4 ± 1.8	33.9 ± 1.5
Splice	10.0 ± 1.0	10.1 ± 0.5	10.5 ± 0.6	11.1 ± 0.6	11.2 ± 0.7
Thyroid	4.5 ± 2.1	4.4 ± 2.2	4.2 ± 2.1	4.7 ± 2.3	4.5 ± 2.1
Titanic	23.3 ± 1.3	22.6 ± 1.2	23.2 ± 2.0	22.4 ± 1.0	22.4 ± 1.1
Twonorm	2.9 ± 0.3	3.0 ± 0.3	2.6 ± 0.2	3.0 ± 0.4	2.7 ± 0.2
Waveform	10.7 ± 1.1	10.8 ± 0.6	9.9 ± 0.4	10.1 ± 0.4	10.1 ± 0.4

Table 4.3: Simulation results on IDA repository for RBF-Network (RBFNet), AdaBoost (AB), Kernel Fisher Discriminant (KFD), Support Vector Machines with L_1-soft margin (SVM L_1) and Support Vector Machines with L_2-soft margin (SVM L_2). The methods are compared in terms the classification error its mean and the standard deviation are computed over the 100 realizations.

	SVM L_1		SVM L_2	
	#SV	(C, σ)	#SV	(C, σ)
Banana	139 ± 8	$(1.48, 0.68)$	210 ± 15	$(5.94, 1.12)$
Breast	113 ± 6	$(38.48, 6.54)$	187 ± 5	$(1.00, 3.71)$
Diabetes	255 ± 8	$(3.85, 5.88)$	414 ± 7	$(1.92, 8.09)$
German	420 ± 12	$(5.94, 5.48)$	613 ± 12	$(1.14, 6.09)$
Heart	98 ± 5	$(1.00, 6.90)$	137 ± 6	$(1.00, 7.80)$
Image	147 ± 8	$(1092.55, 3.71)$	365 ± 11	$(11.40, 2.26)$
Ringnorm	131 ± 7	$(1.00, 2.43)$	132 ± 8	$(12.99, 2.43)$
Flare	473 ± 14	$(28.39, 5.88)$	623 ± 14	$(1.48, 3.00)$
Splice	594 ± 16	$(26.02, 6.90)$	597 ± 16	$(12.99, 7.80)$
Thyroid	20 ± 3	$(11.40, 1.65)$	67 ± 4	$(11.40, 0.58)$
Titanic	69 ± 10	$(3.38, 0.95)$	150 ± 0	$(3.38, 0.58)$
Twonorm	57 ± 7	$(28.39, 5.48)$	123 ± 8	$(1.14, 4.75)$
Waveform	129 ± 9	$(2.96, 5.29)$	175 ± 13	$(2.96, 7.27)$

Table 4.4: Comparison of Support Vector Machines classifiers with L_1-soft and L_2-soft margin on IDA repository in terms of the number of support vectors. The regularization constant C and the kernel width σ which yields the best classification performance is also listed.

number of required columns is correlated with the number of iterations as well as the computational time (c.f. discussion in Section 4.10.1). In all experiments, the scale invariant ε-optimality stopping condition (4.8) with $\varepsilon = 0.001$ was used. The algorithms were implemented in Matlab 6.

The algorithms tested were the Kowalczyk, Mitchell-Demyanov-Malozemov (MDM), improved Mitchell-Demyanov-Malozemov (IMDM) and the Keerthi algorithm. The Kozinec algorithm was found to be incomparably slow for this precision. The results obtained are depicted in Figure 4.5. It can be seen that the trend of the curves resembles the results obtained for the GMNP. The IMDM algorithm required in all experiments the minimal number of columns of the matrix H which is directly correlated with the low computational time. The performance of the remaining algorithms is on average similar with slight superiority of the Kowalczyk algorithm.

4.11 Summary to QP solvers

This chapter analyzes the QP solvers based on known methods to solve the MNP and NPP which were successfully applied to solve large scale problems arising in the learning of the binary SVM classifier with the L_2-soft margin.

The contributions of this part of the thesis involve:

- The application of the Kozinec algorithm for learning the binary SVM classifier with L_2-soft margin was proposed by the author of the thesis in [11, 15]. However, the application of different kinds of algorithms solving the NPP for learning of the SVM L_2-soft margin classifier was proposed earlier by Keerthi et al. [29] and Kowalczyk [31].

- The algorithms for the MNP and NPP were generalized to solve the QP tasks with an additional linear term and arbitrary symmetric positive definite Hessian in the objective function. These generalized formulations are denoted as the GMNP and the GNPP. As a result the generalized algorithms can be applied for optimization problems arising in a broader class of learning methods. Namely, the solvers can be applied for the Reduced Set Density Estimation, computation of the minimal enclosing ball, Support Vector Data Description, learning the single, binary and the multiclass classifiers. Section 4.9 describes these applications in more details.

- A novel proof of the equivalence between the MNP and the QP task associated to the learning of the singleclass SVM with hard and L_2-soft margin was presented. A novel proof was also given for the equivalence between the NPP and the QP task associated to the learning of the binary SVM with hard and L_2-soft margin.

- All the generalized methods were analyzed in a common framework which allows for their comparison. The convergence to the ε-optimal solution in a finite number of iterations was proven for all the methods.

- A novel method, named Improved Mitchell-Demyanov-Malozemov (IMDM) algorithm, was derived by combining of ideas of two algorithms. Namely, the form of the updating rule was adopted from the MDM algorithm. The idea of selecting the optimal rule in each iteration was adopted from the Kowalczyk algorithm.

- The algorithms were experimentally evaluated on large scale problems and compared between each other. The algorithm were proven to solve large problems (even with millions of variables) in reasonable times. The proposed IMDM algorithm outperformed the other algorithms in all experiments.

The listed contributions were partially published in [11, 15, 18].

Figure 4.1: Benchmarking QP solvers on Iris and Wine data sets.

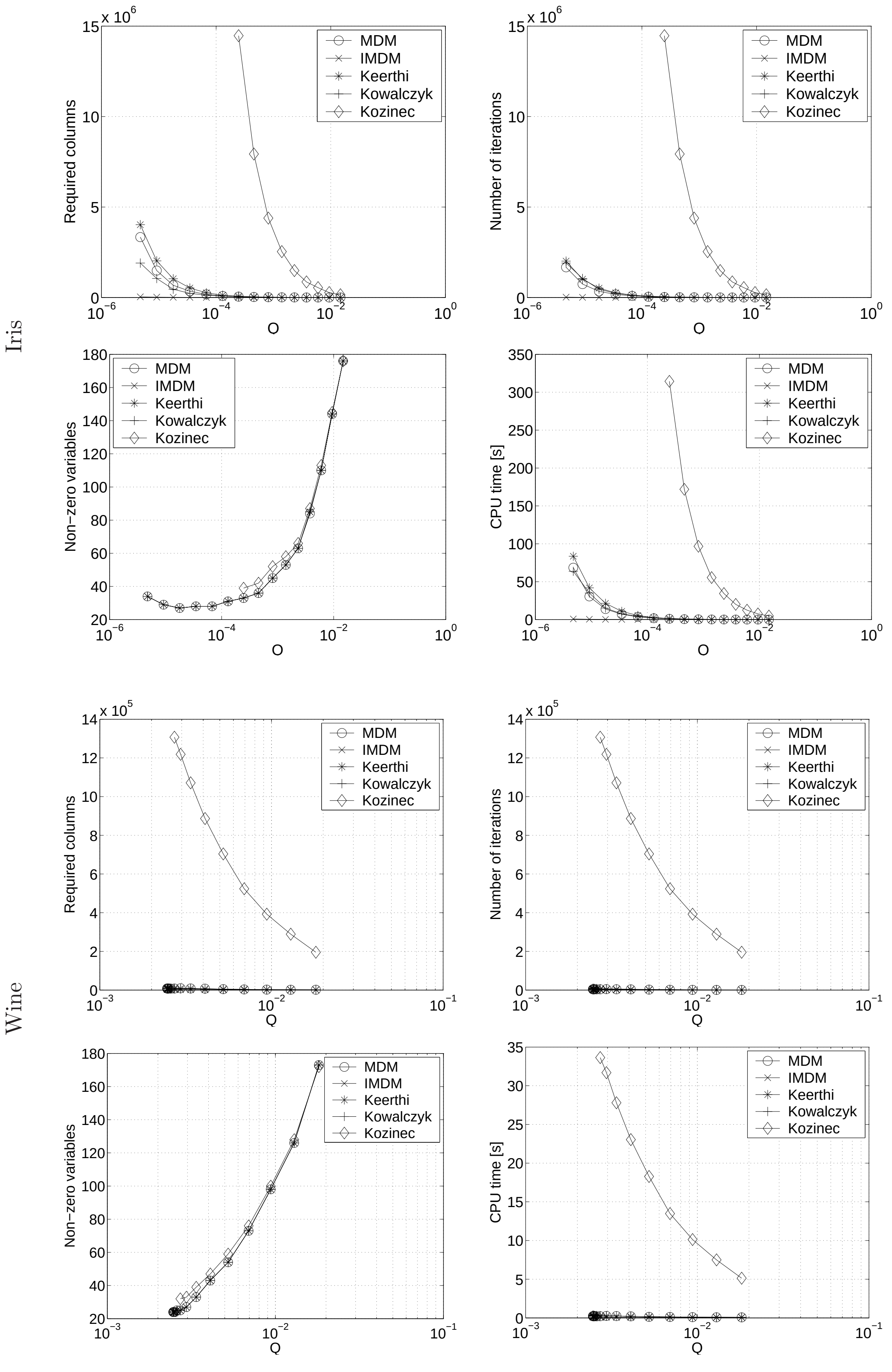


Figure 4.2: Benchmarking QP solvers on Glass and Vehicle data sets.

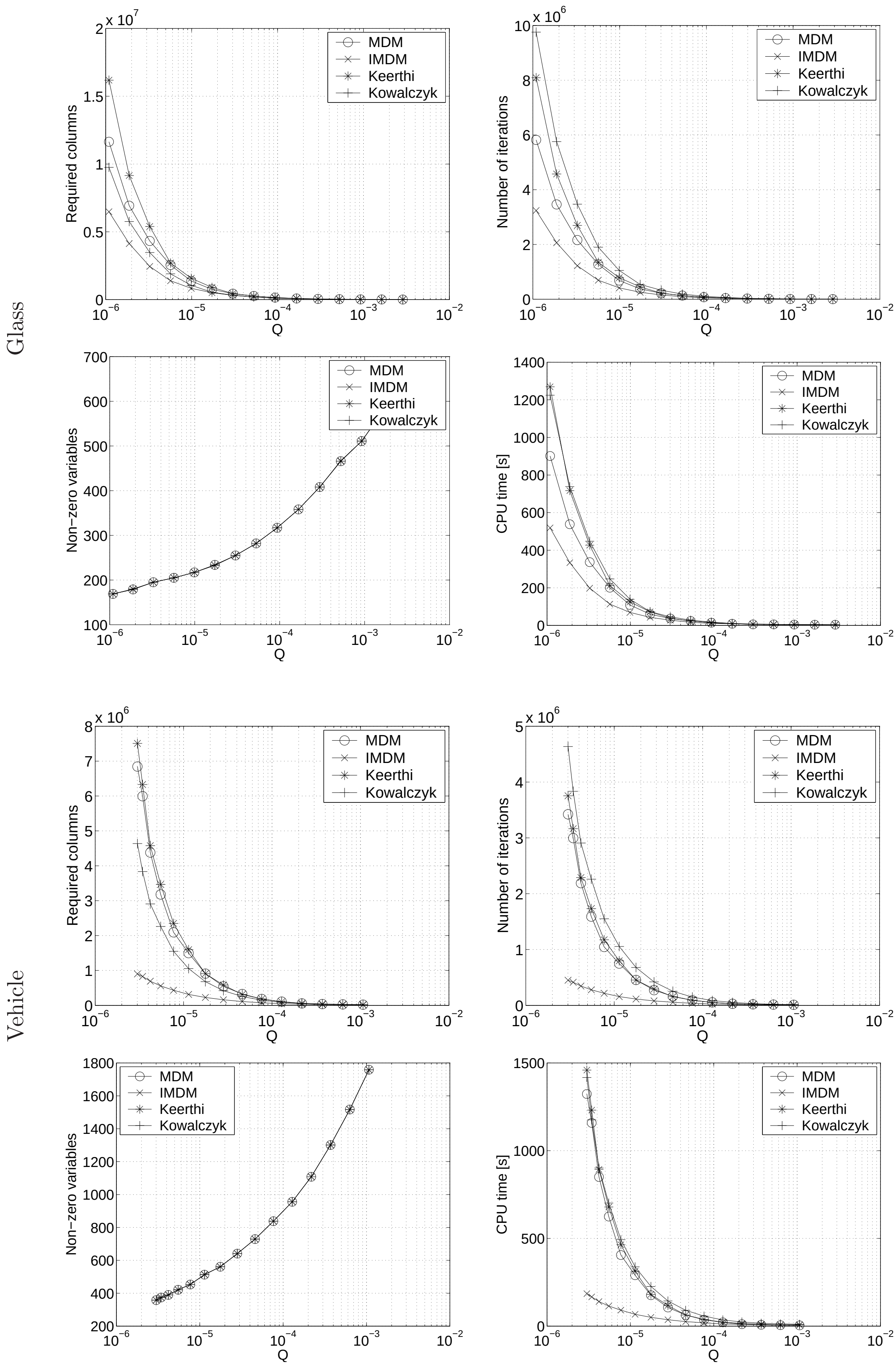


Figure 4.3: Benchmarking QP solvers on Vowel and Segment data sets.

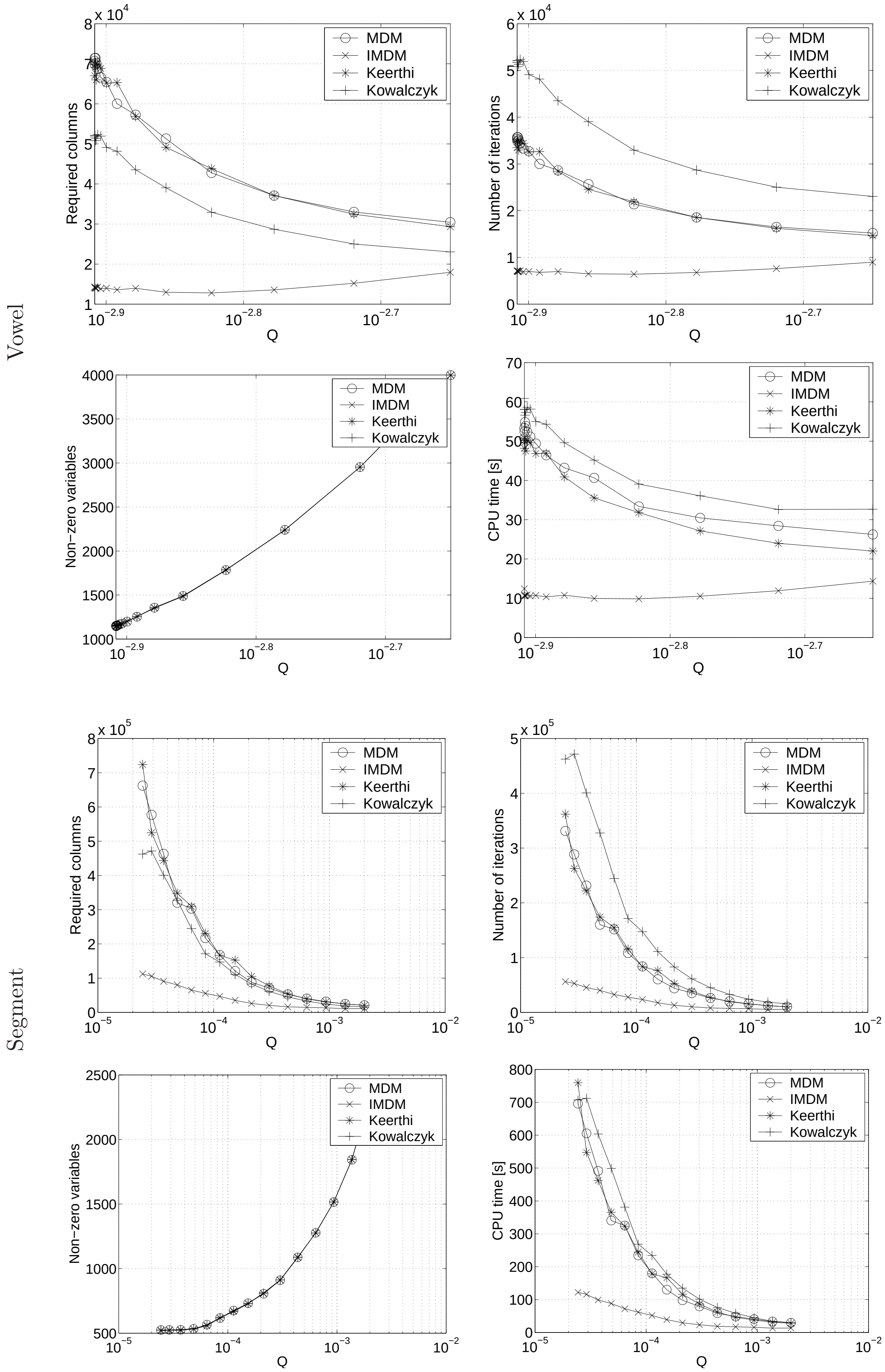


Figure 4.4: Benchmarking QP solvers on OCR data sets.

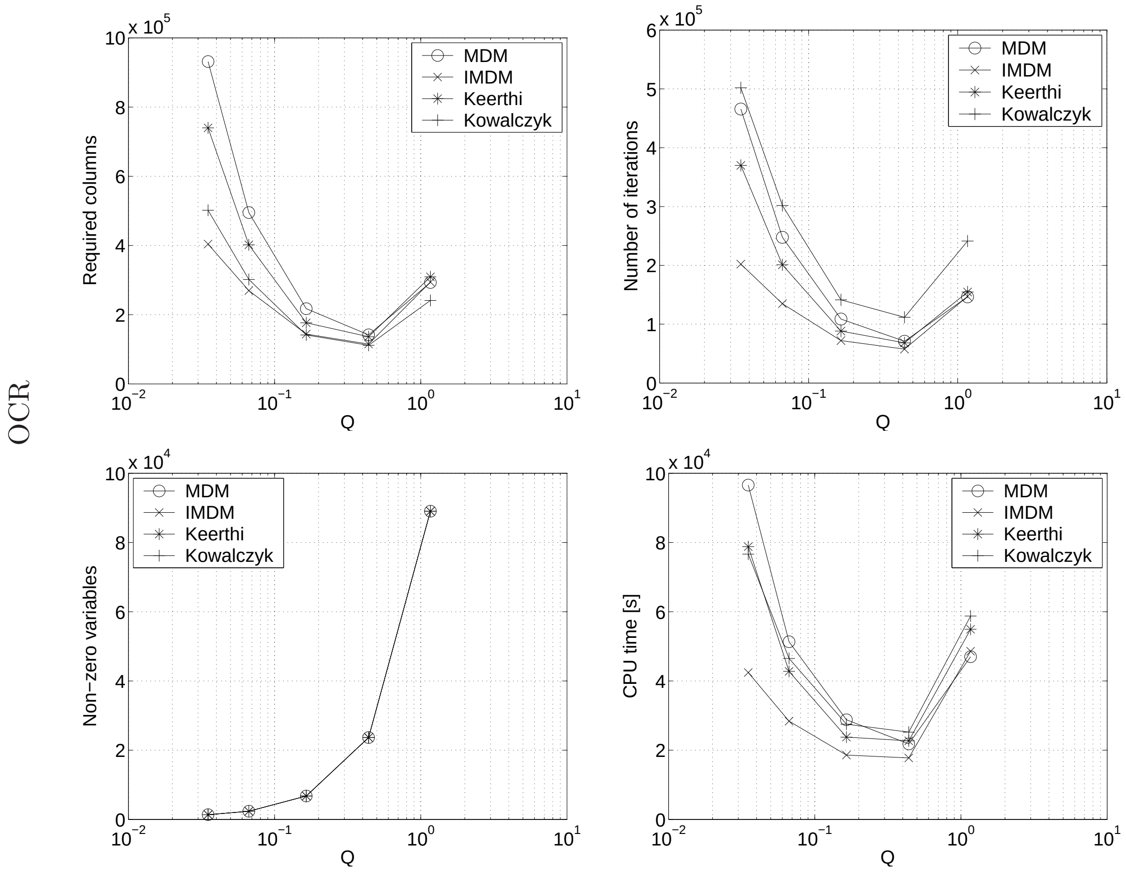
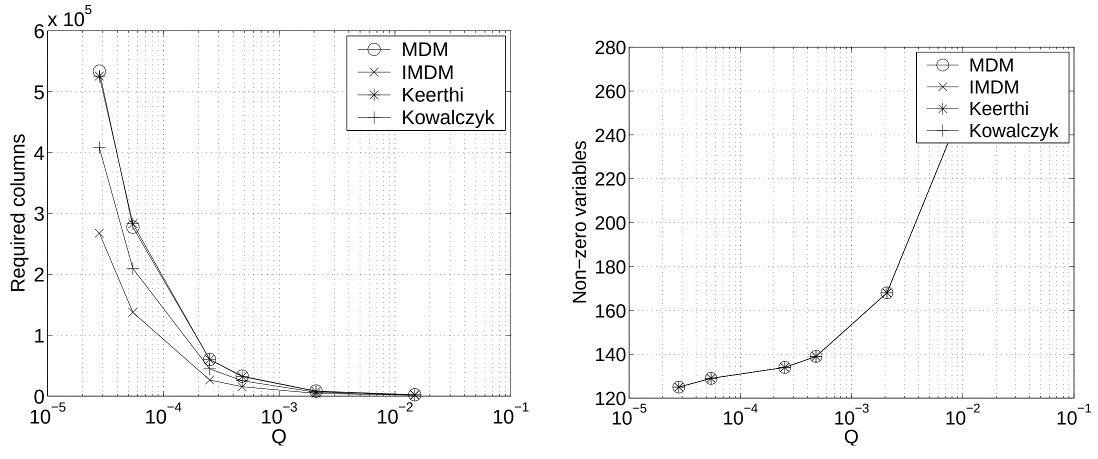
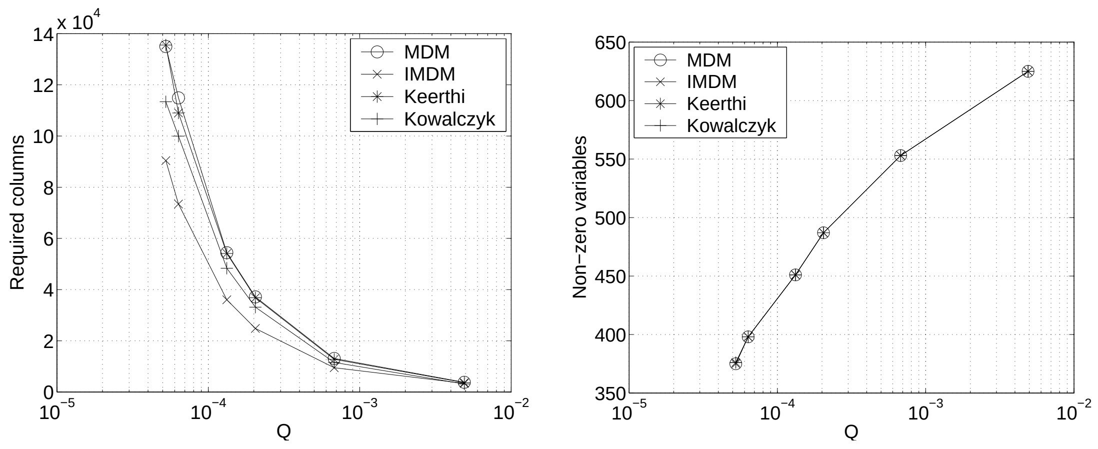


Figure 4.5: Benchmarking QP solvers on Banana, German and Image data sets.

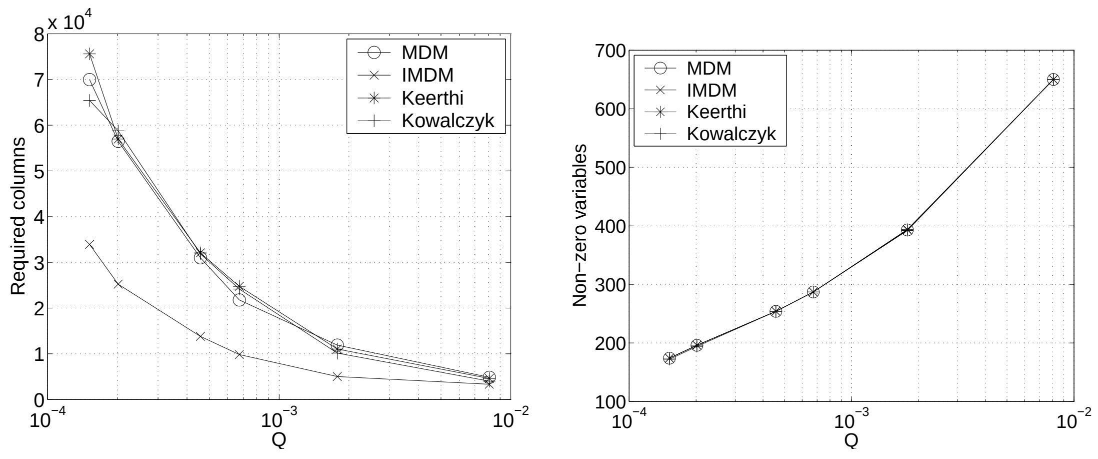
Banana



German



Image



5 Greedy Kernel Principal Component Analysis

5.1 Motivation

The kernel methods in general learn a function in the following form

$$f(x) = \sum_{i=1}^{m} \alpha_i \langle \Phi(x), \Phi(x_i) \rangle + b = \sum_{i=1}^{m} \alpha_i k(x, x_i) + b\,. \tag{5.1}$$

The function f is a linear combination of training examples of observations $\{x_1, \ldots, x_m\}$ mapped to the feature space $\mathcal{H}$. The feature map $\Phi \colon \mathcal{X} \to \mathcal{H}$ of the input data does not have to be known explicitly but it is rather given by a positive definite kernel function $k \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. The information contained in the training observations is expressed by the kernel matrix $\mathrm{K} \in \mathbb{R}^{m \times m}$ the entries of which are $[\mathrm{K}]_{i,j} = k(x_i, x_j)$. Two problems can arise when dealing with the kernel methods:

(i) *The training stage becomes demanding due to the size of the kernel matrix.* The storage of the kernel matrix becomes infeasible for a large training set as the kernel matrix grows quadratically with the number of examples. On the other hand, a frequent evaluation of the entries of the kernel matrix makes a given learning method slow.

(ii) *The evaluation stage becomes demanding due to a large number of non-zero coefficients of the kernel expansion (5.1).* Even though the learning methods like the Support Vector Machines produce sparse solutions the number of non-zero coefficients can be still large. This happens especially when the number of training examples is high or the examples are heavily overlapping. Moreover, some kernel methods do not enforce the solution to be sparse, e.g., the Kernel Principal Component Analysis or the Kernel Fisher Discriminant.

In particular, the second problem is significant in most real applications where the evaluation time is important.

The method proposed in this chapter aims to mitigate both the mentioned problems. Let $\mathcal{T} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ be a training set of examples represented in the feature space $\mathcal{H}$. The idea is to select a subset $\mathcal{S} \subset \mathcal{T}$ of the training examples such that the linear span of $\mathcal{S}$ is similar to the linear span of all examples $\mathcal{T}$. Let $\mathcal{I} = \{1, \ldots, m\}$ denote a set of indices of the training examples $\mathcal{T}$ and $\mathcal{J} = \{j_1, \ldots, j_l\}$ a set of indices of l selected examples $\mathcal{S}$. In such a case $\mathcal{J} \subset \mathcal{I}$. Let $\tilde{\mathcal{T}} = \{\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_m\}$ be an approximation of the training examples $\mathcal{T}$ represented in the basis given by the selected subset $\mathcal{S}$. If the function f is learned from the approximated examples $\tilde{\mathcal{T}}$ then two facts hold true:

(i) The kernel matrix of the approximated data can be factorized to $\tilde{\mathrm{K}} = \mathrm{Z}^T\mathrm{Z}$, where $\mathrm{Z} \in \mathbb{R}^{l \times m}$ is a matrix the columns of which are coordinates of training examples $\mathcal{T}$ represented in the orthonormalized basis of the selected subset $\mathcal{S}$. The validity of this statement will be derived below.

(ii) The learned function lies in the span of selected examples. Thus

$$\tilde{f}(x) = \sum_{j\in\mathcal{J}} \beta_j \langle \Phi(x_j), \Phi(x) \rangle + \theta = \sum_{j\in\mathcal{J}} \beta_j k(x_j, x) + \theta \,.$$

The number l of selected examples $\mathcal{J}$ determines the complexity of the function f (in the worst case).

It can be seen that using the approximated training examples $\tilde{\mathcal{T}}$ yields a reduction both in the training and the evaluation stage. A method to select a small subset $\mathcal{S}$ which allows a good approximation of the training examples is proposed below.

5.2 Problem formulation

The aim is to approximate training examples $\mathcal{T} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ by a new set $\tilde{\mathcal{T}} = \{\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_m\}$ such that

$$\tilde{\boldsymbol{x}}_i = \sum_{j\in\mathcal{J}} \boldsymbol{x}_j [\boldsymbol{\beta}_i]_j \,, \quad \forall i \in \mathcal{I} \,.$$

The set $\mathcal{J} \subset \mathcal{I}$ contains indices of l selected examples $\mathcal{S} = \{\boldsymbol{x}_j \colon j \in \mathcal{J}\} \subset \mathcal{T}$ and $\boldsymbol{\beta}_i \in \mathbb{R}^l$, $i \in \mathcal{I}$, are coefficients of linear combinations. The objective function to minimize is the mean square error

$$\varepsilon_{MS}(\mathcal{T}|\mathcal{J}) = \frac{1}{m}\sum_{i\in\mathcal{I}} \|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|^2 = \frac{1}{m}\sum_{i\in\mathcal{I}} \left\| \boldsymbol{x}_i - \sum_{j\in\mathcal{J}} \boldsymbol{x}_j[\boldsymbol{\beta}_i]_j \right\|^2 \,. \tag{5.2}$$

The approximation error $\varepsilon_{MS}(\mathcal{T}|\mathcal{J})$ of the examples in the set $\mathcal{T}$ depends on the subset $\mathcal{J}$ and the coefficients $\boldsymbol{\beta}_i$, $i \in \mathcal{I}$. However, given the subset $\mathcal{S}$ the coefficients $\boldsymbol{\beta}_i$, $i \in \mathcal{I}$ can be computed optimally to minimize the error $\varepsilon_{MS}(\mathcal{T}|\mathcal{J})$. Therefore they do not appear as an argument of the error function. It is easy to see that the optimal coefficients can be computed as

$$\boldsymbol{\beta}_i = \operatorname*{argmin}_{\boldsymbol{\beta}\in\mathbb{R}^l} \left\| \boldsymbol{x}_i - \sum_{j\in\mathcal{J}} \boldsymbol{x}_j[\boldsymbol{\beta}_i]_j \right\|^2 = (\mathrm{K_s})^{-1}\boldsymbol{k}_s(x_i) \,, \quad \forall i \in \mathcal{I} \,,$$

where $\mathrm{K_s} \in \mathbb{R}^{l\times l}$ is a kernel matrix of the selected examples $\mathcal{S}$, i.e., $[\mathrm{K_s}]_{i,j} = k(x_{j_i}, x_{j_j}) = \langle \boldsymbol{x}_{j_i}, \boldsymbol{x}_{j_j} \rangle$, and the vector $\boldsymbol{k}_s(x_i) = [k(x_{j_1}, x_i), \ldots, k(x_{j_l}, x_i)]^T \in \mathbb{R}^l$ contains kernel functions evaluated at the examples $\mathcal{S}$ and x_i. Having the matrix $\mathrm{K_s}$ and the vector $\boldsymbol{k}_s$ defined the error $\varepsilon_{MS}(\mathcal{T}|\mathcal{J})$ can be written as

$$\varepsilon_{MS}(\mathcal{T}|\mathcal{J}) = \frac{1}{m}\sum_{i\in\mathcal{I}} \left(k(x_i, x_i) - 2\,\mathrm{K_s}\boldsymbol{k}_s(x_i) + \langle \boldsymbol{k}_s(x_i), \mathrm{K_s}\boldsymbol{k}_s(x_i) \rangle \right) \,. \tag{5.3}$$

An important observation is that both the coefficients and the error function can be expressed in terms of dot products and thus the kernel functions can be employed.

The selection of the subset $\mathcal{S}$ is stated as the following optimization problem

$$\mathcal{J}^* = \operatorname*{argmin}_{\substack{\mathcal{J}\subset\mathcal{I} \\ \mathrm{Card}(\mathcal{J})=l}} \varepsilon_{MS}(\mathcal{T}|\mathcal{J}) \,, \tag{5.4}$$

where Card($\mathcal{J}$) denotes the cardinality of the set $\mathcal{J}$. It should be remarked that the task (5.4) is connected to the Kernel Principal Component Analysis (KPCA) described in Section 1.5. The KPCA also aims to minimize the mean square error but the basis vectors are represented as linear combinations of all training examples. The approximation found by the KPCA is optimal with respect to the mean square error but it is the worst possible with respect to the number of training data required for data representation.

The selection of the optimal subset $\mathcal{J}^*$ is a combinatorial problem as there exist $\binom{m}{l}$ possible selections. Instead of solving the problem (5.4) exactly, an approximated solution can be found by a greedy Algorithm 23.

Algorithm 23: Naive Greedy KPCA

```
1. Initialization. Set $\mathcal{J}^{(0)} = \{\emptyset\}$.
2. For $t = 1$ to $l$:
       $(a)$   $j_t \in \operatorname*{argmin}_{j \in \mathcal{I} \setminus \mathcal{J}^{(t-1)}} \varepsilon_{MS}(\mathcal{T}|\mathcal{J}^{(t-1)} \cup \{j\})$ .
       $(b)$   $\mathcal{J}^{(t)} = \mathcal{J}^{(t-1)} \cup \{j_t\}$ .
```

Algorithm 23 implements a greedy strategy to minimization of the task (5.4). There are two reasonable stopping conditions: (i) the algorithm halts if Card($\mathcal{J}^{(t)}$) reaches a given limit or (ii) the algorithm halts if the error $\varepsilon_{MS}(\mathcal{T}|\mathcal{J}^{(t)})$ falls below a specific limit. The use of a particular stopping condition depends on a given application. Even though Algorithm 23 is simple to implement, it is not suitable for practical use due to its high computational requirements. The bottleneck is the Step (b) which requires $O(m^2)$ computations. The evaluation of $\varepsilon_{MS}(\mathcal{T}|\mathcal{J}^{(t)} \cup j)$ using (5.3) requires $O(m)$ operations and it must be repeated m times. There is a need to adopt further approximations to the solution to obtain a practically useful algorithm which is proposed below.

Section 5.3 presents a simple and fast algorithm which minimizes an upper bound on ε_{MS}. This algorithm is fast but the obtained approximation of the input data is too rough. Therefore this simple algorithm is further extended which gives rise to a practically applicable method described in Section 5.4.

5.3 Upper bound minimization

The mean square error to be minimized $\varepsilon_{MS}(\mathcal{T}|\mathcal{J})$ can be upper bounded by

$$\varepsilon_{MS}(\mathcal{T}|\mathcal{J}) = \frac{1}{m} \sum_{i \in \mathcal{I}} \|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|^2 \leq \frac{1}{m}(m-l) \max_{j \in \mathcal{I} \setminus \mathcal{J}} \|\boldsymbol{x}_j - \tilde{\boldsymbol{x}}_j\|^2 , \tag{5.5}$$

where $l = \text{Card}(\mathcal{J})$ is number of selected examples in the set $\mathcal{J}$. The bound (5.5) obviously holds because $(m-l)$ examples included to the set $\mathcal{J}$ are represented without error and the mean cannot be higher than the maximum. A simple method for minimization of the error $\varepsilon_{MS}(\mathcal{T}|\mathcal{J})$ can be based on a greedy minimization of the upper bound (5.5). This method simply adds in each iteration the example with the biggest error to the set of the selected examples $\mathcal{S}$. Moreover, it will be shown that the evaluation of the partial errors $\|\boldsymbol{x}_j - \tilde{\boldsymbol{x}}_j\|$, $j \in \mathcal{J}$ can be efficiently evaluated when the approximated data $\tilde{\mathcal{T}} = \{\tilde{\boldsymbol{x}}_1, \dots, \tilde{\boldsymbol{x}}_m\}$ are represented in the orthonormal basis $\mathcal{W} = \{\boldsymbol{w}_1, \dots, \boldsymbol{w}_l\}$ spanning the selected subset $\mathcal{S} = \{\boldsymbol{x}_{j_1}, \dots, \boldsymbol{x}_{j_l}\}$.

The orthonormal basis $\mathcal{W}$ can be computed gradually as the new examples are added to the set $\mathcal{S}$. The examples $\mathcal{T}$ are represented in the orthonormal basis $\mathcal{W}$ as

$$\tilde{\boldsymbol{x}}_i = \sum_{j=1}^{l} \boldsymbol{w}_j [\boldsymbol{z}_i]_j \,, \qquad \forall i \in \mathcal{I} \,,$$

where the new representation $\mathcal{Z} = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_m\} \in \mathbb{R}^{m \times l}$ is a set of l-dimensional real vectors. The basis vectors $\boldsymbol{w}_j \in \mathcal{H}$ are determined as a linear combination of the selected examples $\mathcal{S}$, i.e.,

$$\boldsymbol{w}_j = \sum_{i=1}^{l} \boldsymbol{x}_{j_i} [\boldsymbol{\alpha}_j]_i \,,$$

where $\boldsymbol{\alpha}_j \in \mathbb{R}^l$, $j = 1, \dots, l$, are l-dimensional vectors. Let the matrix $\mathrm{A} = [\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_l] \in \mathbb{R}^{l \times l}$. The basis $\mathcal{W}$ is required to be orthonormal which means that

$$\mathrm{K_s A K_s} = \mathrm{E} \,,$$

where $\mathrm{E} \in \mathbb{R}^{l \times l}$ is the identity matrix. Having the orthonormal basis $\mathcal{W}$ the vectors $\mathcal{Z} = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_m\}$ can be computed as projections of $\mathcal{T} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}$ onto the basis vectors $\mathcal{W} = \{\boldsymbol{w}_1, \dots, \boldsymbol{w}_l\}$, i.e.,

$$[\boldsymbol{z}_i]_j = \langle \boldsymbol{w}_j, \boldsymbol{x}_i \rangle, \qquad \forall i \in \mathcal{I}, j = 1, \dots, l \,,$$

or in a compact form

$$\boldsymbol{z}_i = \mathrm{A}^T \boldsymbol{k}_s(x_i) \,.$$

The partial approximation errors can be evaluated simply as

$$\varepsilon_i = \|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|^2 = \langle \boldsymbol{x}_i, \boldsymbol{x}_i \rangle - \langle \boldsymbol{z}_i, \boldsymbol{z}_i \rangle \,, \quad i \in \mathcal{I} \,, \tag{5.6}$$

which follows directly from orthonormality of the basis $\mathcal{W}$.

The vectors $\mathrm{A} = [\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_l]$ which makes the basis $\mathcal{W}$ orthonormal can be found by the Gram-Schmidt orthogonalization process

$$\begin{aligned} \boldsymbol{\alpha}_1 &= \frac{1}{\sqrt{\varepsilon_1}} \boldsymbol{\delta}(1) \,, \\ \boldsymbol{\alpha}_t &= \frac{1}{\sqrt{\varepsilon_t}} \left(\boldsymbol{\delta}(t) - \sum_{i=1}^{t-1} \langle \boldsymbol{w}_i, \boldsymbol{x}_{j_i} \rangle \boldsymbol{\alpha}_i \right) \quad \text{for} \quad t > 1 \,, \end{aligned}$$

where the vectors $\boldsymbol{\delta}(t) \in \mathbb{R}^l$ has all entries equal 0 except for the t-th entry equal to 1. The normalization constants ε_t which ensure $\langle \boldsymbol{w}_t, \boldsymbol{w}_t \rangle = 1$ are computed as

$$\varepsilon_t = \langle \boldsymbol{x}_{j_t}, \boldsymbol{x}_{j_t} \rangle - \sum_{i=1}^{t-1} (\langle \boldsymbol{w}_i, \boldsymbol{x}_{j_t} \rangle) \,.$$

The greedy algorithm minimizing efficiently the upper bound on the mean square error is described by Algorithm 24.

Algorithm 24: Greedy minimization of upper bound on $\varepsilon_{MS}(\mathcal{T}|\mathcal{J})$

```
1. Initialization. Set $\mathcal{J}^{(0)} = \{\emptyset\}$ and $\varepsilon_i^{(0)} = k(x_i, x_i)$, $i \in \mathcal{I}$.
2. For $t = 1$ to $l$:
    (a) $j_t \in \operatorname*{argmax}_{j \in \mathcal{I} \setminus \mathcal{J}} \varepsilon_j^{(t-1)}$ .
    (b) $[\boldsymbol{z}_i]_t = \frac{1}{\sqrt{\varepsilon_{j_t}^{(t-1)}}} \left( k(x_i, x_{j_t}) - \sum_{h=1}^{t-1} [\boldsymbol{z}_i]_h [\boldsymbol{z}_{j_t}]_h \right) \quad \forall i \in \mathcal{I}$ .
    (c) $\boldsymbol{\alpha}_t = \frac{1}{\sqrt{\varepsilon_{j_t}^{(t-1)}}} \left( \boldsymbol{\delta}(t) - \sum_{i=1}^{t-1} [\boldsymbol{z}_{j_t}]_i \boldsymbol{\alpha}_i \right)$ .
    (d) $\varepsilon_i^{(t)} = \varepsilon_i^{(t-1)} - ([\boldsymbol{z}_i]_t)^2 \quad \forall i \in \mathcal{I}$ .
    (e) $\mathcal{J}^{(t)} = \mathcal{J}^{(t-1)} \cup \{j_t\}$ .
```

Algorithm 24 finds in each iteration one example $\boldsymbol{x}_{j_t}$ which contributes to the mean square error maximally. This example is included to the set $\mathcal{S}$ (or its index is included to $\mathcal{J}$ respectively) and the approximation errors are recomputed. Algorithm 24 goes from the iteration $t = 1$ to l and each iteration requires $O(mt)$ computations. Therefore the total computational complexity scales with $O(ml^2)$.

It is seen that Algorithm 24 is very simple to implement but the minimized upper bound can be too coarse approximation of the mean square error. The next section proposes an extension of the algorithm which aims to find a more precise approximation.

5.4 Greedy KPCA algorithm

The aim is to find basis vectors nearly as good as those selected by the naive greedy KPCA Algorithm 23. At the same time, the method should have a low computational requirements as Algorithm 24 which minimizes the upper bound on ε_{MS}. The idea is to combine these two algorithms together as described below.

Let $\mathcal{Z} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m\} \in \mathbb{R}^{l \times m}$ be a finite-dimensional representation of the training examples $\mathcal{T} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ found by Algorithm 24. Let it be further assumed that the number l of the selected basis vectors $\mathcal{S}$ is sufficiently high so that the mean square error $\varepsilon_{MS}(\mathcal{T}|\mathcal{J})$ is zero (in the worst case $l = m$). If the error $\varepsilon_{MS}(\mathcal{T}|\mathcal{J})$ is zero than obviously $\langle \boldsymbol{z}_i, \boldsymbol{z}_j \rangle = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$, $i, j \in \mathcal{I}$, which follows directly from (5.6). Now, the aim is to select a single vector $\boldsymbol{x}_{j_1}$ which would minimize the error $\varepsilon_{MS}(\mathcal{T}|\{j_1\})$ maximally, i.e.,

$$
\begin{aligned}
j_1 &\in \operatorname*{argmin}_{j \in \mathcal{I}} \frac{1}{m} \sum_{i=1}^{m} \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle}{\langle \boldsymbol{x}_j, \boldsymbol{x}_j \rangle} \right\|^2 \\
&= \operatorname*{argmin}_{j \in \mathcal{I}} \frac{1}{m} \sum_{i=1}^{m} \left(\langle \boldsymbol{x}_i, \boldsymbol{x}_i \rangle - \frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle^2}{\langle \boldsymbol{x}_j, \boldsymbol{x}_j \rangle} \right) \\
&= \operatorname*{argmin}_{j \in \mathcal{I}} \frac{1}{m} \sum_{i=1}^{m} \left(\langle \boldsymbol{z}_i, \boldsymbol{z}_i \rangle - \frac{\langle \boldsymbol{z}_i, \boldsymbol{z}_j \rangle^2}{\langle \boldsymbol{z}_j, \boldsymbol{z}_j \rangle} \right)
\end{aligned}
$$

$$= \operatorname*{argmax}_{j\in\mathcal{I}} \frac{\boldsymbol{z}_j^T}{\|\boldsymbol{z}_j\|}\left(\sum_{i=1}^m \boldsymbol{z}_i\boldsymbol{z}_i^T\right)\frac{\boldsymbol{z}_j}{\|\boldsymbol{z}_j\|} \,. \tag{5.7}$$

The evaluation of (5.7) yields the best j_1 minimizing $\varepsilon_{MS}(\mathcal{T}|\{j_1\})$. The number of computations is $O(ml^2)$ even if the computations required by Algorithm 24 to obtain $\mathcal{Z}$ are counted in. The idea is to use smaller number $p < l$ of basis vectors $\mathcal{S}$ for approximation of $\mathcal{T}$ but the formula (5.7) is still used. Hence j_1 is not optimal but the number of computations is $O(mp^2)$ which can be set reasonably small. The consequent indices $j_{t+1},\dots,j_l$ can be found using the same procedure but applied in the space perpendicular to previously selected basis vectors $\mathcal{S} = \{\boldsymbol{x}_{j_1},\dots,\boldsymbol{x}_{j_t}\}$.

The idea can be simply implemented by replacing Step (a) of Algorithm 24 by a new procedure described by Algorithm 25. The resulting algorithm will be referred to as the Greedy KPCA algorithm. The idea of selecting the basis vectors is illustrated in Figure 5.1.

Algorithm 25: Basis vector selection for Greedy KPCA (Replacement for Step (a))

1. Initialization. Set $\pi_i^{(0)} = \varepsilon_i^{(t)}$, $i \in \mathcal{I}$.
2. For $k = 1$ to p:

$$(a)\qquad r_k \in \operatorname*{argmax}_{j\in\mathcal{I}} \pi_j^{(k-1)} \,.$$

$$(b)\qquad [\boldsymbol{u}_i]_k = \frac{1}{\sqrt{\pi_{r_k}^{(k-1)}}}\left(k(x_i,x_{r_k}) - \sum_{j=1}^{t-1}[\boldsymbol{z}_i]_j[\boldsymbol{z}_{r_k}]_j - \sum_{j=1}^{k-1}[\boldsymbol{u}_i]_j[\boldsymbol{u}_{r_k}]_j\right) \,, \quad \forall i\in\mathcal{I}\,.$$

$$(c)\qquad \pi_i^{(k)} = \pi_i^{(k-1)} - ([\boldsymbol{u}_i]_k)^2 \,, \quad \forall i \in \mathcal{I}\,.$$

3. Select $j_t = \operatorname*{argmax}_{j\in\mathcal{I}\setminus\mathcal{J}^{(t-1)}} = \dfrac{\boldsymbol{u}_j^T}{\|\boldsymbol{u}_j\|}\left(\displaystyle\sum_{i=1}^m \boldsymbol{u}_i\boldsymbol{u}_i^T\right)\dfrac{\boldsymbol{u}_j}{\|\boldsymbol{u}_j\|}$.

The Greedy KPCA algorithm requires $O(mpl^2)$ operations as the procedure described by Algorithm 25 requires $O(mp(t+p))$ operations and is repeated for $t = 1$ to l and $p \ll l$.

After the algorithm halts, the set $\mathcal{J}^{(l)}$ contains indices of the selected training examples which form the basis $\mathcal{S}$. The matrix $\mathrm{A} = [\boldsymbol{\alpha}_1,\dots,\boldsymbol{\alpha}_l] \in \mathbb{R}^{l\times l}$ defines the orthonormal basis $\mathcal{W}$ which has the same linear span as the set $\mathcal{S}$. The set $\mathcal{Z} = \{\boldsymbol{z}_1,\dots,\boldsymbol{z}_m\} \in \mathbb{R}^{l\times m}$ contains the training examples $\mathcal{T} = \{\boldsymbol{x}_1,\dots,\boldsymbol{x}_m\}$ represented in the orthonormal basis $\mathcal{W}$. The mean square error can be simply evaluated as

$$\varepsilon_{MS}(\mathcal{T}|\mathcal{J}^{(l)}) = \frac{1}{m}\sum_{i=1}^m \varepsilon_i^{(l)} \,. \tag{5.8}$$

Because the vectors $\mathcal{Z}$ represent the training examples $\mathcal{T}$ in the orthonormal basis $\mathcal{W}$, then the following formula clearly holds

$$\langle \tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j\rangle = \langle \boldsymbol{z}_i, \boldsymbol{z}_j\rangle \,.$$

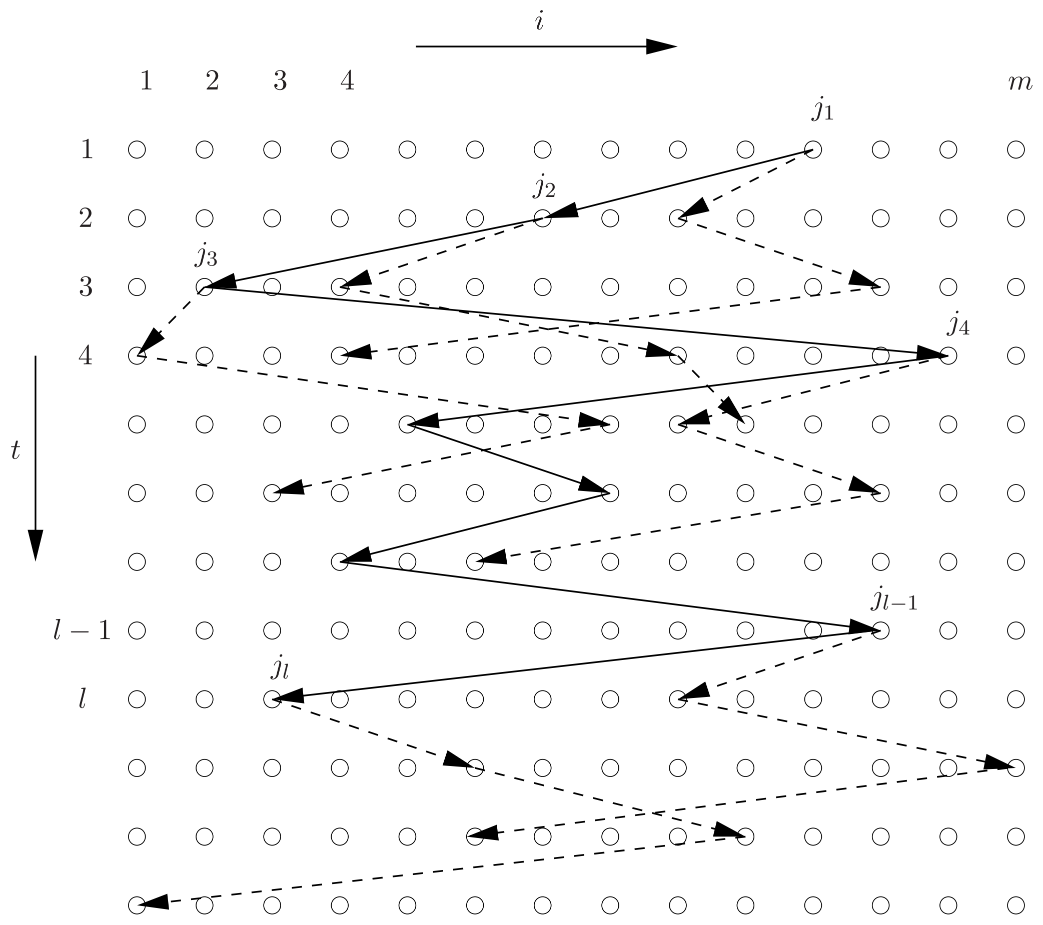


Figure 5.1: The idea of selecting the basis vectors by the Greedy KPCA algorithm. The solid line shows indices of basis vectors selected in Step 3 of the procedure described by Algorithm 25. The dashed line shows indices of $p = 3$ basis vectors selected in Step 2(a) of the same procedure.

Let $\mathrm{Z} = [\boldsymbol{z}_1, \dots, \boldsymbol{z}_m] \in \mathbb{R}^{l\times m}$ be vectors from the set $\mathcal{Z}$ represented in a matrix. The matrix Z can be used to approximate the original kernel matrix K so that

$$\mathrm{K} \approx \mathrm{Z}^T\mathrm{Z}\,.$$

The approximation is perfect, i.e., $\mathrm{K} = \mathrm{Z}^T\mathrm{Z}$, if the mean square error $\varepsilon_{MS}(\mathcal{T}|\mathcal{J}^{(l)})$ is equal to zero which follows from (5.6).

So far the number l of selected basis vectors has been assumed fixed beforehand. It can be also useful in practice to select such a number l that the mean square error $\varepsilon_{MS}(\mathcal{T}|\mathcal{J}^{(t)})$ drops below a prescribed limit. This stopping condition can be implemented simply because the error $\varepsilon_{MS}(\mathcal{T}|\mathcal{J}^{(t)})$ is known in each iteration due to formula (5.8).

5.5 Approximation to regularized risk minimization

Let $\mathcal{T}_{\mathcal{X}\mathcal{Y}} = \{(x_1, y_1), \dots, (x_m, y_m)\} \in (\mathcal{X}\times\mathcal{Y})^m$ be a training set. The input states are assumed to be represented in the feature space $\mathcal{H}$ via the map $\Phi\colon \mathcal{X} \to \mathcal{H}$. The learning of function $f(x) = \langle \Phi(x), \boldsymbol{w} \rangle + b$ can be defined as the regularized risk minimization

$$(\boldsymbol{w}^*, b^*) = \operatorname*{argmin}_{\boldsymbol{w}\in\mathcal{H}, b\in\mathbb{R}} \frac{1}{m}\sum_{i=1}^{m} V(y_i, f(x_i)) + \lambda\|\boldsymbol{w}\|^2 \,, \tag{5.9}$$

where

$$f(x) = \langle \boldsymbol{w}, \Phi(x) \rangle + b = \sum_{i=1}^{m} \alpha_i \langle \Phi(x_i), \Phi(x) \rangle + b = \sum_{i=1}^{m} \alpha_i k(x_i, x) + b \,,$$

according to the Representer Theorem 1.1. The vector $\boldsymbol{w} \in \mathcal{H}$ lies in the linear span of $\mathcal{T} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$.

The Greedy KPCA algorithm selects a subset $\mathcal{S} = \{\boldsymbol{x}_{j_1}, \ldots, \boldsymbol{x}_{j_l}\}$ such that $\mathrm{Span}(\mathcal{S})$ approximates the $\mathrm{Span}(\mathcal{T})$. Let $\tilde{f}(x)$ be a function defined as

$$\tilde{f}(x) = \langle \tilde{\boldsymbol{w}}, \Phi(x) \rangle + \theta = \sum_{j\in\mathcal{J}} \beta_j \langle \Phi(x_j), \Phi(x) \rangle + \theta = \sum_{j\in\mathcal{J}} \beta_j k(x_j, x) + \theta \,, \tag{5.10}$$

i.e., $\tilde{\boldsymbol{w}} \in \mathcal{H}$ lies in the $\mathrm{Span}(\mathcal{S})$ and $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_l]^T \in \mathbb{R}^l$ are its coordinates in the basis $\mathcal{S}$. The function $\tilde{f}(x)$ can be learned using the regularized risk minimization

$$(\tilde{\boldsymbol{w}}^*, \theta^*) = \operatorname*{argmin}_{\tilde{\boldsymbol{w}}\in\mathrm{Span}(\mathcal{S}), \theta\in\mathbb{R}} \frac{1}{m}\sum_{i=1}^{m} V(y_i, \tilde{f}(x_i)) + \lambda\|\tilde{\boldsymbol{w}}\|^2 \,. \tag{5.11}$$

The learning task (5.11) is an approximation to the task (5.9) because $\tilde{\boldsymbol{w}}$ is selected from $\mathrm{Span}(\mathcal{S}) \subset \mathcal{H}$ instead of the whole feature space $\mathcal{H}$. The approximated task (5.11) can be further simplified when the vector $\tilde{w}$ is sought in the orthonormal basis $\mathcal{W} = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_l\}$ instead of the basis $\mathcal{S} = \{\boldsymbol{x}_{j_1}, \ldots, \boldsymbol{x}_{j_l}\}$, i.e.,

$$\tilde{\boldsymbol{w}} = \sum_{i=1}^{l} \beta_i \Phi(x_{j_i}) = \sum_{i=1}^{l} v_i \boldsymbol{w}_i \,, \tag{5.12}$$

where $\boldsymbol{v} = [v_1, \ldots, v_l]^T \in \mathbb{R}^l$ is a vector which determines $\tilde{\boldsymbol{w}}$ in the basis $\mathcal{W}$. Substituting (5.12) to (5.11) gives

$$(\boldsymbol{v}^*, \theta^*) = \operatorname*{argmin}_{\boldsymbol{v}\in\mathbb{R}^l, \theta\in\mathbb{R}} \frac{1}{m}\sum_{i=1}^{m} V(y_i, \tilde{f}(x_i)) + \lambda\|\boldsymbol{v}\|^2 \,, \tag{5.13}$$

and

$$\tilde{f}(x) = \sum_{i=1}^{l} v_i \langle \boldsymbol{w}_i, \Phi(x) \rangle + \theta = \langle \boldsymbol{v}, \mathrm{A}^T \boldsymbol{k}_s(x) \rangle + \theta \,.$$

An advantage is that the learning of task (5.13) requires only the finite-dimensional representation $\mathcal{Z} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m\} \in \mathbb{R}^{l\times m}$ of the input examples $\mathcal{T} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ because $\tilde{f}(x_i) = \langle \boldsymbol{v}, \boldsymbol{z}_i \rangle + \theta$, $i \in \mathcal{I}$. It can be also convenient to work explicitly with the finite-dimensional vectors $\mathcal{Z}$ instead of expressing the learning in terms of the dot products.

5.6 Experiments

5.6.1 Minimization of reconstruction error

In this experiment, the Greedy KPCA algorithm is compared to the ordinary KPCA (based on the eigenvalue decomposition) in terms of the mean square reconstruction error and the sparseness of the data approximation. The aim is to investigate how fast the Greedy KPCA decreases the reconstruction error compared to the optimal solution of the ordinary KPCA. The sparseness of the data approximation means the number of training data which determines the projection onto the basis vectors. The sparseness is linearly proportional to the evaluation time of the data projection.

The training data and the kernel parameters are adopted from the experiment described in Section 4.10.2. The first 6 data sets of the IDA benchmark repository are used only. The comparison was performed on the first realizations of the training part. The kernel parameters which produced the best binary SVM classifier with L_2-soft margin are used here.

The ordinary and the Greedy KPCA algorithms were implemented in Matlab 6 language. The Matlab function `eig` was used to solve the eigenvalue problem of the ordinary KPCA. The implementation was optimized neither for speed nor the memory requirements but it was found sufficient for all tested problems. However, there is still a large space for improvements in this direction.

The number l of basis vectors required to achieve the mean square reconstruction errors $\varepsilon_{MS} \in \{0.1, 0.01, 0.001\}$ was recorded for both the Greedy KPCA and the ordinary KPCA. The results obtained are listed in Table 5.1. As expected, the Greedy KPCA needs more basis vectors to achieve desired error ε_{MS} than the optimal KPCA. This is especially apparent at the first stages ($\varepsilon_{MS} = 0.1$). However, this difference decreases as the number of basis vectors grows. The Greedy KPCA algorithm required on average 30% more basis vectors than the optimal KPCA to achieve the reconstruction error $\varepsilon_{MS} = 0.001$. It is worth mentioning that the Greedy KPCA requires only l training data to represent l basis vectors while the ordinary KPCA always requires all the m training data. This implies that the projection function found by the Greedy KPCA is much sparser and consequently the evaluation of the projection function is faster. The evaluation time for the projection function corresponding to the $\varepsilon_{MS} = 0.001$ is listed in the rightmost column of Table 5.1.

5.6.2 Approximated regularized least squares

This section describes how to use the proposed Greedy KPCA to find an approximated solution of the regularized least squares problem [9]. Let $\mathcal{T}_{\mathcal{XY}} = \{(x_1, y_1), \dots, (x_m, y_m)\} \in (\mathcal{X} \times \mathbb{R})^m$ be a training set of examples. The hidden state is assumed to be a real number $y \in \mathbb{R}$. The input observations $x \in \mathcal{X}$ are assumed to be represented in the feature space $\mathcal{H}$ via a kernel function $k \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. The aim is to learn a function $f(x) \in \mathcal{H}$ such that the quadratic loss function $V(y, f(x)) = (y - f(x))^2$ is minimized. The learning task can stated as the regularized risk minimization

$$f^* = \operatorname*{argmin}_{f \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} (y_i - f(x_i))^2 + \lambda \|f\|^2 \,, \tag{5.14}$$

where $\lambda \in \mathbb{R}^+$ is some regularization constant. The value of λ has to be determined based on other principle, e.g., using cross-validation or independent validation training data.

Problem	Algorithm		ε_{MS}			Eval
			0.1	0.01	0.001	time [%]
Banana	Ordinary KPCA		5	14	22	100.0
(m=400)	Greedy	$p = 1$	10	24	38	9.5
	KPCA	$p = 25$	8	18	28	7.0
		$p = 50$	8	17	28	7.0
Breast	Ordinary KPCA		9	47	96	100.0
(m=200)	Greedy	$p = 1$	25	107	158	79.0
	KPCA	$p = 25$	18	73	121	60.5
		$p = 50$	17	71	118	59.0
Diabetis	Ordinary KPCA		1	8	32	100.0
(m=468)	Greedy	$p = 1$	5	16	57	12.2
	KPCA	$p = 25$	2	12	46	9.8
		$p = 50$	2	12	43	9.2
Flare	Ordinary KPCA		2	12	32	100.0
(m=666)	Greedy	$p = 1$	3	38	154	23.1
	KPCA	$p = 25$	3	38	43	6.5
		$p = 50$	3	17	46	2.6
German	Ordinary KPCA		18	165	400	100.0
(m=700)	Greedy	$p = 1$	45	360	665	95.0
	KPCA	$p = 25$	42	285	524	74.9
		$p = 50$	38	277	513	73.3
Heart	Ordinary KPCA		4	23	73	100.0
(m=170)	Greedy	$p = 1$	9	48	116	68.2
	KPCA	$p = 25$	7	43	101	59.4
		$p = 50$	7	41	97	57.1

Table 5.1: Comparison between the Greedy KPCA and the ordinary KPCA in terms of ability to decrease the mean square reconstruction error. The evaluation time (projection on a single basis vector) required by the Greedy KPCA representation is reported relatively to the time of the ordinary KPCA.

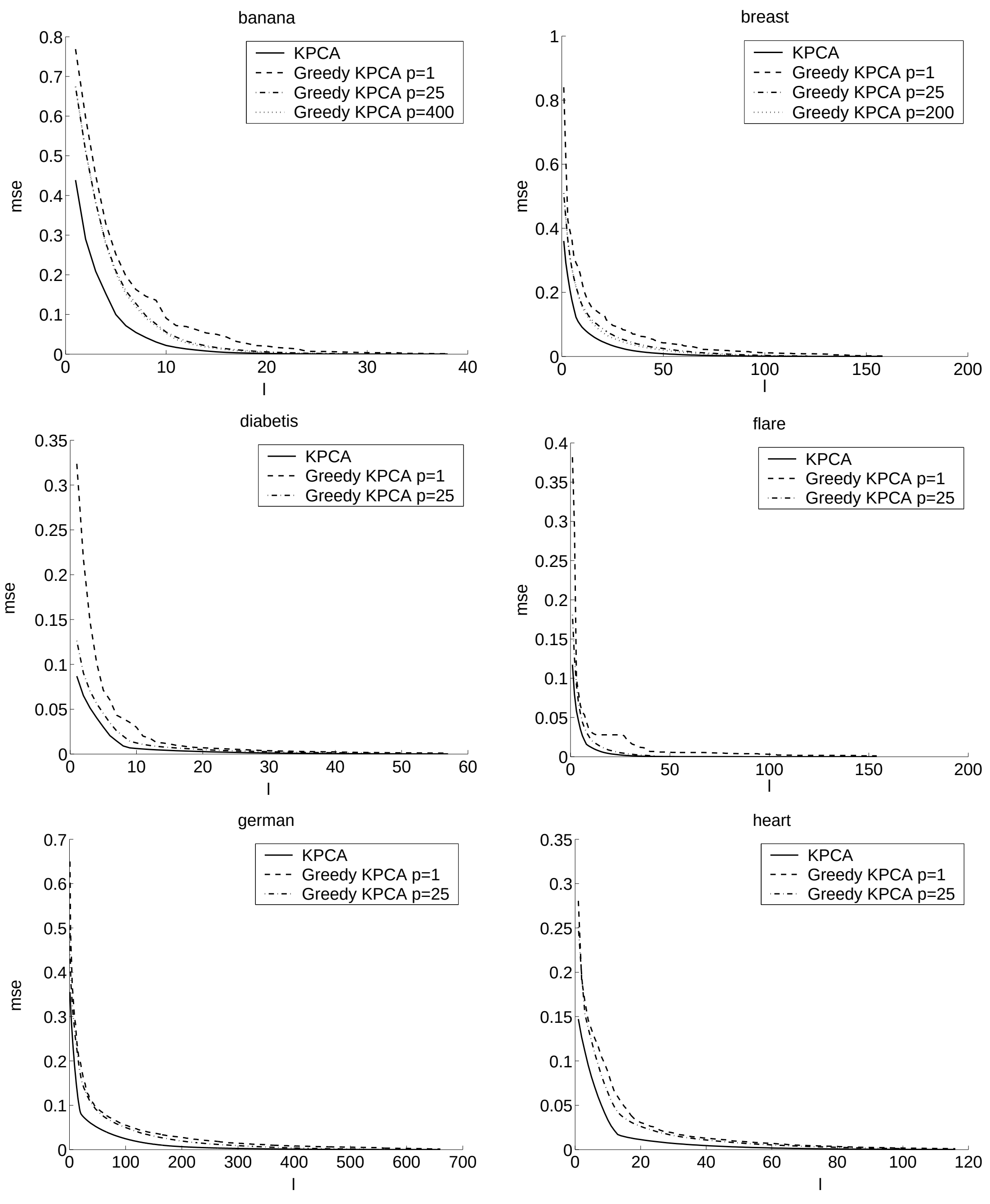


Figure 5.2: Comparison of the Greedy KPCA and the ordinary KPCA algorithm on the first six data sets of the IDA benchmark repository. The mean square reconstruction error ε_{MS} with respect to the number of basis vectors l is displayed.

According to the Representer Theorem 1.1, the resulting function has the form

$$f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x) \,, \tag{5.15}$$

where $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_m]^T \in \mathbb{R}^m$ is an unknown vector. Substituting (5.15) to (5.14) and solving for $\boldsymbol{\alpha}$ gives

$$\boldsymbol{\alpha}^* = (\mathrm{K} + m\lambda\mathrm{E})^{-1}\boldsymbol{y} \,, \tag{5.16}$$

where $\mathrm{K} \in \mathbb{R}^{m\times m}$ is the kernel matrix, $\mathrm{E} \in \mathbb{R}^{m\times m}$ is the identity matrix and $\boldsymbol{y} = [y_1, \dots, y_m]^m \in \mathbb{R}^m$ is a vector which contains the values of hidden states. Thus the learning consists of evaluation formula (5.16) which can be a demanding task due to the computation of matrix inverse when m is large.

The problem can be alleviated using the approximation described in Section 5.5. The Greedy KPCA finds a finite-dimensional representation $\mathcal{Z} = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_m\} \in \mathbb{R}^{l\times m}$ of input examples mapped to the feature space $\mathcal{H}$. The resulting function reads

$$\tilde{f}(x) = \langle \boldsymbol{v}, \mathrm{A}^T \boldsymbol{k}_s(x) \rangle \,,$$

where the vector $\boldsymbol{v} \in \mathbb{R}^l$ is learned by solving

$$\boldsymbol{v}^* = \operatorname*{argmin}_{\boldsymbol{v}\in\mathbb{R}^l} \frac{1}{m} \sum_{i=1}^{m} (y_i - \langle \boldsymbol{v}, \boldsymbol{z}_i \rangle)^2 + \lambda \|\boldsymbol{v}\|^2 = (\mathrm{Z}^T\mathrm{Z} + m\lambda\mathrm{E})^{-1}\mathrm{Z}\boldsymbol{y} \,. \tag{5.17}$$

The matrix $\mathrm{Z} = [\boldsymbol{z}_1, \dots, \boldsymbol{z}_m] \in \mathbb{R}^{l\times m}$ contains the vectors of the set $\mathcal{Z}$ and $\mathrm{E} \in \mathbb{R}^{l\times l}$ is the identity matrix. The learning of the approximated function $\tilde{f}$ can be considerably simpler as the formula (5.17) requires inversion of the matrix of size $l < m$.

The described method is evaluated on a synthetical problem with known ground truth function to be learned from data. It is assumed that the input space is $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^2 \colon \boldsymbol{x} = [x_1, x_2]^T, -2 \le x_1 \le 2, -2 \le x_2 \le 2\}$ and the statistical model is given by

$$P(y|\boldsymbol{x}) = N_1(y; f^\star(\boldsymbol{x}), 0.3) \quad \text{and} \quad P(x) = N_2(\boldsymbol{x}; 0; \mathrm{E}) \,.$$

The symbol $N_1(y; f^\star(\boldsymbol{x}), 0.3)$ stands for an univariate Gaussian distribution with mean value $f^\star(\boldsymbol{x})$ and standard deviation 0.3 and $N_2(\boldsymbol{x}; 0; \mathrm{E})$ stands for bivariate Gaussian with zero mean and the identity covariance matrix. The function $f^\star(\boldsymbol{x})$ is

$$f^\star(\boldsymbol{x}) = x_1 \exp(-(x_1)^2 - (x_2)^2) \,.$$

It is easy to show that the function $f^\star$ minimizes the Bayesian risk with quadratic loss function, i.e.,

$$f^\star(\boldsymbol{x}) = \operatorname*{argmin}_{y'\in\mathbb{R}} \int_{\mathbb{R}} P(y|\boldsymbol{x})(y - y')^2 \mathrm{d}y = \int_{\mathbb{R}} yP(y|\boldsymbol{x}) \mathrm{d}y = \int_{\mathbb{R}} yN(y; f^\star(\boldsymbol{x}), 0.3) \mathrm{d}y \,. \tag{5.18}$$

The aim is to learn a function $f(x)$ which minimizes the Bayesian risk from training examples $\mathcal{T}_{\mathcal{X}\mathcal{Y}} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_m, y_m)\} \in (\mathbb{R}^2 \times \mathbb{R})^m$. The number of $m = 100000$ training examples was generated from the known underlying model. The indices $\mathcal{I} = \{1, \dots, m\}$ of the training examples was split into two subsets $\mathcal{I} = \mathcal{I}_{trn} \cup \mathcal{I}_{val}$ both having $m_{trn} = m_{val} = 50000$ indices.

A common approach in the kernel based machine learning is to replace the Bayesian risk with the regularized risk (5.14). The minimization of (5.14) produces function $f(x)$ which, however, depends on the kernel function $k(x, x')$ and the regularization constant λ. These unknown parameters can be selected based on minimization of the validation risk

$$R_{val}[f] = \frac{1}{m_{val}} \sum_{i \in \mathcal{I}_{val}} (y_i - f(\boldsymbol{x}_i))^2 \,. \tag{5.19}$$

In this case the learning of f becomes intractable because of the computation of the inversion of the matrix K which is of size $m_{trn} = 50000$. Moreover, this inversion must be computed for each pair of a kernel function and a regularization constant. The approximated solution (5.17) can be used instead. In this particular case, the setting was the following:

The set of polynomial kernels $k(\boldsymbol{x}, \boldsymbol{x}') = (\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + 1)^d$ with $d \in \{1, 2, \ldots, 12\}$ and Gaussian kernels $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2})$ with $\sigma \in \{0.1, 0.25, 0.5, 0.75, 1, 1.25, 1.5\}$ was used.

The set of regularization constants was $\lambda \in \{10^{-9}, \ldots, 10^4\}$.

The Greedy KPCA algorithm ran until the number of basis vectors reached number $l = 100$ or the error ε_{MS} dropped below 0.1. The number of basis vectors in the inner loop of the Greedy KPCA was set to $p = 20$.

The results obtained are summarized in Figure 5.3. The validation risk $R_{val}[f]$ for the function learning with the Gaussian kernel is displayed in Figure 5.3(a). The validation risk for the function with polynomial kernel is displayed in Figure 5.3(b). The risk is a function of the kernel parameter and the regularization constant. The displayed values are the minimum values with respect to the regularization constant its influence was only negligible. The function learned with the Gaussian kernel $\sigma = 0.5$ and the $\lambda = 10^{-8}$ had the smallest validation risk $R_{val} = 0.090$ and its shape can be seen in Figure 5.3(c). The ground truth function $f^\star$ given by (5.18) had the validation risk $R_{val} = 0.0898$ and its shape is displayed in Figure 5.3(d).

5.6.3 Reducing complexity of SVM classifier

This section describes how to use the proposed Greedy KPCA to find an approximated solution of the binary SVM problem. This problem arises from the minimization of the regularized risk (5.9) if the hard margin, L_1-soft, or L_2-soft margin loss functions are used (c.f. Section 1.4.1). The approximation of this problem can be computed by (i) applying the the Greedy KPCA on the input training set and (ii) solving the approximated task (5.13). The result is the discriminant function of the form (5.13). Its complexity, i.e., the number l of selected training data, can be set up before the training of the classifier begins. The open question is how to select the number l to obtain a classifier with a low classification error. This question can be answered in the applications in which a limit on the evaluation time is strictly given, thus there is no freedom in tuning the proper value of l. In other cases, l must be tuned similarly to the kernel parameters and the regularization constant.

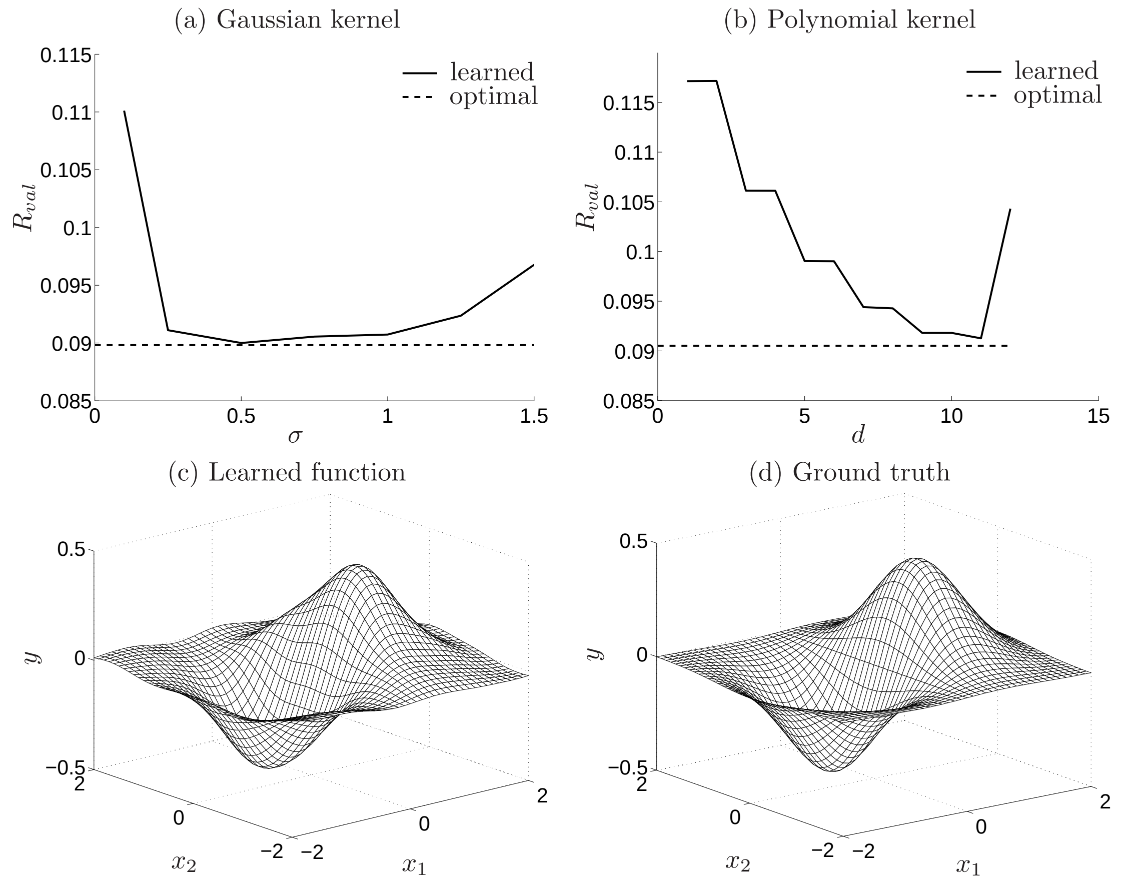


Figure 5.3: Figure (a) shows the value of the validation risk $R_{val}[f]$ with respect to the width σ of the Gaussian kernel. Figure (b) shows $R_{val}[f]$ with respect to the degree d of the polynomial kernel. The ground truth function $f^\star$ is displayed in Figure (c) and the learned function f with the smallest validation error is displayed in Figure (d).

Figure 5.4 illustrates the application of the Greedy KPCA for approximation of the SVM L_1-soft margin classifier on a toy Ripley data set [43]. The Gaussian kernel with $\sigma = 1$ and the regularization constant $C = 2/(m\lambda) = 10$ was used. Figure 5.4(a) shows the SVM classifier without approximation and Figure 5.4(b),(c), and (d) displays the approximated solution with increasing number of selected basis vectors l.

The approximation was also tested on the IDA repository and the associated experimental protocol described in Section 4.10.2. The binary SVM classifier with both the L_1-soft and L_2-soft margin is considered. The free parameters of the SVM, i.e., the regularization constant and the kernel argument of the Gaussian kernel, were adopted from the experiment in Section 4.10.2. The number l of basis vectors which allowed to approximate the training examples with $\varepsilon_{MS} < 0.001$ was used but l could not exceed the half of the number m of the training examples.

The results on the IDA repository are listed in Table 5.2. It is seen that the approximated SVM classifier achieves similar recognition rates compared to the ordinary SVM. Even more, in some cases the results are even better which could be a result of the reduced

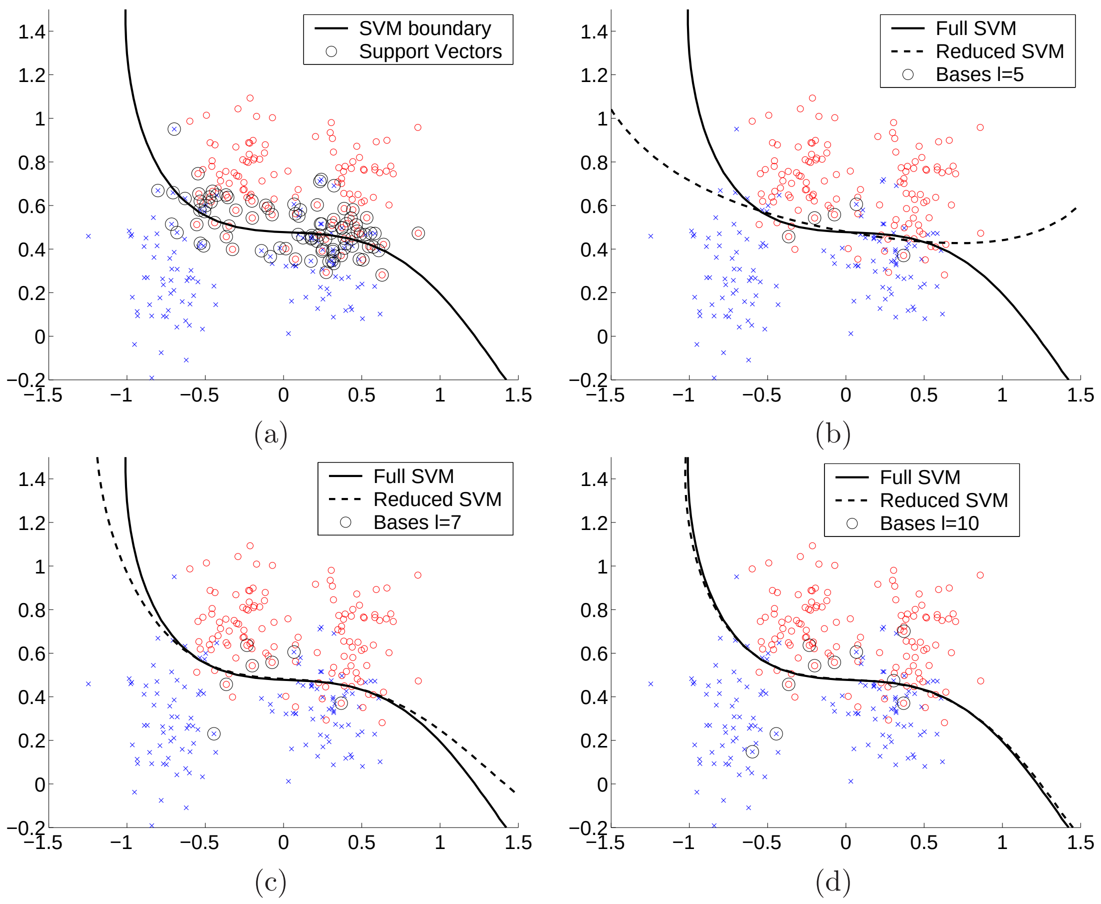


Figure 5.4: Illustration of the Greedy KPCA used to control complexity of the binary SVM classifier with L_1-soft margin. The figures show the SVM boundary without approximation and the boundaries of approximated classifiers constructed in the lower dimensional space represented by l basis vectors. The selected training data, i.e., basis vectors, are marked by circles.

complexity of the function space. It should be pointed out that the kernel parameter and the regularization constant were not optimized for the approximated classifier. The complexity of the ordinary SVM classifier is determined by the number of the support vectors. The complexity of the approximated SVM classifier is given by the number of the basis vectors selected by the Greedy KPCA. The complexity is linearly proportional to the evaluation time of the classifier. The last column of Table 5.2 contains the evaluation time of the approximated classifier relatively to the ordinary SVM classifier. In 60% of cases, the number of basis vectors was smaller than the number of the support vectors which yields the classification rule with reduced evaluation time. In the case of the L_1-soft margin SVM, the average evaluation time was reduced to 60% compared to the ordinary SVM. In the case of the L_2-soft margin SVM, the average evaluation time is reduced to 50%.

It can be seen that the approximation is useful for the cases in which the ordinary SVM classifier has many support vectors, i.e., the cases with highly overlapping training data.

Problem	Ordinary SVM L_1		Greedy KPCA + Linear SVM L_1		Eval time
	err [%]	#SV	err [%]	l	[%]
Banana	10.4 ± 0.4	139 ± 8	10.4 ± 0.4	61 ± 2	43.9
Breast	26.1 ± 4.9	113 ± 6	25.7 ± 4.7	60 ± 2	53.1
Diabetes	23.2 ± 1.7	255 ± 8	23.2 ± 1.7	82 ± 2	32.2
German	23.7 ± 2.2	420 ± 12	23.8 ± 2.2	350 ± 0	83.3
Heart	15.7 ± 3.3	98 ± 5	15.6 ± 3.3	85 ± 0	86.7
Image	3.0 ± 0.5	147 ± 8	3.3 ± 0.6	270 ± 8	183.7
Ringnorm	1.6 ± 0.1	131 ± 7	1.5 ± 0.1	200 ± 0	153.9
Splice	11.1 ± 0.6	594 ± 16	12.5 ± 0.7	500 ± 0	84.2
Thyroid	4.7 ± 2.3	20 ± 3	4.6 ± 2.3	63 ± 3	315.0
Titanic	22.4 ± 1.0	69 ± 10	22.4 ± 1.0	16 ± 9	23.2
Twonorm	3.0 ± 0.4	57 ± 7	3.6 ± 0.4	200 ± 0	350.9
Waveform	10.1 ± 0.4	129 ± 9	10.1 ± 0.4	200 ± 0	155.0

Problem	Ordinary SVM L_2		Greedy KPCA + Linear SVM L_2		Eval time
	err [%]	#SV	err [%]	l	[%]
Banana	10.5 ± 0.5	210 ± 15	10.5 ± 0.5	28 ± 1	13.3
Breast	26.0 ± 4.5	187 ± 5	25.8 ± 4.5	100 ± 0	53.5
Diabetes	23.1 ± 1.8	414 ± 7	23.1 ± 1.8	43 ± 1	10.4
German	23.4 ± 2.3	613 ± 12	23.5 ± 2.2	350 ± 0	57.1
Heart	16.0 ± 3.1	137 ± 6	16.0 ± 3.0	85 ± 0	62.0
Image	3.1 ± 0.6	365 ± 11	3.2 ± 0.6	555 ± 15	152.1
Ringnorm	1.7 ± 0.1	132 ± 8	1.9 ± 0.2	200 ± 0	151.5
Splice	11.2 ± 0.7	597 ± 16	12.2 ± 0.9	500 ± 0	83.8
Thyroid	4.5 ± 2.1	67 ± 4	5.0 ± 2.1	70 ± 0	100.4
Titanic	22.4 ± 1.1	150 ± 0	22.4 ± 1.1	12 ± 2	54.6
Twonorm	2.7 ± 0.2	123 ± 8	2.7 ± 0.2	200 ± 0	162.6
Waveform	10.1 ± 0.4	175 ± 13	10.1 ± 0.4	200 ± 0	114.3

Table 5.2: An evaluation of the Greedy KPCA used to control complexity of the binary SVM classifier on the IDA repository. The stopping condition of Greedy KPCA is a limit on the error $\varepsilon_{MS} < 0.001$ and maximal number of basis vectors $l < \frac{m}{2}$.

5.7 Summary to Greedy KPCA

This chapter describes a novel method named the Greedy KPCA algorithm the preliminary versions of which were published in [14, 16]. The Greedy KPCA algorithm aims to find a lower-dimensional representation of data embedded into the RKHS space similarly to the ordinary KPCA. In contrast to the KPCA, the basis vectors are selected directly from the training set and are not expressed as a linear combination of all the training vectors. As a result, the found representation of data is suboptimal in terms of the reconstruction error but the representation is less complex compared to the ordinary KPCA.

The Greedy KPCA can be employed to reduce computational and memory requirements of an arbitrary kernel method before. The method was tested in connection to the regularized kernel least squares approximation and binary SVM classifier. The method turned out to be useful in cases in which a big portion of training data is required to represent the solution function.

The method was also intended to be used to accelerate evaluation of the multiclass SVM classifier designed for Optical Character Recognition (OCR) system (see Section 6.2.2). However, it turned out that the linear classifier is sufficiently precise in this case and thus the approximation is not needed. Therefore practical applications which would be resolved thanks to the proposed Greedy KPCA algorithm is still sought.

The Greedy KPCA is closely related to the Sparse Greedy Matrix Approximation proposed by Smola and Schöelkopf [46, 48] as described in Section 3.4. The main difference is that their method uses a randomized selection of the basis vectors instead of the deterministic Greedy KPCA algorithm.

6 Multiclass Support Vector Machines

This section focuses on learning of SVM multiclass classifiers. More exactly, a modified multiclass SVM formulation named the Bounded Support Vector Machines (BSVM) [25] is in question. The multiclass BSVM formulation was described is Section 3.5. A novel contribution proposed in this thesis will show that QP task associated with the multiclass BSVM can be transformed to an equivalent singleclass SVM problem. The aim of this transformation is to facilitate the optimization because the optimization of the singleclass problem is simpler compared to the multiclass case. The proposed method for optimization of the multiclass BSVM formulation is evaluated on the standard benchmark data and it helped to solve a real application of Optical Character Recognition (OCR) system.

6.1 From multiclass BSVM to singleclass SVM

Let $\mathcal{T}_{\mathcal{XY}} = \{(x_1, y_1), \dots, (x_m, y_m)\} \in (\mathcal{X} \times \mathcal{Y})^m$ be a training set of examples. The output state is assumed to attain a value from the set $\mathcal{Y} = \{1, \dots, M\}$, where $M > 2$. The input states $\{x_1, \dots, x_m\}$ are represented as vectors $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}$ in the feature space $\mathcal{H}$ given by a selected kernel function $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. The task is to learn a classification rule $q\colon \mathcal{X} \to \mathcal{Y}$ which estimates the output state $y \in \mathcal{Y}$ from the input state $x \in \mathcal{X}$. The classification rule q is composed of discriminant functions $f_y(x) = \langle \boldsymbol{w}_y, \Phi(x) \rangle + b_y$, $y \in \mathcal{Y}$ which are here assumed to be linear in the feature space $\mathcal{H}$. The resulting classification rule is defined as

$$q(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} f_y(x) \,. \tag{6.1}$$

The learning of unknown parameters $\boldsymbol{w}_y \in \mathcal{H}$, $b_y \in \mathbb{R}$, $y \in \mathcal{Y}$ is expressed as the multiclass BSVM formulation. The multiclass BSVM formulation arises from adding the term $\frac{1}{2} \sum_{y \in \mathcal{Y}} (b_y)^2$ to the multiclass SVM formulation (1.21), which yields

$$(\boldsymbol{w}^*, b^*, \boldsymbol{\xi}^*) = \operatorname*{argmin}_{\boldsymbol{w}, b, \xi} \frac{1}{2} \sum_{y \in \mathcal{Y}} (\|\boldsymbol{w}_y\|^2 + b_y^2) + C \sum_{i \in \mathcal{I}} \sum_{y \in \mathcal{Y} \setminus \{y_i\}} (\xi_i^y)^p \,, \tag{6.2}$$

subject to

$$\begin{array}{rcll} \langle \boldsymbol{w}_{y_i}, \boldsymbol{x}_i \rangle + b_{y_i} - (\langle \boldsymbol{w}_y, \boldsymbol{x}_i \rangle + b_y) & \geq & 1 - \xi_i^y \,, & i \in \mathcal{I},\ y \in \mathcal{Y} \setminus \{y_i\} \,, \\ \xi_i^y & \geq & 0 \,, & i \in \mathcal{I},\ y \in \mathcal{Y} \setminus \{y_i\} \,. \end{array} \tag{6.3}$$

The first term, which was modified, controls the complexity of the learned discriminant functions. Of course, in the case of unbiased discriminant functions $b = 0$, the BSVM and the original multiclass SVM formulation (1.21) coincide. The second term in the criterion (6.2) approximates the empirical classification error as in the original formulation (1.21). The constant $p = \{\infty, 1, 2\}$ determines the type of a loss function which penalizes misclassifications.

In the following text, a transformation will be introduced which transforms the multiclass BSVM problem (6.2) to the singleclass SVM problem. The singleclass SVM problem was described in Section 1.4.3. The QP task associated to the singleclass SVM reads

$$(\boldsymbol{w}^*, \boldsymbol{\xi}^*) = \operatorname*{argmin}_{\boldsymbol{w},\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i\in\mathcal{I}}\sum_{y\in\mathcal{Y}\setminus\{y_i\}} (\xi_i^y)^p \,, \tag{6.4}$$

subject to

$$\begin{array}{rcll} \langle \boldsymbol{w}, \boldsymbol{z}_i^u \rangle & \geq & 1 - \xi_i^u \,, & i \in \mathcal{I}\,, y \in \mathcal{Y}\setminus\{y_i\}\,, \\ \xi_i^u & \geq & 0\,, & i \in \mathcal{I}\,, y \in \mathcal{Y}\setminus\{y_i\}\,. \end{array} \tag{6.5}$$

The task (6.4) is better solved in its dual form. The dual forms for individual types of loss functions ($p = \infty, 1, 2$) were introduced in Section 1.4.3. An important point is that the dual formulation requires the data in terms of dot products. The corresponding QP task can be solved efficiently by various optimization algorithms. Chapter 4 contains algorithms suitable for the hard margin ($p = \infty$) and L_2-soft margin ($p = 2$) formulations.

The transformation from the multiclass BSVM problem to the singleclass SVM problem is based on the Kesler's construction[1] [8, 44]. For simplicity, it is assumed that training data are expressed in a finite-dimensional feature space $\mathcal{H} \subseteq \mathbb{R}^n$. The Kesler's construction maps the training data from the n-dimensional feature space $\mathcal{H}$ into to a new $(n+1)M$-dimensional space $\mathcal{Z}$ in which the multiclass problem appears as a singleclass problem. Each training vector $\boldsymbol{x}_i$ is mapped to new $(M-1)$ vectors $\boldsymbol{z}_i^y$, $y \in \mathcal{Y}\setminus\{y_i\}$ defined as follows. Let the coordinates of $\boldsymbol{z}_i^y$ be divided into M slots, i.e., $\boldsymbol{z}_i^y = [\boldsymbol{z}_i^y(1); \ldots; \boldsymbol{z}_i^y(M)]$. The individual slot $\boldsymbol{z}_i^y(j)$, $j \in \mathcal{Y}$ has $n+1$ coordinates which are defined as

$$\boldsymbol{z}_i^y(j) = \begin{cases} [\boldsymbol{x}_i; 1]\,, & \text{for} \quad j = y_i\,, \\ -[\boldsymbol{x}_i; 1]\,, & \text{for} \quad j = y\,, \\ 0\,, & \text{otherwise.} \end{cases} \tag{6.6}$$

Let the vector $\boldsymbol{w} \in \mathcal{H}$ be composed of vectors $\boldsymbol{w}_y$, $y \in \mathcal{Y}$ and scalars b_y, $y \in \mathcal{Y}$ such that

$$\boldsymbol{w} = [[\boldsymbol{w}_1; b_1]; [\boldsymbol{w}_2; b_2]; \ldots; [\boldsymbol{w}_M; b_M]]\,. \tag{6.7}$$

For instance, when $M = 4$ and $y_i = 3$ then the vectors $\boldsymbol{z}_i^y$, $y = \{1,2,3,4\}\setminus\{3\}$ are constructed as

$$\begin{array}{rcccccc} \boldsymbol{z}_i^1 & = & [\; -[\boldsymbol{x}_i;1]\,; & 0\,; & [\boldsymbol{x}_i;1]\,; & 0\,; &] \\ \boldsymbol{z}_i^2 & = & [\; 0\,; & -[\boldsymbol{x}_i;1]\,; & [\boldsymbol{x}_i;1]\,; & 0\,; &] \\ \boldsymbol{z}_i^4 & = & [\; 0\,; & 0\,; & [\boldsymbol{x}_i;1]\,; & -[\boldsymbol{x}_i;1]\,; &] \end{array}$$

Performing the transformation (6.6) yields a set $\mathcal{T}_{\mathcal{Z}} = \{\boldsymbol{z}_i^m \in \mathcal{Z} \colon i \in \mathcal{I}\,, y \in \mathcal{Y}\setminus\{y_i\}\}$ containing $(M-1)\cdot m$ vectors. Having the transformed training set $\mathcal{T}_{\mathcal{Z}}$, constraints (6.3) of the multiclass BSVM problem can be expressed as constraints (6.5). For instance, the constraints

$$\begin{array}{rcl} \langle \boldsymbol{w}_3, \boldsymbol{x}_i \rangle + b_3 - (\langle \boldsymbol{w}_1, \boldsymbol{x}_i \rangle + b_1) & \geq & 1 - \xi_i^1\,, \\ \langle \boldsymbol{w}_3, \boldsymbol{x}_i \rangle + b_3 - (\langle \boldsymbol{w}_2, \boldsymbol{x}_i \rangle + b_2) & \geq & 1 - \xi_i^2\,, \\ \langle \boldsymbol{w}_3, \boldsymbol{x}_i \rangle + b_3 - (\langle \boldsymbol{w}_4, \boldsymbol{x}_i \rangle + b_4) & \geq & 1 - \xi_i^4\,, \end{array}$$

defined for the input example $\boldsymbol{x}_i$ with $y_i = 3$ can be equivalently written as

$$\begin{array}{rcl} \langle \boldsymbol{w}, \boldsymbol{z}_i^1 \rangle & \geq & 1 - \xi_i^1\,, \\ \langle \boldsymbol{w}, \boldsymbol{z}_i^2 \rangle & \geq & 1 - \xi_i^2\,, \\ \langle \boldsymbol{w}, \boldsymbol{z}_i^4 \rangle & \geq & 1 - \xi_i^4\,. \end{array}$$

[1]The same transformation is named "construction of Fisher's classifiers" in [44].

It is obvious that by substituting $\boldsymbol{w}$ to the objective function of the singleclass SVM problem (6.4) the objective function (6.2) of the multiclass BSVM is recovered. Therefore it is seen that the multiclass BSVM formulation can be formally changed to the singleclass SVM formulation via the transformation given by (6.6) and (6.7).

At the first look, the introduced transformation seems to be intractable because of the increased dimension of the new space $\mathcal{Z}$. However, it will be shown that the dot products of the transformed data can be computed efficiently. Thus the singleclass SVM problem and the multiclass BSVM problem, respectively, can be solved efficiently in the dual formulation which uses just the dot products.

Let $\boldsymbol{z}_i^y$ and $\boldsymbol{z}_j^u$ be two vectors from $\mathcal{Z}$ created by the transformation (6.6). Notice that the vector $\boldsymbol{z}_i^y$ has the y_i-th coordinate slot $\boldsymbol{z}_i^y(y_i)$ equal to $[\boldsymbol{x}_i; 1]$, the y-th slot $\boldsymbol{z}_i^y(y)$ equal to $-[\boldsymbol{x}_i; 1]$, and the remaining coordinates are zero. The vector $\boldsymbol{z}_j^u$ is created likewise. Consequently, the dot product $\langle \boldsymbol{z}_i^y, \boldsymbol{z}_j^u \rangle$ is equal to the sum of dot products between $[\boldsymbol{x}_i; 1]$ and $[\boldsymbol{x}_j; 1]$ which occupy the same coordinate slot. The sign of these dot products is positive if $y_i = y_j$ or $y = u$ and negative if $y_i = u$ or $y_j = u$. If all numbers y_i, y_j, y, and u differ, then the dot product is equal to zero. This means that the corresponding kernel matrix is sparse. The construction of the dot product $\langle \boldsymbol{z}_i^y, \boldsymbol{z}_j^u \rangle$ can be easily expressed using the Kronecker delta function $\delta(i,j) = 1$ for $i = j$, and $\delta(i,j) = 0$ for $i \neq j$. The dot product between $\boldsymbol{z}_i^y$ and $\boldsymbol{z}_j^u$ is

$$\langle \boldsymbol{z}_i^y, \boldsymbol{z}_j^u \rangle = (\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + 1)(\delta(y_i, y_j) + \delta(y, u) - \delta(y_i, u) - \delta(y_j, y)) \,.$$

It is seen that only the dot products of the vectors from the feature space are needed to define the dot product of the transformed data. This allows to use the kernel functions to represent the input vectors in the feature space $\mathcal{H}$, i.e., $\mathcal{H}$ does not have to be finite-dimensional as was assumed for simplicity of notation at the beginning. To apply the proposed transformation in the feature space $\mathcal{H}$ it is enough to replace the dot products $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ by the kernel function $k(x_i, x_j)$. The kernel function $k_M(\boldsymbol{z}_i^y, \boldsymbol{z}_j^u)$ involving transformations (6.6) in the feature space $\mathcal{H}$ reads

$$k_M(\boldsymbol{z}_i^y, \boldsymbol{z}_j^u) = (k(x_i, x_j) + 1)(\delta(y_i, y_j) + \delta(y, u) - \delta(y_i, u) - \delta(y_j, y)) \,. \tag{6.8}$$

It implies that solving the dual form of the singleclass SVM problem with the kernel (6.8) is equivalent to solving the dual form of the multiclass BSVM problem (6.2). An arbitrary QP solver for the singleclass SVM applied on the training set $\mathcal{T}_{\mathcal{Z}}$ returns the multipliers α_i^y, $i = \mathcal{I}$, $y \in \mathcal{Y} \setminus \{y_i\}$ corresponding to the transformed vectors $\boldsymbol{z}_i^y$, $i \in \mathcal{I}$, $y \in \mathcal{Y} \setminus \{y_i\}$. The multipliers determine the vectors $\boldsymbol{w}_y \in \mathcal{H}$, $y \in \mathcal{Y}$ and scalars $b_y \in \mathbb{R}$, $y \in \mathcal{Y}$. The corresponding relation can be obtained by reverting the transform (6.7). The normal vector $\boldsymbol{w} \in \mathcal{Z}$ equals

$$\boldsymbol{w} = \sum_{i \in I} \sum_{y \in \mathcal{Y} \setminus \{y_i\}} \boldsymbol{z}_i^y \alpha_i^y \,.$$

The vector $\boldsymbol{w}_j \in \mathcal{H}$ occupies the j-th coordinate slot and it is determined by the weighted sum of vectors $\boldsymbol{z}_i^y$ which have non-zero j-th coordinate slot, so that

$$\begin{aligned}
\boldsymbol{w}_j &= \sum_{i \in \mathcal{I}} \sum_{y \in \mathcal{Y} \setminus \{y_i\}} \boldsymbol{x}_i \alpha_i^y (\delta(j, y_i) - \delta(j, y)) \,, \\
b_j &= \sum_{i \in \mathcal{I}} \sum_{y \in \mathcal{Y} \setminus \{y_i\}} \alpha_i^y (\delta(j, y_i) - \delta(j, y)) \,.
\end{aligned}$$

To classify the input x, the value of the discriminant function $f_j(x) = \langle \boldsymbol{w}_j \cdot \boldsymbol{\phi}(x) \rangle + b_j$ has to be computed by

$$
\begin{aligned}
f_j(x) &= \sum_{i \in \mathcal{I}} k(x, x_i) \sum_{y \in \mathcal{Y} \setminus \{y_i\}} \alpha_i^y (\delta(j, y_i) - \delta(j, y)) + b_j \\
&= \sum_{i \in \mathcal{I}} k(x, x_i) \beta_i^j + b_j \,,
\end{aligned}
$$

which requires to access the data only through the kernel functions.

6.2 Experiments

6.2.1 Benchmarking on UCI repository

This section aims to experimentally compare the multiclass BSVM formulation with other SVM based approaches to solve the multiclass problem. A detailed comparison of a broad class of approaches was published in the paper by Hsu and Lin [25]. The results and experimental protocol from this paper were adopted in this thesis.

The data sets used for testing involve problems selected from the UCI repository. Basic statistics of the selected data sets are listed in Table 6.1. The Gaussian kernel (c.f. Table 1.1) was used in all experiments. The space of free parameters, i.e., the regularization constant C and the width of the kernel σ, was discretized to a grid with 225 nodes having the coordinates $C \in \{2^{-2}, 2^{-1}, \ldots, 2^{12}\}$ and $\sigma \in \{2^{-\frac{5}{2}}, 2^{-\frac{4}{2}}, \ldots, 2^{\frac{9}{2}}\}$. The pair (C, σ) with the lowest classification error is reported in the results. The classification error was estimated by the 5-fold cross-validation. The experimental protocol in the paper [25] is referred for more details.

The methods tested by Hsu and Lin involve the decomposition-based methods represented by the directed acyclic graphs (DAGs) [40], one-against-one and one-against-all decomposition approaches built of the binary SVM with L_1-soft margin. The methods learning the multiclass SVM classifier at once were represented by the multiclass BSVM formulation and by the approach by Crammer and Singer [6], all with the L_1-soft margin loss function.

To complete the comparison done by Hsu and Lin, the multiclass BSVM and the one-against-all decomposition, both with L_2-soft margin loss function, were tested in this thesis. The comparison of the one-against-all decomposition and the multiclass BSVM with both the L_1-soft and L_2-soft margin in terms of the classification error and the number of support vector is presented in Table 6.2. Table 6.3 summarizes the CPU time required for learning and the free parameters which yield the smallest classification error. The reported CPU times are incomparable for the L_1-soft and L_2-soft formulations because the results for the L_1-soft margin were adopted from Hsu and Lin who used different implementations as well as computer platform. However, it can be seen that learning of the multiclass BSVM is in average more time demanding than the one-against-all decomposition approach which holds for both the L_1-soft and L_2-soft margin loss functions.

All the methods are comparable in terms of the classification error. The multiclass BSVM formulations produce the smaller number of the support vectors compared to the one-against-all decomposition approach which holds true for both the L_1-soft and L_2-soft margin formulations. Therefore the BSVM formulation is suitable when the evaluation

Problem	#training data	#class	#features
Iris	150	3	4
Wine	178	3	13
Glass	214	6	13
Vowel	528	11	10
Vehicle	846	4	18
Segment	2310	7	19

Table 6.1: Statistics of selected problems of UCI repository used to benchmark the multiclass SVM classifiers.

Problem	One-against-all SVM L_1		Multiclass BSVM L_1		One-against-all SVM L_2		Multiclass BSVM L_2	
	err [%]	#SV	err [%]	#SV	err [%]	#SV	err [%]	#SV
Iris	3.33	16	2.67	16	4.00	127	4.00	27
Wine	1.01	29	1.13	55	0.56	126	0.56	71
Glass	28.04	129	28.97	124	28.58	172	28.58	158
Vowel	1.515	393	1.515	279	0.51	770	0.61	419
Vehicle	12.53	343	13.00	264	12.63	490	13.12	403
Segment	2.47	446	2.42	358	2.38	525	2.55	295

Table 6.2: Benchmarking of the multiclass BSVM formulation and one-against-all decomposition approach both with the L_1-soft and L_2-soft margin loss functions on the UCI repository. The table contains the classification error in percents and the number of support vectors.

time is more critical. The L_1-soft margin formulations produce the rules with a smaller numbers of support vectors compared to the L_2-soft margin formulations similarly as in the binary case (c.f. Section 4.10.2).

6.2.2 Benchmarking on OCR system

This section describes a real application of the SVM-based learning of multiclass classifier. The problem is used to compare the proposed multiclass BSVM formulation to the one-against-all decomposition approach. The problem to solve is the design of the optical character recognition (OCR) system. This OCR system should be a part of a more complex system for a car license plate recognition. Besides the demands on the low classification error of the OCR, the speed of the resulting classifier is also a very important issue. The SVM-based learning was proven to produce classifiers with a small classification error compared to other state-oft-the-art approaches [33].

The input of the OCR system is a gray-scale image of size 13×13. For classification the image is represented as a column vector. Its entries are gray-scale values taken row-wise from the image matrix, i.e., the input space has dimension $n = 169$. The characters to be recognized involves numerals $0, 1, \ldots, 9$ and capital letters $A, B, \ldots, Z$ except for the letters G, Q, O and W. There are $M = 32$ characters altogether, i.e., different classes of input images. A huge set of labeled examples is provided. The set of examples is split into

Problem	One-against-all SVM L_1		Multiclass BSVM L_1		One-against-all SVM L_2		Multiclass BSVM L_2	
	time [s]	C,σ	time [s]	C,σ	time [s]	C,σ	time [s]	C,σ
Iris	0.10	$2^{9},2^{\frac{2}{2}}$	0.15	$2^{12},2^{\frac{7}{2}}$	0.07	$2^{-2},2^{-\frac{3}{2}}$	0.11	$2^{11},2^{\frac{2}{2}}$
Wine	0.20	$2^{7},2^{\frac{5}{2}}$	0.28	$2^{0},2^{\frac{1}{22}}$	0.06	$2^{-1},2^{\frac{2}{2}}$	0.05	$2^{0},2^{\frac{3}{2}}$
Glass	10.00	$2^{11},2^{\frac{1}{2}}$	7.94	$2^{9},2^{\frac{3}{2}}$	0.08	$2^{5},2^{\frac{1}{2}}$	5.17	$2^{5},2^{\frac{2}{2}}$
Vowel	9.28	$2^{4},2^{-\frac{2}{2}}$	14.05	$2^{3},2^{\frac{0}{2}}$	0.37	$2^{3},2^{-\frac{2}{2}}$	22.61	$2^{12},2^{-\frac{2}{2}}$
Vehicle	142.50	$2^{11},2^{\frac{3}{2}}$	88.61	$2^{10},2^{\frac{3}{2}}$	0.33	$2^{8},2^{\frac{3}{2}}$	79.25	$2^{6},2^{\frac{3}{2}}$
Segment	68.85	$2^{7},2^{\frac{0}{2}}$	66.43	$2^{5},2^{\frac{0}{2}}$	1.12	$2^{6},2^{-\frac{1}{2}}$	153.17	$2^{12},2^{-\frac{1}{2}}$

Table 6.3: Benchmarking of the multiclass BSVM formulation and one-against-all decomposition approach both with the L_1-soft and L_2-soft margin loss functions. The table contains the CPU time required for learning and the free parameters with the lowest classification error.

three parts. The first part forms the training data used for learning of the SVM classifier. The second part is used for selection of the free parameter of the SVM model. Finally, the third part (left out from the training stage) serves for validation of the resulting classifier. The basic statistics of the data are listed in Table 6.4.

The performance of the OCR system is assessed by its evaluation speed and the classification error. The car license plate in question consists of 7 characters. The distribution of characters at different positions of the car license plate is assumed to be independent. The designed OCR system aims to minimize the classification error ε which is the probability that a single character is misclassified regardless its position in the car license plate. The total error ε_{total} is defined as the probability that there is at least one misclassified character between all 7 characters. Assuming independence it holds that

$$\varepsilon_{total} = (1-\varepsilon)^7 \,. \tag{6.9}$$

The total error ε_{total} characterizes the classification performance of the OCR system.

The number of features (image 13×13)	169
The number of classes	32
The number of training data for SVM learning	49030
The number of testing data for model selection	156478
The number of independent validation data	156479

Table 6.4: Basic statistics of the OCR data set.

Three methods for learning the multiclass SVM classifiers were used: (i) the one-against-all decomposition built of the binary SVM classifiers with L_1-soft margin, (ii) the one-against-all decomposition built of the binary SVM classifiers with L_2-soft margin, and (iii) the multiclass BSVM formulation with L_2-soft margin.

The free parameters (not determined by the SVM learning) are the kernel function and the proper regularization constant. These parameters were selected from a finite set of parameters. The linear kernel and the polynomial kernel of 2nd order (c.f. Table 1.1) were used. In the case of the linear kernel, the classifier is composed of linear discriminant functions. In the case of the 2nd order polynomial, the discriminant functions are quadratic. The regularization constant C was selected from the set $\{10,\ 1,\ 0.1,\ 0,01,\ 0.001,\ 0.0001\}$. A particular pair of kernel function and regularization constant C determines one SVM model, i.e., the multiclass SVM classifier learned from the training data with given parameters. The best model was selected based on the minimal classification error computed on the testing data. The best models were assessed by the validation error computed on the validation data. The validation error is the classification error computed on the validation data.

The results obtained are summarized in Table 6.5. The upper part contains results for the linear kernel and the lower part for the 2nd order polynomial kernel, respectively. The table contains the training classification error, the testing classification error and the number of support vectors. The model with the smallest testing error is boldfaced. The last row contains the validation error computed for the best model (with the minimal testing error).

The multiclass BSVM classifiers have the lowest training errors in all the cases. This is obvious because the multiclass BSVM formulation has the minimization of the training error (more exactly, the upper bound on the training error) included explicitly in the objective function. On the other hand, the decomposition approaches minimize only the classification errors of the individual binary subproblems. This difference probably caused that in the case of linear kernel, the classifier learned by the multiclass BSVM had also a lower validation error compared to the one-against-all decomposition approach. In other words, the decomposition approach did not fit the linear model to the training data with sufficient precision.

In the case of the 2nd order polynomial kernel, the validation errors are approximately the same ($\pm 0.02\%$) regardless which multiclass SVM method was used. In average, the validation error for the polynomial kernel is three times lower compared to the linear kernel.

In general, the evaluation time of the SVM classifier is linearly proportional to the number of kernel functions to be evaluated. The evaluation of the linear kernel and the 2nd order polynomial kernel is approximately the same. In case of the 2nd polynomial kernel, the evaluation time is proportional to the number of support vectors. In the case of the linear kernel, the evaluation time is constant regardless of the number of support vectors. The evaluation time is proportional to computation of $M = 32$ linear kernels required to evaluate M discriminant functions.

The minimal number of 1581 support vectors has the polynomial classifier learned using the multiclass BSVM with L_2-soft margin. The classifier learned by the decomposition approach have the number of support vectors more than 3 times higher. The linear classifiers require evaluation of only $M = 32$ linear kernels regardless of the number of support vectors. It implies that the linear classifiers are approximately $50 \approx 1581/32$ times faster compared to the polynomial ones.

Table 6.6 presents overall results according to which the classifier best for a given application was selected. The table contains an estimate of the total error ε_{total} computed using (6.9) and with the validation error substituted for the true classification error ε. The evaluation time for a single character computed relatively to the time of the BSVM L_2 classifier with the 2nd order polynomial kernel is included. For example, 100% of the evaluation time corresponds approximately to 2 milliseconds on common PC with AMD

2800 MHz processor. The linear classifier learned by the multiclass BSVM with L_2-soft margin was found the best for its low evaluation time and the lowest classification error among the linear classifiers.

6.3 Summary to multiclass SVM

This chapter analyzes a modification of the multiclass SVM learning task which can be transformed to the singleclass SVM problem being easier for optimization.

The contributions of this part of the thesis involve:

- The modified task of learning the multiclass SVM classifier called a multiclass BSVM formulation was proposed by the author of the thesis in [12, 13]. The modification to the original multiclass SVM task lies in adding the sum of squared biases of the classification rules to the regularization term. The multiclass BSVM problem was independently proposed by Hsu and Lin [25]. Hsu and Lin, however, did not exploit the possibility of transforming the multiclass problem to the singleclass SVM problem.
- The transformation of the multiclass BSVM problem to the problem of learning the singleclass SVM classifier is proposed. The whole transformation is performed solely by using a special kernel function. As a result, any solver for the singleclass SVM problem can be readily used to solve the multiclass problem. The transformation is based on the well known Kesler's construction [8, 44] for linear classifiers.
- The multiclass BSVM formulation with L_2-soft margin was experimentally compared to other SVM-based methods. The experimental protocol adopted from Hsu and Lin was used, thus the results computed are comparable to a large variety of methods. The classification accuracy of the multiclass BSVM formulation is comparable to the other methods. The time required for learning is considerably longer compared to the others. The multiclass BSVM formulation produces classification rules with less number of support vectors, i.e., the evaluation time is shorter. The multiclass BSVM formulation allows to find classifiers with a smaller training error for a given class of functions compared to the decomposition approaches.
- The multiclass linear classifier learned by the proposed method was found the best for the OCR module being a part of a commercial car license plate recognition system. The method was also applied on learning of the OCR for a module of a robotic sewerage inspection system which was published in [23] (it is not described in the thesis).

Linear kernel (linear discriminant function)									
C	one-against-all SVM L_1			one-against-all SVM L_2			multiclass BSVM L_2		
	error		#SV	error		#SV	error		#SV
	trn	tst		trn	tst		trn	tst	
10	0.38	0.82	4986	0.28	0.80	9078	0	0.60	956
1	0.49	0.77	6351	0.37	0.75	11318	0	0.47	1578
0.1	0.60	0.77	10569	0.49	0.75	17425	0	0.43	3777
0.01	1.04	0.89	21950	0.71	0.81	32735	0	0.51	10603
0.001	2.51	5.19	41797				0	0.95	29442
Valid. err [%]	0.66			0.59			0.36		

2nd order polynomial kernel (quadratic discriminant function)									
C	one-against-all SVM L_1			one-against-all SVM L_2			multiclass BSVM L_2		
	error		#SV	error		#SV	error		#SV
	trn	tst		trn	tst		trn	tst	
1	1.11	0.66	2256	0	0.34	2328	0	0.53	937
0.1	0.57	0.41	2440	0.00	0.28	2912	0	0.49	1038
0.01	0.14	0.26	3620	0.05	0.27	5273	0	0.38	1581
0.001	0.30	0.42	7853	0.19	0.37	12145	0	0.39	3572
0.0001	0.72	0.75	19037	0.50	0.60	28613	0.13	0.56	9677
Valid. err [%]	0.22			0.20			0.24		

Table 6.5: Comparison of different multiclass SVM classifiers on the OCR data set. The training error, testing error and the number of support vectors are listed for corresponding values of the regularization constant C. The last row contains the validation error for the best (boldfaced) model.

Kernel function	one-against-all SVM L_1		one-against-all SVM L_2		multiclass BSVM L_2	
	$\hat{\varepsilon}_{total}$ [%]	Eval time [%]	$\hat{\varepsilon}_{total}$ [%]	Eval time [%]	$\hat{\varepsilon}_{total}$ [%]	Eval time [%]
Linear	4.53	2.0	4.06	2.0	2.49	2.0
2nd polynomial	1.57	229.0	1.39	333.5	1.67	100.0

Table 6.6: Comparison of different multiclass SVM classifiers of the OCR data set. The estimate of the total error (probability of misclassification of at least one out of 7 characters) and the evaluation time is displayed. The evaluation time is measured relatively to the time of the BSVM L_2 classifier with the 2nd order polynomial kernel.

7 Thesis contributions

This chapter summarizes contributions of the thesis. The contributions are accompanied with references to corresponding published works.

7.1 Quadratic Programming Solvers

The thesis contributes to the problem of *Quadratic Programming* (QP) optimization (c.f. Chapter 4). The QP solvers analyzed are based on known methods which solve the *Minimal Norm Problem* (MNP) and the *Nearest Point Problem* (NPP). These methods were successfully applied to solve large scale problems arising in the learning of the binary SVM classifier with the L_2-soft margin. In this thesis, the solvers were extended such that they can solve more general QP tasks named the *Generalized Minimal Norm Problem* (GMNP) and the *Generalized Nearest Point Problem* (GNPP). As a result the generalized QP solvers can be applied for a broader class of problems, e.g., multiclass SVM classifier learning, Reduced Set Density Estimation, computing Minimal Enclosing Ball and Support Vector Data Description. All the generalized QP solvers were described in a common framework which allows for their comparison and better understanding. The convergence to the ε-optimal solution in a finite number of iterations was proven for all proposed QP solvers. A novel algorithm, named *Improved Mitchell-Demyanov-Malozemov algorithm* (IMDM), was derived based on combination of ideas of two known algorithms. The QP solvers were experimentally evaluated on large synthetic and real problems and they were compared to each other. The solvers were proved to solve large problems (even with millions of variables) in reasonable times. The proposed IMDM algorithm outperformed the other algorithms in all experiments.

The work described in Chapter 4 was published partially in [11, 15, 18].

7.2 Greedy Kernel Principal Component Analysis

Chapter 5 describes a novel method called *Greedy Kernel Principal Component Analysis* (Greedy KPCA) algorithm. The Greedy KPCA aims to find a lower-dimensional representation of the data embedded into the *Reproducing Kernel Hilbert Space* (RKHS) similarly to the ordinary *Kernel Principal Component Analysis* (KPCA) [47, 46]. In contrast to the KPCA, basis vectors are selected directly from the training set and not expressed as a linear combination of all training vectors. As a result, the found representation of data is suboptimal in terms of the reconstruction error but the representation is less complex compared to the ordinary KPCA. The idea of the proposed Greedy KPCA is closely related to the *sparse greedy matrix approximation* [46, 48]. The main difference is that the the sparse greedy matrix approximation uses a randomized selection of the basis vectors instead of the deterministic Greedy KPCA. The Greedy KPCA can be used for reduction of computational and memory requirements of an arbitrary kernel method. It can be also applied for reduction of complexity of functions learned by the kernel methods. It was experimentally verified that the method can reduce computational complexity of the Kernel Least Squares Regression and it can speed up evaluation of the SVM classifier.

A preliminary version of the Greedy KPCA was published in [14, 16]. The extended Greedy KPCA algorithm has not been published yet.

7.3 Multiclass Support Vector Machines

The thesis contributes to the problem of efficient learning of the multiclass SVM classifier (c.f. Chapter 6). The contribution concerns the *Bounded Support Vector Machines* (BSVM) formulation [25]. The BSVM formulation arises after adding a sum of squared biases of the classification rules to the regularization term of the original multiclass SVM task. It is shown in the thesis how to transform the *multiclass BSVM problem* to the problem of learning a *singleclass SVM classifier*. The whole transformation is performed solely by using a special kernel function. As a result, any solver for the singleclass SVM problem can be readily used to solve the multiclass problem. The multiclass BSVM formulation was experimentally compared to other SVM-based methods. The BSVM formulation is comparable with the other SVM-based approaches in terms of the classification accuracy but it requires considerably longer learning times. The multiclass BSVM formulation produces classification rules with less support vectors, i.e., the evaluation time is shorter. Thus the BSVM formulation is eligible for applications in which the evaluation time is especially important. The proposed method was found the best for the Optical Character Recognition (OCR) module being a part of a commercial car license plate recognition system. The method was also applied on learning of the OCR for a module of a robotic sewerage inspection system.

The proposed method for optimizing the multiclass BSVM was published in [12, 13]. An application of the method on learning of the Optical Character Recognition (OCR) for a module of a robotic sewerage inspection system which was published in [23]. The benchmarking of the multiclass BSVM and comparison to other multiclass SVM-based approaches has not been published yet. The application of the multiclass BSVM to design of an OCR system for a car license plate recognition has not been published yet.

7.4 Statistical Pattern Recognition Toolbox

All the novel methods proposed in this thesis were incorporated to the Statistical Pattern Recognition Toolbox (STPR toolbox) [17] written in Matlab. The author of this thesis designed and implemented a substantial part of the STPR toolbox. The STPR toolbox can be freely downloaded from `http://cmp.felk.cvut.cz/~xfrancv/stprtool/`. The toolbox contains an ensemble of pattern recognition techniques, e.g., linear discriminant function analysis, feature extraction, density estimation and clustering, Support Vector Machines, kernel methods, etc. There were more than 11,000 downloads between years 2000 and 2005.

8 Future research

The future research will address a further extensions of the Quadratic Programming (QP) solvers introduced in Chapter 4. The general framework of the proposed sequential algorithm can be also applied for QP tasks which have feasible sets more difficult than the GMNP and the GNPP analyzed in the thesis. Let as assume a linear programming (LP) task with a feasible set equal to a given QP task. If the mentioned LP task has an analytical solution then a solver for the corresponding QP task can be simply derived based on the general framework. Some interesting examples of QP tasks which have this property are given below.

Let the feasible set of the QP task be the following

$$\mathcal{A} = \{\boldsymbol{\alpha} \in \mathbb{R}^m \colon \langle \boldsymbol{\alpha}, \boldsymbol{e}_1 \rangle = 1, \langle \boldsymbol{\alpha}, \boldsymbol{e}_2 \rangle = 1, \ldots, \langle \boldsymbol{\alpha}, \boldsymbol{e}_k \rangle = 1, \boldsymbol{\alpha} \geq 0\} , \qquad (8.1)$$

where the vectors $\boldsymbol{e}_i \in \mathbb{R}^m$, $i = 1, \ldots, k$, are defined like in Section 4.1. The GMNP and the GNPP is obtained if $k = 1$ and $k = 2$, respectively. It is easy to extend the proposed algorithms to solve the QP tasks with the feasible set (8.1) and k greater than 2.

The GMNP and the GNPP do not cover the optimization problems connected to the learning of SVM classifiers with L_1-soft margin. Another extension of the proposed QP solvers will follow this direction. The first step has already been made by Tao et al. [50] who, based on the work [15] published by the author of this thesis, extended the Kozinec algorithm for learning of ν-SVM formulation [49]. However, the extension of the Kozinec algorithm for learning of the SVM with L_1-soft margin is also possible, which has not been published yet. A nice property of this extension is that a linear classifier can be learned from a non-separable data without explicit use of the dual variables. This can be potentially useful in applications where the number of examples is so huge that cannot be explicitly stored in the memory (e.g., problems of learning of the Markov random models).

Other algorithm which can be extended for learning of SVM classifiers with L_1-soft margin is the proposed Improved Mitchell-Demyanov-Malozemov (IMDM) algorithm. In fact, it is very related to the popular Sequential Minimal Optimizer (SMO) by Platt [41] and its improved version proposed by Keerthi et al. [30]. The improved SMO algorithm uses a very similar rule for construction of the line segment which approximates the feasible set as the original Mitchell-Demyanov-Malozemov (MDM) algorithm. It seems likely that the improved search for the line segment proposed in the IMDM algorithm will also work well for this QP task.

Bibliography

[1] B.E. Boser, I.M. Guyon, and V.N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of 5th Annual Workshop on Computer Learning Theory*, pages 144–152. ACM Press, New York, NY, 1992.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004.

[3] C.J. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

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

[5] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273, 1995.

[6] K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2):201–233, 2002.

[7] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[8] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley & Sons, 2nd. edition, 2001.

[9] T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

[10] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, New York, USA, 2nd edition, 1990.

[11] V. Franc and V. Hlaváč. A Simple Learning Algorithm for Maximal Margin Classifier. In *Kernel and Subspace Methods for Computer Vision, workshop adjoint to the International Conference on Neural Networks*, pages 1–11. TU Wien, August 2001.

[12] V. Franc and V. Hlaváč. Kernel representation of the Kesler construction for multi-class SVM classification. In H. Wildenauer and W. Kropatsch, editors, *Proceedings of the CVWW'02*, page 7, Wien, Austria, February 2002. PRIP.

[13] V. Franc and V. Hlaváč. Multi-class Support Vector Machine. In R. Kasturi, D. Laurendeau, and Suen C., editors, *16th International Conference on Pattern Recognition*, volume 2, pages 236–239, Los Alamitos, CA 90720-1314, August 2002. IEEE Computer Society.

[14] V. Franc and V. Hlaváč. Greedy algorithm for a training set reduction in the kernel methods. In Nikolai Petkov and Michel A. Westenberg, editors, *Computer Analysis of Images and Patterns*, pages 426–433, Berlin, Germany, August 2003. Springer.

[15] V. Franc and V. Hlaváč. An iterative algorithm learning the maximal margin classifier. *Pattern Recognition*, 36(9):1985–1996, September 2003.

[16] V. Franc and V. Hlaváč. Training set approximation for kernel methods. In Ondřej Drbohlav, editor, *Computer Vision — CVWW'03 : Proceedings of the 8th Computer Vision Winter Workshop*, pages 121–126, Prague, Czech Republic, February 2003. Czech Pattern Recognition Society.

[17] V. Franc and V. Hlaváč. Statistical pattern recognition toolbox for Matlab. Research Report CTU–CMP–2004–08, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, June 2004.

[18] V. Franc and V. Hlaváč. Simple solvers for large quadratic programming tasks. In Walter Kropatsch and Robert Sablatnig, editors, *Pattern Recognition, Proceedings of the 27th DAGM Symposium, Lecture Notes in Computer Sciences*. Springer Verlag, 2005. IN PRINT.

[19] T.T. Friess, N. Cristianini, and C. Campbell. The kernel adatron algorithm: A fast and simple learning procedure for support vector machines. In *Proceedings of 15th International Conference on Machine Learning*. Morgan Kaufman Publishers, 1998.

[20] E.G. Gilbert. Minimizing the quadratic form on a convex set. *SIAM journal on Control and Optimization*, 4:61–79, 1966.

[21] M. Girolami and C. He. Probability density estimation from optimally condensed data sample. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10), 2003.

[22] L. Gonzáles, C. Angulo, F. Velasco, and A. Catalá. Unified dual for bi-class SVM approaches. *Pattern Recognition*, 2005. IN PRESS.

[23] K. Hanton, V. Smutný, V. Franc, and V. Hlaváč. Alignment of sewerage inspection videos for their easier indexing. In J.L. Crawley, J.H. Piater, M. Vincze, and L. Paletta, editors, *ICVS2003 : Proceedings of the Third International Conference on Vision Systems*, volume 2626 of *Lecture Notes in Computer Science*, pages 141–150, Berlin, Germany, April 2003. Springer-Verlag.

[24] T.J. Hastie and R.J. Tibshirani. Classification by pairwise coupling. In M.I. Jordan, M.J. Kearns, and S.A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[25] C.W. Hsu and C.J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, March 2002.

[26] T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[27] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of ECML-98, 10th European Conference on Machine Learning*, pages 137–142. Springer Verlag, Heidelberg, DE, 1998.

[28] S. S. Keerthi and E. G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, 46(1–3):351–360, 2002.

[29] S.S. Keerthi, S.K. Shevade, C. Bhattacharya, and K.R.K. Murthy. A Fast Iterative Nearest Point Algorithm for Support Vector Machine Classifier Design. *IEEE Transactions on Neural Networks*, 11(1):124–136, January 2000.

[30] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13(3):637 – 649, 2001.

[31] A. Kowalczyk. Maximal margin perceptron. In P.J. Bartlett, B. Schölkopf, D. Schuurmans, and A.J. Smola, editors, *Advances in Large-Margin Classifiers*. The MIT Press, 2000.

[32] B.N. Kozinec. Rekurentnyj algoritm razdelenia vypuklych obolochek dvuch mnozhestv, in Russian (Recurrent algorithm separating convex hulls of two sets). In V.N. Vapnik, editor, *Algoritmy obuchenia raspoznavania (Learning algorithms in pattern recognition)*, pages 43–50. Sovetskoje radio, Moskva, 1973.

[33] Y. LeCun, L. Botou, L. Jackel, H. Drucker, C. Cortes, J. Denker, I. Guyon, U. Müller, E. Sackinger, P. Simard, and V. Vapnik. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural Networks*, pages 261–276, 1995.

[34] C.J. Lin. A formal analysis of stopping criteria of decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 13(5):1045– 1052, 2002.

[35] O.L. Mangasarian and D.R. Musicant. Successive overrelaxation for support vector machines. *IEEE Transactions on Neural Networks*, 10(5), 1999.

[36] B.F. Mitchell, V.F. Demyanov, and V.N. Malozemov. Finding the point of a polyhedron closest to the origin. *SIAM journal on Control and Optimization*, 12:19–26, 1974.

[37] K.R. Müller, S. Mika, G. Ratsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, March 2001.

[38] E. Osuna, R. Freund, and F. Girosi. An improved training algorithms for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII – Proceedings of the 1997 IEEE Workshop*, pages 276–285. IEEE, 1997.

[39] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *Proceedings of CVPR'97, Puerto Rico*, Washington, DC, USA, 1997. IEEE Computer Society.

[40] J. Platt. Large margin DAGs for multiclass classification. In S.A. Solla, T.K. Leen, and K.R. Muüller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 547–553. The MIT Press, 2000.

[41] J.C. Platt. Fast training of support vectors machines using sequential minimal optimization. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods*, pages 185–208. The MIT Press, 1998.

[42] T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer network. *Science*, pages 978–982, 1990.

[43] B.D. Ripley. Neural networks and related methods for classification (with discussion). *Journal of Royal Statistical Society Series B*, 56:409–456, 1994.

[44] M.I. Schlesinger and V. Hlaváč. *Ten lectures on statistical and structural pattern recognition*. Kluwer Academic Publishers, 2002.

[45] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R.C Williamson. Estimating the support of a high-dimensional distribution. Technical Report TR 87, Microsoft Reasearch, Redmond, WA, 1999.

[46] B. Schölkopf and A. Smola. *Learning with Kernels*. The MIT Press, MA, 2002.

[47] B. Schölkopf, A. Smola, and K.R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. Technical report, Max-Planck-Institute fur biologische Kybernetik, 1996.

[48] A. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proceedings 17th International Conference on Machine Learning*, pages 911–918. Morgan Kaufmann, San Francisco, CA, 2000.

[49] A. Smola, B. Schölkopf, R. Williamnson, and P. Bartlett. New Support Vector Aalgorithms. *Neural Computation*, 12(5):1207–1245, May 2000.

[50] Q. Tao, G.W. Wo, and J. Wang. A generalized s-k algorithm for learning ν-svm classifier. *Pattern Recognition Letters*, 25(10):1165–1171, July 2004. [P01].

[51] D.M.J. Tax and R.P.W Duin. Data domain description by support vectors. In M. Verleysen, editor, *Proceedings ESANN*, pages 251–256, Brussels, 1999.

[52] V.Y. Tikhonov and A.N. Arsenin. *Solutions of ill-posed problems*. Winston-Wiley, New York, 1977.

[53] V. Vapnik. *The nature of statistical learning theory*. Springer Verlag, 1995.

[54] V. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, Inc., 1998.

[55] V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition (in Russian)*. Nauka, Moscow, 1974.

[56] G. Wahba. Support Vector Machines, Reproducing Kernel Hilbert Spaces, and randomized GACV. In B. Schölkopf, C.J.C. Burges, and A. Smola, editors, *Advances in Kernel Methods*. The MIT Press, 1998.

[57] J. Weston and C. Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, Egham, TW20 0EX, UK, 1998.