CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

From Supervised to Unsupervised Support Vector Machines and Applications in Astronomy

Dissertation zur Erlangung des Grades eines
Doktors der Naturwissenschaften

vorgelegt von

Dipl.-Math. Dipl.-Inform.
Fabian Gieseke

29. November 2011

Dekan
Prof. Dr. Thorsten Raabe

Tag der Disputation
29.02.2012

Prüfungskommission
Prof. Dr. Martin Fränzle (Vorsitzender)
Jun.-Prof. Dr. Oliver Kramer (Erstgutachter)
Prof. Dr. Christian Igel (Zweitgutachter)
Dr. Ute Vogel (Mitglied der wiss. Mitarbeiter)

To my parents

Abstract

A common task in the field of machine learning is the classification of objects. The basis for such a task is usually a training set consisting of patterns and associated class labels. A typical example is, for instance, the automatic classification of stars and galaxies in the field of astronomy. Here, the training set could consist of images and associated labels, which indicate whether a particular image shows a star or a galaxy. For such a learning scenario, one aims at generating models that can automatically classify new, unseen images. In the field of machine learning, various classification schemes have been proposed. One of the most popular ones is the concept of support vector machines, which often yields excellent classification results given sufficient labeled data.

However, for a variety of real-world tasks, the acquisition of sufficient labeled data can be quite time-consuming. In contrast to labeled training data, unlabeled one can often be obtained easily in huge quantities. Semi- and unsupervised techniques aim at taking these unlabeled patterns into account to generate appropriate models. In the literature, various ways of extending support vector machines to these scenarios have been proposed. One of these ways leads to combinatorial optimization tasks that are difficult to address.

In this thesis, several optimization strategies will be developed for these tasks that (1) aim at solving them exactly or (2) aim at obtaining (possibly suboptimal) candidate solutions in an efficient kind of way. More specifically, we will derive a polynomial-time approach that can compute exact solutions for special cases of both tasks. This approach is among the first ones that provide upper runtime bounds for the tasks at hand and, thus, yield theoretical insights into their computational complexity. In addition to this exact scheme, two heuristics tackling both problems will be provided. The first one is based on least-squares variants of the original tasks whereas the second one relies on differentiable surrogates for the corresponding objective functions. While direct implementations of both heuristics are still computationally expensive, we will show how to make use of matrix operations to speed up their execution. This will result in two optimization schemes that exhibit an excellent classification and runtime performance.

Despite these theoretical derivations, we will also depict possible application domains of machine learning methods in astronomy. Here, the massive amount of data given for today's and future projects renders a manual analysis impossible and necessitates the use of sophisticated techniques. In this context, we will derive an efficient way to preprocess spectroscopic data, which is based on an adaptation of support vector machines, and the benefits of semi-supervised learning schemes for appropriate learning tasks will be sketched. As a further contribution to this field, we will propose the use of so-called resilient algorithms for the automatic data analysis taking place aboard today's spacecrafts and will demonstrate their benefits in the context of clustering hyperspectral image data.

Zusammenfassung

Ein klassisches Problem des maschinellen Lernens ist die Klassifikation von Objekten. Als Ausgangspunkt stehen hier Trainingsdaten in Form von Mustern und zugehörigen Labeln zur Verfügung. Im Bereich der Astronomie könnten die Trainingsdaten z.B. aus Bilddaten und zugehörigen Labeln bestehen, wobei jedes Label angibt, ob auf dem Bild ein Stern oder eine Galaxie zu sehen ist. Das Ziel des Lernprozesses würde dann darin bestehen, auf Basis der bekannten Trainingsdaten ein entsprechendes Modell zu erstellen, welches bisher unbekannte Bilddaten klassifizieren kann. Ein bekanntes Klassifikationskonzept im Bereich des maschinellen Lernens sind die sogenannten Support Vektor Maschinen. Falls genügend Trainingsdaten vorhanden sind, führt dieses Konzept in vielen Fällen zu Modellen mit einer exzellenten Klassifikationsgüte.

Die Erstellung eines hinreichend großen Datensatzes kann sich für gewisse Anwendungsfälle jedoch als aufwendig erweisen. Im Gegensatz zu solchen gelabelten Trainingsdaten stehen ungelabelte Daten oft in großem Umfang zur Verfügung. Um auch letztere für den Lernprozess verwenden zu können, wurden in der Literatur unter anderem die halb- und unüberwachten Support Vektor Maschinen vorgestellt; beide Erweiterungen führen jedoch zu schwierigen kombinatorischen Optimierungsproblemen.

Die Entwicklung von Optimierungsansätzen für beide Erweiterungen ist eines der zentralen Themen dieser Arbeit. Dabei wird sowohl auf die aufwendige Bestimmung exakter Lösungen als auch auf den Entwurf von effizienten Heuristiken eingegangen. Im Speziellen wird ein Ansatz vorgestellt, der es ermöglicht, exakte Lösungen in polynomieller Laufzeit für einen Spezialfall der Problemstellungen zu bestimmen. Dieser liefert somit wertvolle theoretische Einsichten in die Komplexität beider Optimierungsprobleme. Neben diesem exakten Verfahren werden zwei lokale Suchstrategien vorgestellt. Die erste basiert auf einer *least-squares*-Variante der Problemstellungen wohingegen die zweite auf differenzierbaren Ersatzfunktionen beruht. Der in diesem Zusammenhang geleistete Kernbeitrag besteht in hocheffizienten Implementationen beider Ansätze, welche sich durch geschicktes Ausnutzen von Matrixeigenschaften der Zwischenlösungen ergeben.

Über diese theoretischen Ausführungen hinaus ist die Anwendung von Techniken des maschinellen Lernens auf Daten aus der Astronomie Gegenstand der Arbeit. Der große Umfang aktueller Datensätze in diesem Bereich führt dazu, dass eine manuelle Datenanalyse „per Hand“ nicht mehr möglich ist. In diesem Kontext wird ein Verfahren zur effizienten Vorverarbeitung von Spektraldaten vorgestellt, welches auf einer angepassten Version der oben genannten Support Vektor Maschinen basiert. Weiterhin werden sowohl mögliche Anwendungsgebiete von halbüberwachten Lernverfahren im Bereich der Astronomie als auch der Nutzen von sogenannten robusten Algorithmen zur automatisierten Datenanalyse an Bord heutiger Raumfahrzeuge diskutiert und analysiert.

Acknowledgments

I want to express my gratitude to my supervisor, Jun.-Prof. Oliver Kramer, for giving me the opportunity to write this thesis in a prolific and enjoyable scientific environment. I also want to thank him for numerous helpful discussions related to machine learning and optimization issues that contributed significantly to the completion of this work.

I want to express my gratitude to Prof. Christian Igel who kindly agreed to serve as a second referee. I also want to thank Prof. Martin Fränzle and Ute Vogel for agreeing to complete my defense committee.

I would like to thank Prof. Jan Vahrenhold for giving me the opportunity to discover both the machine learning field as well as its application domains in astronomy. I am also thankful for several interesting discussions with Prof. Xiaoyi Jiang, who drew my attention to support vector machines and their semi- and unsupervised extensions. Thanks to Prof. Tapio Pahikkala, Evgeni Tsivtsivadze, and Antti Airola for having had the opportunity to work with you and for various inspiring discussions. It was great to visit you in Turku and Nijmegen and to host you in Dortmund and Münster. Special thanks go to Kai Lars Polsterer for various prolific discussions, not only related to astronomical issues, and for the interesting research we conducted during the last three years.

Thanks to Gundel Jankord, Marlies Terber, Meike Burke, and Petra Oetken, who made travel arrangements and other administrative stuff much easier for me. Further, many thanks to several people who provided valuable linguistic comments for the work at hand including Wolfgang Hempel, Ulrich Trüloff, Christian Jansen, and Svenja Eßling.

I am also glad about having had the opportunity to conduct research at the TU Dortmund, the Turku Centre of Computer Science, the Institute for Structural Mechanics in Weimar, and the International Computer Science Institute in Berkeley. Further, I would like to thank Jane White for her tremendous hospitality that made my short stay in Berkeley so wonderful.

My particular gratitude goes to my lovely girlfriend Anna Amelung for her continuous support and for keeping me going despite the pressure of work. I also want to thank her for the wonderful cartoons she made for me.

I want to thank my family for the support I received year after year. In particular, I want to thank Carolin Gieseke and Jan Verschraegen for knowing when to offer help, and Noan and Malo Verschraegen for being such groovy nephews. Most of all, I want to thank my parents, Beate Gieseke-Ladner and Arnold Gieseke, who supported me throughout my life, especially during the last years. Thank you very much.

Fabian Gieseke
Oldenburg, November 2011

Mathematical Notation

We will use $\mathbb{N}$ to denote the set of natural numbers (not including zero) and $[n]$ to denote the set $\{1, \ldots, n\}$ for any $n \in \mathbb{N}$. The set of real numbers will be denoted by $\mathbb{R}$, the set of strictly positive real numbers by $\mathbb{R}^+$, and the set of non-negative real numbers by $\mathbb{R}_0^+$. Further, the n-dimensional vector space over $\mathbb{R}$ will be denoted by $\mathbb{R}^n$ and the vector space of all $n \times m$ matrices with real coefficients by $\mathbb{R}^{n \times m}$.

Throughout the thesis, vectors and matrices are written in boldface where numbers/scalars are written in plain text. All vectors are assumed to be column vectors and the superscript T is used to denote the transpose of a matrix or a vector, i. e., $\mathbf{x}^{\mathrm{T}}$ is a row vector and $\mathbf{M}^{\mathrm{T}} \in \mathbb{R}^{m \times n}$ is the transpose of the matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$. Further, we use x_i to denote the i-th coordinate of the vector $\mathbf{x} \in \mathbb{R}^n$ and $[\mathbf{M}]_{i,j}$ to denote the element in the i-th row and j-th column of $\mathbf{M} \in \mathbb{R}^{n \times m}$. For two sets $R = \{i_1, \ldots, i_r\} \subseteq [n]$ and $S = \{k_1, \ldots, k_s\} \subseteq [m]$ of indices, we use $\mathbf{M}_{R,S}$ to denote the submatrix that contains only the rows and columns of $\mathbf{M} \in \mathbb{R}^{n \times m}$ that are indexed by R and S, respectively. Moreover, we set $\mathbf{M}_{R,[m]} = \mathbf{M}_R$. The identity matrix is denoted by $\mathbf{I} \in \mathbb{R}^{n \times n}$.

The d-dimensional Euclidean space for fixed dimension $d \in \mathbb{N}$ is denoted by $\mathbb{R}^d$. For two vectors $\mathbf{x}$ and $\mathbf{z}$ in $\mathbb{R}^d$, we will use $\langle \mathbf{x}, \mathbf{z} \rangle := \mathbf{x}^{\mathrm{T}}\mathbf{z} = \sum_{i=1}^{d} x_i z_i$ to denote the standard inner product and $\|\mathbf{x}\| := (\mathbf{x}^{\mathrm{T}}\mathbf{x})^{1/2}$ to denote the Euclidean norm of $\mathbf{x}$. The Euclidean distance between them is given by $d(\mathbf{x}, \mathbf{z}) := \|\mathbf{x} - \mathbf{z}\|$ and we will use

$$d(P, Q) := \inf\{d(\mathbf{x}, \mathbf{z}) \mid \mathbf{x} \in P \wedge \mathbf{z} \in Q\} \tag{1}$$

to denote the natural extension of this metric to non-empty sets $P, Q \subseteq \mathbb{R}^d$. Note that some letters and symbols will be (re-)used multiple times throughout this thesis; the context will make clear which meaning is intended.

Contents

CHAPTER 1

Introduction

The field of *machine learning* [5, 15, 73, 105] has gained more and more attention in recent years. One of the reasons for this phenomenon is the fact that the data volume in various (scientific) fields has increased dramatically during the last decade. This is the case, for instance, in astronomy where recent projects like the *Sloan Digital Sky Survey* [132] or future ones like the *Large Synoptic Sky Telescope* [95] produce or will produce data volumes in the tera- and petabyte range. For such projects, the sheer data volume renders a manual analysis impossible. Machine learning techniques aim at retrieving useful information in an automatic manner and the corresponding tools have been recognized as "increasingly essential in the era of data-intensive astronomy" [17]. In general, expert knowledge is required to teach the machines how to automatically perform a task (like predicting a class for an object). Such a teaching process is usually based on expert knowledge in terms of labels, i. e., a machine has access to a finite set of data items with associated labels. This type of task is called *supervised learning* [5, 15, 73, 105].

Ideally, a large amount of labeled data items should be available for the machines to yield a satisfying classification performance on unseen data. Depending on the task at hand, however, acquiring such labeled data can be extremely time-consuming and expensive. In contrast to labeled data, unlabeled data (i. e., data items without associated labels) can often be obtained in great quantities without much additional effort. Both so-called *semi-* and *unsupervised learning* schemes aim at making use of the (additional) information provided by the unlabeled patterns to generate appropriate models. In this chapter, we will sketch the main ideas of supervised, semi-supervised, and unsupervised learning settings and will provide an overview of the work at hand.

1.1 Motivation

The content presented in this work mostly deals with a well-known supervised learning technique, the so-called *support vector machines* [40, 73, 124, 135], and their extensions to semi- and unsupervised learning scenarios. In this section, we will briefly sketch the basic ideas behind these concepts.

1.1.1 Support Vector Machines

The concept of support vector machines can be used to address *binary classification* tasks [73]. For such a learning setting, there are data from two types of objects along with associated class labels. In most cases, each data item, also called *pattern*, is represented by a set of real-valued *features* describing the objects. These features and the class labels constitute the *training set* $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathbb{R}^d \times \{-1, +1\}$.

Briefly speaking, the goal of a support vector machine consists in finding a hyperplane which separates both classes well such that the induced distance (or *margin*) between the hyperplane and the patterns is maximal. At the same time, patterns lying within the corridor (induced by the margin and the patterns) are penalized. As we will see below, this idea can be formalized mathematically in terms of the following optimization problem:

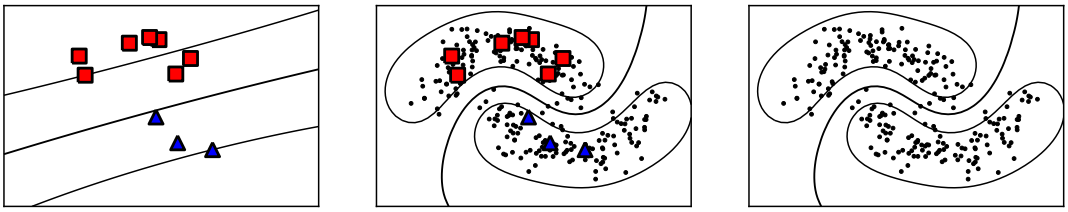
$$\begin{aligned} \underset{\mathbf{w}\in\mathbb{R}^d,\, b\in\mathbb{R},\, \boldsymbol{\xi}'\in\mathbb{R}^l}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C' \sum_{i=1}^{l} \xi'_i \\ \text{s.t. } & y'_i(\langle \mathbf{w}, \mathbf{x}'_i \rangle + b) \geq 1 - \xi'_i,\ \xi'_i \geq 0 \end{aligned} \tag{1.1}$$

Here, the first term of the objective corresponds to maximizing the margin whereas the second term corresponds to penalizing the patterns lying within the corridor [40, 73, 124, 135]. The parameter $C' > 0$ determines the trade-off between these two aims.

This concept can be extended by using so-called *kernel functions* $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, which render non-linear decision hyperplanes possible [124, 135]. In Figure 1.1 (a), an illustration of this concept is shown. Here, the red squares and the blue triangles depict the two classes and the middle line represents the (non-linear) decision hyperplane. The margin is indicated by the regions between the middle and each of the two outer lines. For unseen patterns, one can then resort to the decision hyperplane to assign appropriate labels.

1.1.2 Semi- and Unsupervised Extensions

Support vector machines belong to the class of supervised learning techniques since they only make use of labeled data for generating an appropriate decision hyperplane. Ideally, a



(a) Supervised (b) Semi-Supervised (c) Unsupervised

FIGURE 1.1. *For standard support vector machines, a labeled training set (red squares and blue triangles) is given and the goal consists in finding a decision hyperplane (middle line) which maximizes the margin (indicated by the regions between the middle and each of the two outer lines), see Figure (a). In real-world settings, labeled data are usually scarce while unlabeled data (black points) can often be obtained without much additional effort. As shown in Figure (b), unlabeled data can reveal more information about the structure of the data. This additional information is taken into account by semi-supervised support vector machines. A similar task is addressed by unsupervised support vector machines, see Figure (c). However, while the corresponding model still captures the structure of the data, it might not infer the correct class for a new pattern due to missing labeled data.*

large amount of labeled patterns should be available for this concept to yield reasonable results. In real-world scenarios, however, this type of data is usually scarce and the resulting hyperplane might not be a good candidate for classifying unseen patterns. For this reason, both semi- and unsupervised extensions of support vector machines have been proposed in the literature which aim at incorporating unlabeled patterns $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$ in a reasonable manner.

Semi-Supervised Support Vector Machines

The semi-supervised extension takes both the labeled set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathbb{R}^d \times \{-1, +1\}$ and the unlabeled set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$ of patterns into account. In a nutshell, the idea is to consider the same objectives on the labeled part as a support vector machine but, at the same time, to additionally enforce the decision hyperplane not to go through high-density areas induced by the unlabeled patterns. An illustration of this extension is given in Figure 1.1 (b). Here, the decision hyperplane still maximizes the margin with respect to the labeled patterns (to some degree) but simultaneously avoids the unlabeled patterns (black points). In this case, the unlabeled patterns clearly provide useful information and the resulting model is better suited to classify unseen patterns. From an optimization point of view, semi-supervised support vector machines aim at identifying a partition of the unlabeled patterns into two classes such that a subsequent application of a *modified* support vector machine yields the best overall result.[1]

[1] We would like to point out that there exists a variety of other possibilities to extend the concept of support vector machines to semi-supervised settings. In this thesis, we will focus on the described combinatorial extension, which depicts one of the main research directions in this context.

(a) Image Classification (b) Fore- and Background Separation

FIGURE 1.2. *Both examples demonstrate possible application domains of semi- and unsupervised classification schemes. In Figure (a), the task of classifying handwritten digits is shown. Besides the data (i.e., the grayscale images [42]), appropriate labels are needed for generating machine learning models. This labeling usually requires a visual inspection by human beings. Another example is shown in Figure (b). Here, instead of classifying different types of images, one aims at detecting the two most dominant groups of pixels. In this case, these two groups correspond to the fore- and the background of the image.*

Unsupervised Support Vector Machines

The optimization task induced by semi-supervised support vector machines is strongly related to the one of unsupervised support vector machines. For the unsupervised case, one is only given the set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$ of unlabeled patterns and the goal consists in finding a partition of these patterns into two classes such that a subsequent application of a standard support vector machine yields the best overall result, see Figure 1.1 (c) for an illustration. Note that a trivial solution for this task is to assign all patterns to one class. In order to avoid such (undesired) degenerated solutions, one usually considers an additional constraint which ensures that at least one pattern (or a specific amount of patterns) is assigned to each of the two classes. Exactly as for the semi-supervised case, this task leads to a combinatorial optimization problem which is difficult to address.

1.1.3 Application Examples

As mentioned above, the process of labeling data can be time-consuming and expensive for specific tasks and, hence, leads to a general lack of labeled data for such settings. Further, in some cases, no labels at all are given for the task at hand and one could, e. g., aim at detecting the most dominant classes present in the data. The following three real-world examples sketch possible application domains for support vector machines and their extensions to both semi- and unsupervised settings.

Spam Detection

A typical binary classification task is the automatic classification of emails into two groups, in particular the detection of email spam. In such settings, each email can be represented as an unordered collection of words (occurring in the text) and the labels indicate whether a user classifies a given email as `spam` or `not spam`. Support vector machines are known

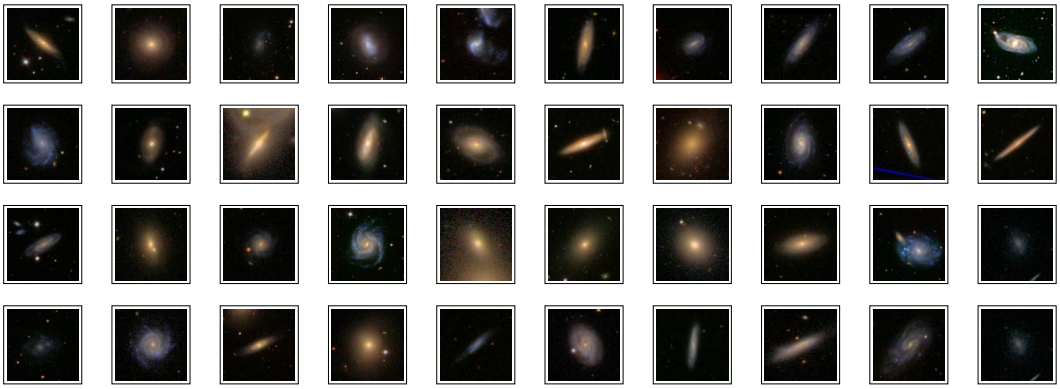


FIGURE 1.3. *An important task in the field of astronomy is the classification of galaxies based on their shapes [6, 17]. In this context, a binary classification task could consist, for instance, in discriminating elliptical and spiral galaxies [132].*

to perform well on text data in general [84]. However, to generate personalized spam filters, the user has to manually label a relatively large amount of emails. In contrast to obtaining such labeled patterns, unlabeled ones can usually be obtained for free in this context. Thus, semi-supervised learning schemes like the corresponding extension of support vector machines have the potential to reduce the amount of manual interaction by the user dramatically.

Image Classification

Another application domain is the automatic classification of images. A popular example is depicted in Figure 1.2 (a). Here, each pattern corresponds to the image of a handwritten digit and the associated labels indicate whether an image is of type `3` or of type `8`. Again, for such scenarios, labeling sufficient images manually can be time-consuming, especially if a variety of classification tasks are given. This is the case, for instance, in the field of astronomy, where one could aim at classifying the different types of galaxies, see Figure 1.3. Similar to generating personalized spam filter models, unlabeled data might be used to reduce the required amount of visual inspection by human beings for such learning tasks.

Fore- and Background Separation

The two problem settings described above depict reasonable examples in which the additional unlabeled data might be useful. In general, without any labeled data at all, unsupervised classification methods can be used for similar problem settings to detect the most dominant (possibly unknown) classes in the data. An important task in the field of pattern recognition [15] is, for instance, the automatic separation of fore- and background pixels in images. Such a setup is sketched in Figure 1.2 (b). Here, each pattern corre-

sponds to a pixel (represented by three color values) and one aims at detecting the two most distinct groups of pixels in the color space. This task is a typical application domain for unsupervised classification schemes. Note, however, that for this particular task, both supervised and semi-supervised classification schemes might also be useful.[2]

1.2 Related Work

A wide range of supervised as well as semi- and unsupervised learning techniques exist in the field of machine learning [15, 73]. In this work, we will mainly focus on support vector machines and their depicted extensions to semi- and unsupervised learning settings. The concept of support vector machines has gained tremendous attention during the last two decades both from a practical as well as theoretical point of view [124, 135]. Further, various extensions have been proposed including the semi- and unsupervised variants sketched above. Both extensions considered in this thesis lead to combinatorial optimization tasks which are difficult to address. In the related literature, three general research directions for these tasks can be found:

(1) *Exact approaches* (see, e. g., [10, 35, 112]): A small number of techniques aim at solving these combinatorial tasks exactly, i. e., they aim at computing solutions with guaranteed accuracy. While the resulting approaches are interesting from a theoretical point of view, the involved computational complexities usually limit their practical use.

(2) *Relaxations* (see, e. g., [14, 98, 140, 148, 149]): Some of the approaches are based on reformulating the original problem definitions to obtain tasks that are easier to address. On the one hand, these instances can often be solved exactly in an efficient manner (via, e. g., convex optimization techniques [20]) and the resulting solutions might depict reasonable candidates. On the other hand, a solution for these modified problem instances is not necessarily optimal in the sense of the original problem definition.

(3) *Local search schemes* (see, e. g., [32, 33, 38, 39, 55, 83, 104, 127, 131, 146, 153, 156]): The most prominent class of related techniques are local search schemes which aim at computing reasonable solutions in an efficient manner. While these approaches are often very attractive from a computational point of view, no guarantee of the solutions' quality is given. However, such schemes often yield surprisingly good results on both artificial and real-world data sets.

[2] In Figure 1.2 (b), for instance, one could also be interested in separating the seagull from the remaining part of the image. For this particular task, one could manually label some pixels belonging to the seagull and some pixels that do not belong to the seagull.

In the remainder of this work, the above learning concepts will be formalized and the related literature will be discussed in a more detailed manner. Despite these concepts several applications of machine learning techniques in the field of astronomy will be discussed. For an overview of this emerging interdisciplinary field, we refer the reader to the surveys given by Ball and Brunner [6] and Borne [17].

1.3 Overview on this Thesis

The content presented in this work is partially based on several papers [58, 59, 60, 61, 62, 67, 63, 64, 65, 66, 91, 92, 139]. The author's contribution to these publications and manuscripts is depicted in the appendix of this thesis. We will now give an overview of the thesis at hand, which is split into four parts:

Part I: Foundations

In the first part, we will provide the machine learning background for the remaining chapters. For this sake, a brief introduction into *statistical learning theory* will be given, which deals with the formal analysis of machine learning techniques. In addition, the different learning settings that are central for the thesis will be defined. As we will see, addressing machine learning tasks usually becomes increasingly difficult in high-dimensional input spaces. One way to cope with such difficulties are *dimension reduction techniques*, which we will briefly sketch as well. These issues will be subject of Chapter 2.

Most parts of this work are related to support vector machines. In Chapter 3, the mathematical background of this concept will be provided, which will form the basis for deriving its extensions to semi- and unsupervised learning scenarios.

Part II: Semi- and Unsupervised Support Vector Machines

The second part of this thesis deals with the extensions of support vector machines, which lead to combinatorial optimization tasks. Surprisingly, exact approaches, i. e., methods aiming at computing solutions with guaranteed accuracy, have not been investigated extensively in the related literature. In Chapter 4 we will propose an approach that is capable of computing exact solutions (up to machine precision) in polynomial time for special cases of the problem instances [66]. Despite providing these theoretical insights, the approach can also be used to generate benchmark data sets in low-dimensional feature spaces, which is of independent interest. The computational complexity of the algorithm, however, renders an application in high-dimensional learning settings impossible.

Aiming at such learning scenarios, we will therefore propose a simple local search strategy to approach both tasks in Chapter 5. Since a direct implementation is still

computationally expensive, we will show how to make use of matrix-based updates for the intermediate candidate solutions, which will greatly reduce the overall runtime [59, 63]. This renders the approach capable of testing a massive amount of candidate solutions and, thus, paves the way for a more detailed exploration of the search space.

Finally, in Chapter 6, we will depict an alternative optimization perspective for the tasks at hand which leads to continuous and non-convex optimization problems. As we will see, a simple but careful application of gradient based schemes yields extremely efficient optimization frameworks to deal with the tasks [67]. Although being conceptually very simple, the resulting approach yields results which are superior (or, at least, comparable) to state-of-the-art methods, both with respect to the classification as well as to the runtime performance. The chapter concludes with a discussion of critical issues related to global optimization and parameter selection.

Part III: Applications in Astronomy

The data volume in astronomy has increased dramatically in recent years. The third part of this work deals with application domains of machine learning techniques in this field. In particular, we will derive a new, adaptable scheme for extracting the *continuum* (rough shape) of a given spectrum and will show how to define simple but expressive features based on continuum-subtracted versions of the raw data in the context of detecting quasars, a special type of astronomical objects [62]. Further, the possible benefits of semi-supervised learning schemes in astronomy will be discussed including perspectives related to multi-view learning and ranking [139]. These issues will be subject of Chapter 7.

Unsupervised learning schemes have also gained interest in astronomy during recent years. One example is the automatic data analysis taking place on board of today's space-craft systems. In contrast to the analysis on earth, however, such systems are faced with computational problems which are caused by the cosmic radiation. In Chapter 8, these problems will be discussed. In addition, we will show how to make use of sophisticated data structures that render the data analysis in such scenarios more stable (without much additional cost with respect to runtime and space consumption) [61, 64].

Part IV: Summary

Finally, in the fourth part, we will summarize the main results of this work and will discuss future research directions including sophisticated concepts to cope with the lack of labeled data and the huge data volumes in astronomy [58, 60, 65] as well as on-line monitoring systems of important astronomical events [91, 92].

Part I

Foundations

CHAPTER 2

Machine Learning Background

In this chapter, we will formalize the concept of *learning*. Briefly speaking, this concept deals with the design of algorithms yielding *models* based on observed data. The key idea is that these models capture as many of the characteristics of the data at hand as possible such that they can make reasonable predictions for unseen data patterns. As an example, consider again the task of classifying handwritten digits. Here, the training data are given as grayscale images with associated class labels and the goal is to derive a model which can automatically assign appropriate class labels to new images.

One of the main issues in the field of machine learning is the fact that such models have to be generated based on a *finite* amount of data. Predictions, however, usually have to be made for *any* possible new data item. This renders simple look-up strategies unsuitable in most cases. Further, learning tasks become increasingly difficult to approach in high-dimensional input spaces. To cope with this phenomenon, known as *curse of dimensionality*, one can try to reduce the input dimension of the data or to increase the amount of labeled data used for generating the models (or a combination thereof). The former idea is addressed by so-called *dimension reduction techniques* that aim at obtaining a low-dimensional but meaningful description of the given data. In some cases, however, such a reduction might not yield satisfying results. This normally necessitates the use of more labeled data or the application of other sophisticated strategies.

Outline. The mathematical formalization of the concept of learning and the definition of the different learning settings will be subject of Section 2.1 and Section 2.2, respectively. Most machine learning approaches depend on parameters and the specific assignments

for these parameters play a crucial role for the performance of the resulting models. In Section 2.3, two well-known model selection techniques will be sketched. Finally, in Section 2.4, the curse of dimensionality along with dimension reduction methods will be briefly explained, followed by concluding remarks given in Section 2.5.

2.1 Statistical Learning in a Nutshell

The concept of *statistical learning theory* [19, 73, 100, 135, 141] provides the mathematical framework for the theoretical analysis of machine learning techniques. Basis for such an analysis is an *input space* $\mathcal{X} \subseteq \mathbb{R}^d$, an *output space* $\mathcal{Y} \subseteq \mathbb{R}$, and random variables $(X, Y) \in \mathcal{X} \times \mathcal{Y}$ with unknown joint distribution $P(\mathbf{x}, y)$.[1] The goal of the learning process is to identify a *prediction function* $f \in \mathcal{H}$ in a *hypothesis space* $\mathcal{H} \subseteq \{g : \mathcal{X} \to \mathcal{Y}\}$ which assigns predictions to unseen *patterns* $\mathbf{x} \in \mathcal{X}$ in an optimal manner. This usually has to be accomplished given a finite *training set*

$$T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \mathcal{Y} \tag{2.1}$$

consisting of $l \in \mathbb{N}$ independent and identically distributed pairs $(\mathbf{x}'_i, y'_i) \in \mathcal{X} \times \mathcal{Y}$ sampled according to the distribution $P(\mathbf{x}, y)$ [19, 100, 135]. To measure the quality of a given prediction function $f \in \mathcal{H}$, one can resort to the *expected*, *empirical*, and the *regularized risk*, which we will describe next.

2.1.1 Expected and Empirical Risk

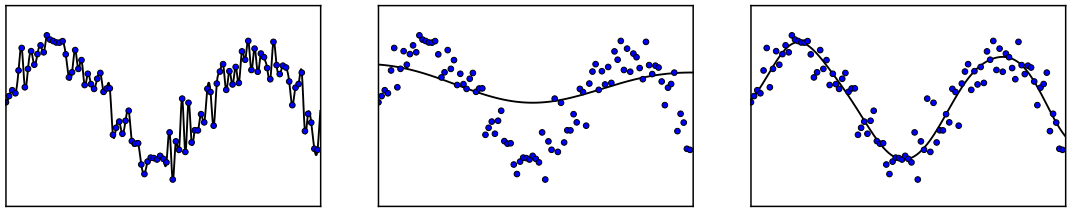
The definition of these risks is based on *loss functions* $\mathcal{L} : \mathcal{Y} \times \mathbb{R} \to [0, \infty)$ measuring the disagreement $\mathcal{L}(y, f(\mathbf{x}))$ of a given label $y \in \mathcal{Y}$ and a corresponding prediction $f(\mathbf{x})$ for a particular pattern $\mathbf{x} \in \mathcal{X}$. Well-known loss functions are, for instance, the *0-1 loss* $\mathcal{L}(y, f(\mathbf{x})) = \mathbf{1}_{\{f(\mathbf{x}) \neq y\}}$ and the *square loss* $\mathcal{L}(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$. For a given prediction function $f \in \mathcal{H}$, the *expected risk* $R[f]$ is then defined as

$$R[f] := \int_{\mathcal{X} \times \mathcal{Y}} \mathcal{L}(y, f(\mathbf{x})) dP(\mathbf{x}, y). \tag{2.2}$$

In principle, one would like to identify the prediction function $f \in \mathcal{H}$ which has minimal expected risk over all possible functions.[2] However, since the joint distribution $P(\mathbf{x}, y)$

[1] For the precise definition of the spaces and the joint distribution, one has to resort to the concept of *measure theory*. In this context, the spaces $\mathcal{X}$ and $\mathcal{Y}$ are arbitrary *measurable spaces* with associated σ-algebras and one considers *measurable functions* as possible prediction functions. The definition of this setup, however, is beyond of the scope of this work and we refer the reader to appropriate textbooks [7, 47, 135, 141] for details.

[2] The infimum of $R[f]$ over all possible measurable functions is called the *Bayes risk* [19].



(a) λ=small (b) λ=large (c) λ=medium

FIGURE 2.1. *Three possible models (black curves) for a given set of data points. In Figure (a), the model fits the data well but is too complex (due to a small λ). In Figure (b), the model is simple (due to a large λ), but does not fit the data. An appropriate model is the one shown in Figure (c) with a reasonable trade-off between model complexity and data fit.*

is unknown, one cannot resolve the integral (2.2). Instead, one usually considers the *empirical risk* $R_{emp}[f]$ as surrogate for the expected risk which is defined as

$$R_{emp}[f] := \frac{1}{l} \sum_{i=1}^{l} \mathcal{L}(y_i', f(\mathbf{x}_i')). \tag{2.3}$$

Unfortunately, there are two difficulties arising when using the empirical risk instead of the expected risk in this context. Firstly, there might be many possible prediction functions in the hypothesis space $\mathcal{H}$ rendering the search for an optimal function difficult. Secondly, it is always possible to select a prediction function among *all* possible prediction functions having small empirical risk but high expected risk [19]. For instance, the function $g : \mathcal{X} \to \mathcal{Y}$ defined as

$$g(\mathbf{x}) := \begin{cases} y_i' & \text{if } \mathbf{x} = \mathbf{x}_i' \text{ for a } i \in \{1, \ldots, l\} \text{ and} \\ 0 & \text{otherwise,} \end{cases} \tag{2.4}$$

yields small empirical risk (assuming, e. g., the 0-1 loss) but does not provide any information about unseen patterns. Thus, simply identifying a function with low expected risk is not useful in general. To cope with both difficulties, one usually restricts the hypothesis space or modifies the objective by adding a term which penalizes complex functions.

2.1.2 Regularized Risk

One possible concept to deal with these problems is *regularization* [19, 73]. The basic idea consists in choosing a large hypothesis space $\mathcal{H}$ and to add a *regularization term* to the objective (i. e., to the empirical risk). This leads to the *regularized risk* $R_{reg}[f]$ defined as

$$R_{reg}[f] := R_{emp}[f] + \lambda \Omega(f) \tag{2.5}$$

for a particular prediction function $f \in \mathcal{H}$, where $\Omega : \mathcal{H} \to \mathbb{R}$ is the so-called *regularizer* which penalizes complex functions. Thus, the first term of the right hand side of (2.5) measures how well the prediction function f fits the patterns in the training set whereas the second term measures the complexity of the function. The additional *regularization parameter* $\lambda \in \mathbb{R}^+$ determines the trade-off between both objectives, i. e., between a good data fit and a small complexity, see Figure 2.1 for an illustration. Typically, the regularizer $\Omega(\cdot)$ is given in terms of a norm $\|\cdot\|_{\mathcal{H}}$ defined on the hypothesis space $\mathcal{H}$. Thus, the concept of regularization is based on the assumption that appropriate prediction functions exhibit a smooth behavior [73]. As we will see in Chapter 3, support vector machines sketched in the introductory chapter are based on this concept as well.

2.1.3 Generalization Bounds

Statistical learning theory aims at providing the theoretical framework for the analysis of algorithms yielding prediction functions of the above form. In general, a prediction function $f \in \mathcal{H}$ generated by such a *learning algorithm* is designated to have small expected risk $R[f]$. Again, since the joint distribution $P(\mathbf{x}, y)$ is unknown, one cannot approach this task directly. One of the goals of statistical learning is to provide *bounds* [19] of the form

$$R[f] \leq R_{emp}[f] + B(l, \mathcal{H}), \tag{2.6}$$

which are fulfilled with high probability for each $f \in \mathcal{H}$ and where $B(l, \mathcal{H}) \to 0$ holds for $l \to \infty$. Here, the term $B(l, \mathcal{H})$ captures the complexity of $\mathcal{H}$ and is supposed to be small if not too complex functions are contained in $\mathcal{H}$. Thus, such bounds essentially state that if the hypothesis space $\mathcal{H}$ contains simple functions (i. e., small $B(l, \mathcal{H})$) and, at the same time, one of the considered functions f can describe the data well (i. e., small empirical risk), the empirical risk $R_{emp}[f]$ is likely to be close to the expected risk $R[f]$. A detailed discussion of these issues, however, goes beyond the scope of this work and we refer the reader to appropriate surveys [19] and textbooks [135, 141] for an overview.

2.2 From Supervised to Unsupervised Learning

We will now define the central learning tasks for the work at hand and will subsequently sketch simple algorithms to approach them.

2.2.1 Supervised Learning

The setup described above is called *supervised learning* since an algorithm has access to a *labeled training set* $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \mathcal{Y}$ consisting of patterns $\mathbf{x}'_i$ with

associated labels y'_i. The two most common supervised learning tasks are *classification* and *regression* [73]. For the former one, the output space $\mathcal{Y}$ consists of a finite set of discrete variables which represent the *classes*. Here, the special case of two classes, i. e., $\mathcal{Y} = \{-1, +1\}$, induces *binary classification* tasks.[3] For the latter one, regression, the output space is usually given as $\mathcal{Y} = \mathbb{R}$. For such tasks, the goal consists in finding a prediction function which assigns appropriate real-valued labels to unseen patterns $\mathbf{x} \in \mathcal{X}$. In this thesis, we will mainly focus on binary classification tasks as well as its extensions to unsupervised and semi-supervised settings.

2.2.2 Unsupervised Learning

At early stages of the learning process, no labels at all are usually given for the patterns (in some cases, it is even not clear what to search for). For such settings, one is only given an *unlabeled training set* $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$ and the induced learning tasks belong to *unsupervised learning* scenarios. The unsupervised analogon for classification is *clustering* (or *unsupervised classification*). Here, the goal is to partition the patterns into groups such that patterns within the same group exhibit similar properties and those being in different groups exhibit dissimilar ones. Note, however, that the desired classification and clustering models do not necessarily coincide, see Figure 2.2. This depends on the data at hand and on the particular objectives for both settings (e. g., on the similarity measure used for defining the clustering partitions). Despite clustering, dimension reduction schemes depict well-known unsupervised learning concepts as well (which will be sketched below).

2.2.3 Semi-Supervised Learning

A recent direction in the field of machine learning is *semi-supervised learning*. Briefly speaking, it can be seen as a mixture of the above two learning settings where both a labeled $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \mathcal{Y}$ and an unlabeled training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$ are given. The key idea of semi-supervised learning is that the unlabeled patterns reveal more information about the structure of the data (compared to supervised learning settings). Thus, an algorithm aims at taking both parts of the data into account for generating appropriate models. Again, a variety of semi-supervised learning tasks can be defined and we refer to the surveys given by Chapelle *et al.* [34] and Zhu and Goldberg [157] for an introduction. In the following, we will consider semi-supervised binary classification settings. There are two slightly different semi-supervised learning scenarios, namely *inductive* and *transductive semi-supervised* learning.

[3] Note that any classification task with $|\mathcal{Y}| > 2$ can be considered as a set of binary classification tasks (e. g., via an one-versus-all strategy).

Inductive Scenarios

The first of these two variants is called *inductive semi-supervised learning* [34, 157]. Here, an algorithm has access to both the labeled and the unlabeled training set. Similar to supervised settings, the goal of the learning process consists in deriving a function $f \in \mathcal{H}$ having a good performance on unseen patterns. Thus, for such settings, these patterns are neither contained in T_L nor in T_U and are therefore not available for generating the model.

Transductive Scenarios

The second setting is called *transductive semi-supervised learning* [34, 157]. Exactly as before, one aims at deriving a function $f \in \mathcal{H}$ based on both the labeled training set T_L and the unlabeled one T_U. However, in contrast to inductive semi-supervised learning scenarios, one is interested in the performance of $f \in \mathcal{H}$ on the unlabeled training set itself. The induced learning tasks are therefore easier to approach since the patterns used for testing are already known during the construction phase of the model. Note that for real-world settings, such a setup is often given: Despite a small amount of labeled patterns, one usually has access to a huge amount of unlabeled patterns. At the same time, one is interested in reasonable labels for these (already given) unlabeled patterns.[4]

2.2.4 Elementary Algorithms

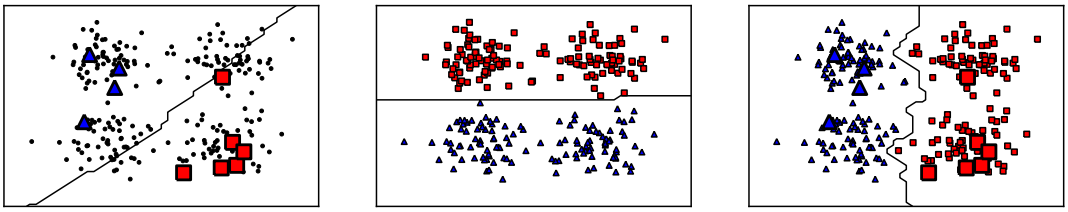
We will now sketch elementary classification algorithms for each of the learning settings depicted above to demonstrate the key ideas of these concepts.

K-Nearest Neighbors

One of the most popular supervised classification algorithms is the *k-nearest neighbor classifier* [73]. The key idea of this approach is to assign a label to a new pattern $\mathbf{x} \in \mathcal{X}$ based on its neighborhood in the training set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \mathcal{Y}$. More precisely, for binary classification scenarios with $\mathcal{Y} = \{-1, +1\}$, the prediction function $f : \mathcal{X} \to \mathcal{Y}$ is defined as

$$f(\mathbf{x}) := \begin{cases} 1 & \text{if } \sum_{\mathbf{x}'_i \in N_k(\mathbf{x})} y'_i > 0 \text{ and} \\ -1 & \text{if } \sum_{\mathbf{x}'_i \in N_k(\mathbf{x})} y'_i \leq 0, \end{cases} \tag{2.7}$$

[4] As an example, consider today's astronomical catalogs. Here, a small portion of objects is usually labeled manually by experts in this field. The majority of objects, however, is not labeled. For such scenarios, it is interesting to build machine learning models which can automatically label these unlabeled but already available patterns.



(a) Supervised (b) Unsupervised (c) Semi-Supervised

FIGURE 2.2. *The figures demonstrate the key ideas of (a) supervised, (b) unsupervised, and (c) semi-supervised classification, where the large squares and triangles depict labeled and the black points unlabeled patterns. Small squares and triangles represent unlabeled patterns that have been assigned a label by one of the approaches. The final models (i.e., decision functions) are indicated by the black lines. Clearly, the k-nearest neighbor model (with $k = 3$), shown in Figure (a), seems to be inappropriate for the data at hand since it does not take the structure induced by the unlabeled patterns into account. The clustering solution shown in Figure (b) obtained via the k-means approach (with $k = 2$) does capture the structure of the data. However, it does not yield the (desired) classification partition induced by the labeled patterns. This is achieved via the semi-supervised propagating 1-nearest neighbor scheme as depicted in Figure (c).*

where $N_k(\mathbf{x})$ denotes the k-nearest neighbors of $\mathbf{x}$ in the training set T_L for a user-defined $k \in \mathbb{N}$. For the definition of the neighborhood, arbitrary distance measures can be used. A popular one is the Euclidean distance, see Figure 2.2 (a) for an illustration. One of the advantages of this classification concept is the fact that it can be extended to multiclass classification settings in a straightforward manner. As pointed out by Hastie *et al.*, this type of classifier "is often successful where each class has many possible prototypes, and the decision boundary is very irregular" [73, p. 465].

K-Means

For clustering scenarios, one of the most popular approaches is the *k-means algorithm* [101]. Given an unlabeled training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X} = \mathbb{R}^d$, this algorithm aims at finding a partition $\{S_1, \ldots, S_k\}$ of the input patterns into a predefined number $k \in \mathbb{N}$ of clusters. Formally, the algorithm aims at finding the global optimum of

$$\underset{\{S_1,\ldots,S_k\}}{\text{minimize}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \mathbf{c}_i\|^2, \tag{2.8}$$

over all possible partitions, where $\mathbf{c}_i$ denotes the mean of the points contained in the set S_i. This optimization task is known to be NP-hard, even for special cases [4]. If both the number k of designated clusters and the input dimension d are fixed, however, the optimal solution can be obtained in $\mathcal{O}(u^{kd+1})$ time [82]. Due to these computational complexities, one usually resorts to heuristics for addressing the above optimization task that yield (possibly suboptimal) solutions in an efficient manner. One of them is the k-means algorithm. The framework is given in Algorithm 2.1: After initializing the set of

Input: A set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$ and a user-defined number $k \in \mathbb{N}$ of clusters.
Output: A partition of the set T_U into k clusters along with cluster centers $\{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$.
1: Randomly initialize cluster centers $\mathbf{c}_1, \ldots, \mathbf{c}_k \in T_U$.
2: **repeat**
3: Assign each pattern in T_U to its nearest cluster center $\mathbf{c}_i$.
4: Recompute positions of the cluster centers (mean of assigned patterns).
5: **until** no changes in this iteration.
6: **return** Partition of T_U and cluster centers $\{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$.

ALGORITHM 2.1. *The k-means algorithm [101] computes a partition of the input data into $k \in \mathbb{N}$ clusters. Throughout its execution, a set of candidate centers is maintained and the approach adjusts these centers until a convergence criterion is fulfilled.*

candidate centers with k points (sampled, e. g., uniformly at random from the set of input patterns T_U), the algorithm proceeds in multiple iterations. In each of these iterations, every pattern is assigned to its nearest candidate center (Step 3) and at the end of the iteration, each candidate center is updated to the mean of all patterns assigned to it (Step 4). The algorithm terminates as soon as an iteration does not result in any change and returns both the partition of T_U as well as the corresponding cluster centers, see Figure 2.2 (b) for an illustration.

Propagating 1-Nearest Neighbor

Finally, let us consider a simple semi-supervised classification approach which belongs to the class of so-called *self-training* [157] models. Briefly speaking, the corresponding learning schemes use their own predictions to (iteratively) classify the unlabeled patterns T_U (and to rebuild the classifier after each iteration). One possible semi-supervised variant of the k-nearest neighbor model is the *propagating 1-nearest neighbor classifier* [157]. Its algorithmic framework is shown in Algorithm 2.2: In each step, the unlabeled candidate pattern closest to any of the already labeled patterns is selected. The label for this pattern is determined by the label of its nearest neighbor in the current set of labeled patterns. This iterative process is repeated until no unlabeled patterns are left. The result of this scheme is illustrated in Figure 2.2 (c). In this case, the additional information provided by the unlabeled patterns clearly leads to a better classification model.

2.2.5 Related Work

One of the concepts related to the settings described above is, for instance, *constraint clustering* [8]. Here, the goal consists in deriving a partition of the data patterns into groups so that similar patterns belong to the same group and dissimilar ones to different

Input: A labeled $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \mathcal{Y}$ and an unlabeled training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$.
Output: A set of labels $\{y_1, \ldots, y_u\}$ for the set T_U.
1: Initialize $L = T_L$ and $U = T_U$.
2: **repeat**
3: Select $\mathbf{x} = \text{minimize}_{\mathbf{x} \in U} \min_{\mathbf{x}' \in L} d(\mathbf{x}, \mathbf{x}')$.
4: Add $(\mathbf{x}, y)$ to L, where y is the label of the nearest neighbor of $\mathbf{x}$ in L.
5: Remove $\mathbf{x}$ from U.
6: **until** U is empty

ALGORITHM 2.2. *The propagating* 1*-nearest neighbor classifier belongs to the class of self-training models [157]. The key idea is to make use of the intermediate classification models to iteratively classify the unlabeled part of the data. Instead of the Euclidean distance d, arbitrary distance functions can be used.*

groups. But, in addition to the set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$ of unlabeled patterns, one is given additional information in terms of, e. g., *must-link* constraint (two patterns $\mathbf{x}_i$ and $\mathbf{x}_j$ have to be in the same group) or *cannot-link* constraints (two patterns $\mathbf{x}_i$ and $\mathbf{x}_j$ must not be be in the same group) [157]. A wide range of other learning techniques is to be found in the literature including, e. g., semi-supervised regression frameworks [94] or corresponding extensions of dimension reduction methods [151]. A general discussion of such methods, however, is beyond the scope of this work and we refer the reader to corresponding textbooks and surveys [5, 15, 34, 73, 157] for an overview of various supervised, semi-supervised, and unsupervised learning schemes.

2.3 Model Selection

The task of *model selection* describes the problem of selecting appropriate machine learning techniques (i. e., algorithms) along with associated model parameters. For instance, the regularization parameter $\lambda \in \mathbb{R}^+$ for problems of the form (2.5) has to be set appropriately for a given training set instance. Another example is the parameter $k \in \mathbb{N}$ which has to be set for the k-nearest neighbor classifier. Below, we will sketch two well-known strategies which can be used if (at least some) labeled patterns are given, i. e., for supervised and semi-supervised scenarios.[5]

2.3.1 Training, Validation, and Test Set

For a data-rich situation with sufficient labeled patterns, a common approach consists in splitting up the data at hand into three parts: the *training*, *validation*, and *test set*. A

[5] For unsupervised settings, however, the situation is more difficult since the corresponding tools, "so useful for model selection in supervised learning, cannot be utilized in this context" [73, p. 519].

model (e. g., an algorithm along with appropriate assignments for the involved parameters) is then applied to the training set and its performance is evaluated on the validation set. Such training and validation phases are repeated for a set of possible parameter assignments and the best-performing parameter setting is selected to obtain the final model. For the final evaluation, one can then resort to the test set (which has *not* been used during the training/validation phases). As performance measure for all three phases, the empirical risk (induced by the specific loss function) is usually considered. This results in the *training*, *validation*, and *test error*. As pointed out by Hastie *et al.* [73], one typically uses half of the data set as training set and one fourth for each the validation and the test set.

2.3.2 K-Fold Cross-Validation

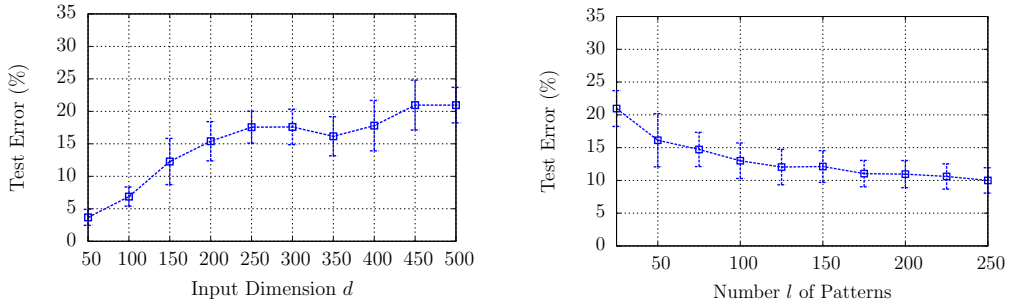
The above setting is well-suited for data-rich situations. But, in real-world settings, labeled data patterns are usually scarce. For such scenarios, a common technique is *K-fold cross-validation* [73]. The basic idea is to split up the labeled data set $T_D = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_N, y'_N)\}$ into smaller subsets. More precisely, for the training and validation phases, one partitions the data into $K \in \mathbb{N}$ disjoint and (almost) equal-sized sets $\mathcal{P}_1, \ldots, \mathcal{P}_K \subset T_D$. Then, $K-1$ out of the K sets are used for training the model (given a fixed parameter setup), and the remaining set is used to validate the model (i. e., $T_D \setminus \mathcal{P}_i$ is used for the training phase and $\mathcal{P}_i$ is used for the validation phase). Such a training and validating phase is repeated K times such that each hold-out set is used once to validate the model. The final selected model is the one with the best average performance measured on all K hold-out validation sets. The benefit of such a strategy is that each pattern in the data set is used for both training and validating the model. Typically, one performs 5-fold or 10-fold cross-validation [73]. This procedure can be extended in a natural way to incorporate the testing phase.

2.4 The Curse of Dimensionality

The *curse of dimensionality* [73] describes the phenomenon that problems often become more difficult to approach as soon as the number of involved variables increases. We will demonstrate this phenomenon by means of a simple artificial classification task and will sketch two general strategies to reduce the drawbacks of this phenomenon.

2.4.1 The Hughes Effect

To demonstrate the curse of dimensionality, also called *Hughes effect* [80], let us consider two Gaussian clusters generated by drawing $N/2$ patterns ($N = 500$) from each of two



(a) (b)

FIGURE 2.3. *Figure (a) shows how the classification performance (test error) decreases with increasing input dimension d (given a training set of fixed size $l = 25$). Taking more training data into account (for fixed $d = 500$) again leads to a better performance, see Figure (b).*

multivariate distributions $X_i \sim \mathcal{N}(\mathbf{m}_i, \mathbf{I})$ with $\mathbf{m}_1 = (-2.5, 0.0, \ldots, 0.0)^{\mathrm{T}} \in \mathbb{R}^d$ and $\mathbf{m}_2 = (+2.5, 0.0, \ldots, 0.0)^{\mathrm{T}} \in \mathbb{R}^d$. The class label of a pattern corresponds to the distribution it has been drawn from. As classification model, we consider the k-nearest neighbor classifier (with $k = 5$) and use half of the data set as training and the other half as test set.

First, let us restrict the size of the training set to $l = 25$ patterns (ignoring the remaining patterns) and let us vary the dimension d of the input space from 50 to 500. In Figure 2.3 (a), the average test error and the one standard deviation over 10 random partitions into training and test patterns is given (for each assignment of d). Clearly, by increasing the dimension of the input space, the performance of the classifier gets worse. Now, let us fix the dimension $d = 500$ and let us increase the amount of (labeled) patterns to $l = 250$. As expected, the (average) test error decreases, see Figure 2.3 (b). Thus, the performance is improved by taking more labeled data into account for generating the model. Obtaining sufficient labeled data, however, is not always possible in real-world scenarios. A possible approach to alleviate this problem consists in reducing the dimension of the input space by means of appropriate strategies, as we will sketch now.

2.4.2 Dimension Reduction

Dimensionality reduction methods offer one possible way to cope with the Hughes effect [73]. The basis of such methods is to reduce the dimension d of the input space (assuming $\mathcal{X} = \mathbb{R}^d$) without loosing too much information. The corresponding class of techniques can be divided into *feature selection* and *feature extraction* methods.

Feature Selection

The idea of *feature selection* [71, 73] is based on the assumption that only few dimensions might be relevant for a specific task and that the remaining ones are less important (or

even harmful) and can therefore be removed. This is the case, for instance, for the artificial data set described above. Here, only the first dimension is important for the classification task whereas the remaining ones are misleading. Thus, for this particular data set, it would be reasonable to ignore all input variables except the first one.

In real-world settings, however, one usually does not know the important features in advance. A simple (but time-consuming) approach consists in testing every possible subset of features via an appropriate machine learning model and to select the best one with respect to, e. g., the induced performance on the validation set. This kind of approach is called *subset selection* [73].[6] Working on such reduced feature sets can have several advantages including a faster generation as well as a better performance of the models.

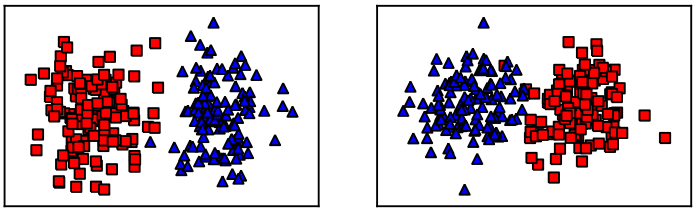
For a detailed discussion of these issues, we refer the reader to the survey given by Guyon [71]. For the work at hand it is important to point out that such methods depict an alternative way to cope with the curse of dimensionality compared to incorporating unlabeled data. However, a meaningful reduction to a low-dimensional space might be time-consuming or even impossible for specific data sets.

Feature Extraction

Exactly as for feature selection techniques, the goal of *feature extraction* [73] methods consists in reducing the dimension of the input space without losing too much information. A well-known candidate is the *principal component analysis* [111], which performs a linear transformation into a low-dimensional space. The transformation is based on the so-called *principal components* which define the new system of coordinates. They are selected in such a way that the projections of the (original) patterns onto the first principal component have maximal variance. The remaining principal components are ordered with respect to the variances of the projected data, i. e., the projections of the patterns onto the second component exhibit the second largest variance, and so on, see Hastie *et al.* [73] for an introduction.

In Figure 2.4, such a reduction of the artificial data set sketched above is shown: Here, the first two dimensions of the artificial data set are depicted in Figure (a). Further, the outcome of a projection onto the first two principal components is given in Figure (b). Note that, in this case, a classification model based on the projected data would have a comparable performance as a model trained on the first two dimensions of the original data set. Thus, the application of such dimension reduction methods depicts another way to cope with the curse of dimensionality. However, whether such a mapping preserves the structure of the data or not is unclear and depends on the specific setting.

[6] Since the task of selecting an optimal subset of features (with a given user-defined cardinality) is of combinatorial nature, one usually resorts to greedy hill climbing strategies to select a reasonable subset.



(a) Selected Features (b) Projections

FIGURE 2.4. *The first two dimensions (features) of the artificial data set described in Section 2.4.1 are shown in Figure (a). The projections of the high-dimensional patterns ($d = 500$) onto the first two principal components are given in Figure (b).*

A variety of other dimension reduction methods including non-linear variants like *locally linear embedding* [123] and *Isomap* [137] have been proposed in the literature, see e. g., Fodor [52] for an overview. All these techniques aim at deriving a meaningful low-dimensional representation of the data in an automatic manner.

2.5 Concluding Remarks

In this chapter, some basics related to statistical learning theory have been provided. In particular, the idea of regularization techniques was sketched. As we will see in the next chapter, the concept of support vector machines belongs to such regularization methods as well. Despite the different learning settings and model selection issues, the curse of dimensionality has been briefly introduced. For supervised tasks, the negative effects of this phenomenon can be shortened by (a) dimension reduction methods or by (b) using more labeled data (or combinations thereof). The reduction of the input space's dimension can be addressed by means of feature selection and extraction methods. However, depending on the data at hand, such a reduction might also fail. On the other hand, sufficient labeled data are often not available in real-world settings. As sketched in Section 2.2, incorporating unlabeled data can also help to reveal more information about the structure of the data and, thus, to circumvent the curse of dimensionality. The extension of support vector machines to these scenarios will be subject of the second part of this thesis.

CHAPTER 3

Support Vector Machines Revisited

Support vector machines are among the most popular classification schemes in the field of machine learning. As outlined in Chapter 1, the key idea of this concept is to identify the hyperplane separating both classes such that the induced distance between the hyperplane and the training patterns is maximal. The so-called kernel functions are one of the reasons for the success of support vector machines. For patterns in the Euclidean space $\mathbb{R}^d$, such functions render non-linear decision functions possible and, thus, provide the basis for flexible models that can adapt to the structure of the data at hand. In addition, such kernel functions can also be defined on arbitrary sets of objects (like sets of graphs or strings), which paves the way for the applicability of support vector machines in various learning scenarios. In this chapter, support vector machines will be described in a detailed manner. More specifically, both the linear and the non-linear case will be derived, where the former one corresponds to linear decision hyperplanes (with patterns in $\mathbb{R}^d$) and the latter one to non-linear decision hyperplanes induced by kernel functions (with patterns in arbitrary sets).

Outline. We start by deriving the concept of support vector machines for patterns in the d-dimensional Euclidean space $\mathbb{R}^d$ in Section 3.1. The general perspective based on kernel functions will be described in Section 3.2. As we will see, both linear and non-linear support vector machines lead to convex optimization tasks. These tasks as well as associated computational aspects will be discussed in Section 3.3. In Section 3.4, the effect of high-dimensional input spaces on the classification performance of support vector machines will be sketched, followed by concluding remarks provided in Section 3.5.

3.1 Linear Support Vector Machines

In this section, we will consider training patterns in the d-dimensional Euclidean space $\mathbb{R}^d$ and derive the concept of *linear support vector machines* [1, 40, 124, 135].

3.1.1 Preliminaries

We will start by providing some elementary definitions [45, 46] related to hyperplanes in $\mathbb{R}^d$, which will be important for Chapter 4 as well.

Hyperplanes

Let $P = \{\mathbf{p}_0, \ldots, \mathbf{p}_k\}$ be a set of $k+1$ points in $\mathbb{R}^d$. A point $\mathbf{p} \in \mathbb{R}^d$ is called a *linear combination* of P if it can be written as a sum $\mathbf{p} = \sum_{i=0}^{k} \beta_i \mathbf{p}_i$ with real coefficients $\beta_0, \ldots, \beta_k$. Further, if $\sum_{i=0}^{k} \beta_i = 1$, then $\mathbf{x}$ is called an *affine combination* and, in case both $\sum_{i=0}^{k} \beta_i = 1$ and $\beta_0, \ldots, \beta_k \geq 0$ hold, a *convex combination* of P. The *linear*, the *affine*, and the *convex hull* of P denote the set of all linear, affine, and convex combinations of P, respectively. If there is no point $\mathbf{p}_i \in P$ which is a linear or affine combination of $P - \{\mathbf{p}_i\}$, then P is called *linearly* or *affinely independent*, respectively. In case of an affinely independent point set P, the affine combination of P is called a *k-flat* and the number k is called the *dimension* of the k-flat. A $(d-1)$-flat in $\mathbb{R}^d$ is named *hyperplane*. For any hyperplane h in $\mathbb{R}^d$, let h^+ denote the open half-space not containing the origin $\mathbf{0}$ and let h^- denote the other open half-space induced by h.

Offset Term

Any hyperplane h in $\mathbb{R}^d$ can be written as

$$h = h(\mathbf{w}, b) = \{\mathbf{x} \in \mathbb{R}^d \mid \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}, \tag{3.1}$$

where $\mathbf{0} \neq \mathbf{w} \in \mathbb{R}^d$ is normal to the hyperplane and $b \in \mathbb{R}$ [45]. Let $\mathbf{x} \in h$ and let $\bar{\mathbf{x}} = r\mathbf{w}$ with $r \in \mathbb{R}$ be the orthogonal projection of $\mathbf{x}$ on the subspace spanned by $\mathbf{w}$. Since $\langle r\mathbf{w}, \mathbf{w} \rangle - \langle \mathbf{x}, \mathbf{w} \rangle = \langle \bar{\mathbf{x}} - \mathbf{x}, \mathbf{w} \rangle = 0$, we have $r = \langle \mathbf{w}, \mathbf{x} \rangle / \langle \mathbf{w}, \mathbf{w} \rangle$. Hence, if $\|\mathbf{w}\| = 1$, then the product $\langle \mathbf{w}, \mathbf{x} \rangle$ denotes the length of $\mathbf{x}$ with respect to $\mathbf{w}$. If $\|\mathbf{w}\| \neq 1$, then $\langle \mathbf{w}, \mathbf{x} \rangle / \|\mathbf{w}\|$ denotes this length as we have

$$\bar{\mathbf{x}} = r\mathbf{w} = \frac{\langle \mathbf{w}, \mathbf{x} \rangle}{\langle \mathbf{w}, \mathbf{w} \rangle} \mathbf{w} = \frac{\langle \mathbf{w}, \mathbf{x} \rangle}{\|\mathbf{w}\|} \frac{\mathbf{w}}{\|\mathbf{w}\|}. \tag{3.2}$$

Consequently, for any hyperplane h of the form (3.1) and $\mathbf{x} \in h$, the term $\frac{\langle \mathbf{w}, \mathbf{x} \rangle}{\|\mathbf{w}\|} = \frac{-b}{\|\mathbf{w}\|}$ determines the *offset* of the hyperplane to the origin $\mathbf{0}$ along the direction given by $\mathbf{w}$.

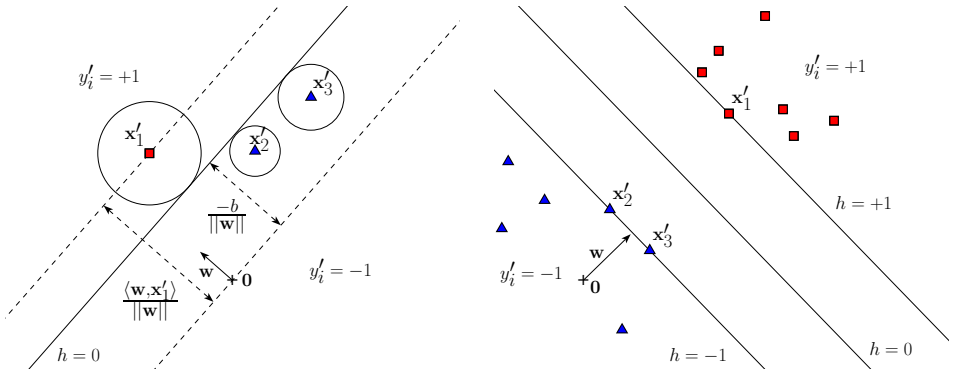


FIGURE 3.1. *The concept of the geometrical margin for a training set* $T_L = \{(\mathbf{x}'_1, +1), (\mathbf{x}'_2, -1), (\mathbf{x}'_3, -1)\}$ *is illustrated in Figure (a). In case a hyperplane* $h = h(\mathbf{w}, b) \subset \mathbb{R}^d$ *fulfills the additional constraint (3.6), the geometrical margin with respect to the labeled training set is given by* $1/\|\mathbf{w}\|$. *This follows from* $\langle \mathbf{w}/\|\mathbf{w}\|, \mathbf{x}'_1 - \mathbf{x}'_2 \rangle = 2/\|\mathbf{w}\|$ *for two training instances* $(\mathbf{x}'_1, +1)$ *and* $(\mathbf{x}'_2, -1)$ *with* $\langle \mathbf{w}, \mathbf{x}'_1 \rangle + b = +1$ *and* $\langle \mathbf{w}, \mathbf{x}'_2 \rangle + b = -1$, *see Figure (b) [124].*

3.1.2 Large Margin Separation

Given a training set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathbb{R}^d \times \{-1, +1\}$ with patterns $\mathbf{x}'_i \in \mathbb{R}^d$ and associated class labels $y'_i \in \{-1, +1\}$, a pair $(\mathbf{x}'_i, y'_i)$ is called *negative training instance* if $y'_i = -1$ and *positive training instance* otherwise.[1] For any hyperplane $h(\mathbf{w}, b)$ in $\mathbb{R}^d$, the corresponding *decision function* $f_{(\mathbf{w},b)} : \mathbb{R}^d \to \{-1, +1\}$ is defined as

$$f_{(\mathbf{w},b)}(\mathbf{x}) := \operatorname{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b) \tag{3.3}$$

where $\operatorname{sgn}(t) = 1$ for $t \geq 0$ and $\operatorname{sgn}(t) = -1$ otherwise. We say that the training set T_L is *linearly separable* if there exists a hyperplane $h(\mathbf{w}, b)$ so that $f_{(\mathbf{w},b)}(\mathbf{x}'_i) = y'_i$ for $i = 1, \ldots, l$. Note that, without loss of generality, one can assume that the separating hyperplane $h(\mathbf{w}, b)$ does not contain any of the points $\mathbf{x}'_1, \ldots, \mathbf{x}'_l$.

Separable Case

The *geometrical margin* [124] of a pair $(\mathbf{x}, y) \in \mathbb{R}^d \times \{-1, +1\}$ with respect to a given hyperplane $h(\mathbf{w}, b) \subset \mathbb{R}^d$ is defined as

$$\rho_{(\mathbf{w},b)}(\mathbf{x}, y) := y \frac{(\langle \mathbf{w}, \mathbf{x} \rangle - (-b))}{\|\mathbf{w}\|} \tag{3.4}$$

[1] We assume the training set T_L to contain at least one positive and one negative training instance. Further, we do not consider multisets of patterns, i.e., we assume that $\mathbf{x}'_i \neq \mathbf{x}'_j$ for $i \neq j$.

and the one of the labeled training set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathbb{R}^d \times \{-1, +1\}$ with respect to $h(\mathbf{w}, b) \subset \mathbb{R}^d$ is defined as

$$\rho_{(\mathbf{w},b)}(T_L) := \min_{i=1,\ldots,l} \rho_{(\mathbf{w},b)}(\mathbf{x}'_i, y'_i), \tag{3.5}$$

see Figure 3.1 (a) for an illustration of both definitions. For a given training set T_L, the goal of a support vector machine is to find a hyperplane $h(\mathbf{w}, b)$ with $f_{(\mathbf{w},b)}(\mathbf{x}'_i) = y'_i$ for all $i = 1, \ldots, l$ that has maximal geometrical margin $\rho_{(\mathbf{w},b)}(T_L)$. Note that the existence of such a hyperplane is guaranteed (by definition) for linearly separable training sets. However, there exists an infinite number of hyperplanes as solutions to this optimization problem.[2] One therefore imposes the following additional constraint:

$$\min_{i=1,\ldots,l} |\langle \mathbf{w}, \mathbf{x}'_i \rangle + b| = 1 \tag{3.6}$$

In this case, the geometrical margin of the training set with respect to the optimal hyperplane is given by $1/\|\mathbf{w}\|$, see Figure 3.1 (b) for an explanation. Thus, both the goal to maximize the geometrical margin with respect to the training set T_L as well as the additional constraint described above lead to the optimization task to be solved for *hard-margin support vector machines* [1, 124]:

$$\begin{aligned} \underset{\mathbf{w}\in\mathbb{R}^d,\, b\in\mathbb{R}}{\text{minimize}}\ & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t. } & y'_i(\langle \mathbf{w}, \mathbf{x}'_i \rangle + b) \geq 1 \end{aligned} \tag{3.7}$$

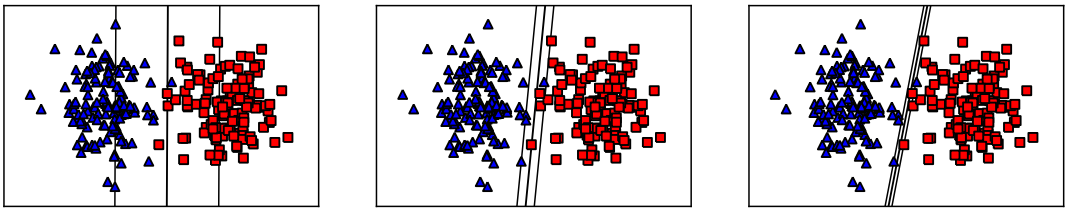
Note that one minimizes $\frac{1}{2}\|\mathbf{w}\|^2$ instead of maximizing $\frac{1}{\|\mathbf{w}\|}$ due to computational considerations (since it leads to a convex optimization problem [20], see below).

Non-Separable Case

The concept of hard-margin support vector machines is based on the assumption that the training set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathbb{R}^d \times \{-1, +1\}$ is linearly separable. In real-world scenarios, however, this assumption is rarely fulfilled. To cope with non-separable training sets, one relaxes the constraints by allowing patterns to lie within the margin or even on the wrong side of the hyperplane. More precisely, so-called *slack variables* $\xi'_1, \ldots, \xi'_l \in \mathbb{R}$ are introduced to impose

$$y'_i(\langle \mathbf{w}, \mathbf{x}'_i \rangle + b) \geq 1 - \xi'_i,\ \xi'_i \geq 0 \tag{3.8}$$

[2] Given $h(\mathbf{w}, b)$ that separates T_L, then $h(s\mathbf{w}, sb)$ separates T_L as well for any $s > 0$.



(a) $C' = 0.001$ (b) $C' = 1$ (c) $C' = 1000$

FIGURE 3.2. *The blue triangles depict negative training instances and the red squares positive ones. The decision hyperplane and both (margin) boundaries are indicated by black lines. For small C', the goal of maximizing the margin has priority whereas a large value leads to a heavy penalization of patterns lying within the margin (or even on its wrong side).*

for $i = 1, \ldots, l$. This leads to the following optimization task [18, 124, 135]:

$$\begin{aligned} \underset{\mathbf{w}\in\mathbb{R}^d,\, b\in\mathbb{R},\, \boldsymbol{\xi}'\in\mathbb{R}^l}{\text{minimize}} \; & \frac{1}{2}\|\mathbf{w}\|^2 + C' \sum_{i=1}^{l} \xi_i' \\ \text{s.t. } & y_i'(\langle \mathbf{w}, \mathbf{x}_i' \rangle + b) \geq 1 - \xi_i',\ \xi_i' \geq 0 \end{aligned} \tag{3.9}$$

Here, the first term of the objective corresponds to maximizing the margin and the second term captures the violation of the (previous) strict constraints. The parameter $C' \in \mathbb{R}^+$ determines the trade-off between both aims, see Figure 3.2. This modified optimization task is called *soft-margin support vector machine* [124] and can also be written in an alternative manner: Both types of constraints can be combined by imposing

$$\xi_i' \geq \mathcal{L}(y_i', \langle \mathbf{w}, \mathbf{x}_i' \rangle + b) \tag{3.10}$$

for $i = 1, \ldots, l$, where $\mathcal{L}(y, t) = \max(0, 1 - yt)$ is the *hinge loss* [135], see Figure 3.3 (a). Since the objective (3.9) is minimal if equality holds for (3.10), one obtains

$$\underset{\mathbf{w}\in\mathbb{R}^d,\, b\in\mathbb{R}}{\text{minimize}} \; \frac{1}{l} \sum_{i=1}^{l} \max(0, 1 - y_i'(\langle \mathbf{w}, \mathbf{x}_i' \rangle + b)) + \lambda \|\mathbf{w}\|^2 \tag{3.11}$$

with $\lambda = \frac{1}{2lC'} \in \mathbb{R}^+$ as equivalent unconstrained optimization task.

3.2 Non-Linear Support Vector Machines

We will now derive a more general definition of support vector machines that is based on so-called *kernel functions*. This type of function depicts, among other things, one of the main reasons for the popularity of support vector machines. In the following, we will assume that the input space $\mathcal{X}$ is an arbitrary set and that the output space $\mathcal{Y}$ is either

given by $\mathcal{Y} = \mathbb{R}$ or by $\mathcal{Y} = \{-1, +1\}$, depending on the context.

3.2.1 Kernels and Feature Spaces

A *kernel function* can be seen as a similarity measure and gives rise to an associated *feature space*. In this section, we will provide the basics related to both of these concepts.

Kernel Functions

Given a non-empty arbitrary set $\mathcal{X}$, a real-valued function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called *kernel* or *kernel function* [124, 135].[3] Applied to a particular subset of $m \in \mathbb{N}$ patterns $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathcal{X}$, a kernel gives rise to a *kernel matrix* (or *Gram matrix*) $\mathbf{K} \in \mathbb{R}^{m \times m}$ with entries $[\mathbf{K}]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. A kernel k is called *positive definite kernel* [125, 135] if the kernel matrix induced by any sequence $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{X}$ with $m \in \mathbb{N}$ is positive definite.[4] In case the associated kernel matrix is strictly positive definite, then the kernel is called a *strictly positive definite kernel*. In the following, we will focus on positive definite kernels and will therefore omit the term *(strictly) positive definite*. For the special case in which the d-dimensional Euclidean space is given as input space, i. e., for $\mathcal{X} = \mathbb{R}^d$, well-known kernel functions $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ are

- the *linear kernel* with $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$,
- *polynomial kernels* with $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle^p$ and $p \in \mathbb{N}$,
- and the *radial basis function* (RBF) *kernel* with $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2})$ and associated *kernel width* $\sigma \in \mathbb{R}^+$.

It should be pointed out that a variety of other kernel functions can be found in the literature which are not restricted to operate on $\mathbb{R}^d$. For instance, kernels can be defined on sets of graphs or strings, see, e. g., Borgwardt [16] and Hofmann *et al.* [76]. This is one of the advantages of kernel functions since the corresponding methods can be applied to arbitrary sets of objects. Another positive aspect of kernel functions is the fact that their specific properties can lead to efficient (optimization) schemes for the different learning tasks. For instance, the positive definiteness of the associated kernel matrix $\mathbf{K}$ often yields convex optimization problems [20] and invertible matrices.[5]

[3] The latter definition can also be extended to complex-valued functions but we will focus on real-valued ones in this work.

[4] A symmetric matrix $\mathbf{G} \in \mathbb{R}^{m \times m}$ is called *positive definite*, $\mathbf{z}^{\mathrm{T}}\mathbf{G}\mathbf{z} \geq 0$ holds for all $\mathbf{z} \in \mathbb{R}^m$. If equality is obtained only for $\mathbf{z} = \mathbf{0}$, then the matrix is called *strictly positive definite* [68, 77].

[5] The matrix $\mathbf{K} + \gamma\mathbf{I}$ with $[\mathbf{I}]_{i,j} = \delta_{ij}$ is strictly positive definite for $\gamma \in \mathbb{R}^+$ and, thus, invertible [68, 77].

Feature Spaces

We can now provide the definition of the *feature space* associated with a kernel function: Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel and let $\mathbb{R}^{\mathcal{X}} = \{f : \mathcal{X} \to \mathbb{R}\}$ denote the space of all functions mapping elements from $\mathcal{X}$ into $\mathbb{R}$. Then the *feature map* $\Phi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}}$ [124, 125] is given by

$$\Phi(\mathbf{x}) := k(\cdot, \mathbf{x}). \tag{3.12}$$

As pointed out by Schölkopf *et al.* [124, 125], the image $\Phi(\mathbf{x})$ of a particular pattern $\mathbf{x}$ can be seen as a function that measures the similarity of $\mathbf{x}$ with respect to all other patterns in $\mathcal{X}$. The feature map can be used to obtain a vector space $\mathcal{H}_k \subset \mathbb{R}^{\mathcal{X}}$ with elements of the form

$$f(\cdot) = \sum_{i=1}^{m} \alpha_i k(\cdot, \mathbf{z}_i) \text{ and } g(\cdot) = \sum_{j=1}^{\hat{m}} \beta_j k(\cdot, \hat{\mathbf{z}}_j) \tag{3.13}$$

with $m \in \mathbb{N}, \hat{m} \in \mathbb{N}$, $\alpha_i \in \mathbb{R}, \beta_j \in \mathbb{R}$, and arbitrary $\mathbf{z}_i \in \mathcal{X}, \hat{\mathbf{z}}_j \in \mathcal{X}$. By defining

$$\langle f, g \rangle_{\mathcal{H}_k} := \sum_{i=1}^{m} \sum_{j=1}^{\hat{m}} \alpha_i \beta_j k(\mathbf{z}_i, \hat{\mathbf{z}}_j), \tag{3.14}$$

one obtains a dot product (space) with associated norm $\|f\|_{\mathcal{H}_k} = \sqrt{\langle f, f \rangle_{\mathcal{H}_k}}$ [124]. Adding the limits of sequences that are convergent with respect to $\|\cdot\|_{\mathcal{H}_k}$ leads to a Hilbert space $\mathcal{H}_k$, called *reproducing kernel Hilbert space* (RKHS) or *feature space* of the associated kernel k. We omit giving further details and refer to Schölkopf *et al.* [124] and Steinwart and Christmann [135] for the corresponding definitions and derivations.

3.2.2 Generalized Representer Theorem

As shown in Chapter 2, the concept of regularized risk minimization aims at finding an optimal prediction function $f \in \mathcal{H}$ in a hypothesis space $\mathcal{H} \subseteq \{g : \mathcal{X} \to \mathcal{Y}\}$ with respect to

$$R_{reg}[f] = R_{emp}[f] + \lambda \Omega(f), \tag{3.15}$$

where the parameter $\lambda \in \mathbb{R}^+$ determines the trade-off between data fit and complexity of the function (which is in turn measured via the regularizer $\Omega(f)$). In the following, we will use the feature space $\mathcal{H}_k$ depicted above as hypothesis space and the associated squared norm as regularizer, i. e., $\mathcal{H} = \mathcal{H}_k$ and $\Omega(f) = \|f\|^2_{\mathcal{H}_k}$ for $f \in \mathcal{H}_k$. This leads to problems of the form

$$\underset{f \in \mathcal{H}_k}{\text{minimize}} \sum_{i=1}^{l} \mathcal{L}(y'_i, f(\mathbf{x}'_i)) + \lambda \|f\|^2_{\mathcal{H}_k}, \tag{3.16}$$

for a given loss function $\mathcal{L} : \mathcal{Y} \times \mathbb{R} \to [0, \infty)$ and training set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \mathcal{Y}$ with patterns $\mathbf{x}'_i \in \mathcal{X}$ and real-valued labels $y'_i \in \mathcal{Y} \subseteq \mathbb{R}$. This setup leads to the question of how the above optimization task can be addressed efficiently. The following theorem plays an important role in this context:

Fact 3.1 *([124, p. 90]) Given a non-empty set $\mathcal{X}$, a training set $T = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \mathbb{R}$, a positive definite kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, a strictly monotonic increasing function $g : [0, \infty[\to \mathbb{R}$, and an arbitrary cost function $c : (\mathcal{X} \times \mathbb{R}^2)^l \to \mathbb{R} \cup \{\infty\}$. Then, any $f \in \mathcal{H}_k$ minimizing*

$$c((\mathbf{x}'_1, y'_1, f(\mathbf{x}'_1)), \ldots, (\mathbf{x}'_l, y'_l, f(\mathbf{x}'_l))) + g(\|f\|_{\mathcal{H}_k}) \tag{3.17}$$

admits a representation of the form

$$f(\cdot) = \sum_{i=1}^{l} \alpha_i k(\cdot, \mathbf{x}'_i) \tag{3.18}$$

with appropriate coefficients $\alpha_1, \ldots, \alpha_l \in \mathbb{R}$.

This fact, known as the *representer theorem*, essentially shows that it is always possible to express an optimal solution for the task (3.17) based only on the patterns given in the training set T_L, although the hypothesis space $\mathcal{H}_k$ contains a much richer class of functions (namely linear combinations of kernel functions centered on *arbitrary* patterns in $\mathcal{X}$) [124]. This paves the way for efficient optimization frameworks that address such problems.

There is a direct extension of the above fact, the so-called *semiparametric representer theorem* [124], which is needed for capturing the offset term $b \in \mathbb{R}$ present in the standard formulation of support vector machines:

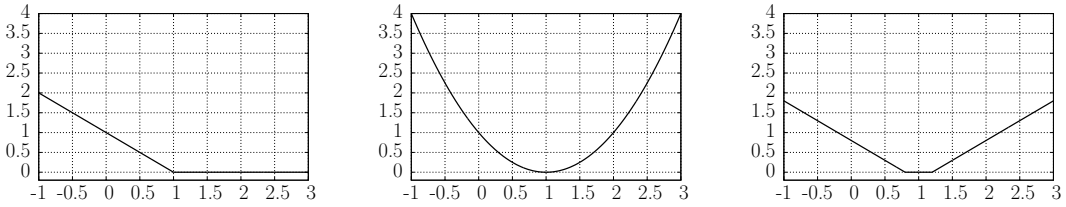
Fact 3.2 *([124, p. 91]) In addition to the assumptions of Fact 3.1, let us assume that we are given a set $\{\psi_1, \ldots, \psi_M\} \subset \mathbb{R}^{\mathcal{X}}$ of $M \in \mathbb{N}$ real-valued functions so that the matrix $\mathbf{M} \in \mathbb{R}^{l \times M}$ with $[\mathbf{M}]_{i,p} = \psi_p(\mathbf{x}'_i)$ has rank M. Then, any $\tilde{f} = f + h$ with $f \in \mathcal{H}_k$ and $h \in span\{\psi_1, \ldots, \psi_M\}$ which minimizes*

$$c((\mathbf{x}'_1, y'_1, \tilde{f}(\mathbf{x}'_1)), \ldots, (\mathbf{x}'_l, y'_l, \tilde{f}(\mathbf{x}'_l))) + g(\|f\|_{\mathcal{H}_k}) \tag{3.19}$$

admits a representation of the form

$$\tilde{f}(\cdot) = \sum_{i=1}^{l} \alpha_i k(\cdot, \mathbf{x}'_i) + \sum_{p=1}^{M} \beta_p \psi_p(\cdot) \tag{3.20}$$

with additional coefficients $\beta_1, \ldots, \beta_M \in \mathbb{R}$.



(a) Hinge Loss (b) Square Loss (c) ε-insensitive Loss

FIGURE 3.3. *A well-known loss function used in classification settings is the hinge loss* $\mathcal{L}(y,t) = \max(0, 1-yt)$ *that leads to standard support vector machines, see Figure (a). The square loss* $\mathcal{L}(y,t) = (y-t)^2$*, depicted in Figure (b), is a common loss function for regression problems. However, it is also applied in the context of classification scenarios, leading to least-squares support vector machines. The so-called* ε*-insensitive loss* $\mathcal{L}(y,t) = \max(0, |y-t| - \varepsilon)$*, shown in Figure (c) with* $\varepsilon = 0.2$*, yields the concept of support vector regression.*

3.2.3 Support Vector Classification

The above fact allows to investigate slightly modified optimization tasks of the form

$$\inf_{f \in \mathcal{H}_k, b \in \mathbb{R}} \frac{1}{l} \sum_{i=1}^{l} \mathcal{L}\big(y_i', f(\mathbf{x}_i') + b\big) + \lambda \|f\|_{\mathcal{H}_k}^2 \tag{3.21}$$

with $\lambda \in \mathbb{R}^+$. Depending on the learning task at hand, different loss functions can be plugged into the above task, which leads to various learning schemes. In the context of binary classification with a labeled training set $T_L = \{(\mathbf{x}_1', y_1'), \ldots, (\mathbf{x}_l', y_l')\} \subset \mathcal{X} \times \{-1, +1\}$, the hinge loss $\mathcal{L}(y,t) = \max(0, 1-yt)$ shown in Figure 3.3 (a) is a commonly considered loss function and leads to

$$\inf_{f \in \mathcal{H}_k, b \in \mathbb{R}} \frac{1}{l} \sum_{i=1}^{l} \max\big(0, 1 - y_i'(f(\mathbf{x}_i') + b)\big) + \lambda \|f\|_{\mathcal{H}_k}^2. \tag{3.22}$$

In this case, due to Fact 3.2 with strictly monotonic increasing function $g(x) = \lambda x^2$, real-valued function $\psi_1(\mathbf{x}) = 1$, and cost function

$$c((\mathbf{x}_1', y_1', \tilde{f}(\mathbf{x}_1')), \ldots, (\mathbf{x}_l', y_l', \tilde{f}(\mathbf{x}_l'))) = \frac{1}{l} \sum_{i=1}^{l} \max(0, 1 - y_i' \tilde{f}(\mathbf{x}_i')) \tag{3.23}$$

for $\tilde{f} = f + h$ with $f \in \mathcal{H}_k$ and $h \in span\{\psi_1\}$, any $f^*(\cdot) + b^*$ minimizing (3.22) admits a representation of the form

$$f^*(\cdot) + b^* = \sum_{j=1}^{l} \alpha_j k(\cdot, \mathbf{x}_j') + \beta \tag{3.24}$$

with coefficients $\alpha_1, \ldots, \alpha_l, \beta \in \mathbb{R}$. Following Steinwart and Christmann [135], one can define

$$\mathbf{w}(\boldsymbol{\alpha}) := \sum_{j=1}^{l} \alpha_j k(\cdot, \mathbf{x}'_j) = \sum_{j=1}^{l} \alpha_j \Phi(\mathbf{x}'_j) \tag{3.25}$$

and can thus rewrite the task (3.22) as

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^l,\, b\in\mathbb{R}}{\text{minimize}} \; \frac{1}{l} \sum_{i=1}^{l} \max\left(0, 1 - y'_i(\langle \mathbf{w}(\boldsymbol{\alpha}), \Phi(\mathbf{x}'_i)\rangle_{\mathcal{H}_k} + b)\right) + \lambda \|\mathbf{w}(\boldsymbol{\alpha})\|^2_{\mathcal{H}_k}. \tag{3.26}$$

The above problem depicts the generalization of linear support vector machines to the general case with arbitrary kernel functions. Note the resemblance of the above formulation with the task (3.11) derived for the linear case. However, instead of having an explicit hyperplane parameter $\mathbf{w} \in \mathbb{R}^d$, we are now given functions $\mathbf{w}(\boldsymbol{\alpha})$ in the feature space $\mathcal{H}_k$ that are parameterized by the finite dimensional vector $\boldsymbol{\alpha} \in \mathbb{R}^l$. The linear case with $\mathcal{X} \subseteq \mathbb{R}^d$ and $k(\mathbf{x}'_i, \mathbf{x}'_j) = \langle \mathbf{x}'_i, \mathbf{x}'_j \rangle$ is, in turn, a special case of the above formulation since one has

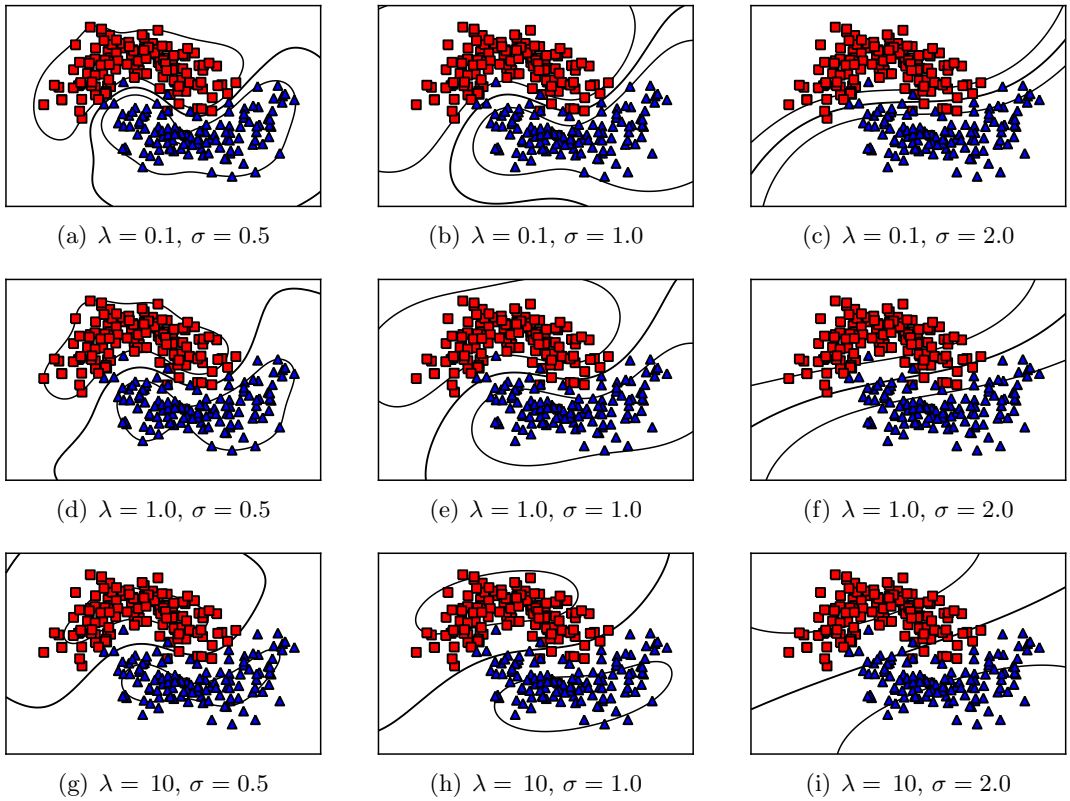
$$\mathbf{w}(\boldsymbol{\alpha}) = \sum_{j=1}^{l} \alpha_j k(\cdot, \mathbf{x}'_j) = \sum_{j=1}^{l} \alpha_j \langle \cdot, \mathbf{x}'_j \rangle = \langle \cdot, \sum_{j=1}^{l} \alpha_j \mathbf{x}'_j \rangle = \langle \cdot, \mathbf{w} \rangle \tag{3.27}$$

with $\mathbf{w} = \sum_{j=1}^{l} \alpha_j \mathbf{x}'_j \in \mathbb{R}^d$. Thus, one can rewrite the general optimization task (3.26) to obtain the former problem (3.11). The feature space $\mathcal{H}_k$ is based on the considered kernel function k. Depending on the given input space $\mathcal{X}$ and the particular training set T_L, the kernel function allows to adapt to the task at hand.

A possible learning scenario is depicted in Figure 3.4: Here, the outcome of a support vector machine based on the RBF kernel for a two-dimensional toy example is shown given varying assignments for the model parameters. It can be seen that, with more regularization (larger λ), the optimal prediction function $f^*(\cdot) + b^*$ for the task (3.22) gets simpler. In addition, the kernel width σ has a similar effect, i. e., larger values lead to simpler models for this data set instance. It should be pointed out that there exists a huge amount of variants of standard support vector machines. For instance, considering the square loss for the task (3.21) leads to so-called *least squares support vector machines* [119, 136], see Figure 3.3 (b). For a detailed overview, we refer the reader to Steinwart and Christmann [135].

3.2.4 Support Vector Regression

We will briefly sketch how to apply the ideas depicted above for regression problems. For such settings, one is given a training set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \mathbb{R}$ with arbitrary real-valued labels. Using the *ε-insensitive loss* $\mathcal{L}(y, t) = \max(0, |y - t| - \varepsilon)$ for



(a) $\lambda = 0.1$, $\sigma = 0.5$ (b) $\lambda = 0.1$, $\sigma = 1.0$ (c) $\lambda = 0.1$, $\sigma = 2.0$

(d) $\lambda = 1.0$, $\sigma = 0.5$ (e) $\lambda = 1.0$, $\sigma = 1.0$ (f) $\lambda = 1.0$, $\sigma = 2.0$

(g) $\lambda = 10$, $\sigma = 0.5$ (h) $\lambda = 10$, $\sigma = 1.0$ (i) $\lambda = 10$, $\sigma = 2.0$

FIGURE 3.4. *Two-dimensional binary classification example addressed by a standard support vector machine with RBF kernel. The kernel width $\sigma \in \mathbb{R}^+$ is increased from left to right and the regularization parameter is increased from top to bottom. Again, the blue triangles depict negative training instances and the red squares positive ones. The contour lines $\{\mathbf{x} \in \mathcal{X} \mid f^*(\mathbf{x}) + b^* = 0\}$ as well as $\{\mathbf{x} \in \mathcal{X} \mid f^*(\mathbf{x}) + b^* = 1\}$ and $\{\mathbf{x} \in \mathcal{X} \mid f^*(\mathbf{x}) + b^* = -1\}$ are indicated by black lines. Note that these lines are more complex for small λ and σ whereas they become simpler for large λ and σ.*

$\varepsilon \in \mathbb{R}^+$ leads to

$$\inf_{f \in \mathcal{H}_k, b \in \mathbb{R}} \frac{1}{l} \sum_{i=1}^{l} \max\left(0, |y_i' - (f(\mathbf{x}_i') + b)| - \varepsilon\right) + \lambda \|f\|_{\mathcal{H}_k}^2. \tag{3.28}$$

Thus, the ε-insensitive loss function penalizes predictions which are more than ε away from the known labels in a linear manner, see Figure 3.3 (c). Exactly as for the classification case, one can apply Fact 3.2 to obtain a finite-dimensional (convex) optimization problem. For further details see Schölkopf and Smola [124].

3.3 Computational Considerations

In this section, we will sketch an efficient way of solving the optimization task induced by support vector classification with $\mathcal{Y} = \{-1, +1\}$. By making use of $\|\mathbf{w}(\boldsymbol{\alpha})\|_{\mathcal{H}_k}^2 = \boldsymbol{\alpha}^{\mathrm{T}} \mathbf{K} \boldsymbol{\alpha}$

and $\langle \mathbf{w}(\boldsymbol{\alpha}), \Phi(\mathbf{x}'_i) \rangle_{\mathcal{H}_k} = \sum_{j=1}^{l} \alpha_j k(\mathbf{x}'_i, \mathbf{x}'_j)$, one can rewrite the problem (3.26) as

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^l,\, b \in \mathbb{R}}{\text{minimize}} \; \frac{1}{l} \sum_{i=1}^{l} \max\left(0, 1 - y'_i \left(\sum_{j=1}^{l} \alpha_j k(\mathbf{x}'_i, \mathbf{x}'_j) + b\right)\right) + \lambda \boldsymbol{\alpha}^{\mathrm{T}} \mathbf{K} \boldsymbol{\alpha}. \tag{3.29}$$

This is a convex function on the convex set $\mathbb{R}^{l+1}$: The first term is convex since the sum of convex functions (defined on the same convex set) is convex. Note that each summand is convex due to the convexity of the hinge loss and the fact that affine maps preserve convexity [20, p. 79 ff.]. Furthermore, the second term is convex since the kernel matrix is positive definite [20]. Thus, in principle, one can resort to simple derivative-free optimization methods to approach this task [108]. Another possible strategy is to resort to differentiable surrogates for the hinge loss so that efficient gradient based solvers can be used, see Chapelle [31]. The most common approach, however, is to reformulate the task (3.29) as quadratic programming (QP) [20] problems, which "can often be solved faster than general convex problems" [135, p. 414]. We will now provide the details.

3.3.1 Primal and Dual Problems

Due to the convexity of the above problem, two related quadratic programs are considered in the literature, the *primal problem* and the *dual problem* associated with it [20].

Primal Problem

Exactly as for the linear case, one can reformulate the optimization task (3.29) as

$$\begin{aligned} \underset{\boldsymbol{\alpha} \in \mathbb{R}^l,\, b \in \mathbb{R},\, \boldsymbol{\xi}' \in \mathbb{R}^l}{\text{minimize}} \; & \frac{1}{2} \boldsymbol{\alpha}^{\mathrm{T}} \mathbf{K} \boldsymbol{\alpha} + C' \sum_{i=1}^{l} \xi'_i \\ \text{s.t. } & y'_i \left(\sum_{j=1}^{l} \alpha_j k(\mathbf{x}'_i, \mathbf{x}'_j) + b\right) \geq 1 - \xi'_i, \xi'_i \geq 0 \end{aligned} \tag{3.30}$$

with $C' = \frac{1}{2l\lambda} \in \mathbb{R}^+$. Again, due to the positive definiteness of the kernel matrix, the first term in the objective is convex. Further, the constraints $\sum_{j=1}^{l} \alpha_j k(\mathbf{x}'_i, \mathbf{x}'_j)$ are linear with respect to the optimization variables. Therefore, one ends up with a (convex) quadratic program, called the *primal optimization problem* [124, 135].

Dual Problem

The *Lagrange dual function* [20, p. 216] $G : \mathbb{R}^l \times \mathbb{R}^l \to \mathbb{R}$ is defined as

$$G(\boldsymbol{\beta}, \boldsymbol{\eta}) := \inf_{\boldsymbol{\alpha} \in \mathbb{R}^l,\, b \in \mathbb{R},\, \boldsymbol{\xi}' \in \mathbb{R}^l} L(\boldsymbol{\alpha}, b, \boldsymbol{\xi}', \boldsymbol{\beta}, \boldsymbol{\eta}) \tag{3.31}$$

with associated *Lagrangian* $L : \mathbb{R}^{2l+1} \times \mathbb{R}^l \times \mathbb{R}^l \to \mathbb{R}$ given by

$$L(\boldsymbol{\alpha}, b, \boldsymbol{\xi}', \boldsymbol{\beta}, \boldsymbol{\eta}) := \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{K}\boldsymbol{\alpha} + C' \sum_{i=1}^{l} \xi_i' + \sum_{i=1}^{l} \beta_i (1 - y_i' (\sum_{j=1}^{l} \alpha_j k(\mathbf{x}_i', \mathbf{x}_j') + b) - \xi_i') - \sum_{i=1}^{l} \eta_i \xi_i'$$

and *dual variables* $\boldsymbol{\beta} \in \mathbb{R}^l$ and $\boldsymbol{\eta} \in \mathbb{R}^l$. The coefficients $\beta_1, \ldots, \beta_l, \eta_1, \ldots, \eta_l$ are called *Lagrange multipliers* [20]. The *Lagrange dual problem* [20, p. 223] is given by

$$\begin{aligned} &\underset{\boldsymbol{\beta} \in \mathbb{R}^l, \boldsymbol{\eta} \in \mathbb{R}^l}{\text{maximize}} \; G(\boldsymbol{\beta}, \boldsymbol{\eta}) \\ &\text{s.t. } \beta_i, \eta_i \geq 0 \end{aligned} \tag{3.32}$$

and depicts a convex optimization task (with respect to the dual variables). Further, it yields the best lower bound on the optimal objective value of the original primal optimization task that can be obtained via the Lagrange dual function [20, p. 225]. Since the primal optimization task (3.30) is convex (with differentiable objective and constraint functions), the so-called *Karush-Kuhn-Tucker* (KKT) conditions are necessary and sufficient for optimality [20, p. 244] and imply (among other things) that the derivatives with respect to the primal variables must vanish, i. e.,

$$\nabla_b L(\boldsymbol{\alpha}, b, \boldsymbol{\xi}', \boldsymbol{\beta}, \boldsymbol{\eta}) = -\sum_{i=1}^{l} \beta_i y_i' \overset{!}{=} 0, \tag{3.33}$$

$$\nabla_{\xi_p'} L(\boldsymbol{\alpha}, b, \boldsymbol{\xi}', \boldsymbol{\beta}, \boldsymbol{\eta}) = C' - \beta_p - \eta_p \overset{!}{=} 0 \tag{3.34}$$

for $p \in \{1, \ldots, l\}$. Thus, we have $C' \sum_{i=1}^{l} \xi_i' - \sum_{i=1}^{l} \beta_i \xi_i' - \sum_{i=1}^{l} \eta_i \xi_i' = 0$ and the Lagrangian can be simplified to:

$$\begin{aligned} L(\boldsymbol{\alpha}, \boldsymbol{\beta}) &= \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{K}\boldsymbol{\alpha} - \sum_{i=1}^{l} \beta_i y_i' \sum_{j=1}^{l} \alpha_j k(\mathbf{x}_i', \mathbf{x}_j') + \sum_{i=1}^{l} \beta_i \\ &= \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{K}\boldsymbol{\alpha} - \sum_{j=1}^{l} \alpha_j \left(\sum_{i=1}^{l} \beta_i y_i' k(\mathbf{x}_i', \mathbf{x}_j') \right) + \sum_{i=1}^{l} \beta_i \end{aligned} \tag{3.35}$$

By further imposing

$$\nabla_{\boldsymbol{\alpha}} L(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{K}(\boldsymbol{\alpha} - \mathbf{Y}'\boldsymbol{\beta}) \overset{!}{=} \mathbf{0} \tag{3.36}$$

with $\mathbf{Y}' = \operatorname{diag}(y_1', \ldots, y_l')$ and by assuming the kernel matrix $\mathbf{K}$ to be strictly positive definite (and, thus, invertible), one obtains

$$\boldsymbol{\alpha} = \mathbf{Y}'\boldsymbol{\beta}. \tag{3.37}$$

Note that, in case $\mathbf{K}$ is only positive definite (and therefore not invertible), $\boldsymbol{\alpha} = \mathbf{Y}'\boldsymbol{\beta}$ still depicts one of the possible solutions [120]. Thus, since we have $\alpha_i = \beta_i y_i'$, one can eliminate the primal variables in (3.35) and rewrite the Lagrange dual problem (3.32) as

$$\begin{aligned} \underset{\boldsymbol{\beta}\in[0,C']^l}{\text{maximize}} \; & \sum_{i=1}^{l} \beta_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \beta_i\beta_j y_i' y_j' k(\mathbf{x}_i', \mathbf{x}_j') \\ \text{s.t.} \; & \sum_{i=1}^{l} \beta_i y_i' = 0, \end{aligned} \tag{3.38}$$

where the box constraint results from Equation (3.34) and the fact that $\beta_i, \eta_i \geq 0$ for $i = 1, \ldots, l$. The above task is called the *dual optimization problem* [124, 135] in the context of support vector machines and can also be written as

$$\begin{aligned} \underset{\boldsymbol{\beta}\in[0,C']^l}{\text{maximize}} \; & \mathbf{1}^{\mathrm{T}}\boldsymbol{\beta} - \frac{1}{2}\boldsymbol{\beta}^{\mathrm{T}}(\mathbf{K} \odot \mathbf{y}'\mathbf{y}'^{\mathrm{T}})\boldsymbol{\beta} \\ \text{s.t.} \; & \boldsymbol{\beta}^{\mathrm{T}}\mathbf{y}' = 0, \end{aligned} \tag{3.39}$$

where $\odot$ denotes the element-wise product and where $\mathbf{1} \in \mathbb{R}^l$ is the vector of ones. Therefore, the dual optimization problem is also a quadratic program which, however, exhibits a simpler structure compared to the primal one.[6] The final classifier is then given by

$$g(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{l} \beta_i^* y_i' k(\cdot, \mathbf{x}_i') + b^*\right) \tag{3.40}$$

for optimal $\boldsymbol{\beta}^* \in \mathbb{R}^l$ and $b^* \in \mathbb{R}$. The optimal offset term b^* can be obtained via the KKT conditions that additionally imply

$$\beta_i^*(y_i'(\sum_{j=1}^{l} \beta_j^* y_j' k(\mathbf{x}_i', \mathbf{x}_j') + b^*) - 1 + \xi_i') \stackrel{!}{=} 0 \tag{3.41}$$

for $i = 1, \ldots, l$ [20, p. 244]. Thus, given an arbitrary (strictly) positive Lagrange multiplier $\beta_i^* \in \mathbb{R}^+$, one can compute the optional offset parameter b^*. The patterns $\mathbf{x}_i'$ with corresponding strictly positive multiplier $\beta_i^* \in \mathbb{R}^+$ are called *support vectors* and gave rise to the name of the overall concept.[7]

[6] The quadratic program is actually given by minimizing $F(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^{\mathrm{T}}(\mathbf{K} \odot \mathbf{y}'\mathbf{y}'^{\mathrm{T}})\boldsymbol{\beta} - \mathbf{1}^{\mathrm{T}}\boldsymbol{\beta}$ subject to the constraints. Note that the objective is convex since the element-wise product of two positive definite matrices is positive definite [77, p. 485].

[7] As pointed out by Rifkin *et al.* [118], such a pattern is guaranteed to exist under extremely mild assumptions. Further, necessary and sufficient conditions for the uniqueness of a computed solution are given by Burges and Crisp [25].

3.3.2 Mathematical Optimization

As pointed out above, one could address the convex optimization task (3.29) by simple derivative-free optimization schemes. Besides, both subgradient methods and differentiable surrogates for the hinge loss are used in the related literature to speed up such schemes [31, 126]. The quadratic programs derived above are, however, more amenable to mathematical optimization. In particular, it is possible to obtain solutions with guaranteed accuracy in polynomial time. The special structure of both quadratic programs has also led to sophisticated optimization frameworks that depict another reason for the success of support vector machines.

Solutions in Polynomial Time

No algorithms are known that can solve general quadratic programming problems exactly with *infinite precision*, as required in, e. g., the *real random access machine model* of computation [116]. Still, one can approximate the optimal solution up to machine precision and the number of needed arithmetic operations is cubic with respect to the involved variables and constraints.

Fact 3.3 *([12, 89]) A quadratic program with n variables and constraints can be solved up to machine precision in $\mathcal{O}(n^3 I_L)$ time, where I_L denotes the* input length, *i. e., the number of bits used to encode all the rational data of the problem.*[8]

This fact can be applied to both quadratic programming problems derived above, which yields the following upper runtime bounds:

Lemma 3.1 *Given a non-empty set $\mathcal{X}$, a labeled training set $T = \{(\mathbf{x}'_1, y'_1), \dots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \{-1, +1\}$, and a positive definite kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Assuming that the corresponding kernel matrix $\mathbf{K} \in \mathbb{R}^{l \times l}$ is precomputed, one can obtain an optimal solution up to machine precision for both the primal (3.30) and the dual (3.39) problem in $\mathcal{O}(l^3 I_L)$ arithmetic operations, where I_L denotes the input length of the problems.*

Proof: The primal problem (3.30) has $2l + 1$ variables and $2l$ constraints, whereas the dual problem (3.39) has l variables and $l + 1$ constraints. In both cases, we have at most $4l + 1$ variables and constraints. □

For common kernel functions like the linear or the RBF kernel with $\mathcal{X} \subseteq \mathbb{R}^d$, the computation of the kernel matrix can be performed in $\mathcal{O}(dl^2)$ time. Consequently, cubic time is needed in total for computing solutions up to machine precision in these cases.

[8] Personal communication with Michael J. Todd and Marshall W. Bern [12].

Sophisticated Optimization Frameworks

For both quadratic programming problems, there exists a variety of sophisticated optimization schemes. A recent runtime analysis for solving both quadratic programming problems has been given by Hush *et al.* [81] and List and Simon [99, 128]. Among other things, Hush *et al.* [81] show that $\mathcal{O}(l^2(c_k+1))$ arithmetic operations are needed to solve the primal problem up to machine precision, where l is the number of patterns and where c_k is an upper bound on the runtime needed to compute an entry of the kernel matrix $\mathbf{K} \in \mathbb{R}^{l \times l}$. Similar to the above bounds, the hidden constants are affected by the distribution of the input data (in addition to the desired accuracy): In short, Hush *et al.* [81] solve the dual optimization problem with accuracy

$$\varepsilon_D := \left(2\sqrt{2\mathcal{K}_l} + 8\sqrt{\lambda}\right)^{-2} \lambda \varepsilon_P^2 \tag{3.42}$$

to obtain a solution with accuracy $\varepsilon_P \in \mathbb{R}^+$ for the primal problem, where $\mathcal{K}_l$ defined as

$$\mathcal{K}_l := \max_{1 \le i \le l} k(\mathbf{x}'_i, \mathbf{x}'_i) \tag{3.43}$$

depends on the particular kernel k and training set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\}$. By using the linear kernel with $\mathcal{X} = \mathbb{R}^d$, for instance, the value $\mathcal{K}_l$ can become arbitrarily large for general training sets. For the RBF kernel, however, the term $\mathcal{K}_l$ is bounded by one for all possible training sets. It should be pointed out that special cases of the optimization tasks can sometimes be approached more efficiently. This is the case, for instance, given linear support vector machines, which can (essentially) be addressed in linear time [86]. However, special assumptions are made for the distribution of the input data for such settings. We omit providing a detailed discussion of the vast amount of possible optimization frameworks and refer the reader to Steinwart and Christmann [135, p. 420 ff.] and Hush *et al.* [81] for comprehensive overviews.

Offset Term b

The standard support vector machine task includes the offset term $b \in \mathbb{R}$. In contrast to the remaining optimization variables, this term is not regularized since it does not occur in the regularization term $\lambda \|f\|^2_{\mathcal{H}_k}$ of the task (3.21). From both the theoretical as well as the practical point of view, the additional term b does not yield any known advantages for kernel functions like the RBF kernel [120, 135]. For the sake of simplicity, it is therefore sometimes omitted. This yields problems of the form

$$\inf_{f \in \mathcal{H}_k} \; \frac{1}{l} \sum_{i=1}^{l} \mathcal{L}\big(y'_i, f(\mathbf{x}'_i)\big) + \lambda \|f\|^2_{\mathcal{H}_k} \tag{3.44}$$

in lieu of the task (3.21). However, for the linear case, the offset term makes a difference since it addresses translated data. In case such an offset effect is needed for a particular learning task, there exists an alternative way to reobtain this flexibility that is based on adding an extra dimension of ones to the input data. More precisely, by transforming each input pattern $\mathbf{x}'_i \in \mathbb{R}^d$ to $\hat{\mathbf{x}}'_i = ({\mathbf{x}'_i}^{\mathrm{T}}, 1)^{\mathrm{T}} \in \mathbb{R}^{d+1}$ [30, 120], one obtains

$$\begin{aligned} \underset{\hat{\mathbf{w}} \in \mathbb{R}^{d+1},\, \boldsymbol{\xi}' \in \mathbb{R}^l}{\text{minimize}} \; & \frac{1}{2}\|\hat{\mathbf{w}}\|^2 + C' \sum_{i=1}^{l} \xi'_i \\ \text{s.t. } & y'_i(\langle \hat{\mathbf{w}}, \hat{\mathbf{x}}'_i \rangle) \geq 1 - \xi'_i,\; \xi'_i \geq 0 \end{aligned} \tag{3.45}$$

instead of the task (3.9). Since $\langle \hat{\mathbf{w}}, \hat{\mathbf{x}}'_i \rangle = \langle \mathbf{w}, \mathbf{x}'_i \rangle + \hat{w}_{d+1}$, the variable $\hat{w}_{d+1}$ can also be seen as a regularized offset term (via $\|\hat{\mathbf{w}}\|^2$).[9]

3.4 The Hughes Effect Revisited

We will now briefly sketch the influence of high-dimensional feature spaces on the performance of support vector machines. For this purpose, we conduct a similar experiment as the one performed for the k-nearest neighbor classifier in Section 2.4.1.

3.4.1 Experimental Setup

Exactly as before, we consider the artificial data set composed of two Gaussian clusters generated by drawing $N/2$ patterns ($N = 500$) from each of two multivariate distributions $X_i \sim \mathcal{N}(\mathbf{m}_i, \mathbf{I})$ with $\mathbf{m}_1 = (-2.5, 0.0, \ldots, 0.0)^{\mathrm{T}} \in \mathbb{R}^d$ and $\mathbf{m}_2 = (+2.5, 0.0, \ldots, 0.0)^{\mathrm{T}} \in \mathbb{R}^d$. Again, the class label of a point corresponds to the distribution it was drawn from and half of the resulting data set is used as training and the other half as test set. To address the support vector machine optimization task, we resort to the `LIBSVM` implementation (with default parameter settings) provided by Chang *et al.* [29] and use a linear kernel. The tradeoff-parameter $C' \in \{2^{-10}, \ldots, 2^{10}\}$ for the support vector machines as well as the parameter $k \in \{1, \ldots, 10\}$ for the nearest neighbor model are tuned on the training set via 5-fold cross-validation. The conducted experiments remain the same, i. e., for the first experiment, we fix the number $l = 25$ of patterns in the data set and vary the input dimension d from 25 to 500, see Figure 3.5 (a). For the second experiment, we fix $d = 500$ and increase the amount of labeled patterns to 250, see Figure 3.5 (b). Again, for all data set instances, the average test error and the one standard deviation over 10 random partitions are reported.

[9] If less regularization is desired, one can add a large constant as additional dimension instead of one.

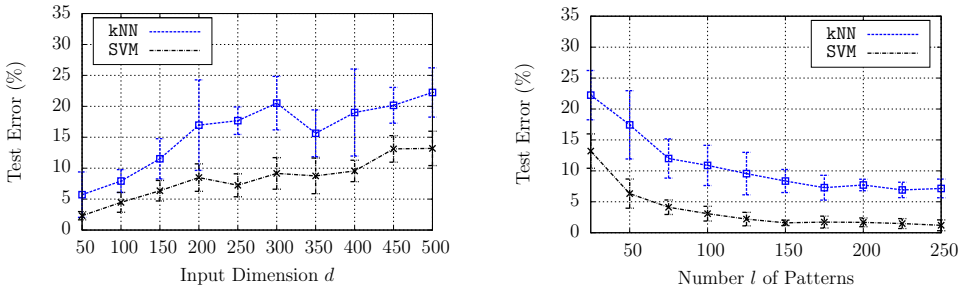


FIGURE 3.5. *Figure (a) shows how the test error decreases with increasing input dimension d (given $l = 25$ patterns) for both a k-nearest neighbor (`kNN`) and a linear support vector machine (`SVM`) model. Taking more labeled data into account (for $d = 500$) again leads to a better performance, see Figure (b).*

3.4.2 Results

As before, increasing the dimension of the data set leads to worse results whereas increasing the amount of labeled patterns for a fixed dimension leads to better ones. Clearly, for both experiments, the classification performance of the linear support vector machine model is superior to the one of the k-nearest neighbor model.[10] However, increasing the amount of labeled patterns (i. e., using sufficient labeled data) is not always possible in real-world settings. This gives rise to the question of how to incorporate unlabeled data into the learning process in order to improve the performance of a support vector machine. This will be subject of the following chapters.

3.5 Concluding Remarks

In this chapter, the concept of support vector machines was derived for linear and general kernel functions. For both settings, we have seen that the learning setup leads to convex optimization tasks. Aiming at efficient optimization schemes, these tasks are usually reformulated as quadratic programming problems. In particular, the resulting programs can be addressed by standard solvers that can yield optimal solutions up to machine precision in cubic time. Similar to the k-nearest neighbor classifier, the Hughes effect can be observed when training a support vector machine model. The following chapters will deal with the extension of support vector machines to semi- and unsupervised settings. As we will see, these extensions have the potential to incorporate unlabeled data in order to reveal more information about the structure of the data and, thus, depict an alternative way to cope with the curse of dimensionality.

[10] Of course, this particular task is well-suited for a linear support vector machine due to its linear structure.

Part II

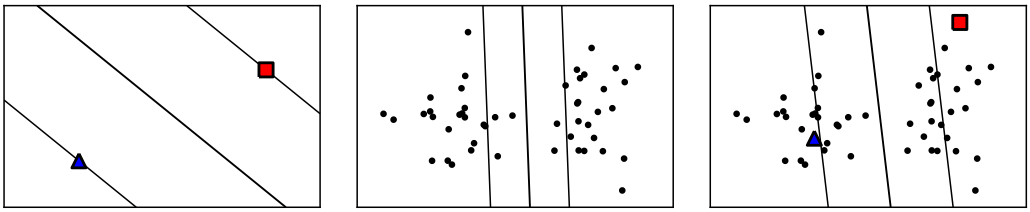
Semi- and Unsupervised Support Vector Machines

CHAPTER 4

Exact Solutions in Polynomial Time

The concept of support vector machines depicts one of the most promising learning schemes in the field of machine learning. However, as for all supervised approaches, sufficient labeled data are needed in general to obtain models of satisfying quality. As depicted in Chapter 2, semi- and unsupervised learning schemes can be used to reveal (more) information about the structure of the data in case (additional) unlabeled patterns are available. In this chapter, we will derive extensions of linear support vector machines to semi- and unsupervised learning settings. As we will see, both extensions lead to combinatorial optimization problems which are difficult to solve. The main result of this chapter is a polynomial-time approach that computes *exact* solutions (up to machine precisision) for these problems. The proposed framework is based on well-known concepts from the field of computational geometry and is among the first ones yielding theoretical upper runtime bounds. Another consequence of independent interest is the fact that the optimization scheme can be used to generate benchmark data sets for low-dimensional input spaces. These data sets depict excellent candidates for the mutual quality assessment of competing approaches for both learning tasks.

Outline. In Section 4.1, the formal definitions related to both extensions of support vector machines will be given. Afterwards, in Section 4.2, some concepts from the field of computational geometry will be introduced. These concepts will form the basis for the polynomial-time approach that is subject of Section 4.3. The experimental evaluation is given in Section 4.4 followed by concluding remarks provided in Section 4.5.



(a) Supervised (b) Unsupervised (c) Semi-Supervised

FIGURE 4.1. *In supervised learning scenarios, we are only given labeled patterns (red squares and blue triangles). Thus, given only a small amount of data, a support vector machine cannot yield a good classification model, see Figure (a). Both unsupervised and semi-supervised support vector machines try to incorporate unlabeled patterns (black dots), see Figures (b) and (c).*

4.1 Mathematical Framework

For the sake of exposition, we will briefly review the main ideas behind both extensions for the linear case: The goal of a support vector machine is to search for a hyperplane separating both classes such that the distance between the hyperplane and the patterns is large, see Figure 4.1 (a). This concept can also be considered in unsupervised learning scenarios. Here, the goal is to find the optimal partition of the data into two classes (given some constraints) such that a *subsequent* application of a support vector machine leads to the best possible result, see Figure 4.1 (b). Semi-supervised support vector machines can be seen as an intermediate approach between supervised and unsupervised support vector machines. Given both the labeled and the unlabeled part of the data, the aim of the corresponding learning task is to find a hyperplane which separates both classes well and, at the same time, passes through a low-density area induced by all patterns, see Figure 4.1 (c). Again, one searches for the optimal assignment of the unlabeled patterns to the two classes such that a subsequent application of a modified support vector machine yields the best overall result.[1]

4.1.1 Learning Tasks

We will now define both tasks from a mathematical point of view while focussing on the extensions of linear support vector machines. The more general cases based on arbitrary kernel functions will be defined in Chapter 5.

Unsupervised Support Vector Machines

The optimization task (3.9) induced by linear support vector machines can be extended to unsupervised learning scenarios in the following manner: Assume that we are given a

[1] Again, as mentioned in Chapter 1, there exists other ways to extend support vector machines to semi-supervised learning settings. In this work, we will focus on these combinatorial extensions.

training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$ of unlabeled patterns in the d-dimensional Euclidean space. Then, the goal of an *unsupervised support vector machine* can be formalized in terms of the following optimization task:

$$\begin{aligned} \underset{\substack{\mathbf{y}\in\{-1,+1\}^u,\\ \mathbf{w}\in\mathbb{R}^d,\, b\in\mathbb{R},\, \boldsymbol{\xi}\in\mathbb{R}^u}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{u}\xi_i \\ \text{s.t.} \quad & y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) \geq 1 - \xi_i,\ \xi_i \geq 0, \\ \text{and} \quad & \left|\frac{1}{u}\sum_{i=1}^{u}\max(0, y_i) - b_c\right| < \varepsilon, \end{aligned} \tag{4.1}$$

where $C \in \mathbb{R}^+$, $b_c \in [0, 1]$, and $\varepsilon \in \mathbb{R}^+$ are manually chosen constants. The second constraint is called the *balance constraint* and is necessary to avoid the two trivial (undesired) solutions which arise by assigning all patterns to only one class.[2]

Obviously, the difficulty of the above optimization task, also known as *maximum margin clustering* (MMC) [149] problem, consists in finding the optimal assignment for the partition vector $\mathbf{y} \in \{-1, +1\}^u$. Since we have both real-valued and integer optimization variables, we are dealing with a *mixed-integer programming* (MIP) [51] problem. This class of optimization tasks is, in general, NP-hard [51, 143] and therefore difficult to approach.

Semi-Supervised Support Vector Machines

Let us now consider semi-supervised scenarios where we are given both a set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathbb{R}^d \times \{-1, +1\}$ of labeled patterns and a set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$ of unlabeled ones. As sketched above, the idea of semi-supervised learning consists in taking advantage of both parts of the data. This leads to *semi-supervised support vector machines* (S^3VMs) [10, 83, 142], originally proposed by Vapnik and Sterin [142]:

$$\begin{aligned} \underset{\substack{\mathbf{y}\in\{-1,+1\}^u,\\ \mathbf{w}\in\mathbb{R}^d,\, b\in\mathbb{R},\, \boldsymbol{\xi}'\in\mathbb{R}^l,\, \boldsymbol{\xi}\in\mathbb{R}^u}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C'\sum_{i=1}^{l}\xi'_i + C\sum_{i=1}^{u}\xi_i \\ \text{s.t.} \quad & y'_i(\langle \mathbf{w}, \mathbf{x}'_i\rangle + b) \geq 1 - \xi'_i,\ \xi'_i \geq 0, \\ \text{and} \quad & y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) \geq 1 - \xi_i,\ \xi_i \geq 0, \\ \text{and} \quad & \left|\frac{1}{u}\sum_{i=1}^{u}\max(0, y_i) - b_c\right| < \varepsilon, \end{aligned} \tag{4.2}$$

where $C' \in \mathbb{R}^+$, $C \in \mathbb{R}^+$, $b_c \in [0, 1]$, and $\varepsilon \in \mathbb{R}^+$. Again, since the optimal partition vector $\mathbf{y} \in \{-1, +1\}^u$ for the unlabeled patterns is unknown, one ends up with a mixed-

[2] The parameters b_c and ε have to be set appropriately to avoid these undesired solutions.

integer programming problem which is difficult to solve [51, 143].

4.1.2 Related Work

In this chapter, we will focus on related approaches which aim at solving the linear extensions exactly, i. e., which yield solutions with guaranteed accuracy (up to machine precision) for the combinatorial tasks at hand. The more general cases along with the related literature will be discussed in Chapters 5 and 6.

Standard Solvers

Since the optimization tasks give rise to mixed-integer programming problems, standard solvers for this type of problems can be applied, see, e. g., Bennett and Demiriz [10]. These approaches can, in general, compute exact solutions up to machine precision if sufficient computational resources are available. However, mixed-integer programming problems are generally NP-hard [51, 143]. Thus, up to know, such general schemes do not yield practical upper runtime bounds.

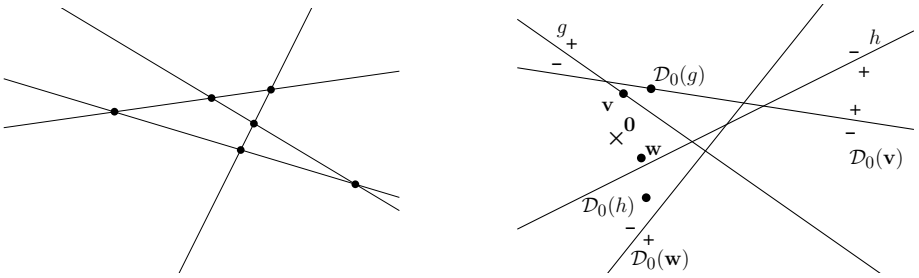
Combinatorial Search Schemes

A trivial approach to solving both combinatorial tasks consists in testing every feasible assignment of the partition vector $\mathbf{y} \in \{-1, +1\}^u$ and to report the best result found during the overall process. Of course, this brute-force approach is only feasible for a very small amount of unlabeled training patterns due to its exponential runtime.

Aiming at exact solutions, Chapelle *et al.* [35] proposed a branch and bound scheme for the semi-supervised case. Their approach recursively splits the search space into smaller subregions, which yields a tree with nodes corresponding to these subregions. The basic idea of such an approach is that one can safely discard large subtrees in the overall search tree based on lower and upper bounds for the objective function. The effectiveness of such a procedure is based on these bounds and on the way the pruning of subtrees takes place. In the worst case, the complete tree has to be examined, which leads to an exponential runtime.

A result competitive to the one provided in this chapter is given by Peng *et al.* [112]. However, their approach only aims at solving the unsupervised setting for the special case of a hard-margin support vector machine (without any slack variables), i. e., a more restrictive version is considered. In addition, a *general position* of the patterns is assumed, i. e., the authors require that "any d points in T_U will define precisely one hyperplane in $\mathbb{R}^d$" [112]. Given this assumption, they describe how to obtain an optimal solution for the hard-margin case in $\mathcal{O}(u^{d+2})$ time.[3]

[3] To the best of the author's understanding, there exists certain point configurations that are not covered



(a) Arrangement (b) Dual Mapping

FIGURE 4.2. *The Figure (a) shows an arrangement in the plane consisting of six vertices,* 16 *edges (eight of which are unbounded), and eleven cells (eight of which are unbounded). Figure (b) illustrates Fact 4.2, where the algebraic signs mark the open half-spaces of a hyperplane. Note, for instance, that* $\mathbf{w} \in h^-$ *and* $\mathcal{D}_0(h) \in \mathcal{D}_0(\mathbf{w})^-$.

4.2 Geometric Background

The optimization framework proposed in this chapter is based on *arrangements*, a well-known concept in the field of computational geometry [43, 45]. In a nutshell, an arrangement is the partition of the d-dimensional Euclidean space $\mathbb{R}^d$ into k-dimensional objects induced by a set of hyperplanes. The popularity of this concept in the field of computational geometry is mainly based on a connection between points and hyperplanes induced by a so-called *dual function*. We will now provide the details.

4.2.1 Arrangements and Duality

Following Edelsbrunner [45], we will introduce both arrangements and the concept of duality along with associated definitions and facts.

Arrangements

The *arrangement* $\mathcal{A}(\mathcal{G})$ of a finite set $\mathcal{G}$ of hyperplanes in $\mathbb{R}^d$ is defined as the partition of $\mathbb{R}^d$ into open convex cells of dimensions $k = 0, \ldots, d$ [45, 70]. An example of an arrangement in the Euclidean plane is shown in Figure 4.2 (a). Here, the plane is subdivided into vertices, edges, and regions. More formally, a *d-face* is defined as a maximal connected region in $\mathbb{R}^d$ that is not intersected by any hyperplane in $\mathcal{G}$. For $k = 0, \ldots, d-1$, a *k-face* is a maximal connected region in the intersection of a subset of hyperplanes, which is not intersected by any other hyperplane in $\mathcal{G}$ and whose dimension is k, i. e., which belongs to a k-flat but not to a $(k-1)$-flat (see Section 3.1 for the definition of a k-flat) [70].

For special values of k, special names are common in the literature. A 0-face is called *vertex*, a 1-face is called *edge*, a $(d-1)$-face is called *facet*, and a d-face is simply called

by the approach of Peng *et al.* [112].

cell. A k-face g and a $(k-1)$-face f are called *incident* if f is contained in the closure of g ($k = 1, \ldots, d$). In this case, g is the *superface* of f and f is a *subface* of g. An arrangement $\mathcal{A}(\mathcal{G})$ with $|\mathcal{G}| \geq d$ is called *simple*, if any d hyperplanes in $\mathcal{G}$ have a common unique intersection point and if any $d+1$ hyperplanes in $\mathcal{G}$ do not have a common intersection point. Following Edelsbrunner [45], we denote by $f_k^{(d)}(\mathcal{G})$ the number of k-faces of $\mathcal{A}(\mathcal{G})$ (for $k = 0, \ldots, d$) and by $i_k^{(d)}(\mathcal{G})$ the number of incidences between k-faces and $(k+1)$-faces (for $k = 0, \ldots, d-1$). The following fact describes the number of all possible faces and incidences in a given arrangement:

Fact 4.1 *([45, Theorem 1.3]) Let $\mathcal{G}$ be an arbitrary non-empty set of hyperplanes in the d-dimensional Euclidean space $\mathbb{R}^d$. Then,*

$$f_k^{(d)}(\mathcal{G}) \leq \sum_{i=0}^{k} \binom{d-i}{k-i} \binom{|\mathcal{G}|}{d-i} \in \Theta(|\mathcal{G}|^d) \tag{4.3}$$

and

$$i_k^{(d)}(\mathcal{G}) \leq 2(d-k) \sum_{i=0}^{k} \binom{d-i}{k-i} \binom{|\mathcal{G}|}{d-i} \in \Theta(|\mathcal{G}|^d). \tag{4.4}$$

Equality is attained if and only if the arrangement $\mathcal{A}(\mathcal{G})$ is simple.

Duality

We will make use of the following *dual function* $\mathcal{D}_0$ [45, Section 1.6] which maps a point $\mathbf{p} \in \mathbb{R}^d - \{\mathbf{0}\}$ to the hyperplane

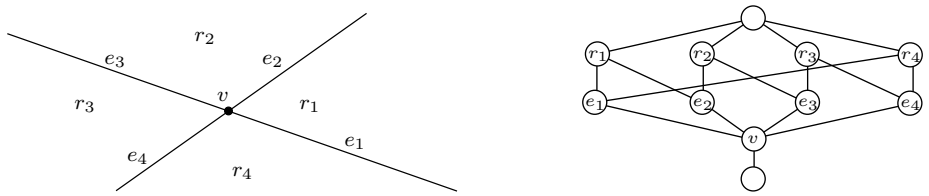
$$\mathcal{D}_0(\mathbf{p}) := \{\mathbf{x} \in \mathbb{R}^d \mid \langle \mathbf{p}, \mathbf{x} \rangle - 1 = 0\}, \tag{4.5}$$

and vice versa, i. e., $\mathcal{D}_0(\mathcal{D}_0(\mathbf{p})) := \mathbf{p}$ for all hyperplanes $\mathcal{D}_0(\mathbf{p})$ not containing the origin $\mathbf{0}$. We extend the definition of $\mathcal{D}_0$ to sets of points and hyperplanes in a natural way. The following fact can be proved easily, see Figure 4.2 (b) for an illustration:

Fact 4.2 *([45, Observation 1.8]) Let $\mathbf{w} \neq \mathbf{0}$ be a point in $\mathbb{R}^d$ and let h be a hyperplane in $\mathbb{R}^d$ not containing the origin $\mathbf{0}$. Then, the transformation $\mathcal{D}_0$ is* incidence *and* order *preserving, i. e.,*

(a) Incidence preservation: The point $\mathbf{w}$ belongs to h if and only if $\mathcal{D}_0(h)$ belongs to $\mathcal{D}_0(\mathbf{w})$.

(b) Order preservation: The point $\mathbf{w}$ is contained in h^+ if and only if the point $\mathcal{D}_0(h)$



(a) Arrangement (b) Incidence Graph

FIGURE 4.3. *In Figure (a), an arrangement in the Euclidean plane is shown. Figure (b) shows its corresponding incidence graph. The graph can be augmented with additional auxiliary information [46].*

is contained in $\mathcal{D}_0(\mathbf{w})^+$ *and the point* $\mathbf{w}$ *is contained in* h^- *if and only if the point* $\mathcal{D}_0(h)$ *is contained in* $\mathcal{D}_0(\mathbf{w})^-$.

4.2.2 Constructing Arrangements

The efficient construction of arrangements will play an important role for the polynomial-time approach proposed in this chapter and is a common problem in the field of computational geometry. Edelsbrunner *et al.* [45, 46] propose a data structure $D(\mathcal{A}(\mathcal{G}))$ called *incidence graph* which represents the mathematical concept of an arrangement $\mathcal{A}(\mathcal{G})$ for a given set $\mathcal{G}$ of hyperplanes in $\mathbb{R}^d$. We will briefly review this data structure as well as its efficient construction. The reader is referred to the descriptions of Edelbrunner *et al.* [45, 46] for a comprehensive introduction.

Incidence Graph

Each k-face in $\mathcal{A}(\mathcal{G})$ is represented by a node in $D(\mathcal{A}(\mathcal{G}))$, where $\mathcal{A}(\mathcal{G})$ is considered to be a $(d+1)$-face and the empty set $\varnothing$ to be a (-1)-face (in the following, we will identify a node with its corresponding face). Further, there exists an edge in $D(\mathcal{A}(\mathcal{G}))$ between two nodes if the corresponding faces are incident, see Figure 4.3. The graph structure can be augmented with auxiliary information, depending on the particular application. Following Edelsbrunner *et al.* [46], we assign a point $p(f)$ to each k-face (and store it in the node itself). For a 0-face, we set $p(f) = f$. If f is an unbounded 1-face, then $p(f)$ is the unique point of f with distance 1 from the (only) incident 0-face. Otherwise, in case f is a bounded 1-face or a k-face with $k \geq 2$, we set

$$p(f) = \left(\sum_{i=1}^{m} p(f_i) \right) / m, \tag{4.6}$$

where $f_1, \ldots, f_m$ are the subfaces of f.

Iterative Construction

Edelsbrunner *et al.* [46] propose an incremental approach to constructing the incidence graph structure of a given arrangement $\mathcal{A}(\mathcal{G})$ induced by a set $\mathcal{G}$ of hyperplanes. The idea is to insert the hyperplanes iteratively. In each iteration, a new hyperplane is added to the already existing graph structure given for the predecessors of the current hyperplane. Since each insertion of such a hyperplane can be performed in $\mathcal{O}(|\mathcal{G}|^{d-1})$ time, the overall runtime is bounded by $\mathcal{O}(|\mathcal{G}|^d)$. We refer to Edelsbrunner *et al.* [46] for details and directly provide the corresponding fact that summarizes the corresponding theoretical analysis:

Fact 4.3 *([46, Theorem 3.3]) For a given set $\mathcal{G}$ of hyperplanes in $\mathbb{R}^d$ with $d \geq 2$, one can construct the incidence graph $D(\mathcal{A}(\mathcal{G}))$ for $\mathcal{A}(\mathcal{G})$ in $\mathcal{O}(|\mathcal{G}|^d)$ time.*

It is worth noting that the incidence graph can deal with degenerated cases (like parallel hyperplanes or the intersection of more than d hyperplanes in a common point). Further, since the maximal complexity of $\mathcal{A}(\mathcal{G})$ is attained if $\mathcal{A}(\mathcal{G})$ is simple, the theoretical bounds are not affected by these cases [46].

4.3 Polynomial-Time Framework

We will now derive the connection of the above geometrical concepts to the combinatorial optimization tasks induced by semi- and unsupervised support vector machines. For the sake of exposition, we will focus on the unsupervised case with a training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$ of unlabeled training patterns. The semi-supervised case can be derived in a similar way, which we will sketch at the end of this section.

4.3.1 Connection to Arrangements

In the following, we will show that there exists a k-face ($k = 0, \ldots, d$) in the arrangement $\mathcal{A}(\mathcal{D}_0(T_U))$ for each optimal partition vector $\mathbf{y}^* \in \{-1, +1\}^u$ of the task (4.1). We will start by considering the case without slack variables and will subsequently show how the ideas can be extended to the more general case with slack variables. For the following theoretical analysis, we will assume that $d \geq 2$ is constant.[4] Further, we will initially assume that both $u \geq d$ and $\mathbf{0} \notin T_U$ hold.

[4]For $d = 1$, the optimization tasks can be solved easily. Again, we do not consider multisets of points, i. e., we assume $\mathbf{x}_i \neq \mathbf{x}_j$ for all $\mathbf{x}_i, \mathbf{x}_j \in T_U$ with $i \neq j$.

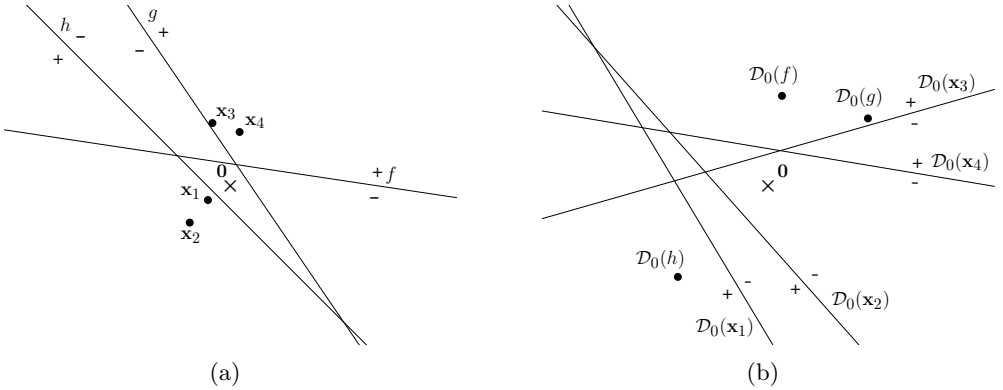


FIGURE 4.4. *In Figure (a), four training patterns $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_4\}$ in the Euclidean plane along with three hyperplanes f, g, and h are shown. Each hyperplane along with the set T_U induce a linearly separable training set. The configuration induced by the dual mapping $\mathcal{D}_0$ is shown in Figure (b). Since $\mathcal{D}_0(f)$ and $\mathcal{D}_0(g)$ are contained in the same cell of the arrangement, the preimages f and g must induce the same linearly separable training set $T'_U = \{(\mathbf{x}_1, -1), (\mathbf{x}_2, -1), (\mathbf{x}_3, +1), (\mathbf{x}_4, +1)\}$.*

Without Slack Variables

Let us initially ignore the slack variables by considering the optimization task induced by a linear hard-margin support vector machine, i. e., we enforce $\xi_i = 0$ for $i = 1, \ldots, u$ in (4.1). In this case, the labeled training set induced by the optimal partition $\mathbf{y} \in \{-1, +1\}^u$ for the task (4.1) is linearly separable. The following lemma follows immediately:

Lemma 4.1 *Let $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$ be a set of points and let $C_{\mathcal{A}}$ be a cell of the arrangement $\mathcal{A}(\mathcal{D}_0(T_U))$. Then, all hyperplanes $\mathcal{D}_0(\mathbf{w})$ with $\mathbf{w} \in C_{\mathcal{A}}$ induce one and the same linearly separable training set with respect to T_U.*

Proof: Application of Fact 4.2. □

Hence, the above lemma implies that every cell (i. e., d-face) in the arrangement $\mathcal{A}(\mathcal{D}_0(T_U))$ corresponds to exactly one linearly separable training set and vice versa (i. e., the mapping between cells in the arrangement and linearly separable training sets is bijective). Further, since we have $\mathcal{O}(u^d)$ cells in the arrangement due to Fact 4.1, there are at most $\mathcal{O}(u^d)$ linearly separable training sets induced by T_U.

These observations are illustrated in Figure 4.4: Here, each hyperplane along with the set $T_U = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ of patterns induce positive and negative training instances. For example, hyperplane f and T_U induce the linearly separable training set $T'_U = \{(\mathbf{x}_1, -1), (\mathbf{x}_2, -1), (\mathbf{x}_3, +1), (\mathbf{x}_4, +1)\}$. Since the hyperplane g and T_U induce the same linearly separable training set T'_U, the above lemma implies that $\mathcal{D}_0(f)$ and $\mathcal{D}_0(g)$ must be contained in the same cell of the arrangement $\mathcal{A}(\mathcal{D}_0(T_U))$. Note that hyperplane h and

T_U induce a different training set. Thus, the point $\mathcal{D}_0(h)$ must be contained in a different cell of the arrangement.

With Slack Variables

Let us now consider the more general case with possibly non-zero slack variables. The next lemma shows that optimal partition vectors fulfill specific properties:

Lemma 4.2 *Given a set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$ of points and let $(\mathbf{y}^*, \mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$ denote an optimal solution for the optimization problem (4.1). Then, one of the two following cases holds for the induced training set $T_U^* = \{(\mathbf{x}_1, y_1^*), \ldots, (\mathbf{x}_u, y_u^*)\}$:*

(a) We have $0 \leq \xi_i^* \leq 1$ for each index i with $y_i^* = +1$. If $\xi_j^* > 1$ holds for a negative training instance $(\mathbf{x}_j, -1) \in T_U^*$, then $d(h(\mathbf{w}^*, b^*), \mathbf{x}_j) \leq d(h(\mathbf{w}^*, b^*), \mathbf{x}_i)$ holds for any positive training instance $(\mathbf{x}_i, +1) \in T_U^*$.

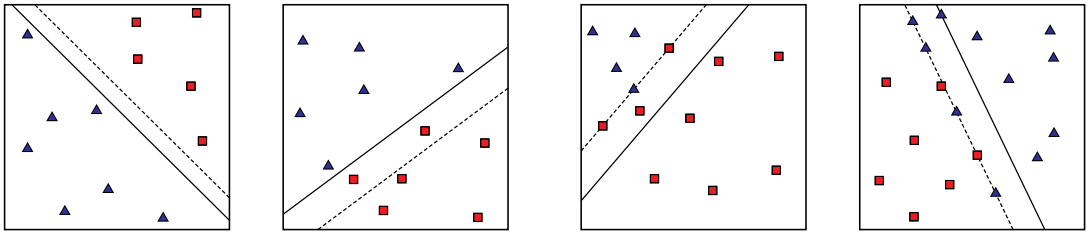
(b) We have $0 \leq \xi_j^* \leq 1$ for each index j with $y_j^* = -1$. If $\xi_i^* > 1$ holds for a positive training instance $(\mathbf{x}_i, +1) \in T_U^*$, then $d(h(\mathbf{w}^*, b^*), \mathbf{x}_i) \leq d(h(\mathbf{w}^*, b^*), \mathbf{x}_j)$ holds for any negative training instance $(\mathbf{x}_j, -1) \in T_U^*$.

Proof: Let us fix the optimal hyperplane $h(\mathbf{w}^*, b^*)$. To prove the first assertion for both cases, let us assume that there exist two training instances $(\mathbf{x}_i, +1)$ and $(\mathbf{x}_j, -1)$ in T_U^* with $\xi_i^* > 1$ and $\xi_j^* > 1$. By exchanging the two associated labels, the corresponding slack variables can be set to 0 without violating the constraints. This leads to a strictly lower objective value, which is a contradiction to the optimality of $h(\mathbf{w}^*, b^*)$.

For the extended assertion of the case (a), let us assume that $0 \leq \xi_i^* \leq 1$ holds for all indices i with $y_i^* = +1$. If $d(h(\mathbf{w}^*, b^*), \mathbf{x}_j) > d(h(\mathbf{w}^*, b^*), \mathbf{x}_i)$ is fulfilled for any negative training instance $(\mathbf{x}_j, -1) \in T_U^*$ with $\xi_j^* > 1$ and any positive training instance $(\mathbf{x}_i, +1) \in T_U^*$, then one can again strictly reduce the objective value by exchanging the labels of $\mathbf{x}_j$ and $\mathbf{x}_i$ without violating the constraints. The extended assertion for the case (b) follows in the same manner. □

The above lemma guarantees the existence of a hyperplane h_{cut} (parallel to an optimal hyperplane $h_{opt} = h(\mathbf{w}^*, b^*)$) that gives rise to an optimal partition vector $\mathbf{y}^* \in \{-1, +1\}^u$.[5] If h_{cut} does not contain any points, then there exists a cell in $\mathcal{A}(\mathcal{D}_0(T_U))$ containing $\mathcal{D}_0(h_{cut})$, see Figure 4.5 (a). Otherwise, up to about u points of T_U might be contained in

[5] Note that, when ignoring the balance constraint, the optimal labels for a *fixed* hyperplane $h(\mathbf{w}, b)$ are given by $y_i^* = f_{(\mathbf{w},b)}(\mathbf{x}_i)$ for all patterns $\mathbf{x}_i \in T_U$ [36]; see the equation (3.3) for the definition of $f_{(\mathbf{w},b)}$. Including the balance constraint, however, renders a more detailed analysis necessary.



(a) Non-Degenerated (b) Degenerated

FIGURE 4.5. *Two non-degenerated cases are shown in Figure (a). Here, the hyperplanes h_{cut} (dashed lines) induce linearly separable training sets. In Figure (b), two degenerated cases are depicted where the hyperplanes h_{cut} contain multiple patterns and, thus, induce training sets which are not linearly separable.*

h_{cut}. In these cases, there exists a corresponding k-face ($k = 0, \ldots, d-1$) in $\mathcal{A}(\mathcal{D}_0(T_U))$ which contains $\mathcal{D}_0(h_{cut})$, see Figure 4.5 (b). Note that the last observation follows directly from the definition of a k-face (which is provided in Section 4.2).

4.3.2 Polynomial-Time Algorithm

The above derivations show that it is sufficient to test all partition vectors which correspond to k-faces in the arrangement $\mathcal{A}(\mathcal{D}_0(T_U))$ (instead of testing all possible $\mathcal{O}(2^u)$ partition vectors). Note that, for the special case with multiple points in the hyperplane h_{cut}, the configuration of points within h_{cut} is not important, i. e., the particular label assignments for these points can be selected in an arbitrary manner. Hence, it is sufficient to check a *single* partition fulfilling the balance constraint out of the $\Theta(2^u)$ possible partitions (induced by all points contained in the hyperplane h_{cut}).

Algorithmic Framework

The two initial assumptions ($u < d$ and $\mathbf{0} \notin T_U$) made at the beginning of this section can be removed as follows: If $u < d$, then one can resort to the simple brute-force approach that tests $\mathcal{O}(2^{d-1}) = \mathcal{O}(1)$ possible partition vectors. Further, since the task (4.1) is invariant under translation, one can handle the case $\mathbf{0} \in T_U$ by simple translation steps in the pre- and post-processing phase.

The main optimization approach is given in Algorithm 4.1: First, the incidence graph $D(\mathcal{A}(\mathcal{D}_0(T_U)))$ for the arrangement $\mathcal{A}(\mathcal{D}_0(T_U))$ is constructed. Given this graph, one iterates over all k-faces (i. e., nodes in the graph). For each k-face, the corresponding partition vector $\mathbf{y}$ is generated, where one can resort to the point $p(f)$ stored for each k-face. If $k = d$, then $p(f)$ is not contained in any dual hyperplane and, hence, induces a unique linearly separable training set (via $\mathcal{D}_0(p(f))$ and T_U) along with the desired partition vector $\mathbf{y}$. If $k < d$, then the hyperplane $\mathcal{D}_0(p(f))$ can contain up to u points

Input: An unlabeled training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$, $C \in \mathbb{R}^+$, $b_c \in [0,1]$, and $\varepsilon \in \mathbb{R}^+$.
Output: An optimal solution (up to machine precision) $(\mathbf{y}^*, \mathbf{w}^*, b^*, \boldsymbol{\xi}^*)$ for the task (4.1).

```
y* = nil
Construct incidence graph D(A(D_0(T_U)))
for each k-face in A(D_0(T_U)) do
    Compute vector y ∈ {−1, +1}^u corresponding to the k-face
    if the vector y fulfills the balance constraint then
        Compute solution (w, b, ξ) for (3.9) given T'_U = {(x_1, y_1), ..., (x_u, y_u)}
        if y* = nil or J(w, b, ξ) < J(w*, b*, ξ*) then
            (y*, w*, b*, ξ*) = (y, w, b, ξ)
        end if
    end if
end for
return (y*, w*, b*, ξ*)
```

ALGORITHM 4.1. *The polynomial-time framework for addressing the task (4.1) induced by linear unsupervised support vector machines: The approach first constructs an incidence graph for the arrangement $\mathcal{A}(\mathcal{D}_0(T_U))$ and then traverses all k-faces of $\mathcal{A}(\mathcal{D}_0(T_U))$. For each k-face, the corresponding (labeled) training set is tested via a standard support vector machine and the best overall result is stored.*

of T_U. However, since their particular configuration is not important, one can generate a unique partition vector that fulfills the balance constraint (if existent), again via $\mathcal{D}_0(p(f))$ and T_U. Given the vector $\mathbf{y}$, the solution $(\mathbf{w}, b, \boldsymbol{\xi})$ with objective value $J(\mathbf{w}, b, \boldsymbol{\xi})$ for the support vector machine task (3.9) with training set $T'_U = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_u, y_u)\}$ is computed. Throughout the overall execution, one keeps track of the best result.

Runtime Analysis

In summary, our approach processes all k-faces in the arrangement $\mathcal{A}(\mathcal{D}_0(T_U))$ and for each k-face, a single partition vector $\mathbf{y}$ is generated and checked via a support vector machine. This leads to the following result:

Theorem 4.1 *Given an unlabeled training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$, one can compute an exact solution (up to machine precision) for the task (4.1) in $\mathcal{O}(u^{d+3} I_L)$ time, where I_L denotes the input length, i.e., the number of bits used to encode all the rational data of the problem.*

Proof: The construction of the graph can be performed in $\mathcal{O}(u^d)$ time. In total, at most $\mathcal{O}(u^d)$ nodes (representing all k-faces) are given in the graph due to Fact 4.1. For each k-face f, the corresponding partition vector $\mathbf{y}$ has to be generated via the point $p(f)$ stored in the node associated with f. This can be done in linear time (also if the point $p(f)$ is contained in up to u dual hyperplanes). Thus, the total runtime is $\mathcal{O}(u^{d+1} + u^d \cdot T_1(u,d))$,

where $T_1(u, d)$ denotes the time needed for computing a solution for the optimization problem (3.9) for fixed $\mathbf{y}$. Since these intermediate tasks give rise to quadratic programs with $u + d + 1$ variables and $2u$ constraints, we have $T_1(u, d) = \mathcal{O}(u^3 I_L)$ due to Fact 3.3 and, thus, $\mathcal{O}(u^{d+3} I_L)$ as total runtime. □

Note that other optimization schemes for the intermediate support vector machine tasks could be used. As pointed out in Chapter 3, recent runtime results provide quadratic upper runtime bounds for general (non-linear) support vector machines. Further, specific schemes like the linear-time approach of Joachims [86] could be employed to address the optimization tasks induced by the special case of a linear support vector machine. The latter scheme, however, assumes that the patterns in $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$ have bounded norm.

Dealing with the Semi-Supervised Case

The combinatorial problem induced by the semi-supervised setting is similar to the one of the unsupervised setting. The only difference is the additional term representing the labeled part of the training data that leads to a slightly different quadratic program. The total runtime is $\mathcal{O}(u^{d+1} + u^d \cdot T_2(l, u, d))$, where $T_2(l, u, d)$ denotes the runtime for solving the task (4.2) for fixed partition vector $\mathbf{y}$. Again, these intermediate tasks give rise to quadratic programs with $l + u + d + 1$ variables and $2(l + u)$ constraints, which can be solved in $\mathcal{O}((l + u)^3 I_L)$ time due to Fact 3.3. Thus, the only difference consists in having slightly modified intermediate optimization tasks in Step 6 of Algorithm 4.1. The remaining ideas can essentially be taken over. This leads to the following result:

Theorem 4.2 *Given a labeled training set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathbb{R}^d \times \{-1, +1\}$ and an unlabeled training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$, one can compute an exact solution (up to machine precision) for the task (4.2) in $\mathcal{O}(u^d (l + u)^3 I_L)$ time, where I_L denotes input length, i.e., the number of bits used to encode all the rational data of the problem.*

4.4 Experimental Analysis

In the remainder of this chapter, we evaluate the runtime behavior of the approach and sketch the generation of benchmark data sets. Again, for simplicity, we focus on the unsupervised case with $d = 2$.

4.4.1 Experimental Setup

We start by describing the experimental setup including details about the considered data sets along with implementation details.

Implementation Details

The implementation is based on the programming language `Python`. The runtimes are measured on a `3.00 GHz Intel(R) Core(TM)2` PC running `Ubuntu 10.04`. To avoid degenerated (dual) hyperplanes, we perform an initial translation such that no points lie around the origin (all points are shifted into the first quadrant); after having applied the optimization scheme, this translation is reverted. For simplicity, we assume that the hyperplanes h_{cut} do not contain any points and that the induced arrangement is simple. Thus, we are only interested in the partition vectors induced by the cells (2-faces) of the arrangement $\mathcal{A}(\mathcal{D}_0(T_U))$.[6] Further, instead of building the complete incidence graph structure in Step 2 of Algorithm 4.1, we resort to the following simple scheme for obtaining these partition vectors: Let

$$\mathbf{s}_{(i,j)} = \mathcal{D}_0(\mathbf{x}_i) \cap \mathcal{D}_0(\mathbf{x}_j) \tag{4.7}$$

be the intersection point of the dual hyperplanes $\mathcal{D}_0(\mathbf{x}_i)$ and $\mathcal{D}_0(\mathbf{x}_j)$ with $i, j \in \{1, \ldots, u\}$ and $i \neq j$. Then, the value

$$v_p(i,j) := \operatorname{sgn}(\langle \mathbf{x}_p, \mathbf{s}_{(i,j)} \rangle - 1) \in \{-1, +1\} \tag{4.8}$$

indicates the relative position of $\mathbf{s}_{(i,j)}$ with respect to the dual hyperplane $\mathcal{D}_0(\mathbf{x}_p)$, see again the definition (4.5) of the dual function $\mathcal{D}_0$.[7] Further, the intersection point $\mathbf{s}_{(i,j)}$ is adjacent to four cells of the arrangement $\mathcal{A}(\mathcal{D}_0(T_U))$, which correspond to the following four partition vectors:

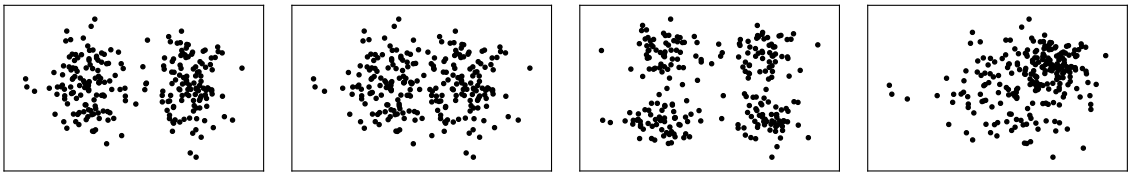
$$\begin{aligned}
\mathbf{y}_1(i,j) &= \{v_1(i,j), \ldots, -1, \ldots, -1, \ldots, v_u(i,j)\} \\
\mathbf{y}_2(i,j) &= \{v_1(i,j), \ldots, +1, \ldots, -1, \ldots, v_u(i,j)\} \\
\mathbf{y}_3(i,j) &= \{v_1(i,j), \ldots, -1, \ldots, +1, \ldots, v_u(i,j)\} \\
\mathbf{y}_4(i,j) &= \{v_1(i,j), \ldots, +1, \ldots, +1, \ldots, v_u(i,j)\}
\end{aligned}$$

Here, for all vectors, the i-th and j-th positions are fixed to ± 1. Thus, by testing these four partition vectors for each of the $\mathcal{O}(u^2)$ intersection points, all cells of the arrangement are processed (multiple times).[8] To approach the intermediate support vector machine tasks in Step 6 of Algorithm 4.1, we resort to the `LIBSVM` implementation provided by Chang *et al.* [29] (using the default parameter settings).

[6] The derivations in Section 4.3 dealing with the special cases are of theoretical interest. Aiming at a robust and simple implementation, we ignore these cases here. For the correct generation of benchmark data sets, we make use of simple assertions to catch these degenerated cases.

[7] Again, we define $\operatorname{sgn}(t) = 1$ for $t \geq 0$ and $\operatorname{sgn}(t) = -1$ for $t < 0$.

[8] Note that each pair of dual hyperplanes intersect since we assume the arrangement to be simple. Further, this approach can be extended to arbitrary dimensions in a natural way.



(a) `Gaussian1` (b) `Gaussian2` (c) `Gaussian3` (d) `Gaussian4`

FIGURE 4.6. *The figures show the different distributions of the two-dimensional data sets. While the cluster structure for the data sets shown in Figures (a) and (b) are obvious, this is not the case for the data sets depicted in Figures (c) and (d).*

Artificial Data Sets

We consider four artificial data sets with $d = 2$ for generating benchmark data of varying complexity. The first data set is the two-dimensional version of the data set already considered the previous two chapters, i.e., $N/2$ points ($N = 250$) are drawn from each of two multivariate Gaussian distributions $\mathbf{X}_i \sim \mathcal{N}(\mathbf{m}_i, \mathbf{I})$ with $\mathbf{m}_1 = (-2.5, 0.0) \in \mathbb{R}^2$ and $\mathbf{m}_2 = (+2.5, 0.0) \in \mathbb{R}^2$, see Figure 4.6 (a). This data set is denoted by `Gaussian1`. The second one, called `Gaussian2`, is a variant of the first one with different centers $\mathbf{m}_1 = (-1.5, 0.0) \in \mathbb{R}^2$ and $\mathbf{m}_2 = (+1.5, 0.0) \in \mathbb{R}^2$, thus leading to two overlapping clusters which are more difficult to separate, see Figure 4.6 (b). The third artificial data set, named `Gaussian3`, is composed of four Gaussian clusters. Here, $N/4$ points are drawn from each of four multivariate Gaussian distributions $\mathbf{X}_i \sim \mathcal{N}(\mathbf{m}_i, \mathbf{I})$ with

$$\mathbf{m}_1 = (-2.5, -2.5) \in \mathbb{R}^2, \qquad \mathbf{m}_2 = (-2.5, +2.5) \in \mathbb{R}^2,$$
$$\mathbf{m}_3 = (+2.5, -2.5) \in \mathbb{R}^2, \qquad \mathbf{m}_4 = (+2.5, +2.5) \in \mathbb{R}^2,$$

see Figure 4.6 (c). Finally, we consider two overlapping clusters (`Gaussian4`), i.e., we generate $N/2$ points from two multivariate Gaussian distributions $\mathbf{X}_i \sim \mathcal{N}(\mathbf{m}_i, \sigma_i \mathbf{I})$ with $\sigma_1 = 4$, $\sigma_2 = 1$ and $\mathbf{m}_1 = (0, 0) \in \mathbb{R}^2$ and $\mathbf{m}_2 = (2, 2) \in \mathbb{R}^2$, see Figure 4.6 (d).

4.4.2 Results

We will now provide the results of the experimental analysis.

Practical Runtime

We consider the `Gaussian1` data set to investigate the practical running time of the implementation. We fix two of the three model parameters, i.e., $C = 1$ and $b_c = 0.5$. The third parameter ε determines the amount of valid partitions (fulfilling the balance constraint), where larger values lead to a less restrictive balance constraint and, hence, to

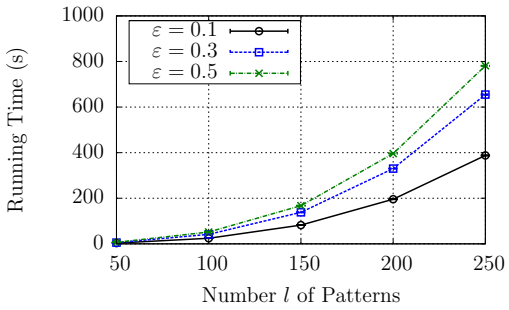


FIGURE 4.7. *The plot depicts the runtime behavior of the approach for varying assignments of ε. Clearly, the more partitions have to be checked by a support vector machine, the more runtime is needed.*

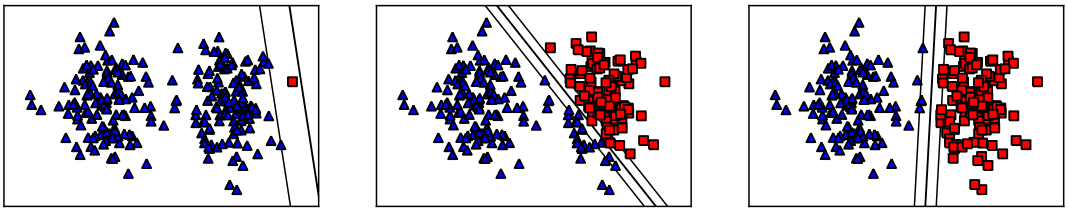
more valid partitions. For this parameter, we consider three possible assignments: $\varepsilon = 0.1$, $\varepsilon = 0.3$, and $\varepsilon = 0.5$. The practical runtimes for this setup are given in Figure 4.7. It can be seen that the runtime increases moderately with increasing size of the training set. Further, as expected, larger values for ε lead to an increased runtime time as well.

Generating Benchmark Data Sets

Model selection and performance evaluation is a difficult issue in the field of semi- and unsupervised support vector machines due to the lack of (sufficient) labeled data. In most cases, the authors resort to the true labels (given in the test set) for tuning the non-fixed model parameters (like C, b_c, and ε) [36]. This way of dealing with the involved parameters is not realistic but, nevertheless, a meaningful procedure since one can investigate the flexibility of the proposed models, i. e., one can test if the corresponding approach is, in principle, capable of adapting to the structure of the data. Note that, even if a good assignment for the parameters is given, one still has to solve the combinatorial optimization tasks. Thus, a comprehensive comparison on artificial and challenging benchmark data sets can be seen as an important pre-evaluation step before approaching real-world data.

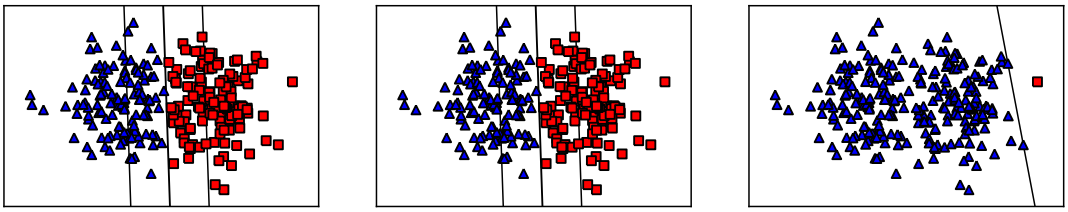
Such an experimental setup is, for instance, a well-established procedure in the field of stochastic optimization [13], where one resorts to difficult non-convex real-valued functions (with many local optima) for the comparison of competing heuristics. In the remainder of this section, we exemplify the generation of benchmark data sets which could serve as the basis for a similar experimental setup in the context of semi- and unsupervised support vector machines. Below, we will see that different data distributions along with specific parameter assignments can lead to various non-obvious optimal solutions. For all remaining experiments, we fix $N = u = 250$.

Scenario 1: Balance Constraint. Most real-world experimental setups aim at detecting obvious cluster structures given in the data set. The `Gaussian1` data set could



(a) $b_c = 0.1$ (b) $b_c = 0.3$ (c) $b_c = 0.5$

FIGURE 4.8. *Influence of the balance parameter b_c on the* `Gaussian1` *data set for fixed $C = 1$ and $\varepsilon = 0.1$. Depending on the particular assignment for the parameter b_c both unbalanced and balanced clustering solutions can be enforced.*



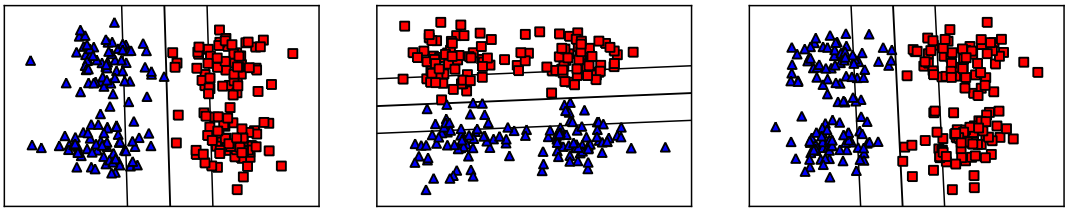
(a) $\varepsilon = 0.1$ (b) $\varepsilon = 0.3$ (c) $\varepsilon = 0.5$

FIGURE 4.9. *Influence of the parameter ε on the* `Gaussian2` *data set for fixed $C = 0.01$ and $b_c = 0.5$. Given a tight balance constraint induced via $\varepsilon = 0.1$, a balanced clustering solution is obtained. Relaxing this constraint by increasing ε leads to unbalanced solutions. Note that in Figure (c), only the hyperplane $\{\mathbf{x} \in \mathbb{R}^2 \mid \langle \mathbf{w}^*, \mathbf{x} \rangle + b^* = -1\}$ is visible.*

be seen as such an easy task due to its well-separated Gaussian clusters. However, the model parameters can also be used to induce more difficult problem instances: Let us fix $C = 1$ and $\varepsilon = 0.1$ and let us consider three assignments for the balance parameter b_c. In Figure 4.8, the optimal partitions along with the induced classification models are shown for $b_c \in \{0.1, 0.3, 0.5\}$. As expected, the assignment $b_c = 0.5$ leads to the obvious cluster separation while the other two ones lead to non-obvious partitions.

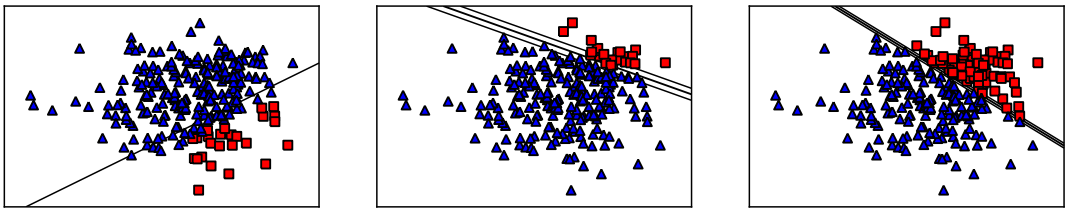
The next experiment is based on the `Gaussian2` data set which exhibits a more overlapping cluster structure. Let us now fix $C = 0.01$ and $b_c = 0.5$. By imposing tight balance constraints via $\varepsilon = 0.1$ and $\varepsilon = 0.3$, one gets decision hyperplanes separating the two obvious clusters. Relaxing the constraint via $\varepsilon = 0.5$, however, yields a completely different partition, see Figure 4.9. Hence, for both data sets, the balance constraint determined by b_c and ε can have a significant influence.[9] We would like to point out that this can be a quite sensitive issue for real-world data scenarios: Local search strategies (like

[9]Note that besides avoiding undesired solutions, the balance constraint can also be used to enforce certain ratios between the positive and the negative class. As a real-world example, consider the task of fore- and background separation for image data (where each pixel corresponds to a vector in $\mathbb{R}^3$). Here, the foreground could be a small object whereas the background could encompass most of the pixels. To obtain a satisfying separation, a small (or a large) value for b_c might be important.



(a) s_1 (b) s_2 (c) s_3

FIGURE 4.10. *Sensitivity with respect to variations of the data set's distribution for* `Gaussian3` *given fixed* $C = 0.01$, $b_c = 0.5$, *and* $\varepsilon = 0.1$. *In this case, using different seeds for the generation of the input data leads different optimal partitions.*



(a) $C = 0.001$ (b) $C = 1$ (c) $C = 1000$

FIGURE 4.11. *Influence of the model parameter* C *on the* `Gaussian4` *data set for fixed* $b_c = 0.2$ *and* $\varepsilon = 0.1$. *By assigning small values to* C, *large margins are prioritized. Assigning large values to* C, *on the other hand, leads to a heavy penalization of patterns lying within the margin. Note that in Figure (a), all positive patterns (red squares) are misclassified, i.e., they are lying on the wrong side of the decision hyperplane (again, only the hyperplane* $\{\mathbf{x} \in \mathbb{R}^2 \mid \langle \mathbf{w}^*, \mathbf{x} \rangle + b^* = -1\}$ *is visible).*

label-switching schemes) are often based on a good estimate for the ratio b_c. However, since a good estimate for this ratio is often not available, one is often enforced to relax the constraint via ε. As we have seen, this can lead to undesired partitions (even if one is able to compute the exact solution).

Scenario 2: Minor Variations. The second scenario is based on the `Gaussian3` data set. Due to the symmetry present in this data set, two obvious partitions are possible (i. e., the vertical and horizontal partition). Thus, given appropriate assignments for the model parameters, the optimal partition should depend on minor changes of the data set's distribution. To investigate this issue, we fix $C = 0.01$, $b_c = 0.5$, and $\varepsilon = 0.1$ and use three different seeds for the generation of the data (i. e., for the generation of random numbers). The result is shown in Figure 4.10. Clearly, minor changes of the data set's distribution have a significant influence on the final (optimal) partition.

Scenario 3: Hard- and Soft-Margins. For the next scenario, we consider different assignments for the parameter $C \in \{0.001, 1, 1000\}$ and fix the other two remaining parameters ($b_c = 0.2$ and $\varepsilon = 0.1$). Thus, by using assignments for the model parameter C,

FIGURE 4.12. *The* `COIL` *data set consisting of grayscale images of* 20 *objects.*

we relax and stress the importance of patterns lying within the induced margin, respectively. Further, the balance constraint used in this setup allows partitions having different sizes. The final optimal partitions are depicted in Figure 4.11. Again, the particular assignment for a single parameter leads to completely different partitions: Small values for the parameter C leads to a prioritization of a large margin whereas large values leads lead to the hard-margin case, i. e., to a heavy penalization of patterns lying within the margin.

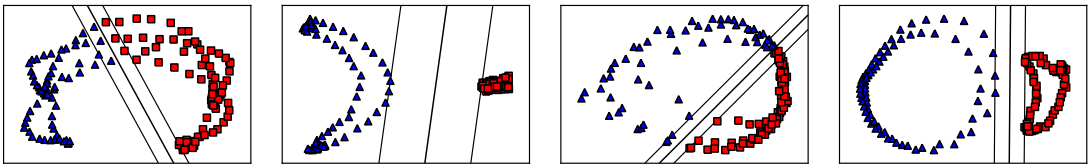
Toy Example: Clustering Image Data

Following Peng *et al.* [112], we sketch the principle combination of dimensionality reduction methods and the proposed clustering approach. As high-dimensional data, we consider the `COIL` [107] data set, which consists of images $1,440$ images (72 images for each of 20 objects), see Figure 4.12. Here, we use $\texttt{COIL}(i, j)$ to denote the binary task induced by the objects i and j, where the labeling in Figure 4.12 is from left to right and top to bottom.

In Figure 4.13, the projections on the first two principal components of four binary tasks obtained via the principal component analysis procedure are shown. In case the induced two-dimensional data sets exhibit a well-separated cluster structure, the exact clustering approach ($C = 1$, $b_c = 0.5$, $\varepsilon = 0.1$) can subsequently separate both classes. In case the reduced data sets do not exhibit such a structure, the clustering approach does not yield satisfying results. For the latter cases, one can try to add more flexibility to the overall model by making use of non-linear dimensionality reduction approaches (like the kernel variant of the principal component analysis [124]). We would like to point out that this example shall only demonstrate the principle application of the exact clustering approach for high-dimensional data. Naturally, the reduction to a low-dimensional feature space can result in a big loss of information about the structure of the data.

4.5 Concluding Remarks

The key contribution of this chapter is a polynomial-time approach for linear semi- and unsupervised support vector machines. The proposed algorithm is the first one that yields non-trivial theoretical upper runtime bounds for the tasks at hand (considering the more



(a) COIL(1,2) (b) COIL(1,8) (c) COIL(3,6) (d) COIL(4,5)

FIGURE 4.13. *Four two-dimensional embeddings along with the corresponding clustering results.*

general soft-margin extensions). Its algorithmic framework is based on a connection between all possible partition vectors and all k-faces in the arrangement induced by the training patterns. The efficient construction of this arrangement plays an important role for the theoretical runtime analysis of the proposed approach. Constructing arrangements is, in turn, an important building block for a variety of algorithms in the field of computational geometry and the optimality of the corresponding algorithm "relies heavily on a nontrivial combinatorial fact" [45, 46].

We believe that the derivations provided in this chapter will stimulate the collaboration between both the field of machine learning and the field of computational geometry with respect to these problems (similar to the results [9, 12, 41, 102, 150] for standard support vector machines). Note that a comparable research direction exists for the k-means algorithm, which aims at obtaining good candidate solutions for its associated combinatorial optimization task (2.8). Similar to the results presented in this chapter, computing exact solutions for this task takes $\mathcal{O}(u^{kd+1})$ time for fixed number $k \in \mathbb{N}$ of designated clusters with patterns in $\mathbb{R}^d$ [82]. Despite these theoretical insights, the polynomial-time approach can be used to generate benchmark data sets in low-dimensional feature spaces. These data sets depict excellent baseline tasks for the mutual quality assessment of competing optimization frameworks. As pointed out above, such an experimental setup is common in the field of stochastic optimization [13], where difficult tasks (e. g., non-convex functions with many local optima) are used for the comparison of optimization schemes.

CHAPTER 5

Speedy Local Search

The computational complexities of the exact approaches proposed in the previous chapter indicate that the optimization tasks are difficult to approach, even for special cases. Since the idea of extending support vector machines to such semi- and unsupervised learning settings is very appealing for real-world data, both extensions have received considerable attention in recent years from a practical point of view. In the literature, a variety of heuristic optimization schemes can be found that are, on the one hand, computationally efficient, but, on the other hand, only yield (possibly suboptimal) candidate solutions without any guaranteed accuracy. In this chapter, we will propose such a local search scheme for addressing both tasks. While a direct implementation is still computationally expensive, we will show how to accelerate the search by means of matrix-based updates for the intermediate solutions. More precisely, these computational shortcuts will render linear-time updates per iteration possible (instead of cubic-time operations needed by a naive implementation). This will greatly reduce the overall runtime of the approach and permits a huge amount of possible candidate solutions to be tested in an efficient kind of way. In addition, we will show how to efficiently integrate well-known (kernel matrix) approximation schemes into the framework that pave the way for large-scale learning settings.

Outline. After motivating the basic ideas in Section 5.1, we will formalize both tasks from a general perspective including non-linear kernels in Section 5.2. The algorithmic framework along with the matrix-based speed-ups will be described in Section 5.3, followed by experiments given in Section 5.4. Concluding remarks will be provided in Section 5.5.

Input: A labeled training set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathbb{R}^d \times \{-1, +1\}$, an unlabeled training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathbb{R}^d$, $C' \in \mathbb{R}^+$, $C \in \mathbb{R}^+$, $b_c \in [0, 1]$, and $\varepsilon \in \mathbb{R}^+$.
Output: An approximation $(\mathbf{y}, \mathbf{w}, b, \boldsymbol{\xi}', \boldsymbol{\xi})$ for the semi-supervised learning task (4.2).
1: Compute solution $(\mathbf{w}, b, \boldsymbol{\xi}')$ for the support vector machine task (3.9) given T_L.
2: Initialize vector $\mathbf{y} \in \{-1, +1\}^u$ via $y_i = f_{(\mathbf{w},b)}(\mathbf{x}_i)$ for $i = 1, \ldots, u$.
3: **repeat**
4: Switch a positive and a negative entry of $\mathbf{y}$ if this leads to a strictly better objective (4.2). In case several pairs leading to a smaller objective exist, consider the one that yields the best improvement.
5: **until** No label-switches leading to a smaller objective possible anymore.
6: Compute solution $(\mathbf{w}, b, \boldsymbol{\xi}', \boldsymbol{\xi})$ for the task (4.2) for fixed $\mathbf{y} \in \{-1, +1\}^u$.
7: **return** $(\mathbf{y}, \mathbf{w}, b, \boldsymbol{\xi}', \boldsymbol{\xi})$

ALGORITHM 5.1. *Local search scheme for linear semi-supervised support vector machines proposed by Joachims [83]: The initial candidate solution is obtained via a linear support vector machine trained on the labeled part of the data. The unlabeled patterns are initialized via the decision function $f_{(\mathbf{w},b)}$ that is defined in (3.3). The main loop (Steps 3–5) iterates until no label-switchings leading to a lower objective are possible anymore. Throughout the overall execution, only partition vectors fulfilling the balance constraint are generated. Further, the approach also gradually increases the influence of the unlabeled patterns (which is not depicted here).*

5.1 Motivation

One of the first practical optimization schemes for semi-supervised scenarios was given by Joachims [83]: The main idea of his approach is to first train a standard support vector machine on the labeled part of the data and to subsequently improve this initial guess by incorporating the unlabeled part via a label-switching strategy, see Algorithm 5.1. Although being one of the first methods for the semi-supervised learning task, this kind of local search strategy is still among the state-of-the-art methods, most probably due to its conceptual simplicity and the publicly available source code.

For the unsupervised case, this scheme was reformulated by Zhang *et al.* [153]. Since no labeled part is available for these settings, they resort to a simple clustering approach for getting reasonable initial candidate solutions. Exactly as Joachims, they suggest to subsequently improve the objective value by an iterative label-switching strategy. One of their key insights was the replacement of the hinge loss by the ε-insensitive loss such that "it is easier to flip the labels if needed" [153]. Their experimental evaluation indicates that such a replacement is an important algorithmic ingredient, at least for unsupervised scenarios. In addition to the ε-insensitive loss, they also suggest to make use of the square loss (instead of the hinge loss), which led to competitive results.

One of the main disadvantages of the two schemes depicted above is the fact that one has to retrain a support vector machine for each of the intermediate candidate solutions.

Depending on the particular assignments for the involved model parameters, such a recurrent training can be extremely time-consuming. In this chapter, we will propose a simple local search scheme similar to the one depicted in Algorithm 5.1 that can be used to address both semi- and unsupervised support vector machines. Further, motivated by the results of Zhang *et al.* [153] for the ε-insensitive and the square loss, we will consider least-squares variants for the original problem formulation. While a naive implementation of the resulting local search scheme is computationally very demanding, we will provide an efficient implementation of this framework that is based on matrix updates of the intermediate candidate solutions. This will result in a conceptually very simple but extremely effective approach for both learning tasks.

5.2 General Classification Framework

We will now formalize the classification tasks by extending the definitions of linear semi- and unsupervised support vector machines to the general cases with arbitrary kernels.

5.2.1 Non-Linear Extensions

In the following, let $\mathcal{X}$ be an arbitrary set, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a kernel function, and $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \mathcal{Y}$ a labeled training set with $\mathcal{Y} = \{-1, +1\}$. As depicted in Chapter 3, support vector machines are a special case of regularization problems of the form

$$\inf_{f \in \mathcal{H}_k,\, b \in \mathbb{R}} \frac{1}{l} \sum_{i=1}^{l} \mathcal{L}\big(y'_i, f(\mathbf{x}'_i) + b\big) + \lambda \|f\|^2_{\mathcal{H}_k} \tag{5.1}$$

with user-defined parameter $\lambda \in \mathbb{R}^+$, loss function $\mathcal{L} : \mathcal{Y} \times \mathbb{R} \to [0, \infty)$ and feature space $\mathcal{H}_k$ that is associated with the kernel function k. Similar to linear support vector machines, this more general concept can be extended to both semi- and unsupervised settings in the following manner.

Unsupervised Support Vector Machines

The direct extension of the above regularization problem to the unsupervised case for a given unlabeled training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$ has the form [149]

$$\begin{aligned} \underset{\mathbf{y} \in \{-1,+1\}^u,\, f \in \mathcal{H}_k,\, b \in \mathbb{R}}{\text{minimize}} \quad & \frac{1}{u} \sum_{i=1}^{u} \mathcal{L}\big(y_i, f(\mathbf{x}_i) + b\big) + \lambda \|f\|^2_{\mathcal{H}_k} \\ \text{s.t.} \quad & \left| \frac{1}{u} \sum_{i=1}^{u} \max(0, y_i) - b_c \right| < \varepsilon \end{aligned} \tag{5.2}$$

with parameters $\lambda \in \mathbb{R}^+$, $b_c \in [0,1]$, and $\varepsilon \in \mathbb{R}^+$. Again, the balance constraint is used to enforce a certain ratio between positive and negative assignments determined by $\mathbf{y} \in \{-1,+1\}^u$.[1] Plugging in the hinge loss leads to the problem statement usually addressed in the literature for unsupervised support vector machines, also named *maximum margin clustering* [149]. In this case, one obtains a mixed-integer programming problem [51] via the semiparametric representer theorem (Fact 3.2) since, for fixed partition vector $\mathbf{y} \in \{-1,+1\}^u$, any optimal $f^* \in \mathcal{H}_k$ is of the form

$$f^*(\cdot) = \sum_{i=1}^{u} c_i k(\cdot, \mathbf{x}_i) \tag{5.3}$$

with real coefficients $c_1, \ldots, c_u \in \mathbb{R}$.

Semi-Supervised Support Vector Machines

For semi-supervised classification, we are given both a set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \{-1,+1\}$ of labeled patterns as well as a set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$ of unlabeled patterns. Taking both parts into account leads to

$$\begin{aligned} \underset{\mathbf{y} \in \{-1,+1\}^u,\, f \in \mathcal{H}_k,\, b \in \mathbb{R}}{\text{minimize}} \quad & \frac{1}{l} \sum_{i=1}^{l} \mathcal{L}\big(y'_i, f(\mathbf{x}'_i) + b\big) + \gamma \frac{1}{u} \sum_{i=1}^{u} \mathcal{L}\big(y_i, f(\mathbf{x}_i) + b\big) + \lambda \|f\|^2_{\mathcal{H}_k} \\ \text{s.t.} \quad & \left| \frac{1}{u} \sum_{i=1}^{u} \max(0, y_i) - b_c \right| < \varepsilon \end{aligned} \tag{5.4}$$

with parameters $\lambda \in \mathbb{R}^+$, $\gamma \in \mathbb{R}^+$, $b_c \in [0,1]$, and $\varepsilon \in \mathbb{R}^+$ [10, 83, 141, 142]. Thus, in addition to the parameter λ, we are also given the parameter γ that determines the influence of the unlabeled part with respect to the overall objective value. Note that, depending on the data at hand, the particular assignment for γ can have a significant influence on the overall model which necessitates its existence.[2] It should be pointed out that, in contrast to the unsupervised setting, the balance constraint is not necessary for the semi-supervised case due to the labeled part of the data.[3] However, a good estimate for the true ratio between positive and negative patterns is usually beneficial for local search strategies due to the more restrictive search space. Using the hinge loss leads to the problem formulation for standard semi-supervised support vector machines and, again, to a mixed-integer programming problem via the semiparametric representer theorem (Fact 3.2).

[1] As pointed out above, these parameters have to set accordingly to avoid trivial solutions.

[2] A very small value corresponds to a pure classification task whereas a large value corresponds to a pure clustering task.

[3] If at least one negative training and one positive training instance are given in T_L.

5.2.2 Related Work

Both mixed-integer programming problems can be addressed by standard solvers. Although exact solutions (up to machine precision) can be obtained this way, it is not feasible for settings with more than, e. g., 80 unlabeled patterns [10]. For that reason, a variety of heuristics and other techniques has been proposed in the literature.

Unsupervised Support Vector Machines

Xu *et al.* [149] were among the first ones who formalized the extension of support vector machines to unsupervised learning scenarios. Their optimization approach is essentially based on reformulating the original problem definition as semidefinite programming problem [20], which can then be addressed efficiently via standard solvers. Note that the resulting approach computes only an approximation of the original problem formulation (without any accuracy guarantees). A variant of this approach is given by Valizadegan and Jin [140] who show how to reduce the number of involved optimization variables and how to incorporate a kernel learning scheme. Further, an extension of this optimization framework to multi-class settings has been proposed by Xu *et al.* [148].

As mentioned above, Zhang *et al.* [153] proposed an optimization scheme that improves an initial candidate solution by iteratively applying a support vector regression model. While being computationally quite efficient, their approach depends heavily on the considered initial guess (which can, for instance, be obtained via the k-means clustering scheme). A recent local search approach for the linear case is given by Wang *et al.* [146, 156]. It is based on applying the so-called *constrained concave-convex procedure* [133, 152], which is an iterative optimization strategy yielding a local optimum for a non-convex optimization problem [134, 152]. Wang *et al.* [146] apply this iterative scheme by deriving appropriate quadratic programming problems (with many constraints) to be solved per iteration. The intermediate tasks are addressed by the cutting plane approach for linear support vector machines [86]. As pointed out by Wang *et al.* [146], the resulting framework needs $\mathcal{O}(T \cdot su)$ time for u unlabeled patterns, where T is the number of iterations needed by the constrained concave convex procedure and where s represents the sparsity of the data (i. e., the average number of nonzero features in a given pattern). A direct extension to multi-class scenarios is also given by Zhao *et al.* [156]. Another approach, also based on a cutting plane framework, is provided by Li *et al.* [98]. Their scheme is similar to the one proposed by Xu *et al.* [149], but with tighter constraints.

Semi-Supervised Support Vector Machines

The idea of using unlabeled data in the context of support vector machines stems from Vapnik, Sterin, and Joachims [83, 141, 142] under the name of *transductive inference* or

transductive support vector machines and from Bennett and Demiriz [10] under the name of *semi-supervised support vector machines*. Among the first approaches was the local search scheme of Joachims [83] sketched above. During the last decade, a variety of other optimization schemes has been proposed that are based on the concave-convex procedure [38, 39, 55], gradient descent [32], semidefinite programming [14, 148], deterministic annealing [131], and other methods [10, 33]. In addition, the problem has recently gained attention in the field of evolutionary computation [104, 127]. In Chapter 6, we will see that it is possible to restate the tasks as continuous but non-convex optimization problems. In this context several approaches [32, 33, 38, 39, 55] have been proposed and we defer their discussion to Chapter 6. In general, semi-supervised classification settings can also be addressed by various other techniques that aim at using both the labeled part as well as the unlabeled one to improve the generalization performance of the models. For a comprehensive overview of related concepts and methods, we refer the reader to the overviews given by Chapelle *et al.* [34, 36] and Zhu *et al.* [157]. In this work, however, we will focus on the semi-supervised extension of support vector machines defined above.

5.3 Algorithmic Framework

We will start by depicting the algorithmic framework including the definition of the considered least-squares variants as well as the local search scheme used for addressing the induced tasks. Subsequently, we will show how to speed up the intermediate optimization problems of the local search scheme by means of efficient matrix updates.

5.3.1 Least-Squares Variants

Standard support vector machines are associated with the hinge loss. In the literature several variants of this classification concept can be found that are based on replacing the hinge loss by other loss functions. One of these surrogates is the concept of *least-squares support vector machines* [136], also known under the name of *regularized least-squares classification* [120]. We will now describe this supervised variant as well as its extensions to semi- and unsupervised learning settings.

Least-Squares Support Vector Machines

Least-squares support vector machines stem from the replacement of the hinge loss by the square loss $\mathcal{L}(y,t) = (y-t)^2$. Plugging in this loss function into the task (5.1) leads to

$$\inf_{f \in \mathcal{H}_k} \frac{1}{l} \sum_{i=1}^{l} \left(y_i' - f(\mathbf{x}_i')\right)^2 + \lambda \|f\|_{\mathcal{H}_k}^2, \tag{5.5}$$

where $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \{-1, +1\}$ is the labeled training set and where $\lambda \in \mathbb{R}^+$. Note that the latter formulation does not include the offset term $b \in \mathbb{R}$ that addresses translated data. In the remainder of this chapter, we will omit this term for the sake of simplicity.[4] Again, due to the representer theorem (Fact 3.1), any minimizer $f^* \in \mathcal{H}_k$ of the above task has the form

$$f^*(\cdot) = \sum_{i=1}^{l} c_i k(\cdot, \mathbf{x}'_i) \tag{5.6}$$

with appropriate coefficients $\mathbf{c} = (c_1, \ldots, c_l)^{\mathrm{T}} \in \mathbb{R}^l$. Thus, using $\|f^*\|^2_{\mathcal{H}_k} = \mathbf{c}^{\mathrm{T}}\mathbf{K}\mathbf{c}$, one can rewrite the regularization problem (5.5) as

$$\underset{\mathbf{c} \in \mathbb{R}^l}{\text{minimize}} \, \frac{1}{l}(\mathbf{y}' - \mathbf{K}\mathbf{c})^{\mathrm{T}}(\mathbf{y}' - \mathbf{K}\mathbf{c}) + \lambda \mathbf{c}^{\mathrm{T}}\mathbf{K}\mathbf{c}, \tag{5.7}$$

where $\mathbf{y}' = (y'_1, \ldots, y'_l)^{\mathrm{T}}$ and where $\mathbf{K} \in \mathbb{R}^{l \times l}$ is the kernel matrix induced by the sequence $\mathbf{x}'_1, \ldots, \mathbf{x}'_l$ of patterns. One of the benefits of the above classification model is the fact that it depicts an unconstrained convex optimization task whose solution can be computed analytically in $\mathcal{O}(l^3)$ time using $\mathcal{O}(l^2)$ space. For a comprehensive overview of this classification concept, we refer to Rifkin [120] and Suykens and Vandewalle [136].

Unsupervised Least-Squares Support Vector Machines

The extension of the least-squares variant depicted above to unsupervised scenarios with a set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$ of unlabeled patterns consists in searching for a function $f^* \in \mathcal{H}_k$ and a partition vector $\mathbf{y}^* \in \{-1, +1\}^u$ for the unlabeled patterns that are optimal with respect to

$$\begin{aligned} \underset{\mathbf{y} \in \{-1,+1\}^u, \, f \in \mathcal{H}_k}{\text{minimize}} \; & \frac{1}{u} \sum_{i=1}^{u} \big(y_i - f(\mathbf{x}_i)\big)^2 + \lambda \|f\|^2_{\mathcal{H}_k} \\ \text{s.t.} \; & \left| \frac{1}{u} \sum_{i=1}^{u} \max(0, y_i) - b_c \right| < \varepsilon, \end{aligned} \tag{5.8}$$

[4] As explained in Chapter 3, a regularized offset effect can be easily obtained for linear support vector machines. Similar modifications can be performed for least-squares support vector machines.

where $\lambda \in \mathbb{R}^+, b_c \in [0,1]$, and $\varepsilon \in \mathbb{R}^+$. By applying the representer theorem (Fact 3.1) for a fixed partition vector $\mathbf{y} \in \{-1,+1\}^u$, one can rewrite the above task as

$$\begin{aligned} \underset{\mathbf{y}\in\{-1,+1\}^u,\, \mathbf{c}\in\mathbb{R}^u}{\text{minimize}} \; & Q(\mathbf{y},\mathbf{c}) = (\mathbf{\Lambda y} - \mathbf{\Lambda K c})^{\mathrm{T}}(\mathbf{\Lambda y} - \mathbf{\Lambda K c}) + \lambda \mathbf{c}^{\mathrm{T}}\mathbf{K c} \\ \text{s.t.} \; & \left| \frac{1}{u}\sum_{i=1}^{u} \max(0, y_i) - b_c \right| < \varepsilon, \end{aligned} \tag{5.9}$$

where $\mathbf{K} \in \mathbb{R}^{u\times u}$ is the kernel matrix induced by the sequence $\mathbf{x}_1, \ldots, \mathbf{x}_u$ and where $\mathbf{\Lambda} \in \mathbb{R}^{u\times u}$ a diagonal matrix with entries of the form $[\mathbf{\Lambda}]_{i,i} = \sqrt{\frac{1}{u}}$ for $i = 1, \ldots, u$.

Semi-Supervised Least-Squares Support Vector Machines

Using the square loss for the semi-supervised case with a set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \{-1,+1\}$ of labeled patterns and a set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$ of unlabeled patterns leads to

$$\begin{aligned} \underset{\mathbf{y}\in\{-1,+1\}^u,\, f\in\mathcal{H}_k}{\text{minimize}} \; & \frac{1}{l}\sum_{i=1}^{l}\big(y'_i - f(\mathbf{x}'_i)\big)^2 + \gamma\frac{1}{u}\sum_{i=1}^{u}\big(y_i - f(\mathbf{x}_i)\big)^2 + \lambda\|f\|^2_{\mathcal{H}_k} \\ \text{s.t.} \; & \left| \frac{1}{u}\sum_{i=1}^{u} \max(0, y_i) - b_c \right| < \varepsilon, \end{aligned} \tag{5.10}$$

where $\gamma \in \mathbb{R}^+$ and $\lambda \in \mathbb{R}^+$ are cost parameters and where the balance constraint is determined by $b_c \in [0,1]$ and $\varepsilon \in \mathbb{R}^+$. Again, due to the representer theorem (Fact 3.1), any optimal function $f^* \in \mathcal{H}_k$ for fixed $\mathbf{y} \in \{-1,+1\}^u$ is of the form

$$f^*(\cdot) = \sum_{i=1}^{l} c_i k(\cdot, \mathbf{x}'_i) + \sum_{i=l+1}^{l+u} c_i k(\cdot, \mathbf{x}_{i-l}) \tag{5.11}$$

with appropriate coefficients $\mathbf{c} = (c_1, \ldots, c_{l+u})^{\mathrm{T}} \in \mathbb{R}^n$ with $n = l + u$. Thus, we obtain

$$\begin{aligned} \underset{\mathbf{y}\in\{-1,+1\}^n,\, \mathbf{c}\in\mathbb{R}^n}{\text{minimize}} \; & J(\mathbf{y},\mathbf{c}) = (\mathbf{\Lambda y} - \mathbf{\Lambda K c})^{\mathrm{T}}(\mathbf{\Lambda y} - \mathbf{\Lambda K c}) + \lambda \mathbf{c}^{\mathrm{T}}\mathbf{K c} \\ \text{s.t.} \; & \left| \frac{1}{u}\sum_{i=l+1}^{n} \max(0, y_i) - b_c \right| < \varepsilon, \\ & \text{and } y_i = y'_i \text{ for } i = 1, \ldots, l, \end{aligned} \tag{5.12}$$

where $\mathbf{K} \in \mathbb{R}^{n\times n}$ is the kernel matrix induced by the sequence $\mathbf{x}'_1, \ldots, \mathbf{x}'_l, \mathbf{x}_1, \ldots, \mathbf{x}_u$ and where $\mathbf{\Lambda} \in \mathbb{R}^{n\times n}$ is a diagonal matrix with entries $[\mathbf{\Lambda}]_{i,i} = \sqrt{\frac{1}{l}}$ for $i = 1, \ldots, l$ and $[\mathbf{\Lambda}]_{i,i} = \sqrt{\frac{\gamma}{u}}$ for $i = l+1, \ldots, n$.

Input: A labeled training set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \{-1, +1\}$, an unlabeled training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$, $\gamma \in \mathbb{R}^+$, $\lambda \in \mathbb{R}^+$, $b_c \in [0, 1]$, and $\varepsilon \in \mathbb{R}^+$.
Output: An approximation $(\mathbf{c}^*, \mathbf{y})$ for the task (5.12).
1: Initialize candidate solution $\mathbf{y} \subseteq \{-1, +1\}^n$ (see text).
2: **while** not converged **do**
3: Generate modified $\bar{\mathbf{y}}$ by flipping a single coordinate $j \in \{l + 1, n\}$ of $\mathbf{y}$.
4: **if** $F(\bar{\mathbf{y}}) < F(\mathbf{y})$ and $\bar{\mathbf{y}}$ is valid **then**
5: Replace $\mathbf{y}$ by $\bar{\mathbf{y}}$.
6: **end if**
7: **end while**
8: Compute $\mathbf{c}^*$ for $\text{minimize}_{\mathbf{c} \in \mathbb{R}^n} J(\mathbf{y}, \mathbf{c})$.
9: **return** $(\mathbf{c}^*, \mathbf{y})$

ALGORITHM 5.2. *Local search scheme for semi-supervised least-squares support vector machines defined via the task (5.12): Starting with an initial candidate solution, one iteratively improves the objective value by flipping single assignments of the partition vector. Once no flips improve the objective value anymore, the corresponding model $\mathbf{c}^* \in \mathbb{R}^n$ is computed and returned (along with the optimal partition vector $\mathbf{y} \in \{-1, +1\}^n$).*

5.3.2 Local Search Strategy

For the sake of exposition we will focus on the description of the semi-supervised case defined in (5.12); the unsupervised case can be handled in the same way. The local search strategy considered in this chapter is given in Algorithm 5.2 and is similar to the approach proposed by Joachims [83] depicted in Section 5.1: Starting with an initial candidate solution, one iterates until a convergence criterion is fulfilled. For each iteration a new candidate solution is generated by flipping a single coordinate and the best performing candidate solution (out of the two current ones) is used for the next iteration. The quality of an intermediate candidate solution $\bar{\mathbf{y}} \in \{-1, +1\}^n$ is measured in terms of the objective (5.12), i. e., via

$$F(\bar{\mathbf{y}}) = \underset{\mathbf{c} \in \mathbb{R}^n}{\text{minimize}}\, J(\bar{\mathbf{y}}, \mathbf{c}). \tag{5.13}$$

Once the overall process is finished, the final candidate solution $\mathbf{y} \in \{-1, +1\}^n$ along with its corresponding optimal vector $\mathbf{c}^* \in \mathbb{R}^n$ is returned. Throughout the execution of the algorithm, it is ensured that only *valid* candidate solutions are generated, i. e., partition vectors fulfilling the balance constraint.

Initial Candidate Solutions

The generation of initial candidate solutions in Step 1 of Algorithm 5.2 plays an important role for the outcome of the local search. We will consider the following two schemes.

Random Initialization. The first one consists in initializing the unlabeled patterns randomly while taking the balance parameter b_c into account, i.e., we set $y_i = 1$ with probability b_c and $y_i = -1$ with probability $1 - b_c$ for $i = l+1, \ldots, n$. In case the balance constraint is not fulfilled after this initialization phase, we perform (a small amount of) appropriate flips to ensure its validness. The first l coordinates of $\mathbf{y} \in \{-1, +1\}^n$ are fixed to the known values provided by the labeled training set.

Initial Guess. The second scheme uses the labeled part of the data to train a least-squares support vector machine via the convex task (5.7), which is subsequently used to initialize the unlabeled patterns. More specifically, the predictions of the model are used as assignments for the unlabeled patterns. If the balance constraint is not fulfilled after this assignment phase, one uses an appropriate amount of the largest (real-valued) predictions of the model as positive and the remaining ones as negative class assignments. Thus, this setting is similar to the local search scheme proposed by Joachims [83]. Note that, for unsupervised learning settings, such an initial guess cannot be obtained due to lack of labeled data and one has to resort to, e.g., a random initialization of the unlabeled patterns.

Coordinate-Flips and Convergence

Another important issue is the way the coordinates in Step 3 of Algorithm 5.2 are selected. One obvious scheme is a random selection i.e., the coordinate $j \in \{l+1, \ldots, n\}$ is selected uniformly at random per iteration. Another way consists in selecting the coordinates in a round-robin manner, i.e., one sets $j = l + 1$ for the first iteration, $j = l + 2$ for the second iteration, and so on until the $u + 1$-th iteration is reached, where one restarts with $j = l + 1$. In the following, we will make use of the round-robin scheme and will stop the iterative process if no changes have occurred for u consecutive iterations. The next lemma is straightforward and shows that a finite number of iterations is needed for the approach to converge in this case.

Lemma 5.1 *The local search depicted in Algorithm 5.2 converges in a finite number of iterations in case the above round-robin scheme is used to select the coordinates in Step 3.*

Proof: Let us denote u consecutive iterations/coordinate-flips starting with $j = l + 1$ as *round*. In case the overall objective is not reduced in a round, the local search stops. Otherwise, the new objective (5.12) is strictly less than the one of the previous round. Since we have a finite number of candidate solutions, there can only be a finite number of rounds that lead to a decrease of the objective. □

Note that the above observation is similar to the one provided by Joachims [83] for his local search scheme depicted in Algorithm 5.1.

5.3.3 Convex Intermediate Tasks

Let us now consider how to solve the intermediate optimization tasks (5.13) induced by Step 4 of Algorithm 5.2. For a fixed partition vector $\bar{\mathbf{y}}$, the gradient $\nabla_{\mathbf{c}} J(\bar{\mathbf{y}}, \mathbf{c})$ is given by

$$\nabla_{\mathbf{c}} J(\bar{\mathbf{y}}, \mathbf{c}) = -2(\mathbf{\Lambda K})^{\mathrm{T}}(\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda K c}) + 2\lambda \mathbf{K c} \tag{5.14}$$

and the Hessian by

$$\nabla^2_{\mathbf{c}} J(\bar{\mathbf{y}}, \mathbf{c}) = 2(\mathbf{\Lambda K})^{\mathrm{T}}\mathbf{\Lambda K} + 2\lambda \mathbf{K} \succeq 0. \tag{5.15}$$

Since the kernel matrix $\mathbf{K} \succeq 0$ is positive definite, the above Hessian is also positive definite (for all $\mathbf{c} \in \mathbb{R}^n$).[5] Thus, the intermediate optimization tasks are convex and $\nabla_{\mathbf{c}} J(\bar{\mathbf{y}}, \mathbf{c}) \stackrel{!}{=} \mathbf{0}$ is a necessary and sufficient condition for optimality [20]. This yields

$$\begin{aligned}
& -2(\mathbf{\Lambda K})^{\mathrm{T}}(\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda K c}) + 2\lambda \mathbf{K c} && \stackrel{!}{=} \mathbf{0} \\
\Leftrightarrow \quad & ((\mathbf{\Lambda K})^{\mathrm{T}}\mathbf{\Lambda K} + \lambda \mathbf{K})\mathbf{c} && \stackrel{!}{=} (\mathbf{\Lambda K})^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}} \\
\Leftrightarrow \quad & (\mathbf{\Lambda K})^{\mathrm{T}}(\mathbf{\Lambda K} + \lambda \mathbf{\Lambda}^{-1})\mathbf{c} && \stackrel{!}{=} (\mathbf{\Lambda K})^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}} \\
\Leftrightarrow \quad & (\mathbf{\Lambda K})^{\mathrm{T}}(\mathbf{\Lambda K \Lambda} + \lambda \mathbf{I})\mathbf{\Lambda}^{-1}\mathbf{c} && \stackrel{!}{=} (\mathbf{\Lambda K})^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}}
\end{aligned}$$

Now, if $\mathbf{K}$ is strictly positive definite (and therefore invertible), then

$$\mathbf{c}^* = \mathbf{\Lambda G \Lambda}\bar{\mathbf{y}} \tag{5.16}$$

is the optimal solution with $\mathbf{G} := (\mathbf{\Lambda K \Lambda} + \lambda \mathbf{I})^{-1}$.[6] If $\mathbf{K}$ is only positive definite, then the above equation determines one of the possible solutions since

$$(\mathbf{\Lambda K})^{\mathrm{T}}\mathbf{G}^{-1}\mathbf{\Lambda}^{-1}\mathbf{c}^* = (\mathbf{\Lambda K})^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}} \tag{5.17}$$

holds as well, see Rifkin [120, p. 74] for a similar argumentation. Thus, the intermediate solutions can be obtained analytically via Equation (5.16). Further, one can avoid cubic-time operations for computing the desired objective values $F(\bar{\mathbf{y}})$:

[5] The sum of two positive definite matrices is positive definite. Further, we have

$$\mathbf{z}^{\mathrm{T}}(\mathbf{\Lambda K})^{\mathrm{T}}(\mathbf{\Lambda K})\mathbf{z} = ((\mathbf{\Lambda K})\mathbf{z})^{\mathrm{T}}((\mathbf{\Lambda K})\mathbf{z}) \geq 0$$

for all $\mathbf{z} \in \mathbb{R}^n$, which shows the positive definiteness of the matrix $(\mathbf{\Lambda K})^{\mathrm{T}}(\mathbf{\Lambda K}) \succeq 0$.

[6] Note that $\mathbf{\Lambda K \Lambda} \succeq 0$ is positive definite since $\mathbf{K} \succeq 0$ is positive definite. Hence, $\mathbf{\Lambda K \Lambda} + \lambda \mathbf{I} \succ 0$ is strictly positive definite and thus invertible.

Lemma 5.2 *For each vector $\bar{\mathbf{y}} \in \{-1,+1\}^n$, the associated optimal solution $\mathbf{c}^* \in \mathbb{R}^n$ as well as its objective value $F(\bar{\mathbf{y}})$ for the task (5.13) can be obtained in $\mathcal{O}(n^2)$ time. Further, $\mathcal{O}(n^3)$ preprocessing time and $\mathcal{O}(n^2)$ space is needed.*

Proof: Since one can precompute the matrix $\mathbf{\Lambda G \Lambda} \in \mathbb{R}^{n \times n}$, the optimal solution $\mathbf{c}^* \in \mathbb{R}^n$ for each new partition vector $\bar{\mathbf{y}} \in \{-1,+1\}^n$ can be obtained in $\mathcal{O}(n^2)$ time via Equation (5.16). Further, the corresponding objective value

$$F(\bar{\mathbf{y}}) = (\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda K c}^*)^{\mathrm{T}}(\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda K c}^*) + \lambda(\mathbf{c}^*)^{\mathrm{T}}\mathbf{K c}^* \tag{5.18}$$

can be obtained in $\mathcal{O}(n^2)$ time: Both $\mathbf{\Lambda K c}^*$ as well $\lambda(\mathbf{c}^*)^{\mathrm{T}}\mathbf{K c}^*$ can be computed in quadratic time since $\mathbf{\Lambda}$ is a diagonal matrix and since $\mathbf{c}^* \in \mathbb{R}^n$ is a vector. Further, given $\mathbf{\Lambda K c}^*$, one can compute $(\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda K c}^*)^{\mathrm{T}}(\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda K c}^*)$ in linear time. The preprocessing time is cubic and quadratic space is needed to store the involved matrices. □

The above derivations show that quadratic time is needed to obtain solutions from scratch for each modified $\bar{\mathbf{y}}$. We will now show how to speed up these computations.

5.3.4 Speed-Ups via Matrix Calculus

The recurrent computation of the objective values in Step 4 of Algorithm 5.2 is still cumbersome if many iterations are needed and restarts are performed (to alleviate the problem of bad local optima). However, it is possible to update these intermediate solutions efficiently since only a single coordinate of an intermediate candidate solution is flipped per iteration. We will now provide the details.

Coordinate Flips Revisited

The next lemma states that one can compute the objective values in Step 4 of Algorithm 5.2 in $\mathcal{O}(n)$ time, which greatly speeds up the overall execution of the local search.

Lemma 5.3 *One can compute $F(\bar{\mathbf{y}})$ in Step 4 of Algorithm 5.2 in $\mathcal{O}(n)$ time for each modified $\bar{\mathbf{y}} \in \{-1,+1\}^n$. Further, $\mathcal{O}(n^3)$ preprocessing time and $\mathcal{O}(n^2)$ space is needed.*

Proof: Let $\bar{\mathbf{y}}$ be the current candidate solution and let $\mathbf{y}$ be its predecessor. Further, let $\bar{\mathbf{c}}^* = \mathbf{\Lambda G \Lambda}\bar{\mathbf{y}}$ denote the corresponding intermediate solution for $\bar{\mathbf{y}}$ determined by

Equation (5.16). Then, one can reformulate the objective value $F(\bar{\mathbf{y}})$ as

$$\begin{aligned} F(\bar{\mathbf{y}}) &= (\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda}\mathbf{K}\bar{\mathbf{c}}^*)^{\mathrm{T}}(\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda}\mathbf{K}\bar{\mathbf{c}}^*) + \lambda(\bar{\mathbf{c}}^*)^{\mathrm{T}}\mathbf{K}\bar{\mathbf{c}}^* \\ &= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{\Lambda}\underbrace{\left(\mathbf{I} - \overline{\mathbf{K}}\mathbf{G} - \mathbf{G}\overline{\mathbf{K}} + \mathbf{G}\overline{\mathbf{K}\mathbf{K}}\mathbf{G} + \lambda\mathbf{G}\overline{\mathbf{K}}\mathbf{G}\right)}_{=:\mathbf{H}}\mathbf{\Lambda}\bar{\mathbf{y}} \end{aligned} \tag{5.19}$$

with $\overline{\mathbf{K}} := \mathbf{\Lambda}\mathbf{K}\mathbf{\Lambda}$. Now assume that $\mathbf{H}\mathbf{\Lambda} \in \mathbb{R}^{n\times n}$ and the auxiliary information $\mathbf{H}\mathbf{\Lambda}\mathbf{y} \in \mathbb{R}^n$ for the predecessor is available (in memory). Then, for a flipped coordinate j, one can update the right hand side of the above term via

$$\mathbf{H}\mathbf{\Lambda}\bar{\mathbf{y}} = \mathbf{H}\mathbf{\Lambda}\mathbf{y} - 2y_j(\mathbf{H}\mathbf{\Lambda})_{[n],\{j\}} \tag{5.20}$$

in $\mathcal{O}(n)$ time, where $(\mathbf{H}\mathbf{\Lambda})_{[n],\{j\}}$ denotes the j-th column of $\mathbf{H}\mathbf{\Lambda}$. Further, since $\mathbf{\Lambda} \in \mathbb{R}^{n\times n}$ is a diagonal matrix and since $\mathbf{H}\mathbf{\Lambda}\bar{\mathbf{y}} \in \mathbb{R}^n$ is a (column) vector, the remaining matrix multiplication to compute $F(\bar{\mathbf{y}})$ can be performed in $\mathcal{O}(n)$ time as well since $\mathbf{\Lambda}$ is a diagonal matrix. Again, cubic preprocessing time as well as quadratic space is needed to precompute the necessary matrices and auxiliary information. □

It is worth pointing out that the derivations depicted above are also applicable if a small number $s \geq 1$ of coordinates is flipped per iteration; this leads to an update time of $\mathcal{O}(sn)$ per iteration. Note that updating $\bar{\mathbf{c}}^* \in \mathbb{R}^n$ in linear time and plugging it directly into the objective (5.18) to compute $F(\bar{\mathbf{y}})$ only leads to an update time of $\mathcal{O}(n^2)$. Further, various other update schemes are possible like updating the two terms of (5.18) individually.

In case the initial partition vector $\mathbf{y} \in \{-1, +1\}^n$ is initialized randomly (e. g., for the unsupervised case), the local search is, in general, susceptible to the problem of local optima. This problem is usually addressed by performing restarts (e. g., taking the best result out of 100 runs). The following theorem shows that the computational shortcut depicted above yields a significant computational speed-up for such scenarios:

Theorem 5.1 *The overall runtime of Algorithm 5.2 is bounded by $\mathcal{O}(n^3 + \mathcal{S}\mathcal{T}n^2)$, where $\mathcal{S}$ is the number of performed restarts and where $\mathcal{T}$ is an upper bound on the number of rounds (induced by the round-robin scheme) needed by each of the runs. Further, $\mathcal{O}(n^2)$ space is needed in total.*

Proof: Due to Lemma 5.3 the Step 4 of Algorithm 5.2 can be performed in $\mathcal{O}(n)$ time since only a single coordinate is flipped per iteration. Thus, each round (i. e., $u \leq n$ consecutive iterations) of the round-robin scheme takes $\mathcal{O}(n^2)$ time. Further, $\mathcal{O}(n^3)$ preprocessing time as well as $\mathcal{O}(n^2)$ space is needed to store all involved matrices. □

Thus, the cubic preprocessing time dominates the runtime as long as $\mathcal{ST} \leq n$. In that case, performing the recurrent local search is essentially for free and is computationally as cheap as training a *single* supervised least-squares support vector machine via the task (5.7) for n labeled patterns.

We would like to point that (a) flipping a single coordinate or a small amount of coordinates is quite reasonable for local search schemes like the one given in Algorithm 5.2 and that (b) the proposed shortcut greatly reduces the overall runtime of such optimization schemes. However, for large-scale learning settings, the cubic preprocessing time and the quadratic storage requirements of the proposed approach depict bottlenecks. In the remainder of this chapter, we will propose two ways to shorten these drawbacks.

Low-Rank Approximation Schemes

We will now describe two schemes that approximate the objective $F(\bar{\mathbf{y}})$ for a given vector $\bar{\mathbf{y}} \in \{-1, +1\}^n$. Both schemes are based on a parameter $r \leq n$ that determines a trade-off between computational savings and accuracy of the approximation. In a nutshell, these approximations reduce the preprocessing time and the storage requirements to $\mathcal{O}(nr^2)$ (in practice and up to machine precision) and $\mathcal{O}(nr)$, respectively. Further, each evaluation of the approximated objective for Step 4 of Algorithm 5.2 can be performed in $\mathcal{O}(r)$ time. This leads to the following theorem:

Theorem 5.2 *By applying one of the two approximation schemes depicted below with $r \leq n$, the total runtime of Algorithm 5.2 is bounded by $\mathcal{O}(nr^2 + \mathcal{ST}nr)$ (in practice and up to machine precision), where $\mathcal{S}$ is the number of restarts and where $\mathcal{T}$ is an upper bound on the number of rounds (induced by the round-robin scheme) needed by each of the runs. Further, $\mathcal{O}(nr)$ space is needed in total.*

Proof: As we will show below, both schemes reduce the preprocessing time as well as the storage requirements. Further, due to the efficient update time of $\mathcal{O}(r)$, each round of the local search now takes $\mathcal{O}(nr)$ time. □

Thus, similar to the non-approximation case, the preprocessing time dominates the runtime until as long as $\mathcal{ST} \leq r$. It remains to show how to integrate the approximation schemes efficiently. As sketched above, both ways essentially yield the same time and space complexities. For the sake of completeness we will provide both alternatives.

Subset of Regressors. A popular choice for accelerating the (supervised) regularized least-squares classification approach is the so-called *subset of regressors* [119] method: Assume that only a subset of the coefficients $c_1, \ldots, c_n$ in Equation (5.11) is allowed to

be nonzero. More precisely, let $R_1 = \{i_1, \ldots, i_{r_1}\} \subseteq \{1, \ldots, l\}$ and $R_2 = \{j_1, \ldots, j_{r_2}\} \subseteq \{l+1, \ldots, n\}$ be subsets of indices that are allowed to be nonzero in Equation (5.11). Then, given a fixed partition vector $\bar{\mathbf{y}} \in \{-1, +1\}^n$, one searches for minimizers of the form

$$\hat{f}(\cdot) = \sum_{\nu=1}^{r_1} c_{i_\nu} k(\cdot, \mathbf{x}'_{i_\nu}) + \sum_{\nu=1}^{r_2} c_{j_\nu} k(\cdot, \mathbf{x}_{j_\nu - l}) \in \mathcal{H}_k \tag{5.21}$$

with coefficients $\hat{\mathbf{c}} = (c_{i_1}, \ldots, c_{i_{r_1}}, c_{j_1}, \ldots, c_{j_{r_2}})^{\mathrm{T}} \in \mathbb{R}^r$, where $r = r_1 + r_2$. The basic intuition behind this approximation is that only few patterns (called *basis vectors* or *regressors*) are needed to represent a reasonable hypothesis space while *all* patterns are taken into account for the evaluation of the overall loss. Hence, one essentially reduces the hypothesis space of possible functions and the parameter r determines the degree of this reduction. This yields new intermediate optimization tasks for Step 4 of Algorithm 5.2 that are of the form

$$\hat{F}(\bar{\mathbf{y}}) = \underset{\hat{\mathbf{c}} \in \mathbb{R}^r}{\text{minimize}}\, \hat{J}(\hat{\mathbf{c}}, \bar{\mathbf{y}}) \tag{5.22}$$

with $\hat{J}(\hat{\mathbf{c}}, \bar{\mathbf{y}})$ given by

$$\hat{J}(\hat{\mathbf{c}}, \bar{\mathbf{y}}) = (\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\hat{\mathbf{c}})^{\mathrm{T}}(\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\hat{\mathbf{c}}) + \lambda \hat{\mathbf{c}}^{\mathrm{T}}\mathbf{K}_{R,R}\hat{\mathbf{c}}, \tag{5.23}$$

where $R = R_1 \cup R_2$. The gradient is given by

$$\nabla_{\hat{\mathbf{c}}}\, \hat{J}(\hat{\mathbf{c}}, \bar{\mathbf{y}}) = -2\mathbf{K}_R\mathbf{\Lambda}(\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\hat{\mathbf{c}}) + 2\lambda\mathbf{K}_{R,R}\hat{\mathbf{c}} \tag{5.24}$$

and the Hessian by

$$\nabla^2_{\hat{\mathbf{c}}}\, \hat{J}(\hat{\mathbf{c}}, \bar{\mathbf{y}}) = 2\mathbf{K}_R\mathbf{\Lambda}\mathbf{\Lambda}(\mathbf{K}_R)^{\mathrm{T}} + 2\lambda\mathbf{K}_{R,R} \succeq 0. \tag{5.25}$$

Since the Hessian is positive definite, this new task is convex as well. Thus,

$$\nabla_{\hat{\mathbf{c}}}\, \hat{J}(\hat{\mathbf{c}}, \bar{\mathbf{y}}) = -2\mathbf{K}_R\mathbf{\Lambda}(\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\hat{\mathbf{c}}) + 2\lambda\mathbf{K}_{R,R}\hat{\mathbf{c}} \stackrel{!}{=} \mathbf{0} \tag{5.26}$$

is a necessary and sufficient condition for optimality and leads to the optimal solution given by

$$\hat{\mathbf{c}}^* = (\mathbf{K}_R\mathbf{\Lambda}^2(\mathbf{K}_R)^{\mathrm{T}} + \lambda\mathbf{K}_{RR})^{-1}\mathbf{K}_R\mathbf{\Lambda}\mathbf{\Lambda}\bar{\mathbf{y}} = \widehat{\mathbf{G}}\mathbf{K}_R\mathbf{\Lambda}\mathbf{\Lambda}\bar{\mathbf{y}} \tag{5.27}$$

with $\widehat{\mathbf{G}} := (\mathbf{K}_R\mathbf{\Lambda}^2(\mathbf{K}_R)^{\mathrm{T}} + \lambda\mathbf{K}_{RR})^{-1}$.[7] Note that, since the matrix $\mathbf{K}_R\mathbf{\Lambda}^2(\mathbf{K}_R)^{\mathrm{T}}$ is positive definite and since $\lambda\mathbf{K}_{RR}$ is strictly positive definite, the sum of both matrices is strictly

[7] We will assume $|R|$ to be small enough and R to be selected appropriately such that $\mathbf{K}_{R,R}$ is strictly positive definite (which is a standard assumption for this approximation scheme [120, p. 107]). In case this is not possible, one can replace $\mathbf{K}_{R,R}$ by the shifted variant $\mathbf{K}_{R,R} + \delta\mathbf{I}$, where δ is a small positive constant, called *jitter factor* [147]. This guarantees the strictly positive definiteness of the modified matrix $\mathbf{K}_{R,R}$.

positive definite and therefore invertible. The following lemma shows that the above approximation scheme can be efficiently integrated into the framework.

Lemma 5.4 *The subset of regressors scheme with $R_1 = \{i_1, \ldots, i_{r_1}\} \subseteq \{1, \ldots, l\}$, $R_2 = \{j_1, \ldots, j_{r_2}\} \subseteq \{l+1, \ldots, n\}$ and $r = r_1 + r_2$ leads to a preprocessing time of $\mathcal{O}(nr^2)$ (in practice and up to machine precision) and to a space consumption of $\mathcal{O}(nr)$. In addition, the computation of the approximated objective values $\hat{F}(\bar{\mathbf{y}})$ for Step 4 of Algorithm 5.2 can be performed in $\mathcal{O}(r)$ time per iteration.*

Proof: One can reformulate the new objective as

$$\begin{aligned}\hat{F}(\bar{\mathbf{y}}) &= (\boldsymbol{\Lambda}\bar{\mathbf{y}} - \boldsymbol{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\hat{\mathbf{c}}^*)^{\mathrm{T}}(\boldsymbol{\Lambda}\bar{\mathbf{y}} - \boldsymbol{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\hat{\mathbf{c}}^*) + \lambda(\hat{\mathbf{c}}^*)^{\mathrm{T}}\mathbf{K}_{R,R}\hat{\mathbf{c}}^* \\ &= 1 + \gamma - 2\bar{\mathbf{y}}^{\mathrm{T}}\boldsymbol{\Lambda}^2(\mathbf{K}_R)^{\mathrm{T}}\hat{\mathbf{c}}^* + (\boldsymbol{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\hat{\mathbf{c}}^*)^{\mathrm{T}}\boldsymbol{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\hat{\mathbf{c}}^* + \lambda(\hat{\mathbf{c}}^*)^{\mathrm{T}}\mathbf{K}_{R,R}\hat{\mathbf{c}}^*,\end{aligned}$$

where we used $(\boldsymbol{\Lambda}\bar{\mathbf{y}})^{\mathrm{T}}\boldsymbol{\Lambda}\bar{\mathbf{y}} = 1 + \gamma$ and $(\boldsymbol{\Lambda}\bar{\mathbf{y}})^{\mathrm{T}}\boldsymbol{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\hat{\mathbf{c}}^* = (\boldsymbol{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\hat{\mathbf{c}}^*)^{\mathrm{T}}\boldsymbol{\Lambda}\bar{\mathbf{y}}$. Further, one can simplify the above objective via

$$\begin{aligned}\hat{F}(\bar{\mathbf{y}}) &= 1 + \gamma - \bar{\mathbf{y}}^{\mathrm{T}}\boldsymbol{\Lambda}^2(\mathbf{K}_R)^{\mathrm{T}}\left(2\widehat{\mathbf{G}}\widehat{\mathbf{G}}^{-1}\widehat{\mathbf{G}} - \widehat{\mathbf{G}}\mathbf{K}_R\boldsymbol{\Lambda}\boldsymbol{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\widehat{\mathbf{G}} - \lambda\widehat{\mathbf{G}}\mathbf{K}_{R,R}\widehat{\mathbf{G}}\right)\mathbf{K}_R\boldsymbol{\Lambda}^2\bar{\mathbf{y}} \\ &= 1 + \gamma - \bar{\mathbf{y}}^{\mathrm{T}}\boldsymbol{\Lambda}^2(\mathbf{K}_R)^{\mathrm{T}}\left(\widehat{\mathbf{G}}\left(2\widehat{\mathbf{G}}^{-1} - \mathbf{K}_R\boldsymbol{\Lambda}\boldsymbol{\Lambda}(\mathbf{K}_R)^{\mathrm{T}} - \lambda\mathbf{K}_{R,R}\right)\widehat{\mathbf{G}}\right)\mathbf{K}_R\boldsymbol{\Lambda}^2\bar{\mathbf{y}} \\ &= 1 + \gamma - \bar{\mathbf{y}}^{\mathrm{T}}\boldsymbol{\Lambda}^2(\mathbf{K}_R)^{\mathrm{T}}\left(\widehat{\mathbf{G}}\left(\mathbf{K}_R\boldsymbol{\Lambda}^2(\mathbf{K}_R)^{\mathrm{T}} + \lambda\mathbf{K}_{R,R}\right)\widehat{\mathbf{G}}\right)\mathbf{K}_R\boldsymbol{\Lambda}^2\bar{\mathbf{y}} \\ &= 1 + \gamma - \bar{\mathbf{y}}^{\mathrm{T}}\boldsymbol{\Lambda}^2(\mathbf{K}_R)^{\mathrm{T}}\left(\widehat{\mathbf{G}}\left(\widehat{\mathbf{G}}\right)^{-1}\widehat{\mathbf{G}}\right)\mathbf{K}_R\boldsymbol{\Lambda}^2\bar{\mathbf{y}},\end{aligned}$$

where the last two steps are due to the definition of $\widehat{\mathbf{G}}$. Now, let $\widehat{\mathbf{G}} = \mathbf{B}\mathbf{B}^{\mathrm{T}} \in \mathbb{R}^{r\times r}$ be the *Cholesky decomposition* [69, p. 143] of the strictly positive definite matrix $\widehat{\mathbf{G}}$ with lower triangular matrix $\mathbf{B} \in \mathbb{R}^{r\times r}$ and let

$$(\mathbf{K}_R)^{\mathrm{T}}\mathbf{B} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}} \in \mathbb{R}^{n\times r} \tag{5.28}$$

be the *thin singular value decomposition* [69, p. 72] of $(\mathbf{K}_R)^{\mathrm{T}}\mathbf{B}$ with $\mathbf{U} \in \mathbb{R}^{n\times r}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{r\times r}$, and $\mathbf{V} \in \mathbb{R}^{r\times r}$. Then one can reformulate the objective as

$$\begin{aligned}\hat{F}(\bar{\mathbf{y}}) &= 1 + \gamma - \bar{\mathbf{y}}^{\mathrm{T}}\boldsymbol{\Lambda}^2(\mathbf{K}_R)^{\mathrm{T}}\widehat{\mathbf{G}}\mathbf{K}_R\boldsymbol{\Lambda}^2\bar{\mathbf{y}} \\ &= 1 + \gamma - \bar{\mathbf{y}}^{\mathrm{T}}\boldsymbol{\Lambda}^2(\mathbf{K}_R)^{\mathrm{T}}\mathbf{B}\mathbf{B}^{\mathrm{T}}\mathbf{K}_R\boldsymbol{\Lambda}^2\bar{\mathbf{y}} \\ &= 1 + \gamma - \bar{\mathbf{y}}^{\mathrm{T}}\boldsymbol{\Lambda}^2\mathbf{U}\boldsymbol{\Sigma}^2\mathbf{U}^{\mathrm{T}}\boldsymbol{\Lambda}^2\bar{\mathbf{y}}.\end{aligned} \tag{5.29}$$

Similar to the non-approximation case one can now update the term $\mathbf{U}^{\mathrm{T}}\mathbf{\Lambda}^2\bar{\mathbf{y}}$ via

$$\mathbf{U}^{\mathrm{T}}\mathbf{\Lambda}^2\bar{\mathbf{y}} = \mathbf{U}^{\mathrm{T}}\mathbf{\Lambda}^2\mathbf{y} - 2y_j(\mathbf{U}^{\mathrm{T}}\mathbf{\Lambda}^2)_{[r],\{j\}}, \tag{5.30}$$

where $\mathbf{y}$ is the predecessor of $\bar{\mathbf{y}}$ and where j is the flipped coordinate. This update can be performed in $\mathcal{O}(r)$ time since $\mathbf{U}^{\mathrm{T}}\mathbf{\Lambda}^2 \in \mathbb{R}^{r\times n}$. Further, one can obtain $\hat{F}(\bar{\mathbf{y}})$ in $\mathcal{O}(r)$ per iteration because $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$ is a diagonal matrix. From a computational point of view, the computation of the Cholesky decomposition can be performed in $\mathcal{O}(r^3)$ time [69, p. 145] and the computation of the thin singular value decomposition in $\mathcal{O}(nr^2)$ time [69, p. 254] (in practice and up to machine precision), respectively. The space requirement for storing all matrices and vectors is $\mathcal{O}(nr)$. □

Thus, the efficient integration of the subset of regressors method is based on making use of the low-rank nature of the involved matrices. In the following, we will depict an alternative way to obtain such a speed-up. It should be pointed out that the rather longish derivations depicted above can be simplified in case an update time of $\mathcal{O}(n)$ is sufficient. However, depending on the task at hand, the parameter r can be significantly smaller than n. For such settings, the update time of $\mathcal{O}(r)$ greatly reduces the overall runtime.

Nyström Approximation. Another way to obtain the above runtime and space bounds is based on the *Nyström approximation* [120, 147] that replaces the kernel matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$ by

$$\widetilde{\mathbf{K}} := (\mathbf{K}_R)^{\mathrm{T}}(\mathbf{K}_{R,R})^{-1}\mathbf{K}_R \in \mathbb{R}^{n\times n}, \tag{5.31}$$

where $R \subseteq \{1,\ldots,n\} = [n]$ is a subset of indices.[8] Aiming at the efficient incorporation of this approximation scheme, let us consider the eigendecomposition [69]

$$\overline{\mathbf{K}} = \mathbf{V}\mathbf{D}\mathbf{V}^{\mathrm{T}} \tag{5.32}$$

of the real symmetric (positive definite) matrix $\overline{\mathbf{K}} = \mathbf{\Lambda}\mathbf{K}\mathbf{\Lambda}$ with orthogonal matrix $\mathbf{V} \in \mathbb{R}^{n\times n}$ and diagonal matrix $\mathbf{D} \in \mathbb{R}^{n\times n}$. One can then rewrite the original objective $F(\bar{\mathbf{y}})$ given by (5.19) as

$$\begin{aligned} F(\bar{\mathbf{y}}) &= (\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda}\mathbf{K}\bar{\mathbf{c}}^*)^{\mathrm{T}}(\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda}\mathbf{K}\bar{\mathbf{c}}^*) + \lambda(\bar{\mathbf{c}}^*)^{\mathrm{T}}\mathbf{K}\bar{\mathbf{c}}^* \\ &= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{\Lambda}\left(\mathbf{I} - \overline{\mathbf{K}}\mathbf{G} - \mathbf{G}\overline{\mathbf{K}} + \mathbf{G}\overline{\mathbf{K}}\overline{\mathbf{K}}\mathbf{G} + \lambda\mathbf{G}\overline{\mathbf{K}}\mathbf{G}\right)\mathbf{\Lambda}\bar{\mathbf{y}} \\ &= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{V}\left(\mathbf{I} - 2\mathbf{D}\mathbf{D}_\lambda + \mathbf{D}^2{\mathbf{D}_\lambda}^2 + \lambda\mathbf{D}{\mathbf{D}_\lambda}^2\right)\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}} \end{aligned} \tag{5.33}$$

where $\mathbf{D}_\lambda := (\mathbf{D} + \lambda\mathbf{I})^{-1}$. The last equation needs further explanations: Since the matrix $\mathbf{V} \in \mathbb{R}^{n\times n}$ is orthogonal, we have $\mathbf{V}\mathbf{V}^{\mathrm{T}} = \mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I}$ and, thus, $\mathbf{V}^{-1} = \mathbf{V}^{\mathrm{T}}$ [69]. Hence,

[8] Again, we assume the matrix $\mathbf{K}_{R,R}$ to be a strictly positive definite.

one can rewrite $\mathbf{G}$ as

$$\mathbf{G} = (\overline{\mathbf{K}} + \lambda\mathbf{I})^{-1} = (\mathbf{V}\mathbf{D}\mathbf{V}^{\mathrm{T}} + \lambda\mathbf{V}\mathbf{V}^{\mathrm{T}})^{-1} = (\mathbf{V}(\mathbf{D} + \lambda)\mathbf{V}^{\mathrm{T}})^{-1} = \mathbf{V}\mathbf{D}_{\lambda}\mathbf{V}^{\mathrm{T}}, \tag{5.34}$$

where we used the fact that $(\mathbf{A}_1\mathbf{A}_2)^{-1} = \mathbf{A}_2^{-1}\mathbf{A}_1^{-1}$ holds for two invertible matrices $\mathbf{A}_1 \in \mathbb{R}^{n\times n}$ and $\mathbf{A}_2 \in \mathbb{R}^{n\times n}$ [113]. By using the above equality, one can reformulate

$$\begin{aligned}
\overline{\mathbf{K}}\mathbf{G} &= \mathbf{V}\mathbf{D}\mathbf{V}^{\mathrm{T}}\mathbf{V}\mathbf{D}_{\lambda}\mathbf{V}^{\mathrm{T}} = \mathbf{V}\mathbf{D}\mathbf{D}_{\lambda}\mathbf{V}^{\mathrm{T}},\\
\mathbf{G}\overline{\mathbf{K}} &= \mathbf{V}\mathbf{D}_{\lambda}\mathbf{V}^{\mathrm{T}}\mathbf{V}\mathbf{D}\mathbf{V}^{\mathrm{T}} = \mathbf{V}\mathbf{D}_{\lambda}\mathbf{D}\mathbf{V}^{\mathrm{T}},\\
\mathbf{G}\overline{\mathbf{K}}\overline{\mathbf{K}}\mathbf{G} &= \mathbf{V}\mathbf{D}_{\lambda}\mathbf{V}^{\mathrm{T}}\mathbf{V}\mathbf{D}\mathbf{V}^{\mathrm{T}}\mathbf{V}\mathbf{D}\mathbf{V}^{\mathrm{T}}\mathbf{V}\mathbf{D}_{\lambda}\mathbf{V}^{\mathrm{T}} = \mathbf{V}\mathbf{D}_{\lambda}\mathbf{D}\mathbf{D}\mathbf{D}_{\lambda}\mathbf{V}^{\mathrm{T}},\\
\mathbf{G}\overline{\mathbf{K}}\mathbf{G} &= \mathbf{V}\mathbf{D}_{\lambda}\mathbf{V}^{\mathrm{T}}\mathbf{V}\mathbf{D}\mathbf{V}^{\mathrm{T}}\mathbf{V}\mathbf{D}_{\lambda}\mathbf{V}^{\mathrm{T}} = \mathbf{V}\mathbf{D}_{\lambda}\mathbf{D}\mathbf{D}_{\lambda}\mathbf{V}^{\mathrm{T}},
\end{aligned}$$

and, thus, obtains Equation (5.33). Since the involved diagonal matrices commute, one can further decompose this objective as:

$$\begin{aligned}
F(\bar{\mathbf{y}}) &= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{V}\left(\mathbf{I} - 2\mathbf{D}\mathbf{D}_{\lambda} + \mathbf{D}^2{\mathbf{D}_{\lambda}}^2 + \lambda\mathbf{D}{\mathbf{D}_{\lambda}}^2\right)\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}}\\
&= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{V}\left(\mathbf{I} + (-2\mathbf{I} + \mathbf{D}_{\lambda}\mathbf{D} + \lambda\mathbf{D}_{\lambda})\mathbf{D}\mathbf{D}_{\lambda}\right)\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}}\\
&= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{V}\left(\mathbf{I} + (-2\mathbf{I} + \mathbf{D}_{\lambda}(\mathbf{D} + \lambda\mathbf{I}))\mathbf{D}\mathbf{D}_{\lambda}\right)\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}}\\
&= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{V}\left(\mathbf{I} + (-2\mathbf{I} + \mathbf{I})\mathbf{D}\mathbf{D}_{\lambda}\right)\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}}\\
&= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{V}\left(\mathbf{I} - \mathbf{D}\mathbf{D}_{\lambda}\right)\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}}\\
&= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{\Lambda}^2\bar{\mathbf{y}} - \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{V}\mathbf{D}\mathbf{D}_{\lambda}\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}}\\
&= 1 + \gamma - \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{V}\mathbf{D}\mathbf{D}_{\lambda}\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}}
\end{aligned} \tag{5.35}$$

Hence, given the vector $\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}}$, one can compute $F(\bar{\mathbf{y}})$ in $\mathcal{O}(n)$ time since $\mathbf{D}\mathbf{D}_{\lambda}$ is a diagonal matrix. Further, this auxiliary information can also be updated in linear time given the information for the predecessor $\mathbf{y}$ of $\bar{\mathbf{y}}$ via

$$\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}} = \mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{y} - 2y_j\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}_{[n],\{j\}}. \tag{5.36}$$

Thus, the above scheme depicts an alternative way to update the objective value $F(\bar{\mathbf{y}})$, in addition to the one given by Equation (5.19). However, this new scheme also permits an efficient integration of the Nyström approximation:

Lemma 5.5 *The replacement of the kernel matrix $\mathbf{K}$ by its Nyström approximation $\widetilde{\mathbf{K}} = (\mathbf{K}_R)^{\mathrm{T}}(\mathbf{K}_{R,R})^{-1}\mathbf{K}_R$ with $R \subseteq \{1,\ldots,n\} = [n]$ and $r = |R|$ leads to a preprocessing time of $\mathcal{O}(nr^2)$ (in practice and up to machine precision) and to a space consumption of $\mathcal{O}(nr)$. In addition, the computation of the corresponding approximated objective values in Step 4 of Algorithm 5.2 can be performed in $\mathcal{O}(r)$ time per iteration.*

Proof: To obtain an efficient update scheme, one has to make use of the low-rank nature of the matrix $\widetilde{\mathbf{K}} = (\mathbf{K}_R)^{\mathrm{T}}(\mathbf{K}_{R,R})^{-1}\mathbf{K}_R \in \mathbb{R}^{n\times n}$. More specifically, one can compute the nonzero eigenvalues and corresponding eigenvectors of the modified matrix $\overline{\mathbf{K}} = \mathbf{\Lambda}\widetilde{\mathbf{K}}\mathbf{\Lambda}$ in the following manner: Let

$$(\mathbf{K}_{R,R})^{-1} = \mathbf{B}\mathbf{B}^{\mathrm{T}} \in \mathbb{R}^{r\times r} \tag{5.37}$$

be the Cholesky decomposition [69, p. 143] of the strictly positive definite matrix $(\mathbf{K}_{R,R})^{-1}$ with lower triangular matrix $\mathbf{B} \in \mathbb{R}^{r\times r}$ and let

$$\mathbf{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\mathbf{B} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^{\mathrm{T}} \in \mathbb{R}^{n\times r} \tag{5.38}$$

be the thin singular value decomposition [69, p. 72] with $\mathbf{V} \in \mathbb{R}^{n\times r}$, $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$, and $\mathbf{U} \in \mathbb{R}^{r\times r}$.[9] The r nonzero eigenvalues of

$$\overline{\mathbf{K}} = \mathbf{\Lambda}\widetilde{\mathbf{K}}\mathbf{\Lambda} = \mathbf{\Lambda}(\mathbf{K}_R)^{\mathrm{T}}\mathbf{B}\mathbf{B}^{\mathrm{T}}\mathbf{K}_R\mathbf{\Lambda} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^{\mathrm{T}}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^{\mathrm{T}} \tag{5.39}$$

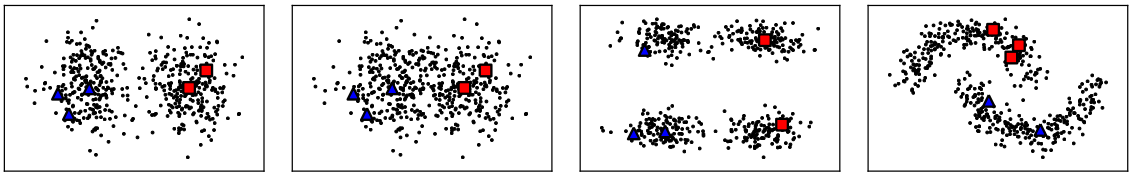
can then be obtained from $\mathbf{\Sigma}^2 \in \mathbb{R}^{r\times r}$ and the matrix $\mathbf{V} \in \mathbb{R}^{n\times r}$ consists of the corresponding eigenvectors (note that we have $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$ [69, 121]).

Again, the computation of the Cholesky decomposition can be performed in $\mathcal{O}(r^3)$ time [69, p. 145] and the computation of the thin singular value decomposition in $\mathcal{O}(nr^2)$ time [69, p. 254], respectively. Thus, due to the low-rank nature of the matrix $\overline{\mathbf{K}}$, the nonzero eigenvalues and corresponding eigenvectors of $\overline{\mathbf{K}}$ can be obtained in $\mathcal{O}(nr^2)$ time (in practice and up to machine precision) using $\mathcal{O}(nr)$ space. Now assume that these r nonzero eigenvalues are the first elements of the diagonal matrix $\mathbf{D} \in \mathbb{R}^{n\times n}$ defined in (5.32) and that the remaining entries of $\mathbf{D}$ are zero. Then, we have $[\mathbf{D}\tilde{\mathbf{D}}]_{i,i} = 0$ for $i = r+1,\ldots,n$ for the two diagonal matrices occurring in Equation (5.35). Thus, considering the computational shortcut (5.35), the remaining eigenvectors with zero eigenvalue (extending the matrix $\mathbf{V}$) do not have to be computed since the last $n-r$ rows of $\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}} \in \mathbb{R}^{n\times 1}$ are multiplied by 0. In other words, it is sufficient to compute only the eigenvectors given in $\mathbf{V} \in \mathbb{R}^{n\times r}$ corresponding to nonzero eigenvalues and to update

$$\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}} = \mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\mathbf{y} - 2y_j(\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda})_{[n],\{j\}} \tag{5.40}$$

in $\mathcal{O}(r)$ time per iteration, where $\mathbf{y}$ is the predecessor of $\bar{\mathbf{y}}$. Thus, all preprocessing matrices can be obtained in $\mathcal{O}(nr^2)$ time (in practice and up to machine precision) using $\mathcal{O}(nr)$ space. Further, the auxiliary information $\mathbf{V}^{\mathrm{T}}\mathbf{\Lambda}\bar{\mathbf{y}}$ can be updated in $\mathcal{O}(r)$ time per single coordinate flip. □

[9] Since the matrix $\mathbf{K}_{R,R}$ is strictly positive definite, its inverse is also strictly positive definite [69].



(a) `Gaussian5` (b) `Gaussian6` (c) `Gaussian7` (d) `Moons`

FIGURE 5.1. *The four artificial data sets used for the experimental evaluation. The red squares and blue triangles depict the labeled patterns and the black dots the unlabeled ones, respectively.*

To sum up, the key observation of the above derivations is the fact that one can reformulate $F(\bar{\mathbf{y}})$ in terms of Equation (5.35) and that one can efficiently compute the r nonzero eigenvalues of $\overline{\mathbf{K}} = \mathbf{\Lambda}\widetilde{\mathbf{K}}\mathbf{\Lambda}$ in $\mathcal{O}(nr^2)$ time using $\mathcal{O}(nr)$ space via the thin singular value decomposition.

5.4 Experimental Analysis

In the following, we will analyze several properties of the proposed local search scheme. For the sake of exposition we will mostly focus on semi-supervised learning settings and will only sketch unsupervised ones. Further, we will defer the comparison with competing semi-supervised methods to Chapter 6.

5.4.1 Experimental Setup

We start by describing the experimental setup including a description of the considered data sets, implementation details, and model selection issues.

Data Sets

In addition to the data sets described in Chapter 4, we will consider additional artificial and real-world data sets to demonstrate certain properties of the optimization framework.

Artificial Data. We extend the artificial data sets presented in Chapter 4 to higher dimensions: The variant of the `Gaussian1` data set, called `Gaussian5` data set, is now based on two Gaussian clusters generated by drawing $N/2$ points from each of two multivariate Gaussian distributions $X_i \sim \mathcal{N}(\mathbf{m}_i, \mathbf{I})$, where $\mathbf{m}_1 = (-2.5, 0.0, \ldots, 0.0) \in \mathbb{R}^d$ and $\mathbf{m}_2 = (+2.5, 0.0, \ldots, 0.0) \in \mathbb{R}^d$. Again, the class label of a point corresponds to the distribution it was drawn from, see Figure 5.1 (a). A similar extension to high dimensions is performed for the old `Gaussian2` data set, now called `Gaussian6`, see Figure 5.1 (b). Further, we consider a variant of the old `Gaussian3` data set, which exhibits a misleading

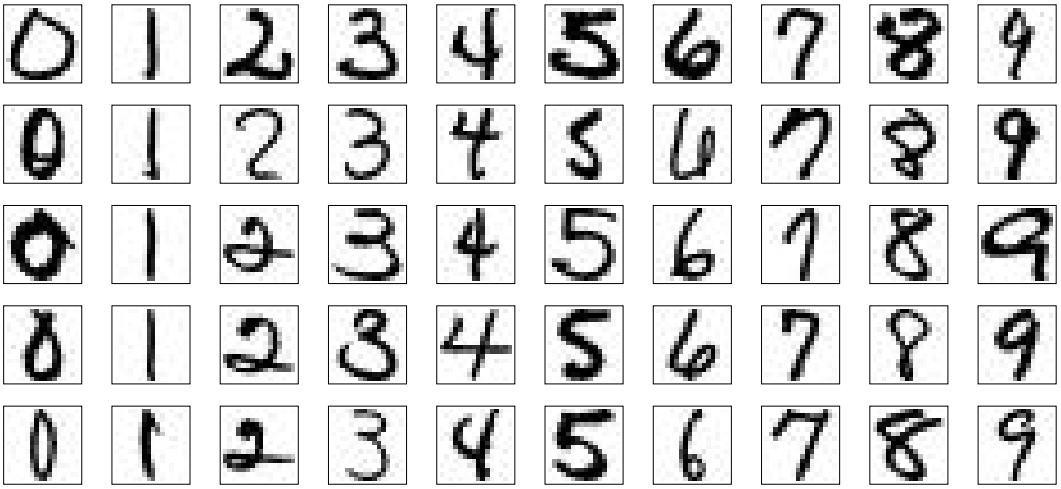


FIGURE 5.2. *The* `USPS` *data set [42, 73] containing images of handwritten digits. Each digit is represented by a* 16×16 *grayscale image (*8 *bits).*

structure for semi-supervised learning tasks: Here, $N/4$ points are drawn from each of four multivariate Gaussian distributions $X_i \sim \mathcal{N}(\mathbf{m}_i, I)$ with:

$$\mathbf{m}_1 = (-2.5, -5.0, 0.0, \ldots, 0.0) \in \mathbb{R}^d \qquad \mathbf{m}_2 = (-2.5, +5.0, 0.0, \ldots, 0.0) \in \mathbb{R}^d$$
$$\mathbf{m}_3 = (+2.5, -5.0, 0.0, \ldots, 0.0) \in \mathbb{R}^d \qquad \mathbf{m}_4 = (+2.5, +5.0, 0.0, \ldots, 0.0) \in \mathbb{R}^d$$

The points drawn from the first two distributions belong to the first class and the remaining one to the second class. Thus, the best pure clustering partition might differ from the pure desired classification partition. We denote this new data set by `Gaussian7`, see Figure 5.1 (c). If not noted otherwise, we use $N = 500$ and $d = 500$ for these three data sets. Finally, we consider the well-known two-dimensional `Moons` data set with $N = 500$ points, see Figure 5.1 (d).

Real-World Data. We consider the `USPS` data set as real-world example in this chapter (consisting of both the training and test set of the original data set [42, 73]) and use `USPS(i,j)` to denote the binary classification task induced by the classes (i. e., numbers) i and j. Further, we rescale the pixels such that the resulting values lie between 0.0 and 1.0. Each digit is represented as a grayscale image (8 bits) with 16×16 pixels ($d = 256$).

Implementation Details

We use the programming language `Python` to implement the local search, where the time-consuming (matrix) operations are implemented via the `Numpy` package. Further, we con-

sider the round-robin scheme to select the coordinate j per iteration and stop the iterative process if no changes have occured for u consecutive iterations. The initial candidate solutions are generated randomly or initialized via a supervised regularized least-squares classification model trained on the labeled patterns, depending on the context. Further, we resort to the subset of regressors method as approximation scheme. We call the resulting approaches *unsupervised regularized least-squares classification* (`URLSC`) and *semi-supervised regularized least-squares classification* (`S`^{2}`RLSC`), respectively.

Model Selection

Both the semi- and the unsupervised task require parameters to be set by the user (e. g., the regularization parameter $\lambda \in \mathbb{R}^+$). In the context of semi- and unsupervised learning, this is in general a quite sensitive issue, as we will describe next.

Lack of Labeled Data. In supervised learning settings, model selection, i. e., assigning reasonable values to the non-fixed parameters of the considered model, is usually performed via cross-validation and grid search on the training set [73]. The final model is then evaluated on the test set, which has not been used during the training phase. However, in realistic semi-supervised learning settings, one usually does not have sufficient labeled data for reliably selecting an appropriate model and this model selection problem is widely considered to be an open issue [34]. The situation is even more severe for unsupervised classification settings since one does not have any labels at all and "cross-validation techniques, so useful for model selection in supervised learning, cannot be utilized in this context" [73, p. 519]. Thus, for unsupervised settings, one has to fix the parameters manually to reasonable values or one has to resort to other measures for evaluating the quality of the computed model [73].

Non-Realistic Scenario. In this chapter, we will therefore consider a *non-realistic scenario* to select the non-fixed model parameters. More specifically, we will make use of the labels given in the test set to tune the parameters and to evaluate the classification performance. The reason for this non-realistic scenario is the fact that, by making use of the test set (with a large amount of labeled patterns), one can evaluate the flexibility of the model, i. e., one can investigate if the model is in principle capable of adapting to the inherent structure of the data. We would like to point out that this non-realistic scenario is often used in the literature for the evaluation of both semi- and unsupervised approaches [34, 149]. Note that, even given a perfect assignment for the involved parameters (provided via an *oracle* or test set), one still has to address the combinatorial tasks. Thus, for the time being, we will ignore these model selection problems by resorting to

the non-realistic scenario; however, a corresponding *realistic scenario* will be the basis for the comprehensive comparison provided in Chapter 6.

Training and Test Set. Each data set consists of N elements. If not noted otherwise, the first half of each data set is used as training and the second half as test set. To induce unsupervised and semi-supervised learning settings, we split up the training set into a labeled and an unlabeled part and use different ratios for the particular setting. The specific amount of patterns is given in brackets for each instance of a data set, where l, u, and t denote the numbers of labeled, unlabeled, and test patterns, respectively (e. g., `Gaussian5[l=25,u=225,t=250]`). If not noted otherwise, we consider 10 random partitions (into these three parts) for a particular experiment and report the average classification error on the test set (along with the one standard deviation).

Model Parameters. We first tune the non-fixed parameters via grid search and subsequently retrain the final model on the training set with the best performing set of parameters. To simplify the experimental setup, we consider a linear kernel as similarity measure for most of the experiments. Further, if not stated otherwise, we set the balance constraint parameters ε to 0.1 and b_c to the (true) ratio estimated on the whole data set, respectively. The cost parameters λ and γ are tuned on a small grid $(\lambda, \gamma) \in \{2^{-10}, \ldots, 2^{10}\} \times \{0.01, 1, 100\}$ of possible assignments. Local search schemes are (in general) susceptible to the problem of local optima. Thus, if not noted otherwise, we take the best out of 25 runs during model selection (i. e., during grid search) and the best out of 200 runs for generating the final model, in both cases with respect to the objective value. As a baseline we consider the regularized least-squares classification approach (`RLSC`) trained on the labeled part of the data and tune the parameter $\lambda \in \mathbb{R}^+$ in the same manner (non-realistic scenario) with $\lambda \in \{2^{-10}, \ldots, 2^{10}\}$.

5.4.2 Semi-Supervised Learning Settings

We will now depict several properties of the local search scheme in the context of semi-supervised learning settings including its dependence on the amount of used labeled and unlabeled data, its performance in classification settings, and its computational behavior.

Amount of Data

As shown in Chapter 3, sufficient labeled data are essential for supervised learning approaches to yield good models. However, for semi- and unsupervised learning schemes, the amount of unlabeled data used for training is an important issue as well. To analyze how much labeled and unlabeled data are needed, we consider the three high-dimensional

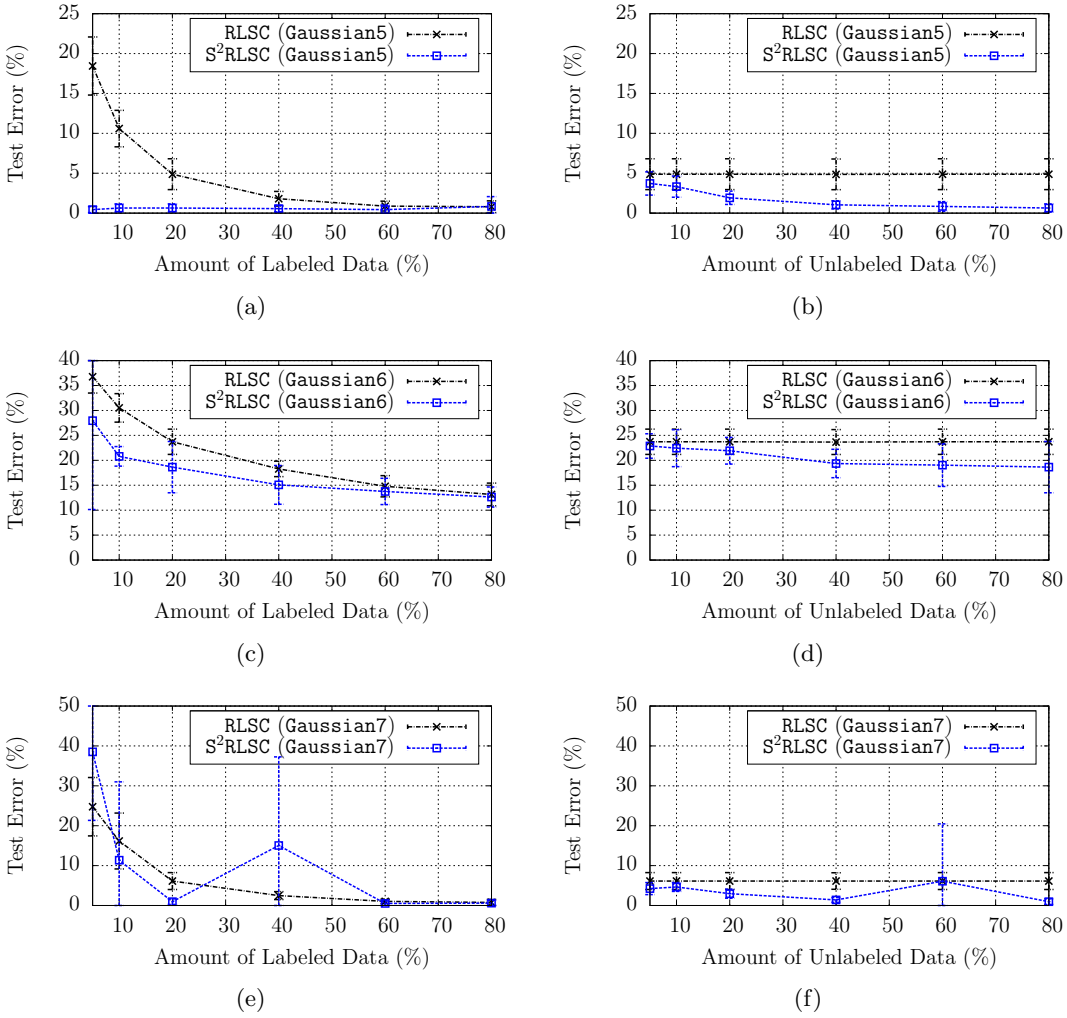


FIGURE 5.3. *If not sufficient labeled data are available, the supervised approach (`RLSC`) fails to yield good models on both data sets. Our semi-supervised approach (`S`^{2}`RLSC`) can successfully incorporate unlabeled data to improve the generalization performance, see Figures (a) and (c). The `Gaussian7` data set exhibits a misleading structure and is, thus, difficult to address by the local search scheme. Sufficient unlabeled data are needed as well for the semi-supervised approach, see Figures (b), (d), and (f).*

artificial data sets and vary the amount of labeled and unlabeled data. Further, we use the supervised `RLSC` approach trained on the labeled part of the data as baseline. Note that, while the induced classification tasks are quite simple to approach for $d = 2$ even in case only few labeled patterns are given, they depict challenging learning tasks for supervised methods in higher dimensions (given only few labeled patterns).

For the first experiment we vary the amount of labeled patterns from 5% to 80% with respect to the size of the training set (i.e., $N/2 = 250$). The remaining training patterns are used as unlabeled data (for each setting, we report the test error averaged over 10 random partitions). In Figures 5.3 (a), (c), and (e), the result of this experiment is

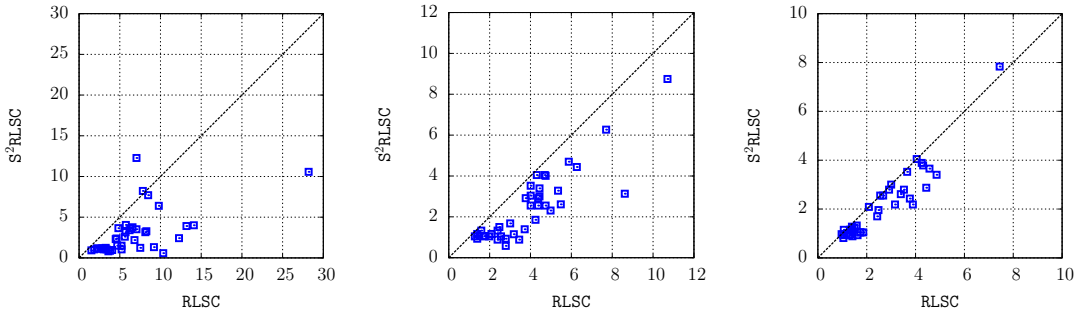


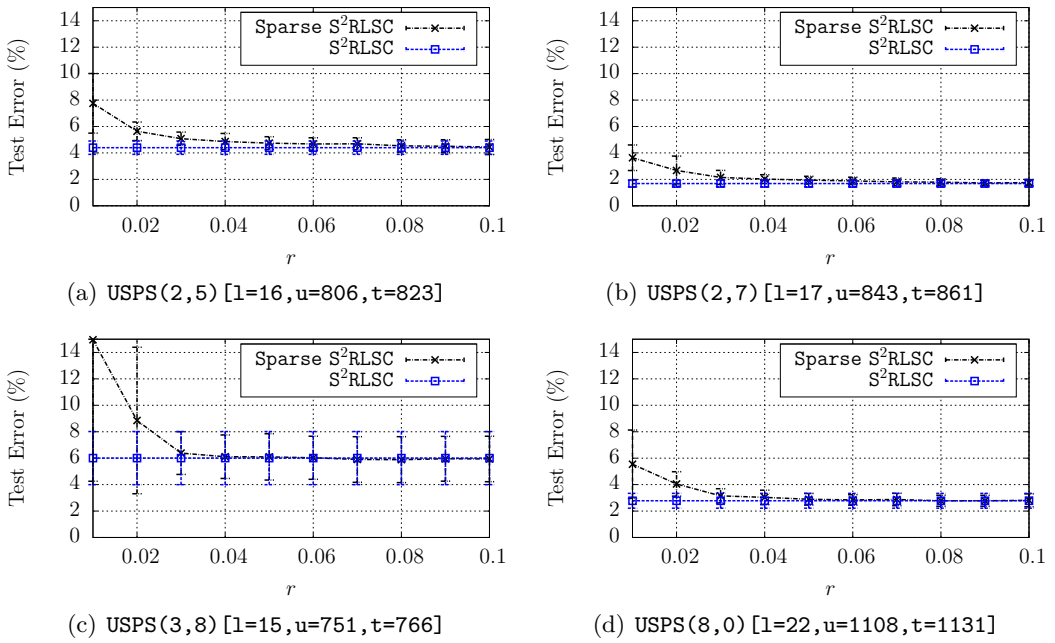
FIGURE 5.4. *All three plots depict the classification performance on the four binary tasks* `USPS(2,5)`, `USPS(2,7)`, `USPS(3,8)`, *and* `USPS(8,0)`, *both for* `RLSC` *and* S^2`RLSC`. *The amount of labeled data is increased from (a)* 1%, *(b)* 2%, *and (c)* 5%; *half of the patterns are used as test set, the remaining ones as unlabeled patterns (in the training set). For each task, the average test error is reported resulting in* 40 *points. The performance of the semi-supervised approach is clearly better compared to the one of the supervised approach (points below the line correspond a better performance of* S^2`RLSC`*).*

shown: For both the `Gaussian5` and the `Gaussian6` data set, the semi-supervised approach performs significantly better compared to the supervised one. For the `Gaussian7` data set, however, the performance of the semi-supervised approach is worse. This is due to the fact that the data set exhibits a misleading structure: The pure clustering solution corresponds to a (reasonable) local optimum and leads to a classification error of about 50%. Thus, for this data set, such a single (undesired) clustering solution can jeopardize the overall averaged classification error.

For the second experiment we fix the amount of labeled patterns to 20% and vary the amount of unlabeled patterns from 5% to 80%, again with respect to the size of the training set, see Figures 5.3 (b), (d), and (f). Clearly, the semi-supervised approach is only capable of generating an appropriate model once sufficient unlabeled data are given. To conclude, given only a small amount of labeled but a large amount of unlabeled patterns, our semi-supervised approach can help to improve the classification performance compared to the supervised `RLSC` baseline. However, incorporating unlabeled data might also lead to worse results (as it is sometimes the case for the `Gaussian7` data set).

Classification Performance

We will briefly sketch the classification performance on real-world data; a detailed comparison with competing approaches will be provided in Chapter 6. The previous experiment demonstrates that (1) not every data set is well-suited for the semi-supervised learning scheme and that (2) both sufficient labeled as well as sufficient unlabeled data is needed for training. To analyze these issues on real-world data, we consider four binary classification tasks induced by the `USPS` data set, namely the `USPS(2,5)`, `USPS(2,7)`, `USPS(3,8)`, and



(a) USPS(2,5)[l=16,u=806,t=823] (b) USPS(2,7)[l=17,u=843,t=861]

(c) USPS(3,8)[l=15,u=751,t=766] (d) USPS(8,0)[l=22,u=1108,t=1131]

FIGURE 5.5. *The influence of the parameter $r = |R|$ is demonstrated. All plots indicate that if a reasonable amount of indices is used, the classification performance is not affected by the approximation scheme.*

USPS(8,0) data set instances. As for the previous experiment, we consider 10 random partitions into labeled, unlabeled, and test patterns for each such task and use the RLSC approach as a baseline. Further, we consider three different amounts of labeled patterns (1%, 2%, and 5% with respect to the size of the training set) and use the remaining patterns as unlabeled data. In Figure 5.4, the outcome of this experiment is shown. Clearly, the semi-supervised S²RLSC approach can successfully incorporate unlabeled data and leads to a better error on the test set. Further, as the amount of labeled data is increased, the performances become more and more similar. Thus, these instances of the USPS data set seem to be adequate candidates for the proposed local search scheme.

Computational Considerations

From a theoretical perspective the runtime of the local search depicted in Algorithm 5.2 is bounded by $\mathcal{O}(n^3 + \mathcal{S}\mathcal{T}n^2)$ for the case with exact objective evaluations and $\mathcal{O}(nr^2 + \mathcal{S}\mathcal{T}nr)$ for the approximation cases, where $\mathcal{S}$ is the number of restarts and where $\mathcal{T}$ is the number of rounds (each involving $u < n$ coordinate-flips). Thus, as long as we have $\mathcal{S}\mathcal{T} \leq n$, the overall time complexity of the local search is no more than training a single RLSC classifier on n labeled training patterns (which takes $\mathcal{O}(n^3)$ time). A similar observation holds for the approximation case (if $\mathcal{S}\mathcal{T} \leq r$) where the training of single classifier takes $\mathcal{O}(nr^2)$ time for n patterns [120]. From a computational point of view two

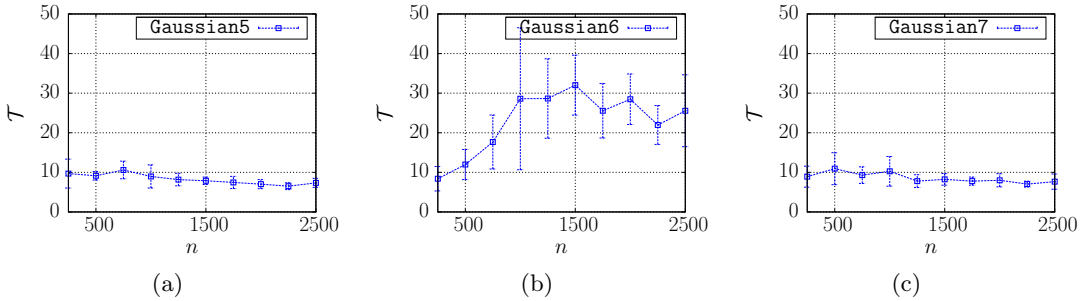


FIGURE 5.6. *Number of rounds $\mathcal{T}$ (i. e., u consecutive iterations) needed for the local search to converge. For the considered data sets a small amount of rounds seems to be sufficient for convergence.*

questions arise, namely (1) how many rounds $\mathcal{T}$ are needed in practice and (2) how is the classification performance affected by the approximation scheme. We will now investigate both questions. Further, we will show that the number $\mathcal{S}$ of restarts can be reduced significantly for semi-supervised learning settings in case one can obtain a reasonable initial guess via the available labeled patterns. Note that we will defer a runtime comparison of several competing semi-supervised methods to Chapter 6.

Sparse Approximation. The size $r = |R| \in \{1, \ldots, n\}$ of the indices for the subset of regressors scheme determines the trade-off between computational savings and accuracy of the approximation. A natural question is how the accuracy is affected by using small values for r. To investigate this issue, we again consider the four instances of the `USPS` data set and vary the assignment for the approximation parameter r from $0.01n$ to $0.1n$ (where $n = N/2$ is the size of the training set). The subset R of indices is selected uniformly at random.[10] For the sake of exposition we fix both cost parameters $\lambda = 1$ and $\gamma = 1$. The result of this experiment is given in Figure 5.5, where the term *sparse* denotes the approach with approximated objective value. All plots indicate that the classification performance is not affected as long as a subset R of reasonable size is selected (i. e., roughly 5% of all possible indices). Naturally, the amount of indices needed to yield a good classification performance depends heavily on the particular data set and the used kernel function, see, e. g., Williams and Seeger [147] who report that one can set $r \ll n$ "without any significant decrease in the accuracy of the solution" for the `USPS` data set.

Number of Rounds. The next experiment is devoted to the question how many rounds $\mathcal{T}$ are needed for the local search to converge in practice. For this purpose we consider the three Gaussian data sets and fix the model parameters to $\lambda = 1$ and $\gamma = 0.1$. Further, we

[10] Kumar *et al.* [93] point out "that uniform sampling without replacement, in addition to being more efficient both in time and space, produces more effective approximations". For the sake of simplicity we therefore consider a random selection of the indices.

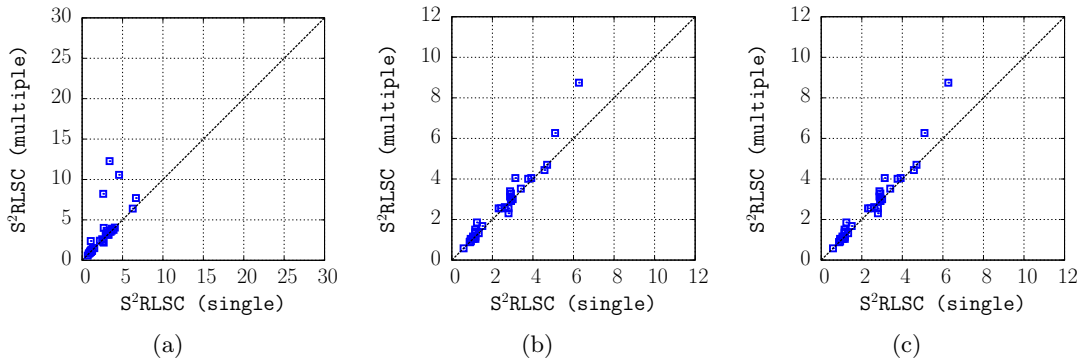


FIGURE 5.7. *Comparison of S²RLSC with multiple restarts and S²RLSC with a single restart (using the predictions of the supervised RLSC model as initial candidate solution). The results are slightly better for the single restart variant. Thus, in this case, improving a single candidate solution seems to be sufficient.*

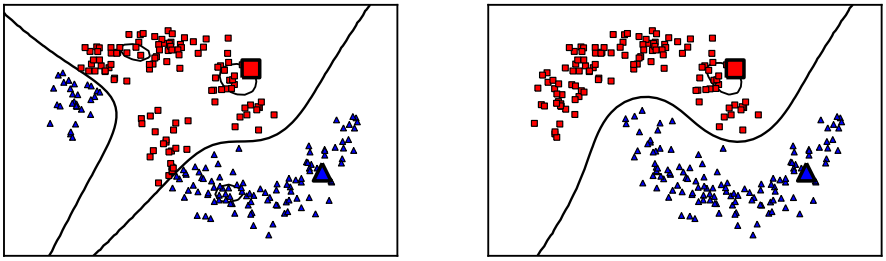
perform a single run and report the number $\mathcal{T}$ of needed rounds averaged over 10 random partitions for each of the data sets. The results are given in Figure 5.6. It can be seen that less than 50 rounds are needed for all data sets and that the number of needed rounds seems to be independent of the number n of training patterns. Thus, although the number $\mathcal{T}$ of rounds can be extremely large from a theoretical point of view, it is relatively small in practice. This renders the local search efficient in practice.

Initial Candidate Solutions. Most of the related optimization heuristics for semi-supervised support vector machines use the labeled part of the data to obtain an initial guess via a supervised model. This guess is then improved by incorporating the available unlabeled patterns. Such schemes are often *deterministic* (since they improve a single initial guess in a deterministic manner) and, thus, do not require any restarts to be performed. On the one hand, this sounds tempting since the computational cost is normally reduced dramatically by this modification. On the other hand, considering only a single candidate solution might not be sufficient for obtaining satisfying results, especially if the initial guess is bad due to a very small labeled set of patterns.

To investigate this issue, we perform a similar experiment as the one depicted in Figure 5.4, i. e., we resort to the four USPS data set instances and analyze the classification performances for different amounts of labeled patterns. More specifically, we compare the classification performances of

(a) S²RLSC multiple: The S²RLSC scheme with multiple restarts (200) and random initialization of the initial candidate solutions.

(b) S²RLSC single: The S²RLSC scheme with a single restart and deterministic initialization of the initial candidate solution via a RLSC model trained on the labeled part of the data.



(a) S^{2}RLSC (single) (21.3 ± 14.9) (b) S^{2}RLSC (multiple) (0.0 ± 0.0)

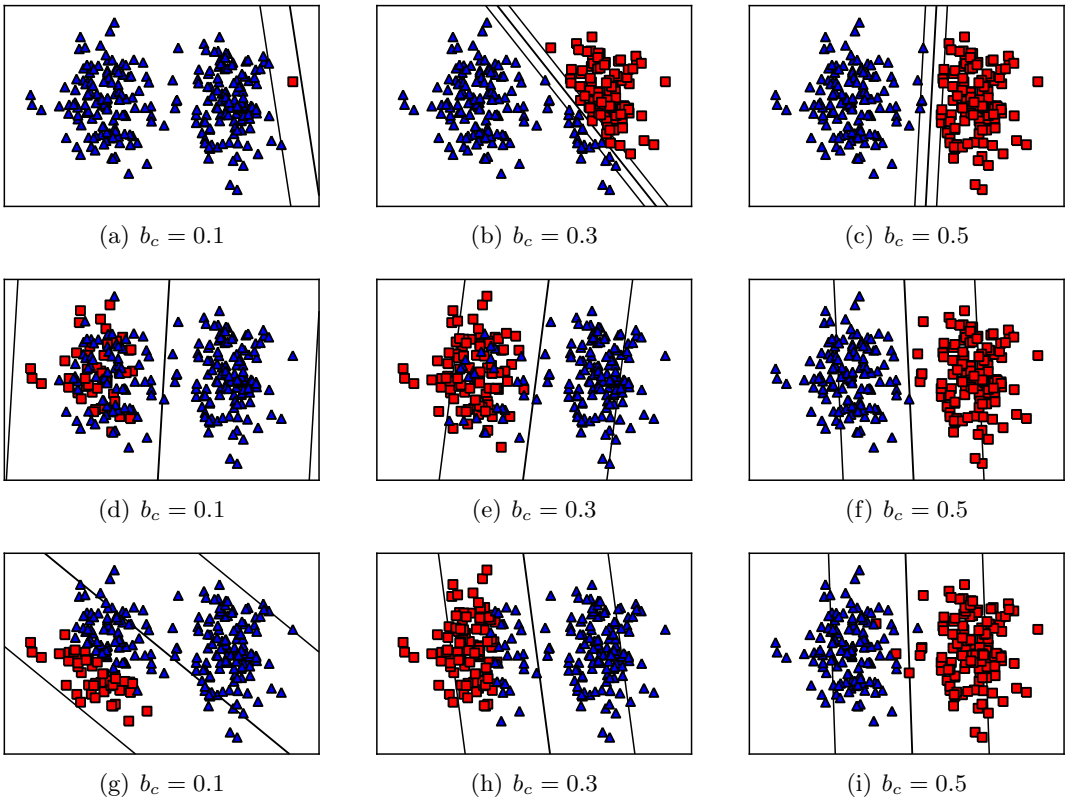
FIGURE 5.8. *If only one restart (with initial guess) is performed, the local search approach can converge to a (bad) local optimum, see Figure (a). Performing sufficient restarts (with random initial candidates) yields good solutions, see Figure (b). The average error (with one standard deviation) on the test set over* 10 *random partitions is given in brackets.*

The new results, depicted in Figure 5.7, show that both approaches exhibit a quite similar classification performance for these data set instances. Thus, for this setup, improving the single initial guess seems to be sufficient (and even better compared to the multi-restart strategy). However, improving only a single initial candidate solution can also lead to worse results: Let us now consider the `Moons` data set and an RBF kernel as similarity measure with kernel width $\sigma = 0.1\hat{\sigma}$, where the value $\hat{\sigma}$ is a rough estimate of the maximum distance between any pair of samples.[11] Further, we fix the two cost parameters $\lambda = 2^{-10}$ and $\gamma = 0.1$. The comparison of S^{2}RLSC `single` and S^{2}RLSC `multiple` is depicted in Figure 5.8. Clearly, the single-restart scheme fails on this particular problem instance while using more restarts leads to the globally optimal solution. To sum up, there are problem instances where the heuristic of improving a single guess works well and other ones where it fails. For the latter ones, it pays off to spend more effort into optimization. We will see below that not only the number of restarts but also the particular local/stochastic search scheme play important roles on difficult learning tasks.

5.4.3 Unsupervised Learning Settings

In the remainder of this section we will sketch unsupervised learning settings. An important difference between the unsupervised and the semi-supervised case is the fact that one cannot resort to initial guesses in the former case and, thus, usually has to explore the search space extensively. This can be addressed, for instance, by performing many restarts (as depicted above for the `Moons` data set). However, making use of more general stochastic optimization frameworks can also be beneficial for specific settings. We will now demonstrate such issues by means of one of the benchmark scenarios given in Chapter 4 and will further provide results in unsupervised real-world learning scenarios.

[11] Given by $\hat{\sigma} = \sqrt{\sum_{j=1}^{d} \left(\max([\bar{\mathbf{x}}_1]_j, \ldots, [\bar{\mathbf{x}}_n]_j) - \min([\bar{\mathbf{x}}_1]_j, \ldots, [\bar{\mathbf{x}}_n]_j) \right)^2}$ for n patterns $\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_n \in \mathcal{X}$.



(a) $b_c = 0.1$ (b) $b_c = 0.3$ (c) $b_c = 0.5$

(d) $b_c = 0.1$ (e) $b_c = 0.3$ (f) $b_c = 0.5$

(g) $b_c = 0.1$ (h) $b_c = 0.3$ (i) $b_c = 0.5$

FIGURE 5.9. *Comparison of the exact approach (top row), the local search strategy (middle row), and a more general stochastic search framework (bottom row) for the first benchmark scenario given in Chapter 4 that is based on the* `Gaussian1` *data set. Except for* $b_c = 0.5$*, the partitions obtained by the local search scheme (middle row) and the more general stochastic search framework (bottom row) differ significantly from the optimal results shown in the top row. Further, the local search strategy got stuck in (obvious) local optima for* $b_c = 0.1$ *and* $b_c = 0.3$ *due to the fact that only single coordinates are flipped per iteration. The more general stochastic search scheme yields better results for these two cases.*

Scenario 1: Balance Constraint – Revisited

The first benchmark scenario given in Chapter 4 aimed at using the balance parameter b_c to induce non-obvious cluster partitions for the `Gaussian1` data set. For this sake two of the parameters were fixed ($C = 1$ and $\varepsilon = 0.1$) whereas the remaining parameter b_c was set to $b_c \in \{0.1, 0.3, 0.5\}$, see Figure 5.9 (top row) for the optimal partitions computed via the exact approach for this setup. Let us now consider the corresponding results obtained via the local search scheme derived in this chapter (with new objective (5.9)). We consider the same experimental setup, i. e., we fix $\varepsilon = 0.1$ and $\gamma = \frac{1}{2uC}$.[12] Further, we consider the same three assignments for the balance parameter $b_c \in \{0.1, 0.3, 0.5\}$ and take the best

[12] Note that, since the square loss is used as surrogate for the hinge loss, different objectives are minimized. Since the `LIBSVM` implementation [29, 48] aims at computing solutions for the objective (3.39), we set $\gamma = \frac{1}{2uC}$ for the `URLSC` implementation to make both objectives as similar as possible.

result out of 200 runs with respect to the objective.

The results are given in Figure 5.9: Except for $b_c = 0.5$, the partitions obtained via the local search scheme (middle row) are quite different from those of the exact approach (top row). Further, the local scheme got stuck in (obvious) bad candidate solutions for both $b_c = 0.1$ and $b_c = 0.3$. The reason for this behavior is the fact that the local search scheme only flips a single coordinate per iteration, which can lead to situations where no labels are flipped anymore although the objective could still be improved. In Figure (d), for instance, the approach is not willing to generate any new red squares due to the balance constraint and, at the same time, does not generate any new blue triangles since this would lead to a worse objective. Hence, these benchmark scenarios depict challenging tasks for the local search scheme and reveal one of its possible disadvantages: Switching only a single coordinate per iteration can be insufficient for specific problem instances (depending on the balance constraint and the structure of the data).

The bottom row of Figure 5.9 shows the results of a more general stochastic search scheme (depicted below) that flips a small amount of coordinates per iteration. While the computed partitions are not optimal as well, the obtained results are clearly better for $b_c = 0.1$ and $b_c = 0.3$ compared to the local search scheme. Hence, for these benchmark scenarios, not only the amount of restarts but also the particular scheme to switch the involved labels is important.

Fast Evolutionary Clustering

While the proposed local search scheme works extremely well on a variety of learning tasks, the above example indicates that it might be useful to resort to other label-switching or stochastic optimization schemes for specific tasks. A well-known class of combinatorial optimization schemes are *evolutionary algorithms* [13]. In Algorithm 5.3, such an approach is given: Here, the starting point of the stochastic search is a *population* $\mathcal{P}_0 = \{\mathbf{y}_1, \dots, \mathbf{y}_\mu\} \subseteq \{-1, +1\}^u$ consisting of $\mu \in \mathbb{N}$ random candidate solutions, also called *individuals*. In Step 2, the *fitness* $F(\mathbf{y})$ is computed for each of these initial individuals, where

$$F(\mathbf{y}) = \underset{\mathbf{c} \in \mathbb{R}^u}{\text{minimize}}\, Q(\mathbf{y}, \mathbf{c}) \tag{5.41}$$

is determined via the task (5.9). The iteration over all *generations* $1, \dots, \tau$ is started in Step 4. For each generation t, one randomly selects ν parental individuals to produce mutated individuals and each of these mutated individuals is created by (uniformly at random) flipping a small number of coordinates of the parental individual. After the computation of the fitness values for the mutated individuals, all resulting individuals are sorted upwards by their objective values yielding a sorted sequence $\mathbf{y}_{i_1}, \dots, \mathbf{y}_{i_{\mu+\nu}}$. Finally, the population $\mathcal{P}_t$ is updated by selecting the μ candidate solutions having the

Input: An unlabeled training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$ and parameters $\mu \in \mathbb{N}$, $\nu \in \mathbb{N}$, $\tau \in \mathbb{N}$, $C \in \mathbb{R}^+$, $b_c \in [0, 1]$, and $\varepsilon \in \mathbb{R}^+$.
Output: An approximation $(\mathbf{c}^*, \mathbf{y}) \in \{-1, +1\}^u \times \mathbb{R}^u$ for the task (5.9).

1: Initialize $\mathcal{P}_0 = \{\mathbf{y}_1, \ldots, \mathbf{y}_\mu\} \subseteq \{-1, +1\}^u$.
2: Compute the fitness $F(\mathbf{y}_j)$ for each $\mathbf{y}_j \in \mathcal{P}_0$.
3: $t = 0$
4: **while** $t \leq \tau$ **do**
5: **for** $i = 1$ **to** ν **do**
6: Randomly select parent $\mathbf{y} \in \mathcal{P}_t$.
7: Generate valid mutated individual $\mathbf{y}_{\mu+i}$.
8: Compute fitness $F(\mathbf{y}_{\mu+i})$.
9: **end for**
10: Compute sorted sequence $\mathbf{y}_{i_1}, \ldots, \mathbf{y}_{i_{\mu+\nu}}$.
11: $\mathcal{P}_{t+1} = \{\mathbf{y}_{i_1}, \ldots, \mathbf{y}_{i_\mu}\}$
12: $t = t + 1$
13: **end while**
14: Compute solution $\mathbf{c}^*$ for minimize$_{\mathbf{c} \in \mathbb{R}^u}$ $Q(\mathbf{y}_{i_1}, \mathbf{c})$
15: **return** $(\mathbf{c}^*, \mathbf{y}_{i_1})$

ALGORITHM 5.3. *Simple stochastic optimization framework for the objective (5.9). In each generation of the search a small amount of labels is flipped. For such settings, the proposed computational shortcuts reduce the runtime of such a scheme dramatically, as shown by Gieseke* et al. *[59].*

best objective values. When generation τ is reached, the best individual along with its corresponding vector $\hat{\mathbf{c}}$ is returned. Throughout the search only valid candidate solutions fulfilling the balance constraint are created.

Similar to the local search considered above, the main computational bottleneck of this more general stochastic search is the recurrent computation of the fitness values in Step 8. However, as mentioned above, the computational shortcuts derived in this chapter can also be applied in case multiple labels are switched per iteration. This greatly reduces the runtime of such an exhaustive search and paves the way for efficiently testing a massive amount of candidate solutions. This is, in turn, important to obtain a good clustering performance: In Table 5.1, a comparison of the classical k-means approach (`k-means`) [101], the iterative scheme proposed by Zhang *et al.* [153] (`IterSVR`), and two instances of the above evolutionary framework on real-world data is given. Here, `EvoMMC` is based on the exact computational shortcut, whereas `FastEvoMMC` is based on the kernel matrix approximation scheme depicted above, see Gieseke *et al.* [59] for details related to the experimental setup. The results indicate that unsupervised support vector machines depict a valuable alternative to classical clustering schemes, at least in case the search space is explored in a sufficient kind of way.

Data	u	k-means	IterSVR	EvoMMC	FastEvoMMC
UCIDigits(3,8)	357	5.3 ± 0.0	3.4 ± 0.0	2.5 ± 0.0	2.6 ± 0.5
UCIDigits(1,7)	361	0.6 ± 0.0	0.6 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
UCIDigits(2,7)	356	3.1 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
UCIDigits(8,9)	354	9.3 ± 0.0	3.7 ± 0.0	3.2 ± 0.4	3.8 ± 4.1
Ionosphere	351	32.0 ± 17.9	32.3 ± 16.6	17.9 ± 6.8	18.8 ± 3.2
Letter	1,555	17.9 ± 0.0	7.2 ± 0.0	3.7 ± 0.3	3.3 ± 1.1
Satellite	2,236	4.1 ± 0.0	3.2 ± 0.0	1.2 ± 0.5	1.1 ± 0.7

TABLE 5.1. *The table shows the clustering error (disagreement of the computed and the true partition) in percent on several real-world data sets of* k-means *[101],* IterSVR *[153],* EvoMMC *[59], and* FastEvoMMC *[59] (averaged errors and standard deviation over 10 random partitions are reported). The results indicate that the concept of unsupervised support vector machines depicts a meaningful alternative to classical clustering schemes. Further, the results obtained by the evolutionary framework (*EvoMMC *and* FastEvoMMC*) are superior to those of the iterative scheme (*IterSVR*), which indicates that putting more effort into optimization pays off for such learning settings.*

5.5 Concluding Remarks

Both the semi- and the unsupervised learning task are promising from a practical point of view. However, the combinatorial nature of the induced optimization problems renders a direct application of the corresponding concepts difficult in real-world settings. In this chapter, we have proposed least-squares variants of the original problem formulations along with simple local search schemes for addressing the resulting optimization tasks. While a naive implementation of the considered search strategies is computationally very demanding, we have shown how to speed-up their execution by means of efficient matrix updates of the intermediate solutions. These computational shortcuts render an exhaustive search possible. The experimental evaluation indicates that the (conceptually very simple) local search scheme yields good classification results in most cases. For difficult problem instances, however, it might be useful to employ more sophisticated search heuristics like general evolutionary algorithms.

CHAPTER 6

Sparse Quasi-Newton Optimization

The efficient implementation of the local search strategy proposed in the previous chapter was based on the fact that, given a fixed partition vector for the unlabeled patterns, the remaining real-valued variables could be directly obtained since they induced a convex optimization task. In this chapter, we will see that a similar observation also holds if one considers the real-valued part of the corresponding optimization tasks to be fixed (and by reformulating the balance constraint). More precisely, the optimal assignments for the remaining integer variables are directly known and, thus, do not occur as optimization variables anymore. This is quite a remarkable property of the learning tasks at hand giving rise to continuous (but non-convex) optimization problems that can be addressed by a completely different branch of mathematical optimization methods. In the following, we will propose the use of quasi-Newton optimization schemes [108] to address the corresponding optimization tasks. The resulting approaches can be implemented easily using black box optimization engines and exhibit an excellent classification and runtime performance that is superior (or at least competitive) to state-of-the-art implementations.

Outline. We will start with a short motivation in Section 6.1 and will subsequently derive the continuous optimization variants in Section 6.2. In general, the resulting objectives are not differentiable. Aiming at the application of gradient based schemes, we will show how to employ surrogate loss functions that render the objective functions at hand more amenable to efficient optimization. The definition of these variants as well as the application of quasi-Newton optimization schemes will be subject of Section 6.3. A detailed experimental evaluation will be given in Section 6.4, followed by a discussion of

optimization and model selection issues in Section 6.5. Finally, concluding remarks will be given in Section 6.6.

6.1 Motivation

As shown in the previous chapter, using the square loss instead of the hinge loss can be appealing since it seems to be well-suited for label-switching optimization schemes (at least in the unsupervised case, as demonstrated by Zhang *et al.* [153]) and can lead to efficient computational update-schemes. Reformulating the original learning tasks and subsequently approaching these corresponding surrogates is a standard procedure for both learning settings. However, whether such a reformulation has a negative effect on the generalization performance of the resulting model or not is unclear. One possible disadvantage of the square loss is, for instance, the fact that outliers (i. e., single patterns lying far away from the remaining ones) can have a significant influence on the overall model [120].

In the following, we will aim at minimizing the problem formulation induced by the hinge loss. As mentioned above, the tasks give rise to continuous but non-convex optimization problems. Using the hinge loss leads to a non-differentiable problem and is therefore not amenable to, e. g., gradient based optimization schemes [108]. A well-known algorithmic idea in the related literature is to resort to differentiable surrogates for the original objectives, see, e. g., Chapelle and Zien [32]. In this chapter, we will propose a particular combination of surrogates for the loss on the labeled and unlabeled parts of the data. This will pave the way for the application of two particular instances of the quasi-Newton family of optimization methods [108]. While the general idea of using differentiable loss surrogates and gradient based optimization schemes is not new in this context, we would like to point out

(a) that the resulting approach is conceptually very simple since one can resort to black box optimization engines,

(b) that computational speed-ups for sparse and non-sparse data can easily be integrated into the overall framework, which is not always the case for competing approaches, and

(c) that the two selected representatives of the quasi-Newton framework depict ideal candidates for the induced optimization tasks (and have not yet been considered in the context of semi-supervised support vector machines).

As we will see below, only the surrogate objective function and its gradient need to be supplied to the main optimization scheme.

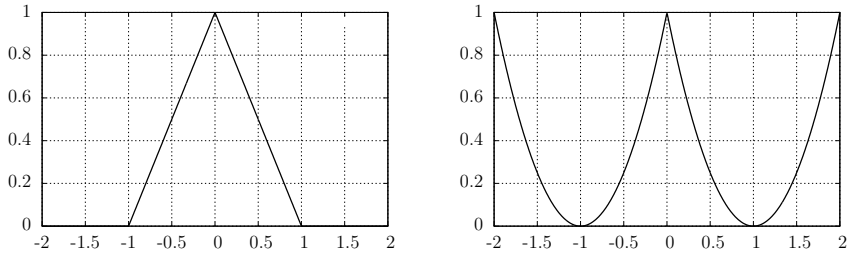


FIGURE 6.1. *The effective loss functions* $\mathcal{L}_e(t) = \max(0, 1 - |t|)$ *and* $\mathcal{L}_e(t) = (t - \operatorname{sgn}(t))^2$ *induced by (a) the hinge loss and (b) the square loss. Both effective loss functions penalize predictions around zero and, thus, enforce the prediction function to avoid high-density areas induced by the unlabeled patterns.*

6.2 Continuous Optimization

We will focus on the semi-supervised case and will show how to obtain a continuous optimization variant for this setting. In general, unsupervised settings can be handled in a similar manner.

6.2.1 Non-Convex Task

Again, we are given both a set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \{-1, +1\}$ of labeled patterns as well as a set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$ of unlabeled ones from an arbitrary set $\mathcal{X}$. As shown in Chapter 5, the semi-supervised setting gives rise to an optimization task of the form

$$\underset{\mathbf{y} \in \{-1,+1\}^u,\, f \in \mathcal{H}_k,\, b \in \mathbb{R}}{\text{minimize}} \; \frac{1}{l} \sum_{i=1}^{l} \mathcal{L}\big(y'_i, f(\mathbf{x}'_i) + b\big) + \gamma \frac{1}{u} \sum_{i=1}^{u} \mathcal{L}\big(y_i, f(\mathbf{x}_i) + b\big) + \lambda \|f\|^2_{\mathcal{H}_k} \tag{6.1}$$

with cost parameters $\lambda \in \mathbb{R}^+$ and $\gamma \in \mathbb{R}^+$, hypothesis space $\mathcal{H}_k$, and loss function $\mathcal{L} : \mathcal{Y} \times \mathbb{R} \to [0, \infty)$. The balance constraint, omitted above, is not obligatory in semi-supervised learning settings (see below). Now, instead of fixing the partition vector $\mathbf{y} \in \{-1, +1\}^u$, let us assume that the hypothesis $(f, b) \in \mathcal{H}_k \times \mathbb{R}$ is fixed. Then, the optimal assignments y_i for the unlabeled patterns $\mathbf{x}_i$ are given by $y_i = \operatorname{sgn}(f(\mathbf{x}_i) + b)$ and are, thus, directly known [32].[1] Therefore, one can rewrite the above task as:

$$\underset{f \in \mathcal{H}_k,\, b \in \mathbb{R}}{\text{minimize}} \; \frac{1}{l} \sum_{i=1}^{l} \mathcal{L}\big(y'_i, f(\mathbf{x}'_i) + b\big) + \gamma \frac{1}{u} \sum_{i=1}^{u} \mathcal{L}_e\big(f(\mathbf{x}_i) + b\big) + \lambda \|f\|^2_{\mathcal{H}_k} \tag{6.2}$$

[1] For this observation, the loss function has to fulfill the reasonable assumption that $\mathcal{L}(+1, t) \leq \mathcal{L}(-1, t)$ for $t \geq 0$ as well as $\mathcal{L}(+1, t) \geq \mathcal{L}(-1, t)$ for $t \leq 0$ holds.

Here, the function $\mathcal{L}_e(t) = \mathcal{L}(\text{sgn}(t), t)$ is called the *effective loss* induced by the loss function $\mathcal{L}$. Note that vector $\mathbf{y} \in \{-1, +1\}^u$ does not occur anymore in the above task.

The two effective loss functions induced by the hinge and the square loss are depicted in Figure 6.1. A common property of both induced functions is the fact that they penalize predictions around zero. Therefore, the second term of the above objective increases if the prediction function passes through over-densities caused by the unlabeled patterns. In other words, the second term favors decision hyperplanes going through low-density areas induced by the unlabeled patterns. Again, due to the semiparametric representer theorem (Fact 3.2), any optimal $(f^*, b^*) \in \mathcal{H}_k \times \mathbb{R}$ for the problem (6.2) can be written as

$$f^*(\cdot) + b^* = \sum_{i=1}^{l} c_i k(\cdot, \mathbf{x}'_i) + \sum_{i=l+1}^{l+u} c_i k(\cdot, \mathbf{x}_{i-l}) + \beta \tag{6.3}$$

with appropriate coefficients $c_1, \ldots, c_{l+u}, \beta \in \mathbb{R}$. Hence, one ends up with searching for appropriate assignments for these real coefficients and, thus, with a continuous optimization task [108]. Plugging in different loss functions leads to various problem instances. The hinge loss leads to the standard formulation and renders the task non-convex and non-differentiable (since the partial derivatives do not exist).

6.2.2 Balance Constraint

The ideas depicted above depend on the assumption that no balance constraint of the form

$$-\varepsilon < \frac{1}{u} \sum_{i=1}^{u} \max(0, y_i) - b_c < \varepsilon \tag{6.4}$$

with $\varepsilon \in \mathbb{R}^+$ is used to enforce partitions of the unlabeled patterns fulfilling a certain ratio of positive and negative elements. While this is, in general, not a problem for the semi-supervised case (since the labeled part prevents unbounded solutions), the unsupervised setting requires an appropriate constraint.[2] It should be pointed out that the balance constraint is also often used in semi-supervised settings since it provides additional information and might help yielding good solutions for local search strategies (given a good estimate for b_c). Incorporating the original balance constraint into the continuous optimization framework, however, seems to be more difficult. This is due to the fact that, by considering the balance constraint, the optimal assignments for the partition vector $\mathbf{y} \in \{-1, +1\}^u$ are not necessarily given by $y_i = \text{sgn}(f(\mathbf{x}_i) + b)$, as shown in Chapter 4. To

[2] For unsupervised settings, assigning all unlabeled patterns to only one class corresponds to an optimal solution and the resulting margin becomes arbitrary large. Note that assigning all unlabeled patterns to only one class can also depict an optimal solution for the semi-supervised setting, but the resulting margin is not unbounded.

add a balance constraint for the continuous optimization perspective, Chapelle *et al.* [32] propose to consider a relaxed version of the form

$$\frac{1}{u}\sum_{i=1}^{u}\langle \mathbf{w}, \mathbf{x}_i\rangle + b = b_c \tag{6.5}$$

for the linear kernel (and for semi-supervised settings). Thus, by preprocessing all patterns such that $\sum_{i=1}^{u} \mathbf{x}_i = \mathbf{0}$, one can enforce the balance constraint by fixing $b = b_c$. Further, for non-linear kernel functions, one can resort to the *kernel PCA map* [124] to implicitly obtain a more flexible model, see Chapelle and Zien [32] for details.[3]

6.2.3 Related Work

Let us finally sketch the related semi-supervised approaches that are based on the continuous optimization perspective. Again, these approaches usually compute an initial guess via the labeled part of the data (as already sketched in the previous chapter) and subsequently improve this guess by means of some kind of local search scheme. Most of the related approaches actually only differ in the way such a local search phase is performed.

As mentioned above, the hinge loss leads to a non-differentiable objective function, which necessitates the use of derivative-free optimization frameworks (or, at least, the use of subgradient methods [108]). A common strategy in the related literature is based on using differentiable surrogates that render the application of sophisticated gradient based optimization schemes possible and usually lead to a better convergence behavior (at least, in practice). Originally, the idea of using such a replacement was proposed by Chapelle and Zien [32]. They consider a similar differentiable surrogate as the one proposed in this chapter and resort to a gradient descent strategy as optimization framework. A refinement of this approach, based on the so-called *continuation method* [108], has later been proposed by Chapelle *et al.* [33]. The key idea of this variant is to minimize a smoothed (convex) version of the original task at the beginning of the optimization process and to iteratively decrease the smoothing (where the outcome of an iteration serves as a starting point for the next iteration). Another approach is given by Collobert *et al.* [38, 39]. Instead of using a differentiable surrogate, they show how to split up the objective term into a convex part and a concave part (using a *ramp loss* as surrogate) and resort to the constrained concave-convex procedure [133, 152] as optimization engine. Zhao *et al.* [155] also consider this kind of optimization framework, but resort to a cutting-plane technique for addressing the intermediate optimization tasks. Finally, Reddy *et al.* [117] propose a

[3] This way, one essentially removes the offset term from the optimization process. If one does not consider such a term at all while using a linear kernel, then the decision hyperplane passes through the origin and the loss on the unlabeled patterns will avoid arbitrary large margins as well (without imposing any ratio between positive and negative entries).

quasi-Newton framework in combination with a label-switching strategy. Further, instead of using differentiable surrogates, they resort to subgradient methods to alleviate the problem of the non-differentiability of the hinge loss.

Both the approach proposed by Chapelle and Zien [32] and the just mentioned quasi-Newton framework proposed by Reddy *et al.* [117] are quite close to what is presented in this chapter. However, compared to the scheme of Reddy *et al.*, we resort to simple differentiable surrogates and propose effective ways to deal with large-scale learning settings. The differentiable surrogates considered in this chapter are related (but not identical) to the ones proposed by Chapelle and Zien [32]. Further, we suggest specific instances of the quasi-Newton family of optimization frameworks to deal with the continuous tasks at hand that seem to be more appropriate than the gradient descent.

6.3 Algorithmic Framework

We will consider two quasi-Newton optimization schemes [108] and will justify their use in this context. For this sake, we will start by deriving an appropriate differentiable surrogate of the objective. Afterwards, we will depict the algorithmic framework as well as computational shortcuts that speed up the overall execution.

6.3.1 Differentiable Surrogates

Aiming at the application of such gradient based optimization schemes, we introduce the following differentiable surrogate loss functions depicted in Figure 6.2. Here, the differentiable replacement for the hinge loss is the modified logistic loss $\mathcal{L}(y,t) = \frac{1}{\eta}\log(1+\exp(\eta(1-yt)))$ with $\eta = 20$ [154] and the one for the effective hinge loss is given by $\mathcal{L}_e(t) = \exp(-st^2)$ with $s = 3$ [32].[4] For the sake of exposition, we will again omit the offset term $b \in \mathbb{R}$. By defining $\bar{\mathbf{x}}_i = \mathbf{x}'_i$ for $i = 1,\ldots,l$ and $\bar{\mathbf{x}}_i = \mathbf{x}_{i-l}$ for $i = l+1,\ldots,n$, the new surrogate objective is given by

$$\begin{aligned} F_\gamma(\mathbf{c}) = \; & \frac{1}{l}\sum_{i=1}^{l}\frac{1}{\eta}\log\left(1+\exp(\eta(1-y'_i f(\bar{\mathbf{x}}_i)))\right) \\ & + \frac{\gamma}{u}\sum_{i=1}^{u}\exp(-3(f(\bar{\mathbf{x}}_{l+i}))^2) + \lambda\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k(\bar{\mathbf{x}}_i,\bar{\mathbf{x}}_j), \end{aligned} \tag{6.6}$$

[4] The surrogate for the unlabeled patterns was initially proposed by Chapelle and Zien [32] in the context of semi-supervised support vector machines. For the labeled part, however, they consider the squared version of the hinge loss. Surprisingly, it seems that this particular combination of surrogates proposed here has not yet been considered in the literature (although they depict perfect differentiable replacements for the original loss functions).

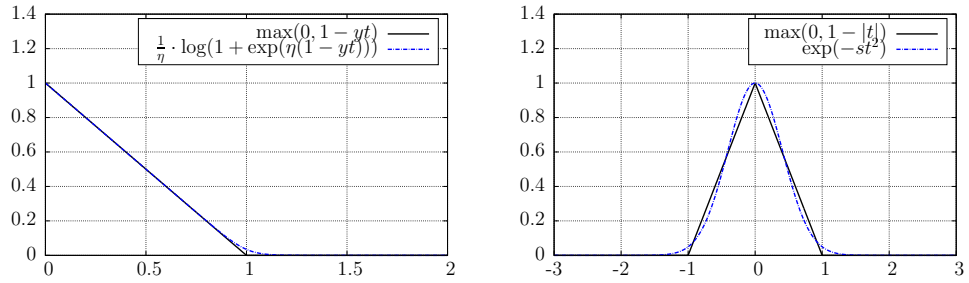


FIGURE 6.2. *The hinge loss* $\mathcal{L}(y,t) = \max(0, 1-yt)$ *and its differentiable surrogate* $\mathcal{L}(y,t) = \frac{1}{\eta}\log(1 + \exp(\eta(1-yt)))$ *with* $y = +1$ *and* $\eta = 20$ *are shown in Figure (a). The effective loss* $\mathcal{L}(t) = \max(0, 1-|t|)$ *induced by the hinge loss along with its surrogate* $\mathcal{L}_e(t) = \exp(-st^2)$ *with* $s = 3$ *are given in Figure (b).*

where $f(\cdot) = \sum_{j=1}^{n} c_j k(\cdot, \bar{\mathbf{x}}_j)$ and $\|f\|_{\mathcal{H}_k}^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k(\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_j)$. The following lemma shows that one can compute both the new objective and its gradient efficiently:

Lemma 6.1 *For a given* $\mathbf{c} \in \mathbb{R}^n$*, one can compute both the objective value* $F_\gamma(\mathbf{c})$ *as well as its gradient* $\nabla F_\gamma(\mathbf{c})$ *in* $\mathcal{O}(n^2)$ *time. The overall space consumption is* $\mathcal{O}(n^2)$*.*

Proof: The partial derivatives are given by

$$\begin{aligned}\frac{\partial F_\gamma(\mathbf{c})}{\partial c_p} &= -\frac{1}{l}\sum_{i=1}^{l}\frac{\exp(\eta(1-y_i' f(\bar{\mathbf{x}}_i)))}{(1+\exp(\eta(1-y_i' f(\bar{\mathbf{x}}_i))))} y_i' k(\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_p) \\ &\quad -\gamma\frac{6}{u}\sum_{i=1}^{u}\exp(-3(f(\bar{\mathbf{x}}_{l+i}))^2) f(\bar{\mathbf{x}}_{l+i}) k(\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_p) + 2\lambda\sum_{i=1}^{n} c_i k(\bar{\mathbf{x}}_i, \bar{\mathbf{x}}_p)\end{aligned}$$

and, thus, the gradient by

$$\nabla F_\gamma(\mathbf{c}) = \mathbf{K}\mathbf{a} + 2\lambda\mathbf{K}\mathbf{c} \tag{6.7}$$

with $\mathbf{a} \in \mathbb{R}^n$ defined via:

$$a_i := \begin{cases} -\frac{1}{l}\cdot\frac{\exp(\eta(1-f(\bar{\mathbf{x}}_i)y_i'))}{1+\exp(\eta(1-f(\bar{\mathbf{x}}_i)y_i'))}\cdot y_i' & \text{for } i = 1,\ldots,l \\ -\gamma\frac{6}{u}\cdot\exp\left(-3(f(\bar{\mathbf{x}}_i))^2\right)\cdot f(\bar{\mathbf{x}}_i) & \text{for } i = l+1,\ldots,n \end{cases} \tag{6.8}$$

Since each $f(\bar{\mathbf{x}}_i)$ can be computed in $\mathcal{O}(n)$ time, one can precompute all predictions $f(\bar{\mathbf{x}}_1),\ldots,f(\bar{\mathbf{x}}_n)$ and, thus, the vector $\mathbf{a} \in \mathbb{R}^n$ in $\mathcal{O}(n^2)$ time. Further, given these predictions, the remaining operations to compute the objective (6.6) and the gradient (6.7)

Input: A labeled training set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathcal{X} \times \{-1, +1\}$, an unlabeled training set $T_U = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset \mathcal{X}$, model parameters $\lambda \in \mathbb{R}^+$ and $\gamma \in \mathbb{R}^+$, an initial (positive definite) inverse Hessian approximation $\mathbf{H}_0$, and a sequence $0 = \alpha_1 < \ldots < \alpha_\tau$ of real (annealing) parameters.
Output: An approximate solution $\mathbf{c} \in \mathbb{R}^n$ for the objective (6.6).

$\mathbf{c}_0 = \mathbf{0}$
for $i = 1$ **to** τ **do**
  $j = 0$
  while termination criteria not fulfilled **do**
    Compute search direction $\mathbf{p}_j$ via (6.10)
    Update $\mathbf{c}_{j+1} = \mathbf{c}_j + \beta_j \mathbf{p}_j$
    Update $\mathbf{H}_{j+1}$ via (6.11)
    $j = j + 1$
  end while
  $\mathbf{c}_0 = \mathbf{c}_j$
end for
return $\mathbf{c}_0$

ALGORITHM 6.1. *The quasi-Newton optimization framework for addressing the differentiable surrogate objective (6.6): Since the first parameter α_1 in the sequence of parameters is zero, one initially trains a supervised model on the labeled part of the data and, thus, also starts with an initial guess for the desired decision function. The influence of the unlabeled part is then increased gradually. For each step, a standard BFGS scheme is applied [108].*

can be performed in $\mathcal{O}(n^2)$ time as well. The space requirements are dominated by the quadratic space consumption of the kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$. □

Note that numerical instabilities might occur when evaluating $\log(\exp(\eta(1 - f(\bar{\mathbf{x}}_i)y'_i)))$ for the computation of the objective (6.6) and $\exp(\eta(1 - y'_i f(\bar{\mathbf{x}}_i)))(1 + \exp(\eta(1 - y'_i f(\bar{\mathbf{x}}_i))))^{-1}$ for the computation of the gradient (6.7). However, one can deal with these possible instabilities in a safe manner since $\log(1 + \exp(t)) - t \to 0$ and $\frac{\exp(t)}{1+\exp(t)} - 1 \to 0$ converge rapidly for $t \to \infty$. Thus, aiming at a numerically stable implementation, one can replace these values by their limits for t larger than, e. g., 500.

6.3.2 Quasi-Newton Optimization

One of the most popular quasi-Newton schemes is the so-called *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) [108] method, which we will now sketch in the context of the given task.

Quasi-Newton Framework

The overall algorithmic framework is provided in Algorithm 6.1: The influence of the unlabeled part is increased gradually via the sequence $0 = \alpha_1 < \ldots < \alpha_\tau$. This sequence can be seen as *annealing sequence* and depicts another well-known strategy for the tasks at hand to create easier problem instances at early stages of the optimization process and

to transform these instances towards the final task throughout the overall execution [33, 83, 131]. Note that we have $\alpha_1 = 0$ for the first parameter. Thus, for the first optimization stage, we essentially train a supervised model only based on the labeled part of the data.[5] For each parameter α_i, a standard BFGS optimization phase is performed (Steps 4–9). For the sake of exposition, we will briefly sketch the main steps [108]: In each of these intermediate optimization stages, a sequence

$$\mathbf{c}_{j+1} = \mathbf{c}_j + \beta_j \mathbf{p}_j \tag{6.9}$$

of candidate solutions is generated, where the *search direction* $\mathbf{p}_j$ is computed via

$$\mathbf{p}_j = -\mathbf{H}_j \nabla F_{\gamma \cdot \alpha_i}(\mathbf{c}_j) \tag{6.10}$$

and where the *step length* β_j is computed via line search [108]. Here, $\mathbf{H}_j$ is an *approximation* of the inverse of the objective's Hessian and is updated via

$$\mathbf{H}_{j+1} = (\mathbf{I} - \rho_j \mathbf{s}_j \mathbf{z}_j^{\mathrm{T}}) \mathbf{H}_j (\mathbf{I} - \rho_j \mathbf{z}_j \mathbf{s}_j^{\mathrm{T}}) + \rho_j \mathbf{s}_j \mathbf{s}_j^{\mathrm{T}} \tag{6.11}$$

with $\mathbf{z}_j = \nabla F_{\gamma \cdot \alpha_i}(\mathbf{c}_{j+1}) - \nabla F_{\gamma \cdot \alpha_i}(\mathbf{c}_j)$, $\mathbf{s}_j = \mathbf{c}_{j+1} - \mathbf{c}_j$, and $\rho_j = (\mathbf{z}_j^{\mathrm{T}} \mathbf{s}_j)^{-1}$. As initial approximation $\mathbf{H}_0$, one usually resorts to $\mathbf{H}_0 = \delta \mathbf{I}$ with $\delta \in \mathbb{R}^+$. An important property of the update scheme is that it preserves the positive definiteness of such an initial approximation, see Nocedal and Wright [108] for details.

Termination Criteria and Convergence

The approach depicted in Algorithm 6.1 performs τ annealing steps, and for each of these steps, a standard BFGS optimization scheme is applied. Thus, the convergence of the overall approach depends on the stopping criterion used in Step 4. A trivial approach consists in stopping the intermediate optimization stages after a fixed (and user-defined) number of iterations $\mathcal{T}$. In general, other stopping criteria can be considered like, e. g., stopping as soon as

$$\| \nabla F_{\gamma \cdot \alpha_i}(\mathbf{c}_k) \| \leq \hat{\varepsilon} \tag{6.12}$$

is fulfilled for a small constant $\hat{\varepsilon} \in \mathbb{R}^+$. However, whether such stopping criteria lead to a (theoretically) guaranteed convergence behavior for (general) non-convex optimization tasks is not clear and is subject of ongoing research [108]. For the experimental evaluation,

[5] By ignoring the second part of the objective (6.6), one obtains a convex optimization task that can be easily addressed by the quasi-Newton scheme. The convexity of the new objective is due to the convexity of the modified logistic loss; the argumentation is the same as for the hinge loss depicted in Section 3.3. As a side note, we would like to point out that these derivations yield an efficient implementation for the standard support vector machine optimization task, especially for sparse data (see below).

stopping the iterative process after $\mathcal{T} = 1,000$ iterations was sufficient for all considered tasks (and only few more function and gradient evaluations were needed).

It should be pointed out that, so far, no rigorous theoretical convergence analyses seem to exist for related local search strategies that address the non-convex task induced by semi-supervised support vector machines. This is, in general, the case for all CCCP-based approaches, where the convergence analysis is subject of ongoing research as well.[6]

6.3.3 Computational Speed-Ups

Two main computational bottlenecks arise for the above scheme: Firstly, the recurrent computation of the objective and gradient needed by the quasi-Newton framework is cumbersome. Secondly, the approximation of the Hessian's inverse is, in general, not sparse, which leads to quadratic-time operations for the quasi-Newton framework itself [108]. In the following, we will show how to alleviate these two problems.

Limited Memory Quasi-Newton

The non-sparse approximation of the Hessian's inverse leads to quadratic time and space requirements. To reduce these computational costs, we consider the L-BFGS methods [108], which depicts a memory and time saving variant of the original BFGS scheme. In a nutshell, the idea consists in generating the approximations $\mathbf{H}_0, \mathbf{H}_1, \ldots$ only based on the last $m \ll n$ iterations and to perform low-rank updates on the fly without storing the involved matrices explicitly (which would take quadratic time and space) [108]. This leads to an update time of $\mathcal{O}(mn)$ for all operations related to the intermediate optimization phases (not counting the time for function and gradient calls). As pointed out by Nocedal and Wright [108], small values for m are usually sufficient in practice (ranging from, e. g., $m = 3$ to $m = 50$). Thus, assuming m to be a relatively small constant, the operations needed by the optimization engine essentially scale linearly with the number n of optimization variables.

Low-Dimensional Search Space

It remains to show how to reduce the second bottleneck, i. e., the recurrent computation of both the objective and the gradient. Similar to the previous chapter, one can resort to, e. g., the subset of regressors method [119, 120] to reduce these computational costs, i. e.,

[6] See, e. g., Sriperumbudur and Lanckriet [134], who report that "though widely used in many applications, the convergence behavior of CCCP has not gotten a lot of specific attention".

one approximates the original hypothesis $f(\cdot) = \sum_{j=1}^{n} c_j k(\cdot, \bar{\mathbf{x}}_j)$ via

$$\hat{f}(\cdot) = \sum_{p=1}^{r} \hat{c}_{j_p} k(\cdot, \bar{\mathbf{x}}_{j_p}), \tag{6.13}$$

where $R = \{j_1, \ldots, j_r\} \subseteq \{1, \ldots, n\}$ is a subset of indices. Using this approximation scheme leads to a slightly modified objective $\hat{F}_\gamma(\hat{\mathbf{c}})$ for $\hat{\mathbf{c}} \in \mathbb{R}^r$, where the predictions $f(\bar{\mathbf{x}}_1), \ldots, f(\bar{\mathbf{x}}_n)$ are replaced by their corresponding approximations $\hat{f}(\bar{\mathbf{x}}_1), \ldots, \hat{f}(\bar{\mathbf{x}}_n)$ in (6.6). Similar derivations as for the non-approximation case show that the gradient $\nabla \hat{F}_\gamma(\hat{\mathbf{c}})$ is then given as

$$\nabla \hat{F}_\gamma(\hat{\mathbf{c}}) \quad = \quad \mathbf{K}_R \, \mathbf{a} + 2\lambda \mathbf{K}_{RR} \, \hat{\mathbf{c}}, \tag{6.14}$$

where f has to be replaced by $\hat{f}$ in the former definition (6.8) of the vector $\mathbf{a} \in \mathbb{R}^n$. It is easy to see that one can compute both the new objective as well as its gradient in an efficient kind of way:

Lemma 6.2 *For a given candidate solution $\hat{\mathbf{c}} \in \mathbb{R}^r$, the approximated objective $\hat{F}_\gamma(\hat{\mathbf{c}})$ as well as its associated gradient $\nabla \hat{F}_\gamma(\hat{\mathbf{c}})$ can be computed in $\mathcal{O}(nr)$ time spending $\mathcal{O}(nr)$ space.*

Proof: All predictions $\hat{f}(\bar{\mathbf{x}}_1), \ldots, \hat{f}(\bar{\mathbf{x}}_n)$ can be computed in $\mathcal{O}(nr)$ time for a $\hat{\mathbf{c}} \in \mathbb{R}^r$. Given these predictions, one can compute the modified vector $\mathbf{a} \in \mathbb{R}^n$ in $\mathcal{O}(n)$ time. Further, the remaining operations for obtaining the new objective $\hat{F}_\gamma(\hat{\mathbf{c}})$ and its gradient $\nabla \hat{F}_\gamma(\hat{\mathbf{c}})$ can be performed in $\mathcal{O}(nr + r^2) = \mathcal{O}(nr)$ time. The space consumption, dominated by $\mathbf{K}_R$, is $\mathcal{O}(nr)$. □

Thus, in combination with the L-BFGS scheme depicted above, both the runtime as well as the space consumption are reduced significantly. Again, as pointed out in Chapter 5, the parameter $r \in \{1, \ldots, n\}$ determines the trade-off between the achieved speed-up and the accuracy of the approximation. Another way to obtain (considerable) speed-ups can be achieved for the special case of a linear kernel, which we will describe next.

Linear Kernel and Sparse Data

Assume that we are given patterns in $\mathcal{X} = \mathbb{R}^d$ and let $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote the data matrix containing the training patterns as rows. In case of the linear kernel, one can write the kernel matrix as $\mathbf{K} = \mathbf{X}\mathbf{X}^{\mathrm{T}} \in \mathbb{R}^{n \times n}$ and can achieve substantial computational savings by avoiding its explicit construction. This is the case, for instance, if the data resides

in a low-dimensional feature space (i. e., $d \ll n$) or due to the data matrix being sparse, meaning that it contains only few nonzero entries.[7]

Lemma 6.3 *For a linear kernel with patterns in $\mathcal{X} = \mathbb{R}^d$, one can compute the objective $F_\gamma(\mathbf{c})$ and the gradient $\nabla F_\gamma(\mathbf{c})$ in $\mathcal{O}(nd)$ time using $\mathcal{O}(nd)$ space for a given candidate solution $\mathbf{c} \in \mathbb{R}^n$.*

Proof: Due to the linear kernel, one can compute

$$\mathbf{K}\mathbf{c} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{c}) \tag{6.15}$$

and thus all predictions $f(\bar{\mathbf{x}}_1), \ldots, f(\bar{\mathbf{x}}_n)$ in $\mathcal{O}(nd)$ time. In the same manner, one can obtain $\mathbf{c}^{\mathrm{T}}\mathbf{K}\mathbf{c}$ and $\mathbf{K}\mathbf{a}$ in $\mathcal{O}(nd)$ time (where the vector $\mathbf{a} \in \mathbb{R}^n$ can be computed in $\mathcal{O}(n)$ time given the predictions). Thus, both the objective $F_\gamma(\mathbf{c})$ and the gradient $\nabla F_\gamma(\mathbf{c})$ can be obtained in $\mathcal{O}(nd)$ time. The space requirements are bounded by the space needed to store the data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, which is $\mathcal{O}(nd)$. □

Hence, compared to a naive computation of the kernel matrix that takes $\mathcal{O}(n^2 d)$ time and $\mathcal{O}(n^2 + nd)$ space, one only needs both $\mathcal{O}(nd)$ time and space, which depicts a significant speed-up. For high-dimensional but sparse data (i. e., if the matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ contains only $s \ll nd$ nonzero entries), one can further reduce the computational cost in the following kind of way:[8]

Lemma 6.4 *For a linear kernel with patterns in $\mathcal{X} = \mathbb{R}^d$ and data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ with $s \ll nd$ nonzero entries, one can compute the objective $F_\gamma(\mathbf{c})$ and the gradient $\nabla F_\gamma(\mathbf{c})$ in $\mathcal{O}(s)$ time using $\mathcal{O}(s)$ space for a given candidate solution $\mathbf{c} \in \mathbb{R}^n$.*

Proof: Without loss of generality, we assume that $s \geq n - 1$ holds (since all but one patterns consisting of zeros can be removed). Similar to the derivations above, one can compute $\mathbf{K}\mathbf{c} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{c})$ and therefore the predictions $f(\bar{\mathbf{x}}_1), \ldots, f(\bar{\mathbf{x}}_n)$ as well as $\mathbf{a} \in \mathbb{R}^n$ in $\mathcal{O}(s)$ time using standard sparse matrix multiplication techniques. In the same way, one can compute $\mathbf{c}^{\mathrm{T}}\mathbf{K}\mathbf{c}$ and $\mathbf{K}\mathbf{a}$ in $\mathcal{O}(s)$ time. Hence, both the objective $F_\gamma(\mathbf{c})$ and the gradient $\nabla F_\gamma(\mathbf{c})$ can be obtained in $\mathcal{O}(s)$ time spending $\mathcal{O}(s)$ space. □

The experimental evaluation in the next section will demonstrate that sparse data sets can be handled in an extremely effective manner based on the shortcut depicted above.

[7] A popular example of sparse data are text data sets, where one usually simply counts the occurrences of words. Here, the dimension d of the input space corresponds to each possible word and is, thus, very large. However, only few of these words occur in a particular text, which leads to a sparse data matrix.

[8] Note that the term s is sometimes used to denote the average number of nonzero entries *per pattern* $\mathbf{x}_i \in \mathcal{X}$ in the training set.

6.3.4 Competitors: Steepest Descent and Newton's Method

Let us briefly sketch the benefits of using the BFGS scheme as well as its limited memory variant compared to its direct competitors, steepest descent and Newton's Method. Since steepest descent only resorts to function and gradient calls as well, the corresponding computational costs are the same and the shortcuts depicted above can be integrated in the same manner. However, the quasi-Newton schemes incorporate second order information via the approximation of the inverse of the Hessian and "the improvement over steepest ascent is dramatic, especially on difficult problems" [108, p. 136]. Thus, it makes sense to resort to the more sophisticated search directions used by the quasi-Newton schemes. Newton's method makes use of the Hessian and, thus, directly resorts to the second order information. This usually results in a better convergence behavior, at least on convex tasks. However, there are two clear disadvantages of this approach in this context:

(a) Firstly, the computational cost per iteration is $\mathcal{O}(n^3)$ (for the exact case), and, thus, the number of iterations has to be far less compared to the quasi-Newton schemes in order to justify this additional effort. Further, for non-convex tasks, the Hessian might not be positive definite and, thus, might not yield a descent direction. Here, additional care has to be taken to ensure the positive definiteness [36].

(b) Secondly, exploiting the specific properties of a linear kernel (for sparse data) as well as a incorporating kernel matrix approximation schemes seem to be more difficult.

Thus, both the simplicity of the approach (no matrices have to be adapted) and its superior computational behavior compared to steepest descent render quasi-Newton schemes well-suited candidates for the task.

6.4 Experimental Analysis

We will now provide the experimental evaluation including further data sets and a comparison of several competing methods.

6.4.1 Experimental Setup

The runtime experiments were performed on a 3 GHz Intel Core™ Duo PC running `Ubuntu 10.04`. We will start by providing the experimental setup, which encompasses implementation details, the considered data sets, model selection issues, and a description of the different learning schemes used for the comparison.

Data Set	N	d	Comment	Data Set	N	d	Comment
Gaussian5	500	500	artificial	USPS(8,0)	2,261	256	rescaled
Gaussian6	500	500	artificial	MNIST(1,7)	2,000	784	rescaled
Gaussian7	500	500	artificial	MNIST(2,5)	2,000	784	rescaled
Moons	500	2	artificial	MNIST(2,7)	2,000	784	rescaled
USPS(2,5)	1,645	256	rescaled	MNIST(3,8)	2,000	784	rescaled
USPS(2,7)	1,721	256	rescaled	TEXT	1,946	7,511	sparse
USPS(3,8)	1,532	256	rescaled				

TABLE 6.1. *All artificial and real-world data sets used in the experimental evaluation.*

Implementation Details

The implementation of the framework proposed in this chapter (QN-S^{3}VM) is based on the programming language Python, the Scipy package (using the L-BFGS implementation optimize.fmin_l_bfgs_b [26] with m = 50, factr = 10^{10}, and maxfun = $1,000$), and the Numpy package. The function and gradient evaluations are based on efficient matrix operations provided by the Numpy package. Further, to avoid numerical instabilities, we make use of $\log(1 + \exp(t)) \approx t$ and $\frac{\exp(t)}{1+\exp(t)} \approx 1$ given $t \geq 500$ for both the function and gradient calls.

Data Sets

In addition to the data sets introduced in the previous chapters, we consider further real-world data sets: The first ones are based on the MNIST database [97], where we focus on similar pairs of classes as for the USPS data set (which seem to be difficult to separate). We also perform the same rescaling for the pixel values as for the USPS data set, i. e., we rescale all pixel values such that they fit into the unit interval $[0, 1]$. Except for the runtime experiments, we restrict the size of each data set instance to $N = 2,000$. In addition, we consider the sparse TEXT data set (with $\frac{s}{Nd} \approx 0.0073$) that is composed of the mac and mswindows classes of the Newsgroup20 data set (and that is preprocessed as described by Chapelle and Zien [32]). See Table 6.1 for an overview of all data sets considered in this chapter.

Model Selection

Model selection is difficult in semi-supervised learning settings since the amount of labeled patterns is usually very small. For this reason, we considered *non-realistic scenarios* in Chapter 5 for tuning the non-fixed parameters, i. e., we used the labeled patterns given in the test set. On the one hand, this scenario can be used to analyze if the model is, in principle, capable of adapting to the structure of the data at hand. On the other hand,

the obtained results are not meaningful for real-world settings.[9]

In this chapter, we will therefore consider an additional *realistic scenario*, where only the labels in the training set are used (which consists of both a small amount of labeled and a large amount of unlabeled patterns). The non-fixed parameters are then tuned on the training set via 5-fold cross-validation [73]. Again, we make use of a linear kernel and an RBF kernel, depending on the particular experiment.

Competing Approaches

We will consider a standard support vector machine implementation as a baseline. Further, in addition to the two semi-supervised local search strategies presented in this work, we will resort to another state-of-the-art method in this field. Note again that the key idea of all semi-supervised approaches is quite similar, i.e., a single initial guess is improved via some kind of local search. However, for each of these schemes, the particular objective function and the local search framework are different. For most experiments, we will use a linear kernel. If an RBF kernel is used, we will tune the kernel width $\sigma \in \mathbb{R}^+$ via the set $\{0.01\hat{\sigma}, 0.1\hat{\sigma}, 1\hat{\sigma}, 10\hat{\sigma}, 100\hat{\sigma}\}$ of possible assignments, where $\hat{\sigma}$ is (as in the previous chapter) an estimate of the maximum distance between any pair of patterns.[10]

LIBSVM. We consider the `LIBSVM` implementation for support vector machines provided by Chang and Lin [29] as a supervised baseline. Except for the two parameters `C` and `gamma`, we resort to the default values for all other parameters. The parameter `C` determines the trade-off between large margin and loss caused by the labeled patterns lying within the margin, and we use the set $\texttt{C} \in \{2^{-10}, \ldots, 2^{10}\}$ as possible assignments. The `LIBSVM` implementation resorts to a slightly different definition of the RBF kernel that is given by $k(\mathbf{u}, \mathbf{v}) = \exp(-\texttt{gamma}\|\mathbf{u} - \mathbf{v}\|^2)$ for two patterns $\mathbf{u} \in \mathbb{R}^d$ and $\mathbf{v} \in \mathbb{R}^d$ with $\texttt{gamma} \in \mathbb{R}^+$. To obtain a similar setting as for the other definition of the RBF kernel, we set $\texttt{gamma} = \frac{1}{2\sigma^2}$ and use the same assignments for σ as depicted above.

UniverSVM. As semi-supervised state-of-the-art method, we consider the `UniverSVM` approach proposed by Collobert *et al.* [38]. The involved parameters are tuned via $(\texttt{C}, \texttt{C*}) \in \{2^{-10}, \ldots, 2^{10}\} \times \{\frac{0.01}{u}, \frac{1.0}{u}, \frac{100.0}{u}\}$. Further, the ratio between the two classes is provided to the algorithm via the `-w` option (which is obtained on the whole data set in the non-realistic scenario; for the realistic scenario, only the labels in the training set are used). Except for the `-S` option (which we set to -0.3), the default values for the remaining parameters are used.

[9] Again, we would like to point out that this non-realistic scenario is often used for the experimental evaluation in the related literature [34, 149].

[10] Given by $\hat{\sigma} = \sqrt{\sum_{j=1}^{d} \left(\max([\bar{\mathbf{x}}_1]_j, \ldots, [\bar{\mathbf{x}}_n]_j) - \min([\bar{\mathbf{x}}_1]_j, \ldots, [\bar{\mathbf{x}}_n]_j) \right)^2}$ for n patterns $\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_n \in \mathcal{X}$.

S^{2}RLSC. Further, we consider the S^{2}RLSC approach derived in Chapter 5 as semi-supervised competitor, where we consider the single-restart variant that improves the initial guess obtained via the labeled part of the data. Concerning the parameters, we resort to the same experimental setup as depicted in Chapter 5, i.e., we use the initial guess obtained via the supervised RLSC model, tune the two non-fixed parameters via $(\lambda, \gamma) \in \{2^{-10}, \ldots, 2^{10}\} \times \{0.01, 1, 100\}$, set $\varepsilon = 0.1$, and set the parameter b_c to an appropriate estimate obtained via the labeled patterns (similar to the UniverSVM approach).

QN-S^{3}VM. Finally, we will consider the QN-S^{3}VM scheme derived in this chapter. The cost parameters λ and γ are tuned on the grid $(\lambda, \gamma) \in \{2^{-10}, \ldots, 2^{10}\} \times \{0.01, 1, 100\}$ of possible assignments. If not noted otherwise, we use a short sequence of annealing steps (i.e., $\alpha_1 = 0, \alpha_2 = 0.01, \alpha_3 = 0.1, \alpha_4 = 1.0$). Further, we do not consider any balance constraint.

6.4.2 Results

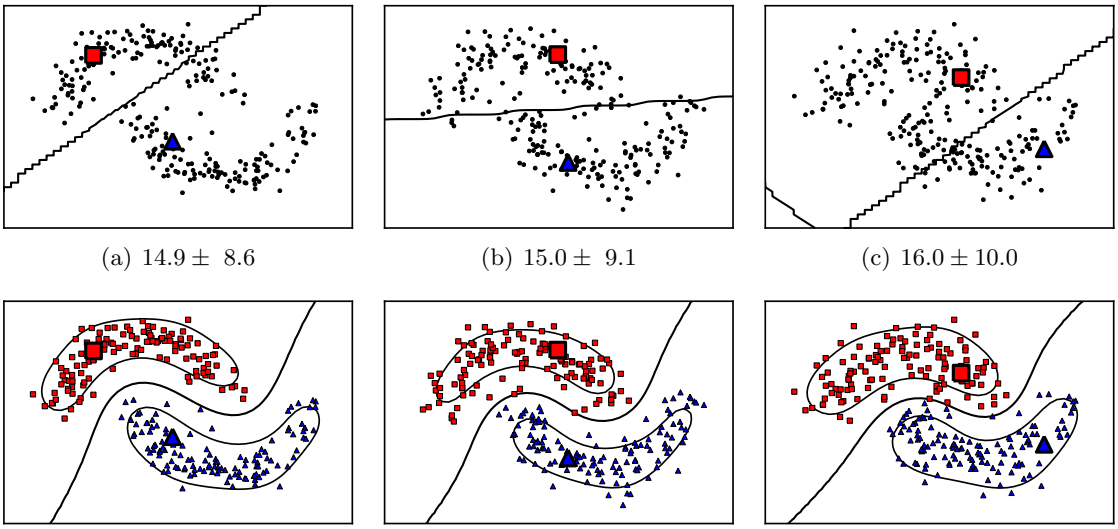
We will provide both experiments related to the model flexibility and the classification performance as well as experiments related to the computational behavior of all methods.

Model Flexibility

Let us again consider the Moons data set, which is said to be a difficult task for semi-supervised support vector machines due to its non-linear structure. In Figure 6.3, the results of the LIBSVM implementation (top row) and of the QN-S^{3}VM implementation (bottom row) are given using slightly varying distributions of the data set (for both methods, an RBF kernel is used). To select the model parameters, we consider the non-realistic scenario. For all figures, the average test error (with one standard deviation) over 10 random partitions into labeled, unlabeled, and test patterns is given. Clearly, the supervised approach is not able to generate reasonable models. The semi-supervised approach, however, can successfully incorporate the additional information provided by the unlabeled patterns, even for the Moons data set instance with a less obvious cluster structure, see Figure 6.3 (f). Note that these examples also demonstrate that *no* balance constraint is needed in principle for semi-supervised support vector machines.

Classification Performance

We will now evaluate the classification performances of all competing approaches given both the non-realistic scenario as well as the realistic one. For each data set, we will further consider two different amounts of labeled, unlabeled, and test patterns. For the sake of simplicity, we will resort to a linear kernel for all methods.



(a) 14.9 ± 8.6 (b) 15.0 ± 9.1 (c) 16.0 ± 10.0

(d) 0.0 ± 0.0 (e) 1.2 ± 2.5 (f) 2.6 ± 2.3

FIGURE 6.3. *The large red squares and blue triangles depict the labeled data; the small black dots the unlabeled data. Further, the smaller red squares and blue triangles depict the partition of the unlabeled patterns computed by the semi-supervised approach. Clearly, the* `LIBSVM` *implementation (top row) is not able to generate appropriate models due to the lack of labeled data, whereas the* `QN-S`^{3}`VM` *approach (bottom row) can successfully incorporate the unlabeled data. The average test errors (with one standard deviation) over* 10 *random partitions are reported.*

Non-Realistic Scenario. In Table 6.2, the test errors (along with the one standard deviations) averaged over 10 random partitions are given for the non-realistic scenario. It can be clearly seen that the semi-supervised approaches yield better results compared to the supervised `LIBSVM` baseline, even if only few labeled patterns are given. Thus, all these approaches can successfully incorporate the additional information provided by the unlabeled data. While the performance of the semi-supervised approaches is promising on all considered data sets, it seems to be especially well-suited for data sets exhibiting a clear low-density area between the two classes. This is the case, for instance, for the `Gaussian5` and for the `Gaussian7` data sets. A significant improvement is also given for the `Text` data set. Here, for one of the two data set instances, the semi-supervised approaches can reduce the error on the test set by about 20%. Further, the approaches seem to perform well on the `USPS` data set, when only a small amount of labeled data is given (e. g., the `USPS(2,5)` data set with only 16 labeled patterns). However, the results are obtained by tuning the model parameters on the test set, which is a non-realistic assumption. In realistic scenarios, only the labeled patterns given in the training set can be used. As we will see now, this (naturally) leads to worse results.

Data Set	l	u	t	LIBSVM	UniverSVM	S^{2}RLSC	QN-S^{3}VM
Gaussian5	25	225	250	13.0 ± 2.9	1.0 ± 0.5	0.6 ± 0.5	0.4 ± 0.4
Gaussian5	50	250	250	5.8 ± 2.2	1.0 ± 0.4	0.6 ± 0.5	0.4 ± 0.4
Gaussian6	25	225	250	32.6 ± 3.2	24.4 ± 3.0	20.7 ± 2.0	17.9 ± 1.4
Gaussian6	50	250	250	25.1 ± 3.2	19.7 ± 3.0	16.5 ± 3.2	16.1 ± 2.4
Gaussian7	25	225	250	17.4 ± 6.6	7.6 ± 12.1	6.8 ± 12.5	1.0 ± 0.5
Gaussian7	50	250	250	6.7 ± 1.5	1.6 ± 0.8	0.8 ± 0.6	0.8 ± 0.6
USPS(2,5)	16	806	823	9.4 ± 5.1	3.2 ± 0.5	2.9 ± 0.4	3.2 ± 2.4
USPS(2,5)	32	790	823	4.7 ± 0.7	3.2 ± 0.5	2.9 ± 0.4	2.8 ± 0.7
USPS(2,7)	17	843	861	4.6 ± 3.0	1.5 ± 0.3	1.0 ± 0.1	1.2 ± 0.2
USPS(2,7)	34	826	861	2.5 ± 1.0	1.4 ± 0.2	1.0 ± 0.1	1.2 ± 0.2
USPS(3,8)	15	751	766	12.0 ± 8.2	4.8 ± 1.1	4.1 ± 1.3	5.7 ± 4.8
USPS(3,8)	30	736	766	6.6 ± 2.1	4.0 ± 1.1	3.9 ± 1.2	3.9 ± 1.3
USPS(8,0)	22	1,108	1,131	4.8 ± 1.7	1.7 ± 0.7	1.2 ± 0.2	1.4 ± 0.6
USPS(8,0)	45	1,085	1,131	2.7 ± 0.8	1.3 ± 0.4	1.1 ± 0.2	1.4 ± 0.6
MNIST(1,7)	40	960	1,000	2.8 ± 1.1	1.9 ± 0.3	1.6 ± 0.3	1.5 ± 0.3
MNIST(1,7)	100	900	1,000	1.9 ± 0.4	1.8 ± 0.3	1.5 ± 0.2	1.4 ± 0.2
MNIST(2,5)	40	960	1,000	4.4 ± 1.0	2.8 ± 0.4	2.3 ± 0.3	2.3 ± 0.3
MNIST(2,5)	100	900	1,000	3.3 ± 0.5	2.9 ± 0.4	2.3 ± 0.3	2.4 ± 0.3
MNIST(2,7)	40	960	1,000	4.0 ± 0.8	2.6 ± 0.3	2.3 ± 0.1	2.5 ± 0.3
MNIST(2,7)	100	900	1,000	2.9 ± 0.7	2.4 ± 0.4	2.3 ± 0.3	2.2 ± 0.4
MNIST(3,8)	40	960	1,000	8.8 ± 1.8	6.9 ± 2.0	6.6 ± 1.3	6.5 ± 1.9
MNIST(3,8)	100	900	1,000	5.9 ± 0.8	4.9 ± 0.5	5.3 ± 3.4	4.7 ± 0.3
TEXT	48	924	974	23.5 ± 6.7	6.5 ± 0.9	5.5 ± 0.9	6.2 ± 1.4
TEXT	389	584	973	4.8 ± 0.7	4.2 ± 0.6	3.8 ± 0.7	4.1 ± 0.5

TABLE 6.2. *Classification performances of all competing approaches for the non-realistic scenario. Clearly, the semi-supervised approaches yield better results in general. This seems to be especially the case if (1) the data exhibits a clear low-density area between both classes and if (2) sufficient unlabeled patterns are used to describe the (high-dimensional) structure of the data.*

Realistic Scenario. For the realistic scenario, the classification performances of all approaches are clearly worse compared to those of the non-realistic scenario, see Table 6.3. A reasonable explanation for this fact is that a small set of labeled patterns might not be sufficient for the model selection phase and that the best performing parameters cannot be selected in a reliable manner. Further, the performances of all semi-supervised approaches is quite similar, and none of the three methods seems to outperform the two other ones. Still, the results of the semi-supervised approaches are *not worse* compared to the ones of the supervised baseline in most cases. This is quite reasonable since all local search schemes improve a single guess obtained via a corresponding supervised model. On the one hand, one gets stable results via such an approach since the subsequent local search schemes only *fine-tune* these initial guesses. However, on the other hand, a small set of labeled patterns might lead to a bad initial guess, and thus, to bad results in general. We

Data Set	l	u	t	LIBSVM	UniverSVM	S^{2}RLSC	QN-S^{3}VM
Gaussian5	25	225	250	13.2 ± 2.8	1.5 ± 0.6	6.6 ± 2.6	1.7 ± 0.6
Gaussian5	50	250	250	6.3 ± 2.4	1.6 ± 0.7	3.3 ± 2.6	1.3 ± 0.9
Gaussian6	25	225	250	38.2 ± 6.7	25.8 ± 3.4	27.5 ± 5.2	27.4 ± 4.9
Gaussian6	50	250	250	25.4 ± 3.2	22.2 ± 3.3	23.1 ± 4.8	24.6 ± 3.9
Gaussian7	25	225	250	20.6 ± 11.5	13.1 ± 15.6	12.0 ± 13.6	1.8 ± 0.5
Gaussian7	50	250	250	6.9 ± 1.6	2.2 ± 1.3	4.1 ± 2.4	2.4 ± 1.2
USPS(2,5)	16	806	823	10.5 ± 4.7	5.6 ± 3.6	6.3 ± 2.8	5.4 ± 1.7
USPS(2,5)	32	790	823	5.4 ± 0.8	3.9 ± 0.6	6.4 ± 2.0	5.1 ± 1.5
USPS(2,7)	17	843	861	4.9 ± 2.9	2.3 ± 1.0	1.4 ± 0.2	3.6 ± 3.3
USPS(2,7)	34	826	861	2.8 ± 1.1	2.1 ± 1.0	1.3 ± 0.2	2.2 ± 0.6
USPS(3,8)	15	751	766	12.9 ± 8.3	8.0 ± 3.5	6.2 ± 2.7	10.8 ± 8.9
USPS(3,8)	30	736	766	7.3 ± 2.1	6.9 ± 2.2	6.4 ± 2.8	6.0 ± 2.9
USPS(8,0)	22	1,108	1,131	5.0 ± 2.0	2.8 ± 1.8	1.8 ± 0.6	3.4 ± 1.3
USPS(8,0)	45	1,085	1,131	3.0 ± 0.8	3.0 ± 1.8	2.0 ± 0.5	2.7 ± 1.0
MNIST(1,7)	40	960	1,000	3.0 ± 1.2	3.4 ± 1.5	2.4 ± 1.0	2.6 ± 0.9
MNIST(1,7)	100	900	1,000	2.0 ± 0.3	2.3 ± 0.8	2.3 ± 0.7	2.0 ± 0.5
MNIST(2,5)	40	960	1,000	4.9 ± 1.2	3.8 ± 1.2	4.2 ± 1.5	4.1 ± 1.3
MNIST(2,5)	100	900	1,000	3.7 ± 0.4	3.6 ± 0.6	3.3 ± 0.8	3.4 ± 0.6
MNIST(2,7)	40	960	1,000	4.3 ± 1.1	3.6 ± 0.6	3.6 ± 1.2	4.0 ± 1.1
MNIST(2,7)	100	900	1,000	3.3 ± 1.0	3.3 ± 0.4	3.2 ± 0.8	3.1 ± 0.6
MNIST(3,8)	40	960	1,000	9.4 ± 2.0	10.1 ± 2.8	9.2 ± 2.2	8.8 ± 2.7
MNIST(3,8)	100	900	1,000	6.9 ± 1.9	5.9 ± 0.7	6.9 ± 1.7	5.8 ± 0.9
TEXT	48	924	974	24.8 ± 9.6	6.7 ± 0.9	6.2 ± 0.9	8.2 ± 3.2
TEXT	389	584	973	4.9 ± 0.7	4.7 ± 0.4	5.0 ± 1.1	4.6 ± 0.6

TABLE 6.3. *Classification performances of all competing approaches for the realistic scenario. As expected, the performances for all approaches (including the supervised* LIBSVM *baseline) are worse compared to the ones for the non-realistic scenario. However, the semi-supervised approaches are still capable of generating better models in general. This is especially the case if a reasonable amount of labeled data is given in the training set so that the model parameters can be selected in a reliable manner. Note that the performance gain is (still) dramatic on some data set instances like the artificial data sets or the* Text *data set.*

would like to point out that all considered methods are *deterministic* (i. e., no restarts are performed). It is an important question if the results can be improved by putting more effort into optimization. We will discuss this question below in a more detailed manner.

Computational Considerations

Let us finally analyze the practical runtime of the considered semi-supervised approaches. Naturally, the runtimes depend heavily on the particular implementation and the used programming language.[11] Thus, the experiments depicted below shall only give a rough idea of the runtimes needed in practice (e. g., for generating the above tables). Again, we

[11] The UniverSVM implementation is based on C, whereas S^{2}RLSC and QN-S^{3}VM are implemented in Python. In general, implementations based on Python are said to be much slower than pure C-implementations.

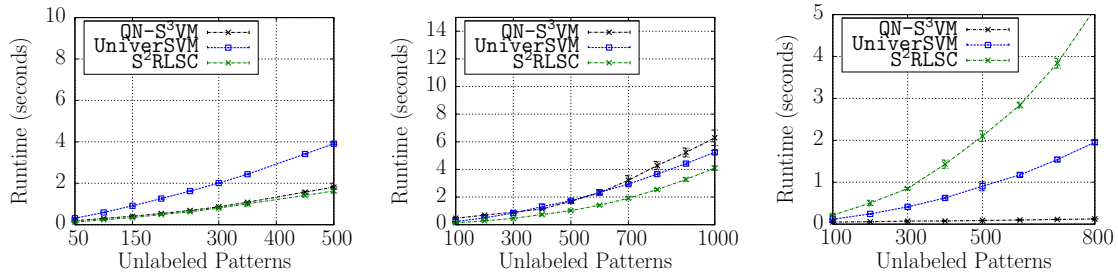


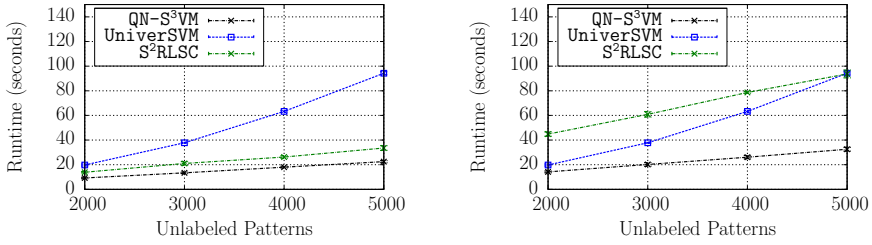
FIGURE 6.4. *Practical runtimes of all semi-supervised competitors on the* `Gaussian5`*, the* `USPS(8,0)`*, and the* `Text` *data set. The amount of labeled patterns is fixed, whereas the amount of unlabeled patterns is varied. For all data sets and all semi-supervised approaches, the average runtimes of* 10 *single executions are reported.*

would like to point out that all schemes only improve a single guess via some kind of local search and that *no* restarts are performed.

Small-Scale Settings. For the analysis of the practical runtimes to obtain the above results, we consider the `Gaussian5`, the `USPS(8,0)`, and the `TEXT` data sets. Again, we resort to a linear kernel and fix the involved model parameters ($\lambda = 1$ and $\gamma = 1$ for both `S`^{2}`RLSC` and `QN-S`^{3}`VM` and `C` $= 1$ and `C`$^* = 1$ for `UniverSVM`). Further, the amount of labeled patterns is fixed to $l = 50$, $l = 22$, and $l = 48$, respectively. In Figure 6.4, the runtime behavior for all approaches given a varying amount of unlabeled patterns is shown. For the `Gaussian5` data set, both the `S`^{2}`RLSC` and the `QN-S`^{3}`VM` implementations seem to outperform the `UniverSVM` implementation. For the `USPS(8,0)` data set, all methods exhibit a quite similar runtime performance. This is not the case for the sparse `Text` data set: Here, the `S`^{2}`RLSC` implementation clearly shows a worse runtime behavior compared to the other two ones. However, for this data, the `QN-S`^{3}`VM` implementation can benefit from the computational shortcut for sparse data depicted above. Note that even with 800 unlabeled examples, the practical runtime is less than a tenth of a second for `QN-S`^{3}`VM`.

Again, we would like to point out that these results should only give a vague idea of the runtime behavior. Especially for the `S`^{2}`RLSC` implementation, it seems that significant speed-ups could be obtained via a pure `C` implementation (the iteration over all iterations forms a bottleneck for the `Python` implementation).

Large-Scale Settings. To sketch the applicability in large-scale scenarios, we consider the `MNIST(1,7)` data set instance, where we now use all patterns and do not restrict the size of the training set (as done for the exhaustive comparison given above): For such a large-scale setting, approximation schemes like the ones depicted for both the `S`^{2}`RLSC` and the `QN-S`^{3}`VM` approach become essential. On some data sets, such a scheme has the potential



(a) $r = 500$ (b) $r = 1,000$

FIGURE 6.5. *Runtime results on the* `MNIST(1,7)` *data set for* $r = 500$ *and* $r = 1,000$. *All approaches exhibit a comparable runtime performance. Further, the kernel matrix scheme has the potential to significantly reduce the practical runtimes.*

to dramatically reduce the runtime while not sacrificing a good classification performance, see Chapter 5. For the following evaluation, we set the number of sub-columns/regressors to $r = 500$ and $r = 1,000$, respectively, and select the sub-columns/regressors uniformly at random; the remaining setup is the same as above. Again, the amount of unlabeled patterns is varied, see Figure 6.5. The runtime performances indicate a similar runtime behavior of all approaches. Naturally, the practical runtime increases for both `S2RLSC` and `QN-S3VM` in case the number of sub-columns/regressors is increased. Thus, in situations where such a kernel matrix scheme can be applied without sacrificing the accuracy, both methods can be accelerated significantly.

6.5 Discussion: Model Selection and Optimization

The above derivations show that (a) the semi-supervised approaches can successfully incorporate unlabeled data in an efficient manner, and that (b) even model selection seems to work (to some degree) in realistic scenarios with only few labeled patterns. However, the out of the box application of such semi-supervised schemes does not work in general for all data sets. We will now discuss possible problems related to model selection and optimization issues that arise in the considered semi-supervised learning settings.

6.5.1 Parameters, Parameters, and Parameters

Let us assume that one is capable of computing exact solutions for the combinatorial/non-convex tasks at hand. Aiming at a comparison of different methods or at an application of such schemes on real-world data, one is still faced with the following problems:

(a) *Data Sets:* Each data set and each partition into labeled, unlabeled, and test patterns give rise to a particular classification task. This renders a meaningful comparison of competing semi-supervised schemes difficult since it requires even more settings to

be compared (in contrast to, e. g., pure supervised learning settings).

(b) *Model Parameters:* Given a particular data set and a fixed partition, each assignment of the model parameters gives rise to a particular optimization task. Thus, one is given a *huge* variety of possible problem instances that have to be evaluated for a meaningful comparison (in contrast to, e. g., the benchmark tasks given in non-convex global stochastic optimization).

(c) *Objective Functions:* As depicted above, a common approach is to employ surrogate loss functions that render the original task more amenable to efficient optimization. While this is a reasonable approach for approximating the tasks at hand, it renders a mutual comparison of competing methods difficult. This is due to the fact that one cannot simply assess the performances based on the objective values (since *different* objectives are minimized).

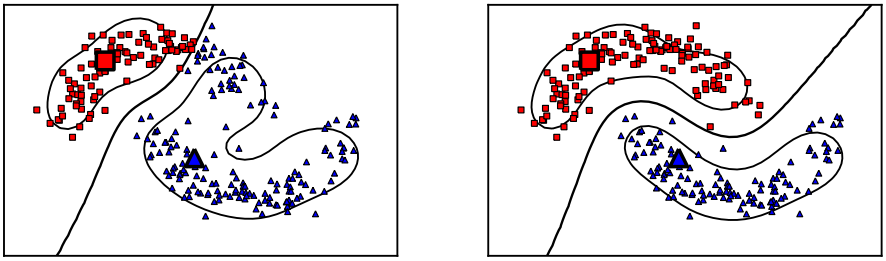
(d) *Labeled Data:* The non-realistic scenario is reasonable since one can analyze if the model is capable of adapting to the structure of the data at hand (e. g., a linear support vector machine cannot adapt to the non-linear structure present in the `Moons` data set). However, in real-world settings, one has only access to a small set of labeled patterns for model selection. As mentioned above, common methods like cross-validation might not yield reasonable parameters in a stable manner. This renders the out-of-the-box application of semi-supervised approaches difficult in real-world settings.

(e) *Misleading Parameters:* Assume that one fixes the parameters beforehand. Then, these parameters can also be misleading and an optimal solution could consists in, for instance, assigning all parameters to one and the same class (for the semi-supervised setting). Thus, the performance would become worse compared to a supervised model even though an optimal solution is computed.

Hence, even if one is able to compute optimal solution in an efficient manner, one will still be faced with the parameter problems depicted above. Note that the local search strategies have the potential to work extremely well on data sets and parameter configurations giving rise to somewhat easy tasks. However in real-world settings, one usually has to fix the parameters beforehand. As we will show now, for such a fixed setup, it makes sense to put more effort into optimization.

6.5.2 More Optimization

For a fixed setup (i. e., fixed data set, fixed partition for the unlabeled patterns, fixed model parameters, and fixed objective function), one has to address the combinatorial/non-



(a) $F_\gamma(\mathbf{c}) = 0.18342$ (b) $F_\gamma(\mathbf{c}) = 0.13661$

FIGURE 6.6. *The large red squares and blue triangles depict the labeled patterns; the final partitions of the unlabeled patterns are shown as small red squares and blue triangles. For this fixed setup, the* QN-S^{3}VM *approach shown in Figure (a) yields a worse solution compared to the more exhaustive stochastic search performed by the* CMA-ES *implementation that is depicted in Figure (b).*

convex optimization task itself. The results provided in Chapter 4 indicate that computing exact solutions is possible but extremely time-consuming. Thus, in order to apply the concept of semi-supervised support vector machines on real-world data, one has to resort to heuristics that compute (possibly) suboptimal solutions. As we have seen above, improving a single initial guess via a local search scheme is a well-known strategy in this field and has the potential to yield good candidate solutions in an efficient kind of way. However, such a local search strategy might also fail. As pointed out by Joachims [83], an important question in this field is if the "results get even better, if we invest more time into search"? We will now investigate this question briefly.

More Optimization Pays Off

The results given in Chapter 5 indicate that using a large number of restarts instead of improving a single guess can lead to better results. We will now perform a similar experiment for the continuous optimization perspective and compare the QN-S^{3}VM approach with another stochastic optimization framework. For this sake, we consider a fixed setup, i. e., we use the Moons data set (with $N = 100$) and a particular partition of the patterns into labeled and unlabeled patterns, see Figure 6.6. Further, we fix the model parameters to $\sigma = 0.1\hat{\sigma}$, $\lambda = 2^{-7}$, and $\gamma = 1$, and use the subset of regressors approximation scheme with $r = 20$. Thus, this setup gives rise to a single non-convex optimization task in $\mathbb{R}^{20}$ with the associated objective $\hat{F}_\gamma(\hat{\mathbf{c}})$ to be minimized.

As a competitive optimization engine, we consider the *covariance matrix adaptation evolution strategy* [109], which depicts a stochastic search method for addressing non-convex tasks. For the considered implementation (CMA-ES) [72], we resort to the default parameter setting. In Figure 6.6, the outcome of a comparison of the QN-S^{3}VM and the CMA-ES implementation is given. It can be seen that the latter implementation yields

a smaller objective (and the desired partition). Hence, this particular example clearly demonstrates that putting more effort into optimization can be useful.[12]

The Curse of Dimensionality

In general, putting more effort into optimization is, of course, a good idea. However, exploring the search space extensively becomes more and more difficult in high-dimensional search spaces (in the above case, the dimension of the search space is determined by the parameter $r \leq n$). This is another example of the curse of dimensionality sketched in Chapter 2: With increasing dimension of the search space, more and more candidate solutions need to be evaluated for a meaningful exploration. In other words, the search space becomes sparsely populated by the candidate solutions in high dimensions.

For the continuous optimization task considered in this chapter, the search space is given by $\mathbb{R}^n$ for the general case and by $\mathbb{R}^r$ for the subset of regressors scheme with $r \leq n$. Hence, global stochastic optimization schemes are (only) feasible as long as n (or r) is not too large. If this is not the case, one has to resort to local search heuristics like the ones used in this chapter.

6.6 Concluding Remarks

In this chapter, we depicted the application of quasi-Newton schemes for the continuous optimization task induced by semi-supervised support vector machines. It turns out that this type of optimization scheme is well-suited for the given task due to its conceptual simplicity and the possibility to greatly reduce the needed runtime by means of the proposed shortcuts, both for sparse and non-sparse data. The experiments conducted in this chapter indicate a similar classification performance of all competing semi-supervised approaches, which was clearly better than the one of the supervised baseline. This was especially the case for data sets exhibiting a low-density between the classes to be separated (as it seems to be the case for, e. g., the `TEXT` data set). The superior performance compared to the supervised model is not surprising since the considered semi-supervised schemes are all based on the idea to improve a single initial guess obtained via a supervised model. Thus, such frameworks can be seen as a rather conservative approach since one aims at computing solutions that are, at least, not worse compared to the initial candidate solution.

[12] It seems that this is especially the case when only few unlabeled patterns are given since the structure of the data is not well described in these cases (and this gives rise to more complicated tasks). Note that a smaller objective value does not necessarily correspond to a better partition, i. e., a non-optimal solution with respect to the objective might correspond to a desired partition of the unlabeled patterns.

Part III

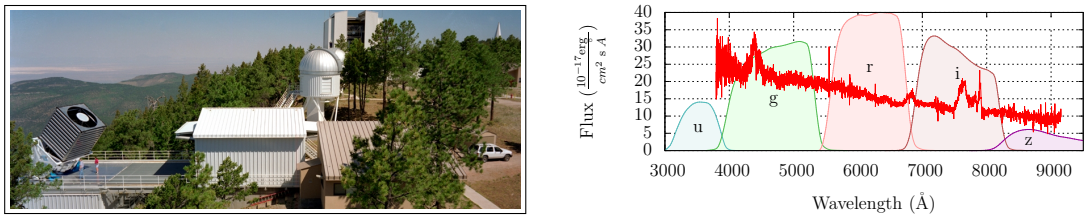
Applications in Astronomy

CHAPTER 7

Machine Learning on Earth

Modern telescopes in the field of astronomy gather huge amounts of data. On the one hand, this offers the opportunity to generate meaningful machine learning models and, hence, to automatically retrieve useful information from the corresponding databases in an efficient kind of way. On the other hand, the sheer data volume and the lack of labeled data lead to problem-specific challenges that render sophisticated strategies essential for taking advantage of the data-rich situation. Surprisingly, machine learning techniques have not been applied extensively in this field and are identified to become essential for the analysis of astronomical data in the future [6, 17]. In this chapter, we will describe the extraction of physically motivated features from raw spectra for the task of identifying so-called quasars (a special type of astronomical objects) in spectroscopic surveys. As we will see, the extraction of such features in the preprocessing phase can lead to a significantly better classification performance compared to a straightforward use of the raw data. Continuum-subtracted versions of the spectra are the basis for this extraction phase. To obtain these versions, we will propose modified support vector regression models and will show (a) how to incorporate adaptable loss functions into the framework and (b) how to address the induced optimization tasks efficiently. In addition to this classification approach, we will sketch possible application domains of semi-supervised learning schemes in astronomy like the ones discussed in the previous chapters.

Outline. In Section 7.1, we will start by describing the data situation for one of the largest astronomical catalogs nowadays available in the field of astronomy and will provide a brief physical background related to quasars. Afterwards, in Section 7.2, we will describe



(a) Apache Point Observatory

(b) Spectroscopic and Photometric Data

FIGURE 7.1. *The telescope at the* Apache Point Observatory *collects both photometric and spectroscopic data [132]. The photometric data is given in terms of grayscale photos that are obtained via five filters covering different wavelength ranges, called the* **u**, **g**, **r**, **i**, *and* **z** *bands, see Polsterer [115] for details. For a small subset of detected objects, detailed follow-up observations in terms of spectra are available.*

the feature extraction scheme as well as the adaptable continuum fitting framework. The potential of semi-supervised learning schemes in the field of astronomy will be sketched in Section 7.3; concluding remarks will be provided in Section 7.4.

7.1 Motivation

The *Sloan Digital Sky Survey* (SDSS) is among the most important astronomical catalogs nowadays available and is currently based on raw data of about 60 terabytes. One of the main objectives of this survey is to find so-called *quasi-stellar radio sources* (quasars). In this section, we will provide a brief overview of the collected data and will sketch the physical background related to these objects.

7.1.1 Massive Data in Astronomy

A wide range of different types of data is collected in the field of astronomy. In this chapter, we will focus on the data gathered for the above catalog. The associated 2.5-meter telescope of this project is equipped with two special-purpose instruments: a 120 mega pixel camera and a pair of spectrographs, see Figure 7.1 (a). The former instrument produces *photometric data* whereas the latter one collects *spectroscopic data*.

Photometric Data

Photometric data corresponds to grayscale photos obtained at different wavelengths. The 120 mega pixel camera, for instance, gathers this type of data through five filters that cover different wavelength ranges, called the *u*, *g*, *r*, *i*, and *z bands* [132], see Figure 7.1 (b). Thus, for each region of the observed sky, five grayscale photos are given as raw data for the anticipated catalog. This raw data usually forms the basis for a preprocessing phase that aims at (a) detecting the visible objects (e. g., stars, galaxies, quasars, ...),

and at (b) retrieving physically motivated features for each of the detected objects. One group of these features are the so-called *magnitudes*, which depict logarithmic measures of the brightness. Among the most established feature extraction schemes (leading to these magnitudes) are the *PSF* and the *Model* approach [132], each yielding a single feature per object and per band. These ten photometric features are stored in the catalog, along with other extracted features and data items. An important issue in this context is the fact that this type of data is used for the *spectroscopic target selection*, i. e., the spectroscopic follow-up observations are usually made based on such extracted photometric features.

Spectroscopic Data

While obtaining photometric data is relatively inexpensive, more detailed observations in terms of spectra are only given for a comparable small amount of objects due to the involved time-consuming collection process.[1] A spectrum of an astronomical object contains precise information about the light reaching the earth at the different wavelengths, see again Figure 7.1 (b). The importance of spectra in the field of astronomy is due to the fact that ground-truth information can often only be obtained via this type of data, i. e., given only the photometric data, an expert in this field might not be able to differentiate between similar objects like special stars and quasars. Thus, these time-consuming follow-up observations are important to obtain information about an object's true nature.

Lack of Labels

Although a huge amount of unlabeled (photometric) data is given, labeled data are usually scarce in astronomy. This is mainly due to two issues: Firstly, obtaining labels usually requires expert knowledge (or at least a careful visual inspection). For instance, obtaining good labels for the task of classifying galaxies based on their shapes is quite a difficult task and led to citizen science projects like *Galaxy Zoo* [57] that aim at gathering a huge amount of labels provided by human beings. Secondly, obtaining reliable labels for, e. g., photometric data items might require time-consuming spectroscopic observations. Both problems lead to a general lack of reliable labels in this field and, thus, render a direct application of pure supervised models difficult for specific learning tasks.

7.1.2 Quasi-Stellar Radio Sources

One of the most interesting challenges in astronomy is the one of detecting distant *quasi-stellar radio sources* (quasars). Due to their extreme luminosities, quasars are among the

[1] The current release of the SDSS contains photometric data of about one billion objects whereas spectra are only available for about two million objects. However, the amount of spectra will greatly increase with future projects like the *Large Sky Area Multi-Object Fiber Spectroscopic Telescope* [138].

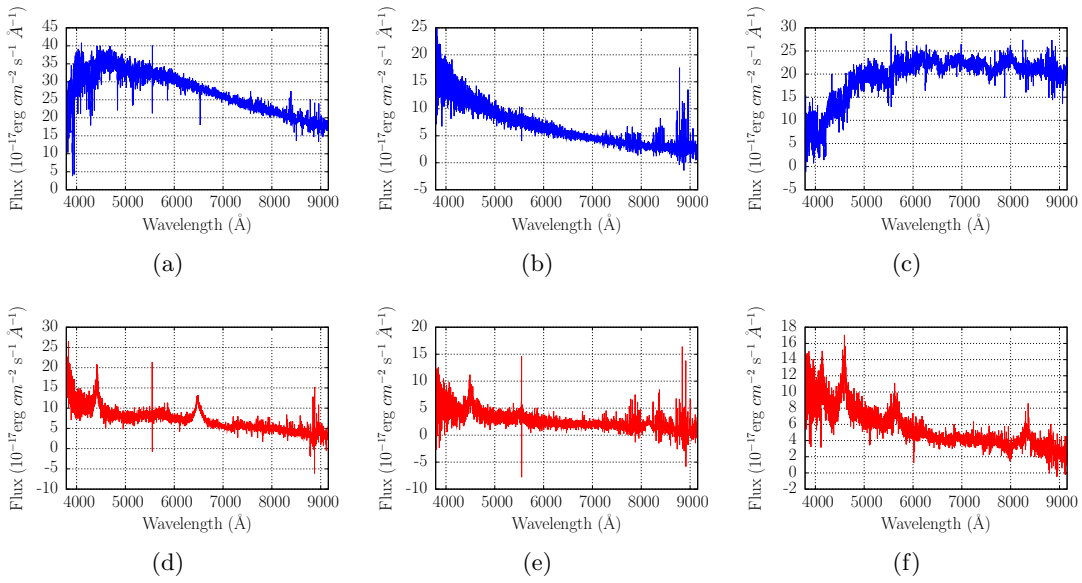


FIGURE 7.2. *Spectroscopic data for objects of type* `other` *(top row) and of type* `quasar` *(bottom row). The typical spectrum of a quasar shows broad emission lines and almost no absorption lines [132].*

most distant objects in the universe that can be observed. Depending on the distance of an object at hand, it takes up to billions of years for the emitted radiation to reach the earth. Therefore, this radiation reveals important information about the long-ago state of a quasar and, thus, about the early universe [62]. Such objects depict *point sources* and often look very similar to stars if only photometric data is considered. However, quasars exhibit remarkable spectra that look quite different from those of other objects. In Figure 7.2, several spectra of other objects (top row) and quasars (bottom row) are shown. Note that the typical spectrum of a quasar shows broad emission lines and almost no absorption lines. Here is an explanation for this phenomenon provided by an expert in this field [62]:

> *"The broad emission lines in the spectrum of a quasar result from the fact that, while observing the direct vicinity of the supermassive black hole, one observes a region with a higher gravitational potential and thus higher orbiting velocities of the accretion disk material. Such high velocities yield to a Doppler broadening of the emission lines, in this strength only present for active galactic nuclei-powered sources."*

In the following, we will derive an efficient and adaptable approach to extract the *continuum* (i. e., the rough shape) of a given spectrum and will demonstrate that such models can be used to extract meaningful, physically motivated features that capture the characteristic properties of quasars.

7.2 Detecting Quasars in Large-Scale Spectroscopic Surveys

Experts in the field of astronomy can easily identify quasars given spectroscopic data. Note that while such features are quite obvious for human beings, they can remain hidden for machine learning models due to the huge number of small emission lines and absorption lines. In this section, we will make use of expert-based features for the task of discriminating quasars from other objects given spectroscopic data. As we will see, these features will lead to a significant better classification performance compared to the direct use of the raw spectra. The basis for the definition of these features are appropriate continuum fits that are obtained via an adapted version of support vector regression schemes. We will now provide the details.

7.2.1 Speedy Adaptable Continuum Extraction

Standard schemes for obtaining continuum fits in the field of astronomy are based on, e. g., *spline models* [75, 132]. As an alternative approach, we propose the use of modified support vector regression models in this context. For this sake, we consider each spectrum as labeled training set $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathbb{R} \times \mathbb{R}$ with patterns corresponding to wavelengths and labels corresponding to flux values. As depicted in Chapter 3, standard regression models are of the form

$$\inf_{f \in \mathcal{H}_k,\, b \in \mathbb{R}} \frac{1}{l} \sum_{i=1}^{l} \mathcal{L}(y'_i, f(\mathbf{x}'_i) + b) + \lambda \|f\|^2_{\mathcal{H}_k} \tag{7.1}$$

with feature space $\mathcal{H}_k$ and appropriate loss function $\mathcal{L} : \mathbb{R} \times \mathbb{R} \to [0, \infty)$. Two issues arise in this context: Firstly, the task of estimating such a continuum actually *differs* from a standard regression task. Given a spectrum of a quasar, for instance, one would like to ignore the typical broad emission lines. This can be achieved by penalizing complex functions; however, too simple models that are not capable of adapting to the spectra at hand are not desired as well. Secondly, recurrently training a support vector regression model for high-dimensional spectra can be quite time-consuming (since the amount of needed operations is usually cubic in the number of input patterns for generating such models).

To alleviate these two problems, we propose a variant of the well-known support vector regression concept: The semiparametric representer theorem (Fact 3.2) states that every optimal hypothesis function for the above task is of the form

$$f(\cdot) = \sum_{j=1}^{l} c_j k(\cdot, \mathbf{x}'_j) + b \tag{7.2}$$

with appropriate coefficients $c_1, \ldots, c_l \in \mathbb{R}$, i. e., the optimal function is composed of the sum of weighted kernel expansions, called *basis vectors*, that are centered on the training patterns (and shifted by the offset term b).[2] The efficient optimization framework we consider in this context is based on a simple yet crucial observation: In contrast to standard regression problems, one is only given a one-dimensional input space (i. e., $\mathcal{X} = \mathbb{R}$). This means that a representation of the above form is lavish since a smooth model (desired in this context) does not require, e. g., thousands of basis vectors.[3] This leads to the subset of regressors scheme [120] that was also considered in the previous chapters, i. e., instead of making use of all l basis vectors, we will consider functions of the form

$$\hat{f}(\cdot) = \sum_{j=1}^{r} \hat{c}_{i_j} k(\cdot, \mathbf{x}'_{i_j}) + b \tag{7.3}$$

with index set $R = \{i_1, \ldots, i_r\}$ and $r \ll l$. This observation can be used to greatly speed up the computation of the desired models (see below).[4]

Non-Symmetric Differentiable Loss Functions

The considered loss function plays an important role in this context. For instance, using the square loss might be a bad choice since large emission and absorption lines are penalized heavily, and, hence, could lead to inappropriate fits for, e. g., quasars. The ε-insensitive loss shown in Figure 7.3 (b) is a better choice for such scenarios since outliers are only penalized in a linear manner. Unfortunately, the ε-insensitive loss is not differentiable, which precludes a direct application of gradient based methods. Aiming at the application of such optimization schemes, we consider

$$\mathcal{L}_{(\zeta_1,\zeta_2,\eta,\varepsilon)}(y,t) = \frac{\zeta_1}{\eta} \log\left(1 + \exp\left(-\eta\left(t - y + \varepsilon\right)\right)\right) + \frac{\zeta_2}{\eta} \log\left(1 + \exp\left(+\eta\left(t - y - \varepsilon\right)\right)\right) \tag{7.4}$$

as surrogate loss function. Here, the parameters ζ_1, ζ_2, and $\varepsilon \in \mathbb{R}^+$ determine the left slope, the right slope, and the region $[y - \varepsilon, y + \varepsilon]$ of predictions that are not penalized, respectively; the parameter $\eta \in \mathbb{R}^+$ defines the smoothness of the hinges at $y \pm \varepsilon$ (and is fixed to $\eta = 20$ throughout this chapter). In Figure 7.3, several loss function instances are shown. Note that this type of surrogate loss function has the following two beneficial properties: Firstly, it can be adapted to the task at hand. For instance, to reduce the

[2] For the RBF kernel, for instance, each such basis vector corresponds to a Gaussian bump.

[3] More precisely, a small amount of basis vectors (e. g., 50) should be sufficient for most spectra. Of course, this depends on the particular fitting task, the used kernel function, and the particular assignments for the (kernel) parameters.

[4] Note that the loss is still evaluated on *all* patterns in this case; simply considering only a small set of patterns (wavelengths) as input might not be reasonable due to large emission lines and absorption lines present in such spectroscopic data.

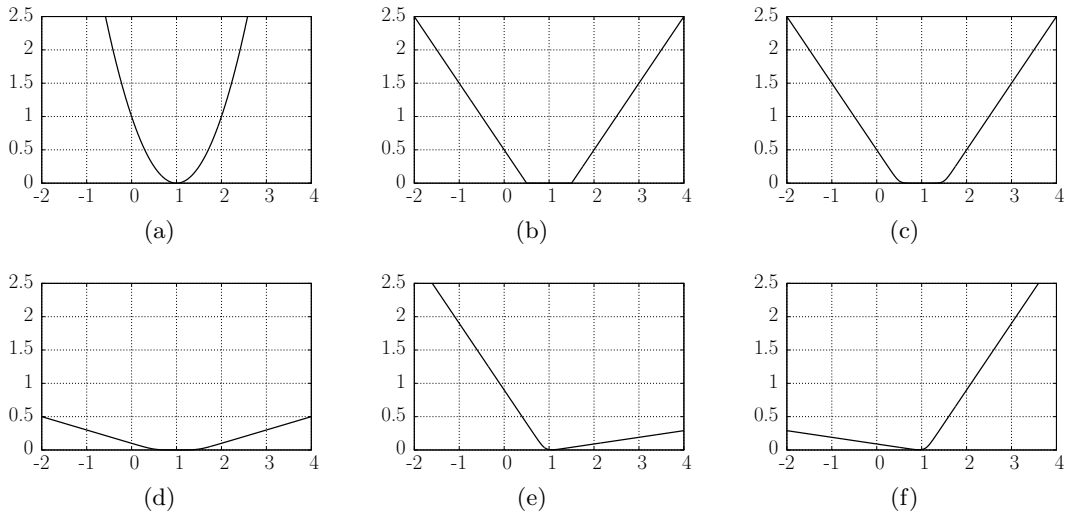


FIGURE 7.3. *The figures depict the square and the ε-insensitive loss as well as (non-symmetric) differentiable surrogates of the latter one defined via Equation (7.4). For all surrogate functions, the parameter η is fixed to* 20*. The appearance of these loss functions can be adapted via the parameters ζ_1, ζ_2, and ε. For instance, Figure (c) stems from $y = 1$, $\zeta_1 = 1.0$, $\zeta_2 = 1.0$, and $\varepsilon = 0.5$, whereas Figure (f) is induced by $y = 1$, $\zeta_1 = 0.1$, $\zeta_2 = 1$, and $\varepsilon = 0.1$.*

influence of broad emission peaks (typical of quasars), one can resort to Figure (f) as loss function since downward deviations are less penalized than upward deviations. Secondly, the loss function is differentiable, which renders the application of gradient based optimization schemes possible. As we will see, these two properties will lead to highly adaptable continuum models that can be obtained in an efficient kind of way.

Gradient Based Optimization

We will now show how to address the induced optimization task. By plugging the differentiable surrogate (7.4) into the objective (7.1) and by making use of the subset of regressors scheme, one obtains

$$\begin{aligned} \underset{\hat{\mathbf{c}}\in\mathbb{R}^r,\, b\in\mathbb{R}}{\text{minimize}}\; F(\hat{\mathbf{c}}) \quad = \quad & \frac{1}{l}\sum_{i=1}^{l}\frac{\zeta_1}{\eta}\log\left(1+\exp\left(-\eta\left(\hat{f}(\mathbf{x}_i')+b-y_i'+\varepsilon\right)\right)\right) \\ & +\frac{1}{l}\sum_{i=1}^{l}\frac{\zeta_2}{\eta}\log\left(1+\exp\left(+\eta\left(\hat{f}(\mathbf{x}_i')+b-y_i'-\varepsilon\right)\right)\right)+\lambda\hat{\mathbf{c}}^{\mathrm{T}}\hat{\mathbf{K}}\hat{\mathbf{c}} \end{aligned} \tag{7.5}$$

as optimization task with kernel matrix $\hat{\mathbf{K}} \in \mathbb{R}^{r\times r}$ and $\hat{f}$ defined via Equation (7.3). Similar to the standard support vector regression framework depicted in Chapter 3, one can easily show the convexity of this new task: The surrogate loss function $\mathcal{L}_{(\zeta_1,\zeta_2,\eta,\varepsilon)}(y,\cdot)$ is convex on $\mathbb{R}$ for fixed $y \in \mathbb{R}$. This yields the convexity of the first two terms since the

sum of convex functions is convex. The last term is convex due to the kernel matrix being positive definite [20]. It is easy to show that both the objective and the gradient can be computed efficiently for an intermediate solution $\hat{\mathbf{c}} \in \mathbb{R}^r$:

Lemma 7.1 *Given a spectrum represented as training set* $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathbb{R} \times \mathbb{R}$*, the above objective* $F(\hat{\mathbf{c}})$ *and gradient* $\nabla F(\hat{\mathbf{c}})$ *can be computed in* $\mathcal{O}(rl)$ *time for each intermediate solution* $\hat{\mathbf{c}} \in \mathbb{R}^r$*.*

Proof: The partial derivative with respect to c_{i_p} is given by

$$\begin{aligned} \frac{\partial F(\mathbf{c})}{\partial c_{i_p}} &= -\frac{1}{l}\sum_{i=1}^{l} \frac{\zeta_1 \exp\left(-\eta\left(\hat{f}(\mathbf{x}'_i) + b - y'_i + \varepsilon\right)\right) k(\mathbf{x}'_i, \mathbf{x}'_{i_p})}{1 + \exp\left(-\eta\left(\hat{f}(\mathbf{x}'_i) + b - y'_i + \varepsilon\right)\right)} \\ &\quad + \frac{1}{l}\sum_{i=1}^{l} \frac{\zeta_2 \exp\left(+\eta\left(\hat{f}(\mathbf{x}'_i) + b - y'_i - \varepsilon\right)\right) k(\mathbf{x}'_i, \mathbf{x}'_{i_p})}{1 + \exp\left(+\eta\left(\hat{f}(\mathbf{x}'_i) + b - y'_i - \varepsilon\right)\right)} + 2\lambda \sum_{j=1}^{r} c_{i_j} k(\mathbf{x}'_{i_j}, \mathbf{x}'_{i_p}). \end{aligned} \tag{7.6}$$

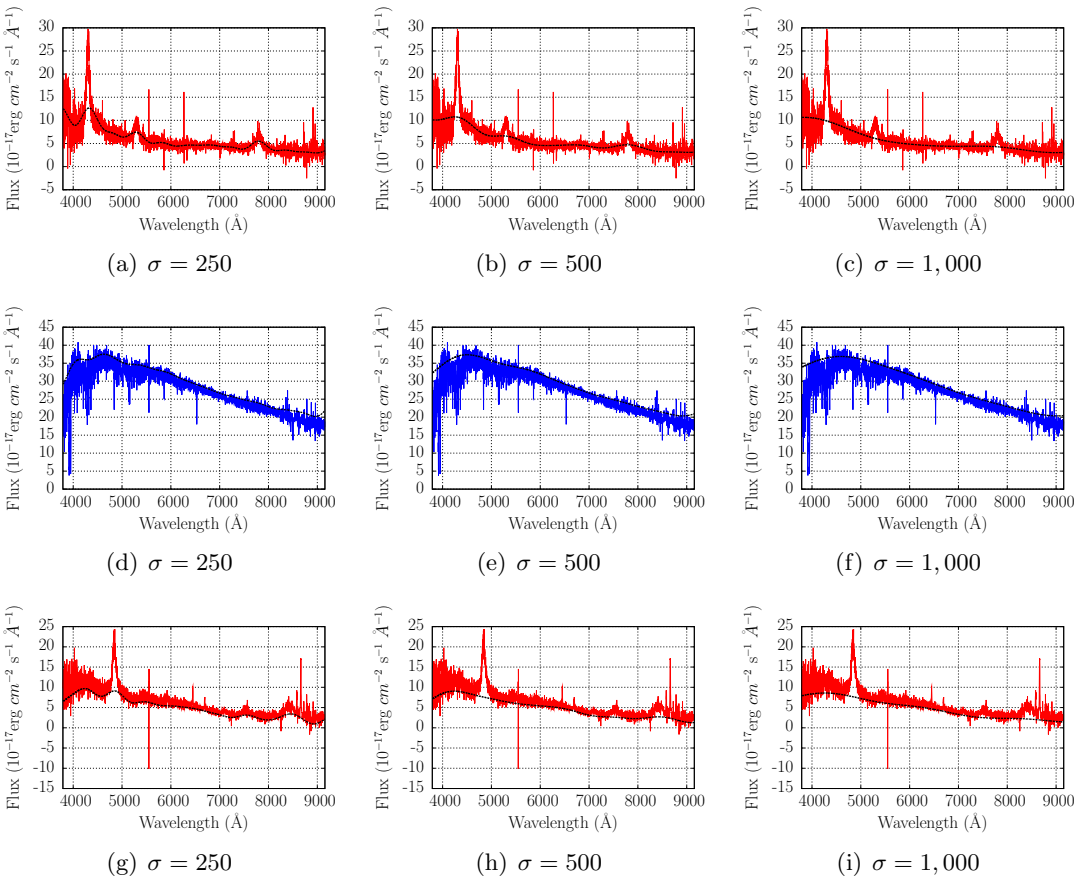
A similar term can be obtained for the partial derivative with respect to b. Since the term $\hat{\mathbf{c}}^{\mathrm{T}}\hat{\mathbf{K}}\hat{\mathbf{c}}$ and all predictions $\hat{f}(\mathbf{x}'_1), \ldots, \hat{f}(\mathbf{x}'_l)$ can be obtained in $\mathcal{O}((r+1)^2) = \mathcal{O}(r^2)$ and $\mathcal{O}((r+1)l) = \mathcal{O}(rl)$ time, respectively, each function and gradient call can be performed in $\mathcal{O}(rl)$ total time. □

Hence, the differentiable surrogate loss renders the application of gradient based optimization engines like Newton or quasi-Newton schemes [108] possible.[5] Note that numerical instabilities might again occur when performing a function or gradient call. However, exactly as in the the previous chapter, one can handle these issues in a safe manner since $\frac{\exp(t)}{1+\exp(t)} - 1 \to 0$ converges rapidly for $t \to \infty$ and, thus, permits a replacement of critical values by the above limit for, e. g., $t > 500$.

Adaptable Continuum Models

Besides rendering the application of gradient based methods possible, the above differentiable surrogate loss function can also be used to adapt to the specific tasks at hand: In Figure 7.4, the obtained continuum models for three different spectra are shown. Here, an RBF kernel with varying kernel width σ is used to generate all models. Further, for each row, different assignments for the parameters ζ_1 and ζ_2 are considered (the top row corresponds to the loss function shown in Figure 7.3 (c), whereas the middle and bottom rows

[5] For the sake of simplicity, we will resort to the latter class of techniques that only require the objective and its associated gradient to be provided. From a practical and theoretical point of view, the application of Newton's method might be beneficial in this context as well and will be subject of future work.



(a) $\sigma = 250$ (b) $\sigma = 500$ (c) $\sigma = 1,000$

(d) $\sigma = 250$ (e) $\sigma = 500$ (f) $\sigma = 1,000$

(g) $\sigma = 250$ (h) $\sigma = 500$ (i) $\sigma = 1,000$

FIGURE 7.4. *The figures depict the continuum fits obtained via the adaptable approach proposed in this chapter. For all figures, an RBF kernel with kernel width* $\sigma \in \{250, 500, 1000\}$ *is used and* λ *is set to* $\lambda = \frac{1}{2l}$. *Each spectrum is represented as labeled training set* $T_L = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\} \subset \mathbb{R} \times \mathbb{R}$ *with* $l = 3,825$ *and* r *is set to* 50. *For the top row, the loss function depicted in Figure 7.3 (c) is used; for the middle and bottom rows, the functions given in Figure 7.3 (e) and (f) are used, respectively. The kernel width determines the influence of neighbored wavelength patterns and is increased from left to right. The final continuum is given as a black dashed line.*

correspond to Figures 7.3 (e) and (f), respectively). Clearly, the models get less complex from left to right (i. e., for increasing kernel widths); a similar effect can be obtained by modifying the parameter λ. In addition, by using the particular loss function instances, one can determine whether the model should go through, above, or below the spectra. Note, for instance, that the continuum models in the last row of Figure 7.3 lie below the considered spectrum and are less influenced by the existent broad emission lines.

It is worth pointing out that even more flexible models can be obtained by considering different loss function instances for specific regions of wavelengths. For instance, one could enforce the continuum model to lie below the spectrum at hand for smaller wavelengths and above the spectrum for larger ones. In addition, other kernel functions than the

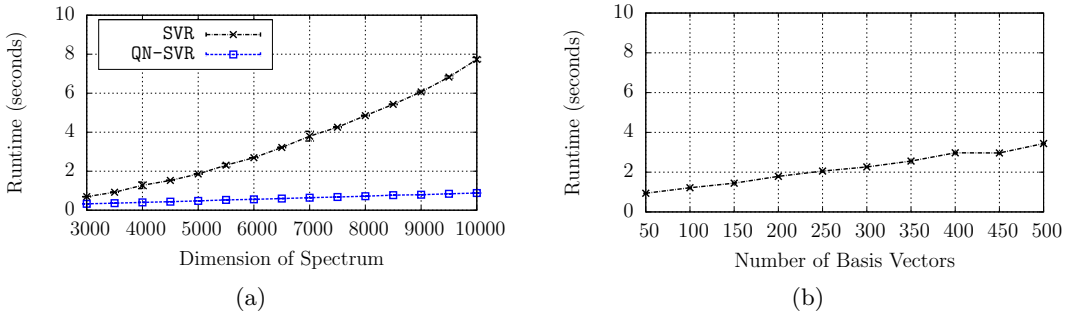


FIGURE 7.5. *The runtime of the adapted continuum model scheme (`QN-SVR`) is compared to the one of a standard support vector regression model (`LIBSVM`) in Figure (a). Further, the dependence of the practical runtime with respect to the number of considered basis vectors is analyzed in Figure (b) for an artificially created input spectrum of dimension* $10,000$.

RBF kernel could be employed. This yields extremely flexible continuum models that are well-suited for approaching a variety of tasks in the field of astronomy.

Computational Considerations

In addition to such flexible continuum models, future spectroscopic surveys will require methods that can handle high-dimensional spectra in an efficient kind of way. To analyze the runtime behavior of the proposed approach in this context, we consider artificially created spectra of increasing dimensions. In Figure 7.5 (a), a runtime comparison between a standard support vector regression implementation (`LIBSVM`) and the above quasi-Newton framework (`QN-SVR`) is given (for fixed parameter assignments and $r = 50$).[6] It can be clearly seen that the latter scheme exhibits a superior runtime performance.[7] In general, a small number r of basis vectors should be sufficient for most spectra (see above). In case more basis vectors are needed for representing appropriate continuum models, one can increase the amount of basis vectors while still being computationally efficient, see Figure 7.5 (b).

From High- to Low-Dimensional Features

Below, we will consider expert-based features that are defined on continuum-subtracted versions of the raw spectra. For this sake, we apply the `QN-SVR` approach depicted above with fixed model parameters (RBF kernel with $\sigma = 1000$ and $\zeta_1 = 0.5$, $\zeta_2 = 1.0$, $\varepsilon = 0.1$, $r = 50$, and $\lambda = \frac{1}{2l}$) and consider the differences between the raw spectra and the resulting

[6] As optimization engine, we resort to the L-BFGS implementation `optimize.fmin_l_bfgs_b` [26] provided by the `Scipy` package with `m` $= 50$, `factr` $= 10^5$, and `maxfun` $= 1,000$. The runtime experiments were conducted on a 2.8 GHz Intel Quad Core™PC running `Ubuntu 11.10`.

[7] Further, the needed function calls for `QN-SVR` were less than 100 for all runs.

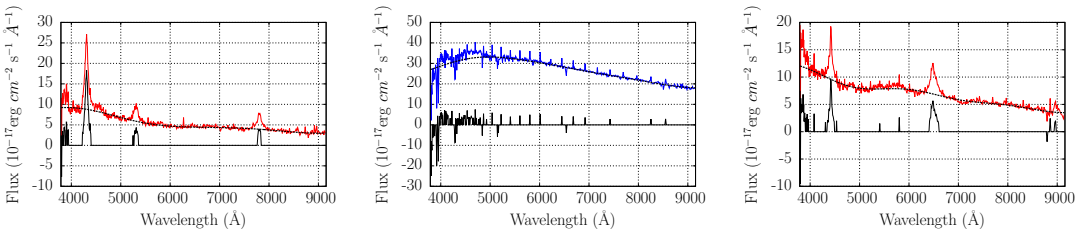


FIGURE 7.6. *Continuum-subtracted versions of raw spectra that form the basis for the extraction of expert-based features. As input for the extraction chain, binned versions of the spectra of dimension $d = 500$ are considered, i. e., consecutive flux values are merged such that the desired dimension is obtained [62].*

continuum models. Further, we set those differences to zero that are smaller than a specific threshold δ (set to the one standard deviation of all differences), and consider binned versions of the raw spectra as input for the extraction phase that stem from averaging consecutive flux values (resulting in spectra of dimension $d = 500$). The output of this preprocessing chain is depicted in Figure 7.6. It can be clearly seen that the characteristic features of quasars (i. e., the broad emission lines) are emphasized. The resulting continuum models and the associated continuum-subtracted spectra can be used to define a small set of ten physically motivated features (the first/last values of the continuum model, the sum of all positive/negative peaks, the width of the broadest negative/positive peaks, the major/minor peak intensity, and the major face of positive/negative peaks), see Gieseke *et al.* [62] for details. As we will see below, such features are well-suited for the task of discriminating quasars from other objects.

7.2.2 Discriminating Quasars from Other Objects

We will now discuss the benefits of the above extraction scheme in the context of identifying quasars in large-scale spectroscopic surveys. It should be pointed out, however, that the generation of adaptable continuum models is of independent interest in astronomy and other application fields.

Experimental Setup

The data set used for the experimental evaluation is based on the sixth data release of the SDSS and contains $N = 1,024$ patterns with 512 objects of type `quasar` and 512 objects of type `other`; the labels for these objects were obtained by an expert in this field [62]. As classification model, we consider a standard support vector machine implementation (`LIBSVM`) [29] and make use of both a linear and an RBF kernel. To train and evaluate the classifier, half of the data set is used as training and the other half as test set. Model

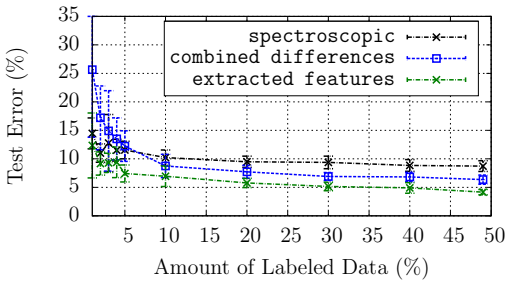


FIGURE 7.7. *The classification performances of a support vector machine based on three different feature sets for a varying amount of labeled input patterns is shown. It can be observed that the extraction of expert-based features leads to the best performance. Further, the feature set based on the continuum-subtracted versions of the spectra exhibits a better performance compared to a direct use of the raw spectra.*

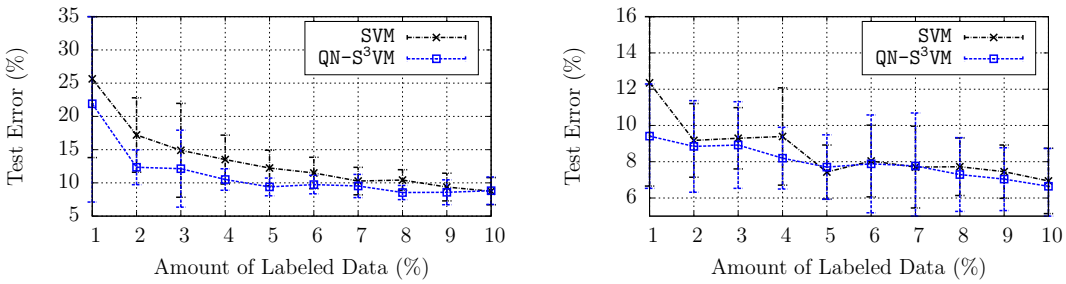
selection is done via 5-fold cross validation performed on the training set, and the final performance is measured via the test error. To tune the non-fixed parameters, we consider $\sigma \in \{2^{-5}\hat{\sigma}, \ldots, 2^{5}\hat{\sigma}\}$ (see page 113 for the definition of $\hat{\sigma}$) and $C \in \{2^{-10}, \ldots, 2^{10}\}$ as possible parameter assignments. Further, we resort to three different feature sets: The first one is based on the binned versions of the raw spectra (`spectroscopic`) with dimension $d = 500$. The second one is based on the continuum models and the continuum-subtracted spectra (`combined differences`). Here, each continuum model and its associated continuum-subtracted spectrum are concatenated ($d = 1000$), and the influence of a continuum value is reduced by a factor of ten. The third set is based on the expert-based features (`extracted features`) described above with $d = 10$. A linear kernel is used for the first two data sets, whereas an RBF kernel is employed for the last one.

Classification Performance

The classification performance of the support vector machine for the three different feature sets is given in Figure 7.7 for a varying amount of labeled patterns (the remaining patterns in the training set are ignored for each setting). It can be seen that the best performance is obtained using the low-dimensional expert-based feature space as input (`extracted features`). Further, the feature set based on the continuum-subtracted versions of the raw spectra (`combined differences`) lead to a superior performance compared to the raw spectra (`spectroscopic`). Hence, for this particular task, emphasizing the characteristic features of quasars via the proposed extraction scheme depicts a simple but effective way to obtain valuable classification models.

7.3 Semi-Supervised Learning Perspectives

Note that the feature extraction framework given above depicts an alternative way to cope with the curse of dimensionality, i. e., to reduce the amount of needed labeled data



(a) `combined differences` (b) `extracted features`

FIGURE 7.8. *The outcome of a semi-supervised support vector machine compared to a support vector machine base. The results depicted in Figure (a) show that significant improvements can be obtained by incorporating the unlabeled data. For the extracted features, only minor improvements can be observed.*

for achieving a satisfying performance. However, such features might not be available for specific learning tasks. Further, the labeling of such objects is time-consuming and might only be given for a small set of labeled objects. In this section, we will depict the use of semi-supervised support vector machines for the above task. In addition, we will sketch multi-view learning settings that might play an important role as well for the automatic analysis of astronomical data sets in the near future.

7.3.1 Semi-Supervised Support Vector Machines for Spectroscopic Data

We will briefly sketch the possible application of semi-supervised support vector machines in the above context.

Experimental Setup

We resort to the `QN-S3VM` implementation derived in the previous chapter (using the experimental setup described in Section 6.4). Again, we increase the amount of labeled patterns (with respect to the size of the training set) and consider the remaining patterns in the training set as unlabeled patterns. The involved model parameters are tuned via 5-fold cross validation on the labeled part of the training set (i. e., the realistic scenario is considered). As supervised competitor, we consider the `LIBSVM` implementation using the experimental setup depicted above.

Classification Performance

The outcome of this experiment is shown in Figure 7.8 for the two feature sets `combined differences` and `extracted features`. For the first one, a linear kernel is used, whereas an RBF kernel is employed for the second one. It can be clearly seen that the semi-supervised scheme outperforms the supervised baseline for the `combined differences`

feature set; for instance, considering only five percent of the patters in the training set as labeled data, an improvement of about three percent with respect to the test error can be observed. Further, slight improvements can also be observed for the second feature set. Hence, these results indicate the potential of semi-supervised support vector machines in this context. Note that a lot of the values in the continuum-subtracted versions of the raw spectra are zero, i. e., such continuum-subtracted spectra exhibit desired sparseness properties. An interesting future research direction is the question whether appropriate machine learning techniques can be applied to take advantage of this data situation.

7.3.2 Multiple Views: Photometric and Spectroscopic Data

Several other semi-supervised concepts can be found in the machine learning field. One of these concepts is *multi-view learning* [22, 23, 122, 129, 130]. Frameworks belonging to this class of learning schemes split up the features into independent sets, called *views*, and the model is built based on these views. A typical example of such views are web pages, where each document can be represented by (a) the words occurring on the page and by (b) the images shown on it. In the context of multi-view learning, the goal of the learning task consists in finding, for each view, a prediction function that performs well on the labeled data of the designated view, and, at the same time, agrees on the unlabeled data with the prediction functions learned on the remaining views. Such multi-view schemes do not only exist for classification scenarios, but also for a variety of other learning tasks like clustering or regression. In the context of astronomy, so-called *ranking* tasks could play an important role in future. In the remainder of this section, we will briefly describe both concepts and will sketch their potential in the field of astronomy.

Co-Regularization Frameworks

The concept of *co-regularization* depicts one possible formalization of the multi-view learning concept [130]. Briefly stated, algorithms based upon such frameworks search for hypotheses $f_1 \in \mathcal{H}_1, \ldots, f_M \in \mathcal{H}_M$ from different reproducing kernel Hilbert spaces $\mathcal{H}_1, \ldots, \mathcal{H}_M$ such that the error of each hypothesis on the labeled part of the data is small and that, at the same time, all hypotheses give similar predictions for the unlabeled data patterns. Within such frameworks, the hypothesis spaces $\mathcal{H}_1, \ldots, \mathcal{H}_M$ can stem from different features (like, e. g., photometric and spectroscopic data) and/or different kernel functions and the disagreement between the prediction functions is taken into account via a so-called *co-regularization term* (see below). In the literature, both empirical [22, 23, 130] and theoretical [122, 129] results indicate the potential of co-regularization frameworks.

Ranking

The goal of *ranking* consists in deriving a model that can order objects. There are several definitions of such ranking tasks. One of them is *object ranking* [56], where one aims at predicting the rank for any unseen subset $O \subseteq C$ of a class of possible objects C. Another popular one is *label ranking*, where the goal consists in predicting, for every instance $\mathbf{x} \in \mathcal{X}$ of the input space $\mathcal{X}$, a preference relation $\mathcal{P}_\mathbf{x} \subseteq \mathcal{L} \times \mathcal{L}$ among a finite set of possible labels/alternatives. Here, a pair $(\rho, \rho') \in \mathcal{P}_\mathbf{x}$ means that the instance $\mathbf{x}$ prefers label ρ to ρ' [56]. A typical label ranking example is, for instance, the task to predict, for each search engine query, a ranking of available web pages.

Outlook: Multi-View Ranking in Astronomy

For general ranking tasks, a variety of supervised learning schemes exist, see, e. g., Herbrich *et al.* [74] and Joachims [85]. A recent semi-supervised approach that combines both the concept of ranking and the one of multi-view learning is given by Tsivtsivadze *et al.* [139].

Co-Regularized Least-Squares Ranking in Short. We will briefly sketch the main idea of this learning approach and refer the reader to Tsivtsivadze *et al.* [139] for a detailed description. As for every semi-supervised learning framework, one is given a labeled training set T_L and an unlabeled one T_U. Further, one is given M different reproducing kernel Hilbert spaces $\mathcal{H}_1, \ldots, \mathcal{H}_M$ and associated kernel functions $k_1, \ldots, k_M$. The goal of the learning process is to find a tuple $\mathbf{f} = (f_1, \ldots, f_M) \in \mathcal{H}_1 \times \ldots \times \mathcal{H}_M$ of prediction functions that minimizes an objective of the form:

$$J(\mathbf{f}) = \sum_{v=1}^{M} c(f_v, T_L) + \beta \sum_{v,u=1}^{M} V(f_v, f_u, T_U) + \lambda \sum_{v=1}^{M} \|f_v\|_{\mathcal{H}_v}^2 \tag{7.7}$$

Here, the function c measures the loss $c(f_v, T_L)$ for each prediction function f_v on the labeled part of the data, and the second term measures the mutual disagreement of all prediction functions on the unlabeled part. The third term penalizes complex prediction functions. Hence, the second summand can be seen as an additional *co-regularization term* that enforces the prediction functions (trained on the different views) to yield similar predictions for unlabeled patterns. Again, the parameters $\lambda \in \mathbb{R}^+$ and $\beta \in \mathbb{R}^+$ can be used to determine the trade-off between all objectives. Both the cost function c for the labeled patterns and the cost function V for the unlabeled ones take the specific properties of the ranking task into account. In contrast to the optimization tasks induced by semi-supervised support vector machines, however, the above learning setting leads to a convex

optimization task whose solution can be obtained analytically by solving a system of linear equations. We omit the lengthy mathematical derivations here and refer the reader to Tsivtsivadze *et al.* [139] for a comprehensive description of this learning concept.

Applications in Astronomy. An interesting future research direction is the question if such sophisticated semi-supervised ranking methods can outperform standard supervised (regression) techniques for specific learning tasks in astronomy. Note that for objects in the Sloan Digital Sky Survey that have already been target of a spectroscopic follow-up observation, one is also given photometric data. Hence, for those objects, one has access to two different views as well. While the information gain might be limited for point-shaped objects like stars or quasars (since one already has the detailed information provided by the associated spectra), photometric data provides valuable additional information for extended objects like galaxies. One possible task could consist, for instance, in ranking all elliptical, spiral, or lenticular galaxies (according to, e. g., the Hubble sequence [78]) while taking both photometric and spectroscopic data into account for generating the anticipated model.

7.4 Concluding Remarks

In this chapter, we have derived an adaptable continuum fitting method that can be applied to spectroscopic data in astronomy and other application fields. The proposed models depict variants of standard support vector regression schemes and can be efficiently obtained by making use of the special properties of the input space in this context. In addition to being computationally extremely efficient (even for high-dimensional spectra), the models can be adapted to the specific tasks at hand. While being of independent interest, we showed the use of this continuum extraction scheme in the context of classifying quasars in spectroscopic surveys. More precisely, we made use of continuum-subtracted versions of the raw spectra to define a small set of physically motivated features. These features led to a significantly better classification performance compared to a straightforward use of the raw data. In addition, we described the use of semi-supervised support vector machines for this particular task. Finally, we sketched the concepts of multi-view learning and ranking, which depict promising learning techniques in the field of astronomy as well.

CHAPTER 8

Machine Learning in Space

The previous chapter dealt with the automatic analysis of astronomical data sets on earth. For such settings, the lack of labeled data and the sheer data volumes render the direct application of machine learning methods challenging. In this chapter, we will focus on the data analysis in space, i. e., on the analysis taking place on today's spacecraft systems. While similar problems have to be tackled for such scenarios as for the ones on earth, one is additionally faced with computational problems that are related to (a) the limited computational resources of such spacecraft systems and to (b) the cosmic radiation present in space. As we will see, both issues can lead to undesired memory corruptions (*bit flips*) that can have a severe influence on the performed computations. To shorten such negative influences, software- and hardware-based countermeasures are usually employed. In this chapter, we will sketch an alternative way to deal with such situations, which is based on so-called *resilient algorithms*. More specifically, we will propose a modification of the classical k-means clustering scheme (that is based on a resilient version of the well-known k-d tree data structure) and will demonstrate its use in the context of clustering hyperspectral image data.

Outline. We will start by describing the computational problems arising on spacecraft systems in Section 8.1. The classical k-means approach along with a (non-resilient) k-d tree-based variant will be depicted in Section 8.2. Afterwards, in Section 8.3, we will describe the modified k-d tree data structure as well as the corresponding adaption of the classical k-d k-means approach. The experimental analysis of the resulting scheme will be subject of Section 8.4, followed by concluding remarks given in Section 8.5.

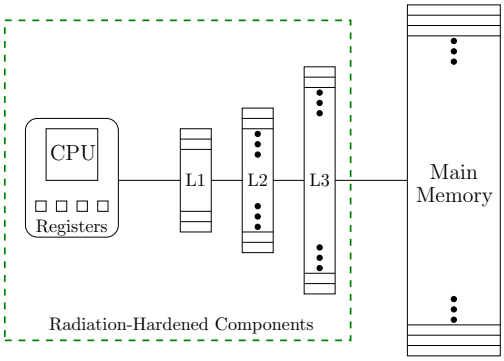


FIGURE 8.1. *Today's spacecraft systems operate in high-radiation environments with possible memory corruptions caused by cosmic rays or alpha particles [103, 144]. For the layers of the memory hierarchy close to the CPU (e. g., registers, L1-cache, L2-cache, L3-cache, . . .), the disturbing influence of memory corruptions is usually addressed by using radiation-hardened components (or by software-based countermeasures). However, due to the increase of mass caused by such components, the remaining layers (e. g., the main memory) can usually not be protected in this manner.*

8.1 Motivation

While the massive amount of data on earth necessitates the use of machine learning techniques to automatically retrieve useful information, the data analysis aboard of spacecrafts has become an important issue in recent years too. For instance, the spacecraft associated with the *Earth Observing-1* (EO-1) mission already makes use of classification schemes to automatically analyze hyperspectral image data such that follow-up observations of interesting regions or events can be made (without any human interaction) [28, 37]. As pointed out by Wagstaff and Bornstein [144, 145], on board autonomy will be an important issue for future missions in general and in particular for those involving large one-way communication times like missions to Jupiter or Saturn. Hence, machine learning techniques have the potential to dramatically reduce idle times by prioritizing data and by creating bandwidth-saving compressed previews for transmission to earth.

Spacecraft systems differ significantly from standard computers in terms of computational power, available memory, electricity consumption, and operating systems. Further, these systems operate in high-radiation environments with interference caused by cosmic rays and alpha particles [103, 144].[1] These disturbing influences can cause memory corruptions (e. g., *bit flips*) in the systems' memory hierarchies which, in turn, can severely affect the computations aboard such systems. For modern spacecraft systems, the influence of memory corruptions in the CPU (e. g., registers) and memory layers close to the CPU (e. g., caches) is usually addressed by using fully radiation-hardened hardware

[1] The influence can even be observed on earth (sea-level) but is already two orders of magnitude larger at 10,000 meters [90].

components [144]. For the remaining parts of the memory hierarchy (e. g., main memory), software- or hardware-based concepts (or combinations thereof) can be employed to reduce the negative effect of the undesired memory corruptions. However, both schemes lead to higher costs, an increase of mass and a reduction in capability and speed in most cases [144], see Figure 8.1. These are undesirable side-effects, especially for spacecraft systems due to the enormous transportation costs. An alternative approach to standard software-based countermeasures are *resilient algorithms* [24, 49, 50, 87, 114] that guarantee reasonable outputs in the presence of memory corruptions. Further, they usually exhibit only a small overhead in space and runtime. In the following, we will describe a resilient version of the k-d tree data structure [11] and will demonstrate how to use it to improve the resiliency of the k-means clustering approach.

8.2 Accelerating K-Means

One of the bottlenecks of a naive k-means implementation is the recurrent computation of nearest neighbors. A variant that aims at accelerating these computations is given by Kanungo *et al.* [88]. Their algorithm first builds a k-d tree [11] for the input data and uses this data structure to speed up the involved nearest neighbor computations. We briefly review their approach since it forms the basis for our resilient clustering scheme.

8.2.1 K-d Trees

The classical k-d tree data structure [11] can be used to partition a given set of patterns $T = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$ in the d-dimensional Euclidean space. An internal node v of such a tree corresponds to (and stores) the d-dimensional box $v.box$, which contains all patterns stored in the subtree rooted at v. Further, each leaf corresponds to (and stores) a single pattern $v.p$. The construction of a k-d tree is usually performed in a level-wise manner from top to bottom ($i = 0, 1, \ldots$), i. e., for each node v on the i-th level, one considers the median of the patterns' coordinates in dimension ($i \bmod d$) to partition the set of patterns associated with v into two (almost) equal-sized subsets. These subsets (along with the corresponding boxes induced by the splitting hyperplane) are then assigned to the children of the node v. Since the median of a set of n numbers can be found in linear time, the construction of such a tree can easily be performed in a recursive manner taking $\mathcal{O}(n \log n)$ overall time [11]. The space consumption is $\mathcal{O}(n)$.

8.2.2 Speed-Up with K-d Trees

Each iteration of the standard k-means approach involves finding the nearest candidate center for each input pattern, see again Algorithm 2.1. The key idea of the variant of Ka-

nungo *et al.* [88] depicted below is to speed up these computations by using an adapted version of the k-d tree data structure. In a nutshell, for each iteration of the k-means scheme, the possible patterns that might be relevant for a (new) cluster center are computed efficiently based on the hierarchical subdivision provided by the tree structure.

Adapted Tree Structure

For the sake of the efficient implementation of k-means, additional information is stored in the k-d tree data structure. More precisely, for each node v of the given tree, one additionally stores the number $v.c$ and the weighted centroid $v.wcent$ (defined as vector sum) of all patterns that are stored in the subtree rooted at v. Hence, one can retrieve, for each node v, the associated centroid of patterns stored in the subtree rooted at v via $v.wcent/v.c$. The filtering approach depicted below involves adding cluster centers and the centroids stored in the nodes of the k-d *tree*. For this sake, each of the k cluster centers z will also be represented via $z.c$ and $z.wcent$, i. e., via the number and the corresponding weighted centroid of patterns associated with the cluster center z. This allows efficient updates of the intermediate cluster centers, as we will describe next.

Filtering Cluster Centers

In each iteration of the classical k-means scheme, the patterns are reassigned to their nearest cluster centers and the positions of the resulting modified cluster centers are updated. This recurrent computation of nearest neighbors and the necessary update of the cluster centers can be very time-consuming. The acceleration proposed by Kanungo *et al.* [88] invokes the procedure given in Algorithm 8.1 on the root of the associated k-d tree as a replacement for these two steps: The algorithm recursively traverses the tree and maintains, for each visited node, a set of possible candidate centers (starting with the root of the k-d tree and all k cluster candidate centers). At each node v, all candidate centers that are strictly farther away from the box associated with v than the candidate center closest to the center of $v.box$ are excluded from further consideration (since no point stored in the subtree rooted at v has to be assigned to them). This filtering step is performed in Steps 7–12. If only one candidate center is left after this filtering phase, all points stored in the subtree will belong to the remaining center, which can then be updated appropriately (Steps 14 and 15). Otherwise, the algorithm recurses (Steps 17 and 18). If the recursion reaches at a leaf w, the point $w.p$ is checked against all remaining candidate centers, and the center closest to $w.p$ is updated accordingly (Steps 3–5).

Hence, the key idea of this filtering scheme is to distribute the points stored in the k-d tree to the k cluster centers in an efficient kind of way. The approach is based on the assumption that Step 13 of Algorithm 8.1 is often fulfilled (which is usually the case for

Input: Node v of the associated k-d tree and the set $Z = \{z_1, \ldots, z_k\}$ of cluster candidate centers.

```
 1: procedure FILTER(v, Z)
 2:     if v is leaf then
 3:         z* = candidate center in Z closest to v.p
 4:         z*.wcent = z*.wcent + v.p
 5:         z*.c = z*.c + 1
 6:     else
 7:         z* = candidate center in Z closest to the center of v.box
 8:         for each z ∈ Z do
 9:             if z* strictly closer to the boundary of v.box than z then
10:                 Z ← Z \ {z}
11:             end if
12:         end for
13:         if |Z| = 1 then
14:             z*.wcent = z*.wcent + v.wcent
15:             z*.c = z*.c + v.c
16:         else
17:             FILTER(v.left,  Z)
18:             FILTER(v.right, Z)
19:         end if
20:     end if
21: end procedure
```

ALGORITHM 8.1. *The filtering scheme proposed by Kanungo* et al. *[88] used to speed up the classical k-means clustering scheme. Initially, the procedure is invoked on the root of the associated k-d tree.*

low-dimensional data). Since updating the corresponding cluster center with the needed information can be performed efficiently using the k-d tree structure (Steps 14–15), these steps usually yield a significant speed-up compared to a naive k-means implementation.

8.3 Resilient K-d K-Means

We will now propose modifications of the above k-d tree based clustering scheme that render it more resilient against memory corruptions. The basis for these modifications is a resilient version of the classical k-d tree, which we will describe next.

8.3.1 Resilient K-d Tree

We start by describing an appropriate model of computation that is usually used to analyze resilient algorithms. Afterwards, we will describe the resilient k-d tree and will analyze both its space consumption and the time needed to construct it.

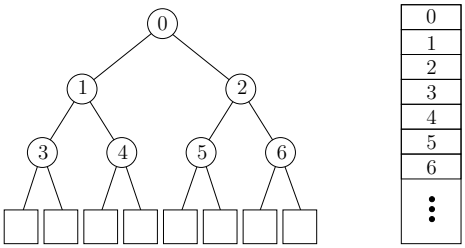


FIGURE 8.2. *The resilient k-d tree (left) and the memory layout used to store its top tree (right). Each node v of the top tree reliably stores the associated d-dimensional box $v.box$, the weighted centroid $v.wcent$, and the number $v.c$ of points stored in the subtree rooted at v (in unsafe memory). The top tree can be embedded in a breadth-first-search order in an array, where the children of a node at index i are stored at the indices $2i+1$ and $2i+2$. Each leaf structure (depicted as square) can accommodate up to $B\delta$ input patterns with an integer label each, which are stored unreliably in unsafe memory.*

Model of Computation

The *faulty-memory RAM model* [49] can be employed to analyze the theoretical properties of resilient algorithms. In addition to the assumptions made for the classical *random access machine* (RAM) model [116], memory cells can get corrupted at any place and at any time during the execution of an algorithm. Further, corrupted and uncorrupted cells cannot be distinguished. The only exception are $\mathcal{O}(1)$ *safe* memory cells (representing, e. g., the CPU registers in real-world architectures), which are not affected by these corruptions. Thus, these cells can be used to store, e. g., loop indices and program counters.

In addition to these safe memory cells, one is given an upper bound $\delta \in \mathbb{N}$ on the number of memory corruptions that may occur during the overall execution of an algorithm. Due to this bound, one can store a data item safely in unsafe memory by replicating it $2\delta + 1$ times in consecutive memory cells. Such a value is called a *reliable value*. Since there are at most δ corruptions taking place during the execution of an algorithm, the majority of all copies are uncorrupted and the original value can be retrieved in linear time using the *majority vote algorithm* that scans all consecutive values while maintaining a candidate and a confidence value, see Boyer [21] for details. Note that a trivial way of ensuring resiliency is to apply this replication to *all* data items. This, however, increases both space and time by a factor of $\Theta(\delta)$. One of the goals of resilient algorithms is to guarantee a certain outcome while storing as little as possible values in a reliable manner.

Note that, due to only $\mathcal{O}(1)$ safe memory cells, specific techniques cannot be easily applied in the context of the fault-memory RAM model. For instance, a direct application of recursion schemes is largely prohibited due recursion stacks of (usually) non-constant sizes. Further, using pointers can lead to a loss of data (unless they are stored reliably).

Structure

The resilient k-d tree proposed below consists of a *top tree* and a number of *leaf structures* and stores the auxiliary information needed for the filtering scheme depicted above (used to accelerate the classical k-means approach): The top tree is a complete binary tree of height $h = \lceil \log(\frac{n}{B\delta}) \rceil$ corresponding to the top part of a standard k-d tree, where $B \in \mathbb{N}$ is a user-defined constant. Further, each data item is stored reliably in unsafe memory, i. e., each internal node v reliably stores (1) the associated d-dimensional box $v.box$, (2) the weighted centroid $v.wcent$, and (3) the number $v.c$ of points stored in the subtree rooted at v. Note that, since the top tree is complete, one does not have to resort to pointers but can store the tree in a breadth-first-search order in an array, i. e., the children of a given node v at index i are stored at the indices $2i + 1$ and $2i + 2$, see Figure 8.2.

Each leaf of the top tree corresponds to at most $B\delta$ points and is associated with exactly one leaf structure. All leaf structures are stored in a common array, and for each leaf structure space is allocated to accommodate up to $B\delta$ patterns along with a cluster label per pattern (used to store the final assignment). Note that one does not need additional pointers to link the leaves of the top tree with their corresponding leaf structures due to the breadth-first-search layout used for storing the top tree (and the fact that the top tree is complete). The following lemma summarizes the space consumption of the resulting data structure.

Lemma 8.1 *Given $B \in \mathbb{N}$, the resilient k-d tree data structure for n patterns in $\mathbb{R}^d$ stores at most $2n + 12n(2\delta + 1)/(B\delta)$ d-dimensional points and $2n + 4n(2\delta + 1)/(B\delta)$ integers in the faulty-memory RAM model of computation with parameter $\delta \in \mathbb{N}$.*

Proof: The binary top tree has height $h = \lceil \log(\frac{n}{B\delta}) \rceil$. At each stage of the recursive construction (of a standard k-d tree) the current point set is split into two sets of (almost) the same size. In the best case (with respect to the space consumption) each leaf of the top tree corresponds to exactly $B\delta$ points, and one is given (exactly) $n/(B\delta)$ leaves and, thus, $n/(B\delta) + (n/(B\delta) - 1) < 2n/(B\delta)$ nodes in the binary top tree (in total). For each node v in the top tree one stores the bounding box (2 points), the weighted centroid (1 point), and the number of points stored in the subtree of v (1 integer). Since one reliably stores this information for each node, the top tree structure occupies $3(2\delta+1) \cdot 2n/(B\delta) = 6n(2\delta + 1)/(B\delta)$ points and $(2\delta + 1) \cdot 2n/(B\delta) = 2n(2\delta + 1)/(B\delta)$ integers. For each leaf structure space is allocated to store $B\delta$ points with an integer label each, which results in n points and n integers to be stored for all $n/(B\delta)$ leaf structures (in the best case).

Hence, for both the leaf structures and the top tree, one needs at most $n + 6n(2\delta + 1)/(B\delta)$ points and $n + 2n(2\delta + 1)/(B\delta)$ integers in the best case. In general, the leaf

structures might not be fully occupied. In the worst case, exactly one of the leaf structures has to store $1 + B\delta/2$ points, whereas all other ones have to store exactly $B\delta/2$ points. Due to the layout of the top tree twice as many leaves (and leaf structures) are needed in this case, which means that twice as much space as in the best case is sufficient for the worst case. □

The size of the top tree (for fixed point set and model parameter δ) is determined by the particular assignment for the parameter $B \in \mathbb{N}$. Since all data items are stored in a resilient manner for the top tree, memory corruptions will not have any influence on the top tree. Thus, the parameter B determines a trade-off between a large resilient top tree and the space requirements needed to accommodate all data items.

Construction

The construction phase of the resilient k-d tree is similar to the one of the classical k-d tree: Starting with the d-dimensional bounding box (stored at the root of the tree) containing all the input points, the box is split into two orthogonal to its longest side (and with respect to the corresponding median). The resulting two boxes and the necessary auxiliary information are stored in the two children of the root. These splitting operations are performed until the maximum level $h = \lceil \log(\frac{n}{B\delta}) \rceil$ of the top tree is reached. The remaining points are then stored in the corresponding leaf structures, where each leaf structure can accommodate up to $B\delta$ points and associated labels. So far, no resilient median selection algorithm has been proposed in the literature. Thus, one has to resort to resilient sorting for computing the medians [50]. This leads to the following lemma.

Lemma 8.2 *Given $B \in \mathbb{N}$, the resilient k-d tree data structure for n patterns in $\mathbb{R}^d$ can be constructed in $\mathcal{O}(n \log^2 n + \delta^2)$ time in the faulty-memory RAM model of computation with parameter $\delta \in \mathbb{N}$.*

Proof: Sorting n data items resiliently can be performed in $\mathcal{O}(n \log n + \alpha\delta)$ worst-case time and linear space, where α is the number of actual memory corruptions that happen during the execution of the algorithm [50]. Without loss of generality, let us assume that n is a power of two. In this case, the recursive construction spends

$$T(n) \quad \in \quad \mathcal{O}\left(n \log n + \alpha_0\delta + \frac{n}{2}\log(\frac{n}{2}) + \frac{n}{2}\log(\frac{n}{2}) + \alpha_1\delta + \ldots\right) \tag{8.1}$$

time, where α_i denotes the number of actual memory corruptions happened during all sorting operations on level i. Since the amount of actual memory corruptions over *all* levels of the tree add up to at most δ, the overall runtime is bounded by $\mathcal{O}(n \log^2 n + \delta^2)$.□

8.3.2 Resilient K-d K-Means

The non-resilient k-d tree-based variant can yield significant practical speed-ups for low-dimensional data compared to a naive implementation of the k-means clustering scheme. However, as reported by Wagstaff and Bornstein [144], it is vulnerable against memory corruptions. We will now discuss the modifications needed to adapt this filtering scheme so that it benefits from the resilient k-d tree variant depicted above.

Safe Flow Control

Since only a constant amount of safe memory cells is available in the faulty-memory RAM model, recursion stacks are prone to corruptions which can result in segmentation faults and other memory-related errors. The recursion stack induced by the FILTER procedure depicted in Algorithm 8.1 has depth $\mathcal{O}(\log(\frac{n}{B\delta}))$ and can, thus, not be stored in safe memory. Due to the breadth-first-search layout used for storing the top tree, however, one can navigate in a pointerless manner (the children of node i are stored at the indices $2i+1$ and $2i+2$). Further, due to this layout, one can deduct from these indices whether a child returned from is a left or a right child when mimicking the recursive calls.[2] Thus, navigating in the induced recursion stack can be performed in a safe manner.

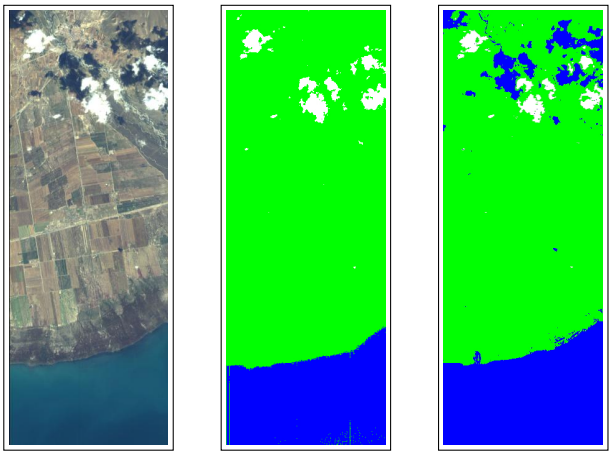
Protecting the Candidate Centers

The corruption of a single candidate center can severely affect the overall outcome of the adapted clustering scheme. To avoid this we store this small (but non-constant) amount of data reliably. Further, while traversing the resilient k-d tree up and down, one needs to reliably reconstruct the set of candidate centers that are already filtered out. For this sake we store a bit vector I_v of length k (bits) for each node v on the recursion path, which indicates whether the i-th candidate center is (still) relevant for v or not. Thus, we explicitly (and reliably) maintain a recursion stack of these bit vectors, which takes $\lceil \log k \rceil (2\delta + 1)$ integers per level of the top tree of the resilient k-d tree structure

Consistency Check for the Filtering Phase

To filter the cluster candidate centers we distinguish between internal nodes and leaves of the top tree. For an internal node we simply resort to Algorithm 8.1. Since each data item of the top tree is stored reliably, memory corruptions will not affect the recursive calls (which are handled as described above). For a leaf of the top tree we make use of the associated leaf structure to update the remaining candidate centers. To avoid

[2] For a given even index j the parental node has index $i = (j-2)/2$; for an uneven one the parental index is given by $i = (j-1)/2$.



(a) RGB (b) LABELS (c) K-MEANS

FIGURE 8.3. *The images depict the* **Qinghai Large** *data set along with a clustering result [144]: (a) RGB image based on the* 11 *bands, (b) manually obtained labels (clouds, ground, water), and (c) standard k-means clustering result (ARI*=0.749*).*

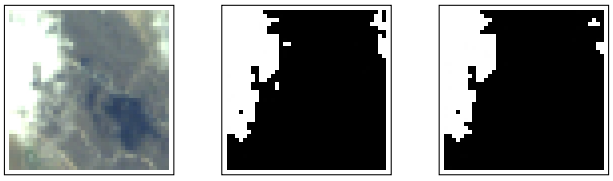
severe corruptions caused by updating candidate centers with corrupted points stored in the leaf structures we make use of a simple consistency check performed on each point before assigning it to one of the centers: Due to the hierarchical structure of the resilient k-d tree, we know that every point in a particular leaf structure must be contained in the box associated with this leaf structure. Since this information is stored reliably in the top tree, one knows that if a point is out of range, it must be corrupted. In this case, one can therefore resort to the center of the corresponding box as surrogate for the point itself.

8.4 Experimental Analysis

For the experimental analysis we consider the standard k-means algorithm (`k-means`), its variant based on k-d trees (`k-d k-means`), and our resilient k-d tree-based approach (`resilient k-d k-means`) as clustering schemes. Further, following Wagstaff and Bornstein [144], we use several real-world data sets to investigate the behavior of all approaches given certain setups for the involved parameters.

8.4.1 Experimental Setup

The results were obtained on a standard Desktop PC having an Intel Dual Core CPU at 2.66 GHz and 4 GB RAM running Debian Linux (kernel version 2.6.26). All algorithms as well as the testbed allowing for injecting memory corruptions were implemented in the programming language `C` and were compiled using the `gcc` compiler (version 4.3.2) with optimization level `-O3`.



(a) RGB (b) Labels (c) K-Means

Figure 8.4. *The* `Qinghai Small` *data set is a subset of the* `Qinghai Large` *data set [144]: (a) RGB image based on the 11 bands, (b) manually obtained labels (clouds, ground), and (c) standard k-means clustering result (ARI=0.940).*

Data Sets

Following Wagstaff and Bornstein [144], we use the `Iris` data set from the UCI repository [53], which consists of $N = 150$ patterns each having $d = 4$ features. Furthermore, we test all approaches on satellite data obtained from the Hyperion instrument that is aboard of the EO-1 Earth orbiter. This instrument collects data at 242 wavelengths, but on board computations can only be performed on a selectable subset of up to 12 bands [28, 144]. For our experiments we consider the same 11 bands used by the on board pixel SWIL classifier described by Castano *et al.* [28]. We consider two particular data sets that are based on observations of the Qinghai province in China made on October 3, 2002.[3] The first data set (`Qinghai Large`) contains $N = 179,200$ patterns (i. e., pixels) each having $d = 11$ features (bands), see Figure 8.3. The second data set (`Qinghai Small`) is a subset of the previous one and contains $N = 1,600$ patterns with $d = 11$ features. For all experiments no model selection is performed (since the true number of clusters is known), and we use all patterns for training ($n = N$).

Clustering Evaluation

For both the `Qinghai Large` and the `Qinghai Small` data set the labels were manually obtained for all pixels (clouds, ground, water), see Figures 8.3 and 8.4. The quality of a computed partition with respect to these true labels is then measured via the *Adjusted Rand Index* (ARI) [79]. This index is a measure for the agreement of two given partitions of a set. A value of 1.0 indicates a perfect agreement of both partitions and a value of 0.0 is obtained in expectation by random partitions for the set. Negative values stem from anti-correlated partitions. Figures 8.3 and 8.4 show the clustering accuracies of the (unmodified) k-means clustering algorithm for both the `Qinghai Large` (ARI=0.749) and the `Qinghai Small` data set (ARI=0.940).

[3] Both data sets were kindly provided by the authors of [144, 145].

Parameters

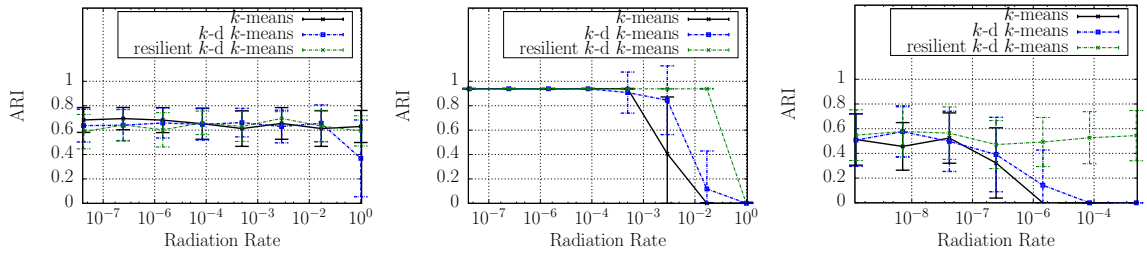
Various parameters need to be set for all approaches. We set the number of designated clusters k to the true number of classes for each data set (i. e., $k = 3$ for `Iris` and `Qinghai Large` and $k = 2$ for `Qinghai Small`). Since all convergence guarantees of the k-means clustering approach are invalid due to the (possible) corruption of memory information, we fix the number of iterations to 30 for all experiments. The k-means approach is, in general, susceptible to the problem of local optima. In addition, the memory corruptions can lead to unstable results. We therefore average the achieved clustering results over 30 runs. For our resilient k-d k-means variant two additional parameters δ and B have to be set. If not noted otherwise, we fix $\delta = 10$ and $B = 5$. The memory corruptions can lead to unpredictable behavior of all implementations and, hence, to unlimited practical runtimes. We therefore restrict all algorithms to finish within a user-defined time interval, which we set to 500 seconds.

Memory Manager

The testbed for performing memory corruptions is based on a *memory manager*. The memory manager can be used to allocate corruptible cells, i. e., bytes of memory which can be affected by bit flips. The $\mathcal{O}(1)$ corruption-free cells can be obtained using the standard `C`-routines for memory administration. In addition to the allocation of sufficient bytes of memory, the memory manager also takes care of an auxiliary data structure which can be used to perform memory corruptions at arbitrary positions of the space allocated through it. A memory corruption consists of a single bit flip. Note that, since all algorithms make use of corruptible indices (e. g., the current assignments of the patterns), access to arbitrary memory positions might occur (see below). Following Wagstaff and Bornstein [144], we do not corrupt the program code itself.

Implementation Issues

To assure a corruption-free execution of the code, we endow all algorithms with a `getIndex`-routine that prevents out-of-bounds accesses to invalid memory locations. The access of a memory position via a possibly corrupted index is safeguarded by this routine, i. e., if a non-valid (corrupted) index is used to access a memory location, the routine redirects the access by setting the index to a predefined valid value. Since Wagstaff and Bornstein [144] observed that the k-d k-means algorithm is most vulnerable if the k-d tree is affected by bit flips, we further improve the resiliency of our competitor by implementing the non-resilient k-d k-means variant using a pointerless k-d tree, i. e., we embed this tree in breadth-first-search order in an array as well.



(a) `Iris` (b) `Qinghai Small` (c) `Qinghai Large`

FIGURE 8.5. *Clustering performance of all approaches under the disturbance of memory corruptions (determined by the radiation rate). Averaged results with one standard deviation are reported.*

Memory Corruptions

Following Wagstaff and Bornstein [144], we consider a *radiation rate* that determines the amount of *single-event upsets* (SEUs) per byte and per second. Hence, the amount of memory corruptions depends on both the runtime and the (current) space usage of an algorithm, i. e., corruptions are performed proportional to the radiation rate, the execution time, and the allocated space (as recorded by the memory manager). In a real-world scenario memory corruptions can happen at any time. To simplify the implementation, we inject a batch of memory corruptions after each iteration (based on the runtime of the last iteration and global space usage). Finally, we do not inject any corruptions during the building phase of both k-d tree-based approaches. This is justified by the insignificant relative running time of the building phase.[4]

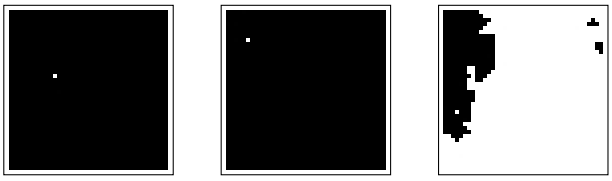
8.4.2 Results

In the remainder of this section we will evaluate the clustering performances of all approaches under the influence of memory corruptions.

Clustering Accuracy

A natural question is how varying radiation rates affect the clustering performances of the different approaches. In Figure 8.5, the behavior of all implementations on the three data sets given varying rates of radiation is given. On the `Iris` data set all approaches exhibit a quite similar clustering performance. On the other two data sets the resilient version shows a superior clustering accuracy. This might be due to the fact that both k-d tree variants are well-suited for data sets with a large amount of patterns (i. e., large n) in a low-dimensional feature space (i. e., small d). Here, the practical runtime depends

[4]Profiling our implementation indicated that consistently less than 6% of the overall running time were spent during the building phase of the (resilient) k-d trees.



(a) (b) (c)

FIGURE 8.6. *Final clustering result on the* `Qinghai Small` *data set of all approaches given a radiation rate of* 10^{-2}. *For (a)* `k-means` *and (b)* `k-d k-means` *one of the two final cluster centers has only a single training pattern assigned to it. The latter outcomes supposedly stem from a single corrupted training pattern (where the exponent of its floating point representation is hit). The* `resilient k-d k-means` *is not affected by such corruptions in this case, see Figure (c).*

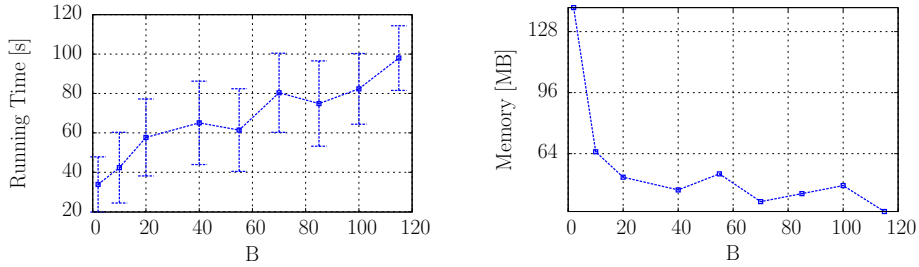
on the efficiency of the pruning approach, which takes place in the upper region of the associated k-d tree. Since this information is stored resiliently for the `resilient k-d k-means` approach, memory corruptions should have less influence on the final clustering performance (compared to `k-d k-means`).

A comparison of the clustering results for our resilient approach on `Qinghai Small` and `Qinghai Large` also indicates that the influence of the radiation rate increases with increasing size of the data set for the non-resilient schemes. An explanation for this behavior might be that even a single bit flip can have a significant influence on the final clustering result for such settings (for instance, if the exponent of a floating point representation is corrupted). This issue is depicted in Figure 8.6. Since the probability for memory corruptions increases with increasing size of the data set (given a fixed radiation rate), we expect worse results when clustering large data sets with non-resilient algorithms. Thus, data scenarios like the one induced by the `Qinghai Large` data set depict excellent candidates for the resilient clustering approach proposed in this chapter.

Trade-Off Parameter B

The parameter B induces a trade-off between space usage and likelihood of resiliently filtering candidate centers: Here, a large value leads to leaf structures corresponding to many points and, thus, to a top tree containing few nodes. In this case, the chances of resiliently filtering candidates before reaching the leaf structures are diminished. The extreme case takes place for $B = n/\delta$, and the resulting algorithm corresponds to the standard k-means scheme.

To analyze the influence of the parameter B on the runtime and space consumption, we consider the `Qinghai Large` data set and fix the model parameter $\delta = 100$. Further, we set the radiation rate to 10^{-8} (SEUs per byte and per second). The results are depicted in Figure 8.7: Clearly, for increasing B, the runtime increases as well. Thus, the benefits of the filtering approach in such settings are not diminished due the additional cost in-



(a) Running Time (b) Memory Usage

FIGURE 8.7. *Running time behavior and space consumption of the* `resilient k-d k-means` *implementation for varying assignments of the parameter B. The plots indicate that the running time increases and the space consumption decreases for increasing B, respectively. A reasonable trade-off between running time and space consumption is given for, e. g., $B = 10$ or $B = 20$.*

duced by storing the data items reliably in the top tree. Further, as expected, the space requirements decrease for increasing B due to the top tree structure becoming smaller. Taking both plots into account indicate that $B = 10$ or $B = 20$ seem to be appropriate assignments for inducing a reasonable space/time trade-off.

8.5 Concluding Remarks

Computational problems in terms of memory corruptions can occur during the automatic data analysis taking place on today's spacecraft systems. In general, software- and hardware-based countermeasures can be employed to shorten the disturbing influence of such memory corruptions. Both concepts, however, lead to an increase of mass and in a reduction of computational capabilities, which are undesired side-effects due to the enormous transportation costs for such systems. Resilient algorithms and data structures depict a valuable alternative to cope with such settings. In this chapter, we have proposed a resilient version of the k-d tree data structure and showed how to incorporate it into the (standard) k-d tree enhanced k-means clustering scheme. The conducted experiments indicate that the resulting clustering framework exhibits a superior clustering accuracy under the influence of memory corruptions than competing methods. This seems to be especially the case for large data sets in low-dimensional feature spaces (like, e. g., hyperspectral image data). In these scenarios, the negative effect of memory corruptions seems to be even more severe since single bit flips can jeopardize the overall outcome of a clustering algorithm.

Part IV

Summary

CHAPTER 9

Summary and Outlook

In this chapter, the main results provided in this thesis will be summarized. In addition to that, we will depict future research directions related to both the extensions of support vector machines to semi- and unsupervised learning settings as well as to possible application domains of machine learning tools in the field of astronomy.

9.1 Summary

After providing machine learning basics as well as a description of support vector machines in Part I, we showed how to extend these concepts to semi- and unsupervised learning scenarios in Part II of this work. Part III was devoted to specific algorithmic issues in the emerging field of astroinformatics [6, 17].

9.1.1 Semi- and Unsupervised Support Vector Machines

While several heuristics can be found in the literature that try to tackle the combinatorial optimization problems induced by semi- and unsupervised support vector machines, only few attempts have been made to approach the question of how to efficiently compute solutions with guaranteed accuracy. In Chapter 4, we studied this question and provided a polynomial-time algorithm that is capable of computing such solutions for the special case of a linear kernel. The resulting framework provides valuable insights into the computational complexity of the induced combinatorial optimization tasks. Further, an interesting connection between the problems at hand and the field of computational geometry was derived that is based on arrangements and associated incidence graphs.

Although the exact approach is interesting from a theoretical point of view, its practical use is limited due to the involved computational complexities. In Chapter 5, we therefore proposed an efficient implementation of a simple label-switching strategy to address the optimization tasks. The idea was to resort to a least-squares variant of support vector machines and to perform efficient matrix-based updates of the intermediate candidate solutions. This led to linear-time operations per iteration of the local search and paved the way for efficiently testing a massive amount of possible solutions. The proposed framework is applicable to both semi- and unsupervised learning settings. While the shortcuts render the local search approach competitive with state-of-the-art schemes in the context of semi-supervised learning, they become even more useful in unsupervised scenarios (due to missing initial guesses provided by labeled patterns).

The alternative continuous optimization perspective on the tasks was depicted in Chapter 6. Here, a conceptually very simple quasi-Newton framework turned out to be well-suited for the induced tasks. For large-scale settings, we demonstrated how to make use of a memory-saving quasi-Newton scheme in combination with kernel matrix approximation techniques as well as in combination with computational shortcuts that make use of the properties of sparse data. The experimental evaluation indicated that the resulting approach yields a superior performance compared to state-of-the-art methods, both with respect to the practical runtime as well as with respect to the classification performance.

9.1.2 Applications in Astronomy

The third part of this thesis dealt with the application of machine learning tools in the field of astronomy. In particular, we considered learning settings on earth as well as learning settings in space. For the data analysis on earth, we approached the task of discriminating quasars from other objects based on spectroscopic data. Aiming at a meaningful and efficient feature extraction, we proposed a variant of standard support vector regression models to obtain continuum-subtracted versions of the raw spectra and showed how to efficiently address the induced optimization tasks. The reduced spectra formed the basis for extracting a small set of expert-based features that led to a significantly better classification performance. In addition to these extraction schemes, we briefly investigated the use of semi-supervised support vector machines in this context and sketched research perspectives related to multi-view learning.

For the data analysis in space, we considered the task of clustering hyperspectral image data. As depicted in Chapter 8, such schemes are already in use aboard today's spacecraft systems to preprocess the data. In contrast to the analysis taking place on earth, the one in space induces computational problems caused by the cosmic radiation that leads to memory corruptions. To cope with such corruptions, one can resort to software- or

hardware-based countermeasures. However, such schemes usually lead to a significant increase of needed physical memory (and, thus, to an increase of mass and transportation costs). As an alternative, we proposed the use of resilient algorithms in this context. In particular, we demonstrated that a resilient version of the classical k-d tree can be used to reduce the negative influence of memory corruptions in an efficient kind of way.

9.2 Research Directions

This work motivates several research directions, which will be described in the following.

9.2.1 Semi- and Unsupervised Learning

Three research directions could be addressed in the context of semi- and unsupervised support vector machines: Firstly, the theoretical complexities of the induced optimization tasks could be analyzed in a more detailed manner. Secondly, the particular optimization heuristics could be improved (while not only focussing on computational aspects). Thirdly, sophisticated model selection schemes could be developed for the two learning scenarios.

Exact Solutions in Less Time

The polynomial-time approach derived in Chapter 4 yields exact solutions with guaranteed accuracy. An interesting future research direction is the question if one can obtain exact solutions with *infinite* accuracy in polynomial time. Note that for hard-margin unsupervised support vector machines, this is the case due to a connection between hard-margin support vector machines and convex hulls [12, 70, 102, 116]. For the soft-margin case, a similar connection to the field of computational geometry is given via the concept of *reduced convex hulls* [102]. However, only exact solutions with guaranteed (but not infinite) accuracy can be obtained this way.

The exact scheme is only applicable for the special case of a linear kernel. A more general approach to obtain exact solutions (for arbitrary kernels) are branch-and-bound strategies like the one proposed by Chapelle *et al.* [35] for semi-supervised support vector machines. For such settings, the computational shortcuts depicted in Chapter 5 should also be applicable to accelerate the overall execution. Hence, in the future, efforts could be made to develop efficient updating schemes for such exact frameworks as well.

Towards Better Heuristics

The local search scheme presented in Chapter 5 is quite simple. In this context, more sophisticated heuristics have the potential to improve the quality of the obtained solutions. The approach given by Adankon *et al.* [2], e. g., resorts to an alternating optimization

framework that is also based on the square loss. This and other schemes can be accelerated by means of the matrix-based updates provided in this work. Future heuristics could also have some kind of greedy character, i. e., they could aim at flipping the pair of labels that leads to the best improvement per iteration. An interesting question is how to detect such pairs efficiently (in, e. g., linear time). Another future research direction is the analysis of the theoretical convergence behavior of such local search schemes for the tasks at hand (similar to the analysis conducted for simple evolutionary algorithms in the field of stochastic optimization [44]).

Almost all semi-supervised state-of-the-art methods are based on a single local search (without any restarts). As shown in Chapters 5 and 6, putting more effort into optimization can lead to better results. In general, more work has to be spent on the mutual assessment of competing optimization frameworks. Since the difficulty of the induced tasks strongly depends on the particular assignments for the involved parameters, one should conduct such an analysis independently of model selection issues. This could be achieved, for instance, by establishing benchmark data sets (with fixed parameter assignments) like it is the case in the field of stochastic optimization [13], where competing approaches are evaluated and compared on challenging benchmark functions.

Model Selection Issues

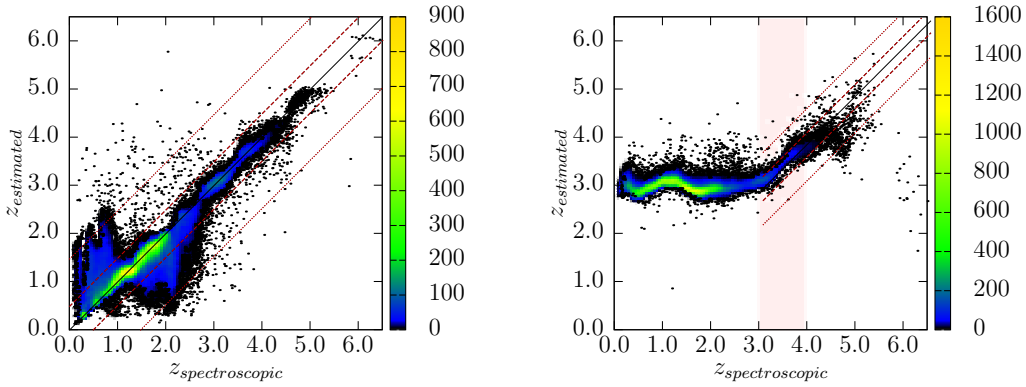
Part II of this work focussed on optimization issues. An important research direction (usually not addressed in the literature) is the question of how to select appropriate model parameters. While one can still fall back on a small set of labeled patterns in semi-supervised learning settings, this is not the case for unsupervised ones anymore. In this context, concepts like *transfer learning* [110] might play an important role in the future.

9.2.2 Astroinformatics: An Emerging Discipline

In today's and near-future astronomy, a variety of different learning tasks have or will have to be addressed. Further, labeled data is usually scarce in this field while unlabeled data is available in huge quantities. Two important research directions arise in astronomy, namely how one can cope with the lack of labeled data and how one can handle the sheer (upcoming) data volume.

Lack of Labeled Data

As sketched in Chapter 7, the lack of labeled data is a crucial issue in astronomy. As an example, consider the task of estimating the redshift of quasars given only photometric data: In Figure 9.1 (a), the outcome of a k-nearest neighbor regression model [73] trained on about $100,000$ photometric patterns is given (where the true redshift labels are obtained



(a) Nearest Neighbor Classifier (b) Linear Support Vector Regression

FIGURE 9.1. *A current research topic in astronomy is the search for new, distant quasars [106]. This task can be approached by standard regression models. In Figure (a), the output of a k-nearest neighbor model is shown, where a data set of about* $100,000$ *objects with four photometric features [65] per object is used for training and evaluating the model; see Polsterer [115] for a detailed description of the experimental setting and a comprehensive analysis of the results. In Figure (b), a support vector regression model is shown that has only been trained on a subset of quasars with redshifts* $z \in [3,4]$ *(highlighted region).*

via the spectroscopic information that is also available for these particular objects). Such models can be used to predict redshift values for photometric objects that have not yet been target of a spectroscopic follow-up observation.[1] While there is a large amount of labeled patterns corresponding to low-redshift quasars, only very few quasars are known that exhibit higher redshifts. This results in very sparsely populated regions in the feature space and, thus, renders such prediction models imprecise for distant quasars. Another important issue is the fact that for detecting new, unseen quasars, no labels at all are given. In Figure 9.1 (b), the output of a linear support vector regression model is shown that is only trained on quasars having a redshift of $z \in [3,4]$. It can be seen that the model is still capable of extrapolating to some degree, i. e., it can make reasonable predictions for quasars with $z \in [4,5]$. An interesting future research direction is, for instance, the question which particular feature space (i. e., combinations of photometric features) gives rise to the best extrapolation performance. Further, semi-supervised approaches have not been considered in this context. Such schemes might be extremely useful to reveal additional information about the structure of the data.

Dealing with Massive Data Sets

From a computational point of view, current and near-future projects in astronomy are faced with the problem that standard algorithmic schemes will not be capable to deal with

[1] It is worth pointing out that this simple classification scheme yields results that are competitive with state-of-the-art frameworks in the field of astronomy [65, 96].

the massive amounts of data. While classical data structures like k-d trees can accelerate search queries for relatively large data sets, such schemes will also fail if huge data sets in the tera- and petabyte range are considered. As an example, consider the task of sorting all about one billion photometric objects in the current release of the Sloan Digital Sky Survey. Even if only 20 data items are considered per object (as double precision numbers), one already has to deal with 16 gigabytes of data. For such settings, not the pure computation time but memory-related issues like the transfer of data items from, e. g., hard disk to internal memory can dominate the overall runtime. *External memory algorithms* aim at dealing with such scenarios in an efficient kind of way. Such algorithms are usually analyzed in the context of the so-called *I/O-model* [3] that defines the parameters N (number of objects of the problem instance), M (size of internal memory), and B (size of objects per disk block). In this model, an *I/O-operation* consists of reading a block of contiguous elements from external into internal memory or writing a block from internal into external memory. Further, computations can only be performed on objects that are given in internal memory. Besides being efficient with respect to the computation time, external memory algorithms also aim at reducing the amount of needed I/O-operations. Hence, this model of computation captures the characteristics of working with massive data sets that are too large to fit into main memory.

From a machine learning point of view, it might be desirable to take as much as possible of the given data into account to construct appropriate prediction models. As sketched above, for instance, the efficient computation of nearest neighbors for large sets of photometric objects could be required. Such computations can also be performed efficiently in the context of the I/O-model [27], where the so-called *well-separated pair decomposition* (WSPD) is one of the key algorithmic ingredients for the overall framework. A possible drawback of such schemes, however, is the fact that the resulting algorithms and data structures are often quite difficult to implement. The *cache-oblivious model* of Frigo *et al.* [54] aims at simplifying the induced implementations. A promising result in this context is given by Gieseke *et al.* [58, 60] who demonstrated that a WSPD can be computed efficiently in the cache-oblivious model as well. This result could pave the way for efficient cache-oblivious nearest neighbor schemes that can deal with huge data sets.

These external memory algorithms could be used in case huge data sets are stored on disk and in case all data items have to be examined (e. g., when sorting all patterns). However, future research projects in astronomy will produce data volumes in the peta- or exabyte range, which cannot even be stored [95, 138]. For such scenarios, on-line monitoring and analysis schemes will play an important role (like, e. g., the automatic detection of rare events). For such settings, techniques like time-series analysis and visual monitoring of events [91, 92] will be important to identify relevant data items.

Appendix

Author's Contribution

Most parts of this work are based on published and not yet published results that stem from collaborations with other authors. This chapter depicts the author's contribution to these co-authored publications and manuscripts.

Chapter 4: The results provided in this chapter are based on the following work:

- F. Gieseke, J. Vahrenhold, and X. Jiang. Exact linear semi-and unsupervised support vector machines in polynomial time. 2011. Manuscript.

The article has mainly been written by me; all the authors contributed to prepare the final manuscript. Xiaoyi Jiang drew my attention to this interesting problem and provided various helpful suggestions and comments with respect to both theoretical as well as practical issues. The polynomial-time scheme for the hard-margin case is based on a discussion with Jan Vahrenhold. The extension of these initial ideas to the soft-margin case, the handling of degenerated cases via the incidence graph structure, the extension to the semi-supervised case, and the classification of the obtained results in the context of the related literature are by me. In addition, the experimental evaluation including the implementation of the approach and the generation of benchmark data sets have been conducted by me.

Chapter 5: This chapter is based on the following two publications:

- F. Gieseke, T. Pahikkala, and O. Kramer. Fast evolutionary maximum margin clustering. In *Proceedings of the 26th International Conference on Machine Learning*, pages 361–368, 2009.

- F. Gieseke, O. Kramer, A. Airola, and T. Pahikkala. Speedy local search for semi-supervised regularized least-squares. In *Proceedings of the 34th Annual German Conference on Artificial Intelligence*, pages 87–98, 2011.

Both papers have mainly been written by me; all the authors have contributed to writing up the manuscripts. The main idea of using matrix-based computational shortcuts, the mathematical derivations, the implementation, and the conducted experiments are for the most part by me. Tapio Pahikkala and Antti Airola provided valuable comments and improvements related to matrix calculus and kernel methods. In addition, Tapio Pahikkala contributed to the depicted computational shortcuts. Oliver Kramer supported me through all the stages of the work and provided valuable comments and improvements related to stochastic search issues and kernel methods.

Chapter 6: The results provided in this chapter are based on the following manuscript:

- F. Gieseke, A. Airola, T. Pahikkala, and O. Kramer. Sparse quasi-Newton optimization for semi-supervised support vector machines. In *Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods*, pages 45–54, 2012.

The work has mainly been conducted by me. Tapio Pahikkala and Antti Airola gave useful comments related to the considered surrogate loss functions and the handling of sparse data. Oliver Kramer gave valuable comments related to optimization issues in high-dimensional search spaces.

Chapter 7: This chapter is partially based on two publications. The first one stems from a collaboration with astronomers of the *Ruhr-Universität Bochum*:

- F. Gieseke, K. L. Polsterer, A. Thom, P. Zinn, D. Bomans, R.-J. Dettmar, O. Kramer, and J. Vahrenhold. Detecting quasars in large-scale astronomical surveys. In *Proceedings of the 9th International Conference on Machine Learning and Applications*, pages 352–357, 2010.

The feature extraction scheme is based on a joint work with Kai Lars Polsterer, Andreas Thom, and Peter Zinn. All co-authors were involved in the collaboration and in preparing the final manuscript. The second publication is a result of a joint work conducted at the *Turku Centre for Computer Science*:

- E. Tsivtsivadze, F. Gieseke, T. Pahikkala, J. Boberg, and T. Salakoski. Learning preferences with co-regularized least-squares. In *Proceedings of the ECML/PKDD Workshop on Preference Learning*, pages 52–66, 2008.

Evgeni Tsivtsivadze had the idea to combine multi-view and preference learning; I contributed to the mathematical formalization of these ideas. All the five authors were involved in preparing the final manuscript.

Chapter 8: This chapter is based on the following two publications:

- F. Gieseke, G. Moruz, and J. Vahrenhold. Resilient k-d trees: K-means in space revisited. In *Proceedings of the 10th IEEE International Conference on Data Mining*, pages 815–820, 2010.

- F. Gieseke, G. Moruz, and J. Vahrenhold. Resilient k-d trees: K-means in space revisited. *Frontiers of Computer Science*, 2011. Accepted.

All the authors contributed to the writing of both papers. I contributed to the experimental setup, the implementation of the memory manager used for injecting memory corruptions, and the implementation of the different k-means variants.

Chapter 9: In addition to providing concluding remarks, future research directions were sketched in this chapter. Among other things, two possible research fields have been suggested that might play an important role for the automatic analysis of astronomical data sets in the near future, namely how to deal with the lack of labeled data and how to cope with the sheer data volumes in this field. For both settings, the following work has (partially) been conducted by me:

- F. Gieseke, J. Gudmundsson, and J. Vahrenhold. Pruning spanners and constructing well-separated pair decompositions in the presence of memory hierarchies. *Journal of Discrete Algorithms*, 8(3):259–272, 2010

- F. Gieseke and J. Vahrenhold. Cache-oblivious construction of a well-separated pair decomposition. In *Proceedings of the 25th European Workshop on Computational Geometry*, pages 341–344, 2009

- O. Kramer and F. Gieseke. Short-term wind energy forecasting using support vector regression. In *Proceedings of the International Conference on Soft Computing Models in Industrial and Environmental Applications*, Advances in Intelligent and Soft Computing, pages 271–280, 2011

- O. Kramer and F. Gieseke. Analysis of wind energy time series with kernel methods and neural networks. In *Proceedings of the 7th International Conference on Natural Computation*, pages 2381–2385, 2011

- F. Gieseke, K. Polsterer, and P. Zinn. Photometric redshift estimation of quasars: Local versus global regression. In *Proceedings of the Astronomical Data Analysis Software and Systems*, 2011. In print

Although these publications are only slightly related to the work at hand, they can serve as a source for further reading. The first two articles are based on my master's thesis. One of the key contributions is a cache-oblivious well-separated pair decomposition that can potentially be used to accelerate nearest neighbor computations in a cache-oblivious manner. Further, I contributed to the writing and the experimental evaluation of the last three publications and manuscripts.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] S. Abe. *Support Vector Machines for Pattern Classification*. Springer, Secaucus, NJ, USA, 2005.

[2] M. Adankon, M. Cheriet, and A. Biem. Semisupervised least squares support vector machine. *IEEE Transactions on Neural Networks*, 20(12):1858–1870, 2009.

[3] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[4] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75:245–248, 2009.

[5] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, 2 edition, 2010.

[6] N. M. Ball and R. J. Brunner. Data Mining and Machine Learning in Astronomy. *ArXiv e-prints*, 2010. arXiv:0906.2173v2.

[7] R. G. Bartle. *The Elements of Integration and Lebesgue Measure*. Wiley-Interscience, 1995.

[8] S. Basu, I. Davidson, and K. Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. Chapman & Hall/CRC, 1 edition, 2008.

[9] K. P. Bennett and E. J. Bredensteiner. Duality and geometry in svm classifiers. In *Proceedings of the 17th International Conference on Machine Learning*, pages 57–64, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[10] K. P. Bennett and A. Demiriz. Semi-supervised support vector machines. In *Advances in Neural Information Processing Systems 11*, pages 368–374. MIT Press, 1999.

[11] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[12] M. W. Bern and D. Eppstein. Optimization over zonotopes and training support vector machines. In *Proceedings of the 7th Workshshop on Algorithms and Data Structures*, number 2125 in Lecture Notes in Computer Science, pages 111–121. Springer, 2001.

[13] H.-G. Beyer and H.-P. Schwefel. Evolution strategies - A comprehensive introduction. *Natural Computing*, 1:3–52, 2002.

[14] T. D. Bie and N. Cristianini. Convex methods for transduction. In *Advances in Neural Information Processing Systems 16*, pages 73–80. MIT Press, 2004.

[15] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[16] K. M. Borgwardt. *Graph Kernels*. PhD thesis, Fakultät für Mathematik, Informatik und Statistik der Ludwig-Maximilians-Universität München, 2007.

[17] K. Borne. Scientific data mining in astronomy. 2009. arXiv:0911.0505v1.

[18] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, pages 144–152, 1992.

[19] O. Bousquet, S. Boucheron, and G. Lugosi. Introduction to statistical learning theory. In *Advanced Lectures on Machine Learning*, volume 3176 of *Lecture Notes in Computer Science*, pages 169–207. Springer, 2004.

[20] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[21] R. S. Boyer and J. S. Moore. MJRTY: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 105–118, 1991.

[22] U. Brefeld and T. Scheffer. Co-EM support vector learning. In *Proceedings of the 21st International Conference on Machine Learning*, page 16, New York, NY, USA, 2004. ACM.

[23] U. Brefeld, T. Gärtner, T. Scheffer, and S. Wrobel. Efficient co-regularised least squares regression. In *Proceedings of the 23rd International Conference on Machine learning*, pages 137–144, New York, NY, USA, 2006. ACM.

[24] G. S. Brodal, R. Fagerberg, I. Finocchi, F. Grandoni, G. Italiano, A. G. Jørgensen, G. Moruz, and T. Mølhave. Optimal resilient dynamic dictionaries. In *Algorithms – ESA 2007. 15th Annual European Symposium*, volume 4698 of *Lecture Notes in Computer Science*, pages 347–358. Springer, 2007.

[25] C. J. C. Burges and D. J. Crisp. Uniqueness of the SVM solution. In *Advances in Neural Information Processing Systems 12*, pages 223–229, 2000.

[26] R. H. Byrd, P. Lu, P. Lu, J. Nocedal, J. Nocedal, C. Zhu, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[27] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42(1):67–90, 1995.

[28] R. Castano, D. Mazzoni, N. Tang, T. Doggett, S. Chien, R. Greeley, B. Cichy, and A. Davies. Learning classifiers for science event detection in remote sensing imagery. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2005.

[29] 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`.

[30] K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. Coordinate descent method for large-scale l2-loss linear support vector machines. *Journal of Machine Learning Research*, 9: 1369–1398, 2008.

[31] O. Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19:1155–1178, 2007.

[32] O. Chapelle and A. Zien. Semi-supervised classification by low density separation. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, pages 57–64, 2005.

[33] O. Chapelle, M. Chi, and A. Zien. A continuation method for semi-supervised svms. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 185–192, 2006.

[34] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[35] O. Chapelle, V. Sindhwani, and S. S. Keerthi. Branch and bound for semi-supervised support vector machines. In *Advances in Neural Information Processing Systems 19*, pages 217–224. MIT Press, 2007.

[36] O. Chapelle, V. Sindhwani, and S. S. Keerthi. Optimization techniques for semi-supervised support vector machines. *Journal of Machine Learning Research*, 9:203–233, 2008.

[37] S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castano, A. Davis, and D. Boyer. Using autonomy flight software to improve science return on earth observing one. *Journal of Aerospace Computing, Information, and Communication*, 2:196–216, 2005.

[38] R. Collobert, F. Sinz, J. Weston, and L. Bottou. Trading convexity for scalability. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 201–208, 2006.

[39] R. Collobert, F. Sinz, J. Weston, and L. Bottou. Large scale transductive svms. *Journal of Machine Learning Research*, 7:1687–1712, 2006.

[40] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

[41] D. J. Crisp and C. J. C. Burges. A geometric interpretation of ν-svm classifiers. In *Advances in Neural Information Processing Systems 12*, pages 244–250, Cambridge, MA, 2000. MIT Press.

[42] L. Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems 2*, pages 396–404, 1990.

[43] M. de Berg, O. Cheong, M. van Krefeld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3 edition, 2008.

[44] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1-2):51–81, 2002.

[45] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, New York, NY, USA, 1987.

[46] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Computing*, 15:341–363, 1986.

[47] J. Elstrodt. *Maß und Integrationstheorie*. Springer, 2004.

[48] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6:1889–1918, 2005.

[49] I. Finocchi, F. Grandoni, and G. F. Italiano. Designing reliable algorithms in unreliable memories. *Computer Science Review*, 1(2):77–87, 2007.

[50] I. Finocchi, F. Grandoni, and G. F. Italiano. Optimal resilient sorting and searching in the presence of dynamic memory faults. *Theoretical Computer Science*, 410(44): 4457–4470, 2009.

[51] C. A. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*. Oxford University Press, USA, 1995.

[52] I. K. Fodor. A survey of dimension reduction techniques. Technical report, Lawrence Livermore National Laboratory, 2002. UCRL-ID-148494.

[53] A. Frank and A. Asuncion. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2010.

[54] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Symposium on Foundations of Computer Science*, pages 285–299, 1999.

[55] G. Fung and O. L. Mangasarian. Semi-supervised support vector machines for unlabeled data classification. *Optimization Methods and Software*, 15:29–44, 2001.

[56] J. Fürnkranz and E. Hüllermeier. Preference learning. *Künstliche Intelligenz*, 19(1): 60–61, 2005.

[57] Galaxy Zoo: Hubble. `http://www.galaxyzoo.org/`, November 2011.

[58] F. Gieseke and J. Vahrenhold. Cache-oblivious construction of a well-separated pair decomposition. In *Proceedings of the 25th European Workshop on Computational Geometry*, pages 341–344, 2009.

[59] F. Gieseke, T. Pahikkala, and O. Kramer. Fast evolutionary maximum margin clustering. In *Proceedings of the 26th International Conference on Machine Learning*, pages 361–368, 2009.

[60] F. Gieseke, J. Gudmundsson, and J. Vahrenhold. Pruning spanners and constructing well-separated pair decompositions in the presence of memory hierarchies. *Journal of Discrete Algorithms*, 8(3):259–272, 2010.

[61] F. Gieseke, G. Moruz, and J. Vahrenhold. Resilient k-d trees: K-means in space revisited. In *Proceedings of the 10th IEEE International Conference on Data Mining*, pages 815–820, 2010.

[62] F. Gieseke, K. L. Polsterer, A. Thom, P. Zinn, D. Bomans, R.-J. Dettmar, O. Kramer, and J. Vahrenhold. Detecting quasars in large-scale astronomical surveys. In *Proceedings of the 9th International Conference on Machine Learning and Applications*, pages 352–357, 2010.

[63] F. Gieseke, O. Kramer, A. Airola, and T. Pahikkala. Speedy local search for semi-supervised regularized least-squares. In *Proceedings of the 34th Annual German Conference on Artificial Intelligence*, pages 87–98, 2011.

[64] F. Gieseke, G. Moruz, and J. Vahrenhold. Resilient k-d trees: K-means in space revisited. *Frontiers of Computer Science*, 2011. Accepted.

[65] F. Gieseke, K. Polsterer, and P. Zinn. Photometric redshift estimation of quasars: Local versus global regression. In *Proceedings of the Astronomical Data Analysis Software and Systems*, 2011. In print.

[66] F. Gieseke, J. Vahrenhold, and X. Jiang. Exact linear semi-and unsupervised support vector machines in polynomial time. 2011. Manuscript.

[67] F. Gieseke, A. Airola, T. Pahikkala, and O. Kramer. Sparse quasi-Newton optimization for semi-supervised support vector machines. In *Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods*, pages 45–54, 2012.

[68] G. H. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore and London, second edition, 1989.

[69] G. H. Golub and C. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore and London, third edition, 1996.

[70] J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, Boca Raton, FL, USA, 1997.

[71] I. Guyon. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[72] N. Hansen. The CMA evolution strategy: A tutorial. Technical report, TU Berlin, ETH Zürich, 2005.

[73] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, 2 edition, 2009.

[74] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *Proceedings of the 9th International Conference on Articial Neural Networks*, pages 97–102, London, 1999. Institute of Electrical Engineers.

[75] B. F. Hildebrand. *Introduction to numerical analysis: 2nd edition.* Dover Publications, Inc., New York, NY, USA, 1987.

[76] T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The Annals in Statistics*, 36(3):1171–1220, 2008.

[77] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, 1990.

[78] E. P. Hubble. Extragalactic nebulae. *Astrophysical Journal*, 64:321–369, 1926.

[79] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1): 193–218, 1985.

[80] G. F. Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory*, 15(1):55–63, 1968.

[81] D. Hush, P. Kelly, C. Scovel, and I. Steinwart. QP algorithms with guaranteed accuracy and run time for support vector machines. *Journal of Machine Learning Research*, 7:733–769, 2006.

[82] M. Inaba, N. Katoh, and H. Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering (extended abstract). In *Proceedings of the 10th Annual Symposium on Computational Geometry*, pages 332–339, New York, NY, USA, 1994. ACM.

[83] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the International Conference on Machine Learning*, pages 200–209, 1999.

[84] T. Joachims. *Learning to Classify Text Using Support Vector Machines: Methods, Theory and Algorithms.* Kluwer, 2002.

[85] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*, pages 133–142, New York, NY, USA, 2002. ACM.

[86] T. Joachims. Training linear SVMs in linear time. In *KDD'06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.

[87] A. G. Jørgensen, G. Moruz, and T. Mølhave. Priority queues resilient to memory faults. In *Algorithms and Data Structures, 10th International Workshop 2007*, volume 4619 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2007.

[88] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[89] M. Kojima, S. Mizuno, and A. Yoshise. A polynomial-time algorithm for a class of linear complementarity problems. *Mathematical Programming*, 44:1–26, 1989.

[90] H. Kopetz. Mitigation of transient faults at the system level – the TTA approach. In *Online Proceedings of the 2nd Workshop on System Effects of Logic Soft Errors*, 2006.

[91] O. Kramer and F. Gieseke. Short-term wind energy forecasting using support vector regression. In *Proceedings of the International Conference on Soft Computing Models in Industrial and Environmental Applications*, Advances in Intelligent and Soft Computing, pages 271–280, 2011.

[92] O. Kramer and F. Gieseke. Analysis of wind energy time series with kernel methods and neural networks. In *Proceedings of the 7th International Conference on Natural Computation*, pages 2381–2385, 2011.

[93] S. Kumar, M. Mohri, and A. Talwalkar. Sampling techniques for the nyström method. *Journal of Machine Learning Research*, 5:304–311, 2009.

[94] J. Lafferty and L. Wasserman. Statistical analysis of semi-supervised regression. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 801–808. MIT Press, Cambridge, MA, 2008.

[95] Large Synoptic Survey Telescope. `http://www.lsst.org`, August 2011.

[96] O. Laurino, R. D'Abrusco, G. Longo, and G. Riccio. Astroinformatics of galaxies and quasars: a new general method for photometric redshifts estimation. 2011. arXiv:1107.3160.

[97] Y. LeCun and C. Cortes. The MNIST database. `http://yann.lecun.com/exdb/mnist/`, July 2011.

[98] Y.-F. Li, I. W. Tsang, J. T. Kwok, and Z.-H. Zhou. Tighter and convex maximum margin clustering. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, pages 344–351. JMLR: W&CP 5, 2009.

[99] N. List and H. U. Simon. General polynomial time decomposition algorithms. *Journal of Machine Learning Research*, 8:303–321, 2007.

[100] U. Luxburg and B. Schölkopf. Statistical learning theory: Models, concepts, and results. In *Handbook of the History of Logic*, volume 10, pages 751–706. Elsevier, 2011.

[101] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[102] M. E. Mavroforakis and S. Theodoridis. A geometric approach to support vector machine (svm) classification. *IEEE Transactions on Neural Networks*, 17(3):671–682, 2006.

[103] T. C. May and M. H. Woods. Alpha-particle-induced soft errors in dynamic memories. *IEEE Transactions on Electron Devices*, ED-26(1):2–9, Jan. 1979.

[104] I. Mierswa. *Non-convex and multi-objective optimization in data mining.* PhD thesis, Technische Universität Dortmund, 2009.

[105] T. M. Mitchell. *Machine Learning.* McGraw-Hill, New York, 1997.

[106] D. J. Mortlock, S. J. Warren, B. P. Venemans, M. Patel, P. C. Hewett, R. G. McMahon, C. Simpson, T. Theuns, E. A. Gonzales-Solares, A. Adamson, S. Dye, N. C. Hambly, P. Hirst, M. J. Irwin, E. Kuiper, A. Lawrence, and H. J. A. Rottgering. A luminous quasar at a redshift of z = 7.085. *Nature*, 474(7353):616–619, 2011.

[107] S. Nene, S. Nayar, and H. Murase. Columbia object image library (coil-100). Technical report, 1996.

[108] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, 2 edition, 2000.

[109] A. Ostermeier, A. Gawelczyk, and N. Hansen. A derandomized approach to self adaptation of evolution strategies. *Evolutionary Computation*, 2(4):369–380, 1994.

[110] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010.

[111] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[112] J. Peng, L. Mukherjee, V. Singh, D. Schuurmans, and L. Xu. An efficient algorithm for maximal margin clustering. *Journal of Global Optimization*, 2011. To appear.

[113] K. B. Petersen and M. S. Pedersen. The matrix cookbook, 2008. Version 20081110.

[114] U. F. Petrillo, I. Finocchi, and G. F. Italiano. The price of resiliency: a case study on sorting with memory faults. In *Algorithms – ESA 2006. 14th Annual European Symposium*, volume 4168 of *Lecture Notes in Computer Science*, pages 768–779. Springer, 2006.

[115] K. L. Polsterer. *The LUCIFER Control Software: The Core System, Instrument Control and Scientific Applications*. PhD thesis, Ruhr-Universität-Bochum, 2011.

[116] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.

[117] I. S. Reddy, S. Shevade, and M. Murty. A fast quasi-Newton method for semi-supervised SVM. *Pattern Recognition*, 44(10-11):2305–2313, 2011.

[118] R. Rifkin, M. Pontil, and A. Verri. A note on support vector machine degeneracy. In *Proceedings of the 10th International Conference on Algorithmic Learning Theory*, pages 252–263, London, UK, 1999. Springer-Verlag.

[119] R. Rifkin, G. Yeo, and T. Poggio. Regularized least-squares classification. In *Advances in Learning Theory: Methods, Models and Applications*. IOS Press, 2003.

[120] R. M. Rifkin. *Everything Old is New Again: A Fresh Look at Historical Approaches in Machine Learning*. PhD thesis, MIT, 2002.

[121] R. M. Rifkin and R. A. Lippert. Notes on regularized least-squares. Technical Report MIT-CSAIL-TR-2007-025, MIT Computer Science and Artificial Intelligence Laboratory, 2007.

[122] D. S. Rosenberg and P. L. Bartlett. The rademacher complexity of co-regularized kernel classes. *Journal of Machine Learning Research - Proceedings Track*, 2:396–403, 2007.

[123] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

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

[125] B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In *Proceedings of the 14th Annual Conference on Computational Learning Theory*, pages 416–426, 2001.

[126] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th International Conference on Machine learning*, pages 807–814, New York, NY, USA, 2007. ACM.

[127] C. Silva, J. S. Santos, E. F. Wanner, E. G. Carrano, and R. H. C. Takahashi. Semi-supervised training of least squares support vector machine using a multiobjective evolutionary algorithm. In *Proceedings of the 11th Conference on Congress on Evolutionary Computation*, pages 2996–3002, Piscataway, NJ, USA, 2009. IEEE Press.

[128] H. U. Simon. On the complexity of working set selection. *Theoretical Computer Science*, 382:262–279, 2007.

[129] V. Sindhwani and D. Rosenberg. An RKHS for multi-view learning and manifold co-regularization. In *Proceedings of the 25th Annual International Conference on Machine Learning*, pages 976–983. Omnipress, 2008.

[130] V. Sindhwani, P. Niyogi, and M. Belkin. A co-regularization approach to semi-supervised learning with multiple views. In *Proceedings of ICML Workshop on Learning with Multiple Views*, pages 74–79, 2005.

[131] V. Sindhwani, S. Keerthi, and O. Chapelle. Deterministic annealing for semi-supervised kernel machines. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 841–848, New York, NY, USA, 2006. ACM.

[132] Sloan Digital Sky Survey. `http://www.sdss.org`, August 2011.

[133] A. J. Smola, S. Vishwanathan, and T. Hofmann. Kernel methods for missing variables. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, pages 325–332, 2005.

[134] B. Sriperumbudur and G. Lanckriet. On the convergence of the concave-convex procedure. In *Advances in Neural Information Processing Systems 22*, pages 1759–1767, 2009.

[135] I. Steinwart and A. Christmann. *Support Vector Machines*. Springer-Verlag, New York, NY, USA, 2008.

[136] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[137] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

[138] The Large Sky Area Multi-Object Fiber Spectroscopic Telescope (LAMOST). `http://www.lamost.org/`, August 2011.

[139] E. Tsivtsivadze, F. Gieseke, T. Pahikkala, J. Boberg, and T. Salakoski. Learning preferences with co-regularized least-squares. In *Proceedings of the ECML/PKDD Workshop on Preference Learning*, pages 52–66, 2008.

[140] H. Valizadegan and R. Jin. Generalized maximum margin clustering and unsupervised kernel learning. In *Advances in Neural Information Processing Systems 19*, pages 1417–1424, 2007.

[141] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[142] V. Vapnik and A. Sterin. On structural risk minimization or overall risk in a problem of pattern recognition. *Automation and Remote Control*, 10(3):1495–1503, 1977.

[143] S. Vavasis. *Nonlinear Optimization: Complexity Issues*. Oxford University Press, 1991.

[144] K. L. Wagstaff and B. Bornstein. K-means in space: A radiation sensitivity evaluation. In *Proceedings of the 26th International Conference on Machine Learning*, pages 1097–1104, 2009.

[145] K. L. Wagstaff and B. Bornstein. How much memory radiation protection do onboard machine learning algorithms require? In *Proceedings of the IJCAI-09/SMC-IT-09/IWPSS-09 Workshop on Artificial Intelligence in Space*, 2009.

[146] F. Wang, B. Zhao, and C. Zhang. Linear time maximum margin clustering. *IEEE Transactions on Neural Networks*, 21(2):319–332, 2010.

[147] C. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001.

[148] L. Xu and D. Schuurmans. Unsupervised and semi-supervised multi-class support vector machines. In *Proceedings of the National Conference on Artificial Intelligence*, pages 904–910, 2005.

[149] L. Xu, J. Neufeld, B. Larson, and D. Schuurmans. Maximum margin clustering. In *Advances in Neural Information Processing Systems 17*, pages 1537–1544, 2005.

[150] M.-H. Yang and N. Ahuja. A geometric approach to train support vector machines. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 430–437. IEEE Computer Society, 2000.

[151] X. Yang, H. Fu, H. Zha, and J. Barlow. Semi-supervised nonlinear dimensionality reduction. In *Proceedings of the 23th International Conference on Machine Learning*, pages 1065–1072, New York, NY, USA, 2006. ACM.

[152] A. L. Yuille and A. Rangarajan. The concave-convex procedure (CCCP). In *Advances in Neural Information Processing Systems 14*, 2002.

[153] K. Zhang, I. W. Tsang, and J. T. Kwok. Maximum margin clustering made practical. In *Proceedings of the 24th International Conference on Machine Learning*, pages 1119–1126, 2007.

[154] T. Zhang and F. J. Oles. Text categorization based on regularized linear classification methods. *Information Retrieval*, 4:5–31, 2001.

[155] B. Zhao, F. Wang, and C. Zhang. CutS3VM: a fast semi-supervised svm algorithm. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 830–838, 2008.

[156] B. Zhao, F. Wang, and C. Zhang. Efficient maximum margin clustering via cutting plane algorithm. In *Proceedings of the SIAM International Conference on Data Mining*, pages 751–762, 2008.

[157] X. Zhu and A. B. Goldberg. *Introduction to Semi-Supervised Learning*. Morgan and Claypool, 2009.