NEW SUPPORT VECTOR ALGORITHMS FOR MULTICATEGORICAL DATA APPLIED TO REAL-TIME OBJECT RECOGNITION

THÈSE N^{O} 2826 (2003)

PRÉSENTÉE À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

Institut des systèmes informatiques et multimédias

SECTION D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Silvio BORER

mathématicien diplômé de l'Université de Zurich
de nationalité suisse et originaire de Himmelried (SO)

acceptée sur proposition du jury:

Prof. W. Gerstner, Prof S. Süsstrunk, directeurs de thèse
Dr P.-Y. Burgi, rapporteur
Prof. P. Fua, rapporteur
Prof. B. Schoelkopf, rapporteur

Lausanne, EPFL
2003

Summary

Humans have the ability to learn. Having seen an object we can recognise it later. We can do this because our nervous system uses an efficient and robust visual processing and capabilities to learn from sensory input. On the other hand, designing algorithms to learn from visual data is a difficult task.

More than fifty years ago, Rosenblatt proposed the perceptron algorithm. The perceptron learns from data examples a linear separation, which categorises the data in two classes. The algorithm served as a simple model of neuronal learning. Two further important ideas were added to the perceptron. First, to look for a maximal margin of separation. Second, to separate the data in a possibly high dimensional feature space, related nonlinearly to the initial space of the data, and allowing nonlinear separations. Important is that learning in the feature space can be performed implicitly and hence efficiently with the use of a kernel, a measure of similarity between two data points. The combination of these ideas led to the support vector machine, an efficient algorithm with high performance.

In this thesis, we design an algorithm to learn the categorisation of data into multiple classes. This algorithm is applied to a real–time vision task, the recognition of human faces. Our algorithm can be seen as a generalisation of the support vector machine to multiple classes. It is shown how the algorithm can be efficiently implemented. To avoid a large number of small but time consuming updates of the variables limited accuracy computations are used. We prove a bound on the accuracy needed to find a solution. The proof motivates the use of a heuristic, which further increases efficiency. We derive a second implementation using a stochastic gradient descent method. This implementation is appealing as it has a direct interpretation and can be used in an online setting.

Conceptually our approach differs from standard support vector approaches because examples can be rejected and are not necessarily attributed to one of the categories. This is natural in the context of a vision task. At any time, the sensory input can be something unseen before and hence cannot be recognised.

Our visual data are images acquired with the recently developed adaptive vision sensor from CSEM. The vision sensor has two important features. First, like the human retina, it is locally adaptive to light intensity. Hence, the sensor has a high dynamic range. Second, the image gradient is computed on the sensor chip and is thus available directly from the sensor in real time. The sensor output is time encoded. The information about a strong local contrast is transmitted first and the weakest contrast information at the end.

To recognise faces, possibly moving in front of the camera, the sensor images have to be processed in a robust way. Representing images to exhibit local invariances is a common yet unsolved problem in computer vision. We develop the following representation of the

sensor output. The image gradient information is decomposed into local histograms over contrast intensity. The histograms are local in position and direction of the gradient. Hence, the representation has local invariance properties to translation, rotation, and scaling. The histograms can be efficiently computed because the sensor output is already ordered with respect to the local contrast.

Our support vector approach for multicategorical data uses the local histogram features to learn the recognition of faces. As recognition is time consuming, a face detection stage is used beforehand. We learn the detection features in an unsupervised manner using a specially designed optimisation procedure. The combined system to detect and recognise faces of a small group of individuals is efficient, robust, and reliable.

Zusammenfassung

Der Mensch besitzt die Fähigkeit zu lernen. Er kann Objekte wiedererkennen, die er zu einem früheren Zeitpunkt gesehen hat. Der Ursprung dieser Fähigkeit liegt in einer effizienten und robusten Verarbeitung der visuellen Information in unserem Nervensystem. Einen Algorithmus zu entwerfen, der von visuellen Daten lernen kann, ist dagegen eine schwierige Aufgabe.

Vor über fünfzig Jahren entwickelte Rosenblatt den Perceptron Algorithmus. Das Perceptron lernt an Hand von Beispielen eine lineare Separation der Daten in zwei Klassen. Der Algorithmus diente als einfaches Modell des neuronalen Lernens. Zwei weitere wichtige Ideen wurden dazugefügt: Erstens sollten die gegebenen Beispiele maximal separiert werden. Zweitens versuchte man, die Beispiele in einem Merkmalsraum von möglicherweise grosser Dimension zu separieren, wobei die Daten nichtlinear in diesen Raum abgebildet werden. Dies ermöglicht ein nichtlineares Separieren der Daten. Wichtig ist dabei, dass man implizit, und deshalb effizient, im Merkmalsraum lernen kann. Dazu benutzt man einen Kern, ein Mass der Ähnlichkeit zwischen zwei Beispielen. Die Kombination dieser Ideen führte zur Support Vector Maschine, einem effizienten, leistungsfähigen Algorithmus.

In der vorliegenden Dissertation entwickeln wir einen Algorithmus, der an Hand von Beispielen lernt, Daten zu klassifizieren. Der Algorithmus wird verwendet, um Gesichter in Echtzeit wiederzuerkennen. Unser Algorithmus kann als Verallgemeinerung der Support Vector Maschine gesehen werden. Es wird gezeigt, wie er effizient implementiert werden kann: Um eine grosse Anzahl kleiner aber zeitaufwändiger Änderungen der Variablen zu verhindern, rechnen wir mit beschränkter Genauigkeit. Wir beweisen eine obere Schranke für die benötigte Genauigkeit, um eine Lösung zu finden. Der Beweis motiviert den Gebrauch einer bestimmten Heuristik, die die Effizienz weiter steigert. Eine zweite Implementierung benützt eine stochastische Gradientenmethode. Diese ist deshalb interessant, da sie eine direkte Interpretation besitzt und in einer online Situation benützt werden kann.

Im Unterschied zum üblichen Kontext fordern wir nicht, dass alle neuen Daten zu einer vorgegebenen Klasse gehören. Neue Daten können zurückgewiesen werden. Für die Anwendung zur Klassifizierung visueller Daten ist das eine natürliche Anforderung. Denn zu jedem Zeitpunkt können neue, zuvor ungesehene sensorielle Daten ankommen, die dann nicht wiedererkannt werden.

Unsere visuellen Daten bestehen aus Bildern, aufgenommen mit einer neu entwickelten Kamera von CSEM. Das Herzstück der Kamera ist ein adaptiver Sensor, der sich durch die folgenden Eigenschaften von herkömmlichen Sensoren unterscheidet: Erstens adaptiert sich der Sensor lokal an die gegebenen Lichtverhältnisse und kann deswegen mit lokal sehr unterschiedlichen Lichtintensitäten umgehen. Zweitens wird der Gradient der

Intensität des Lichtes, das auf den Sensor fällt, direkt im Sensorchip berechnet. Deswegen ist der Gradient in Echtzeit verfügbar. Diese Information wird zeitlich codiert. Der Sensor übermittelt erst die Information von starken und danach diejenige von schwachen Kontrasten.

Damit Gesichter, die sich möglicherweise vor der Kamera bewegen, wiederekannt werden können, werden die Bilder entsprechend verarbeitet. Bilder so darzustellen, dass sie lokal invariant sind, ist ein bekanntes Problem im Gebiet der Computer Vision. Wir entwickeln die folgende Darstellung. Der Gradient der Bilder wird in lokale Histogramme zerlegt. Es wird gezeigt, dass daraus die Invarianz bezüglich kleinen Translationen, Rotationen und Massstabsänderungen folgt. Die Histogramme bezüglich dem Betrag des Gradienten können effizient berechnet werden, da der Sensor die Gradienteninformation in geordneter Reihenfolge ausgibt.

Unser Algorithmus benützt die lokalen Histogramme als Merkmale. Er lernt mit Hilfe einer Menge von Bildern der Gesichter einer kleinen Gruppe von Personen, diese wiederzuerkennen. Da das Erkennen rechenaufwendig ist, wird zuerst eine Detektionsstufe benutzt. Ein speziell entwickeltes Optimierungsproblem wird gelöst, um die Gesichter zu detektieren. Das ganze System zur Detektion und Wiedererkennung von Gesichtern ist robust, effizient und zuverlässig.

Contents

1 Introduction **1**
1.1 Organisation of the thesis 1

2 Reproducing kernel Hilbert spaces **3**
2.1 Reproducing kernel Hilbert spaces 3
2.2 Constructing kernels 5
2.3 The metric induced by a kernel 6
2.4 Application: Maximal margin classification 7
2.5 Discussion 9

3 Regularisation and Statistical learning **11**

4 Support Vector Representation of Multi–categorical Data **15**
4.1 Introduction 15
4.2 The optimisation problem 16
4.3 Discussion 26
4.4 Details and Proofs ∗ 26

5 Efficient optimisation of the multiclass problem **29**
5.1 The algorithm 29
5.2 Performance test 35
5.3 Discussion 39

6 Simple optimisation of the multiclass problem **41**
6.1 Introduction 41
6.2 Learning rule 42
6.3 Experiments 44
6.4 Discussion 45

7 Unsupervised Learning **47**
7.1 Introduction 47
7.2 Modifying the cost function 50
7.3 Modifying the dual problem 52
7.4 Conclusion 58
7.5 Details and Proofs ∗ 58

8 Locally invariant image representation **61**
8.1 The CSEM adaptive vision sensor 61
8.2 Decomposition into local parts 65
8.3 Local histogram 66
8.4 Discussion and links to biology 69

9 Recognition of faces **71**
9.1 Introduction 71
9.2 The database of faces 74
9.3 The detection kernel 76
9.4 Learning multiple features 77
9.5 Learning a hierarchy 80
9.6 Detection results 81
9.7 The recognition kernel 83
9.8 Recognition results 85
9.9 Discussion 87

10 Conclusion **89**
10.1 Main results 89
10.2 Perspectives 90

Acknowledgements

Many people have contributed to making my research work both fruitful and enjoyable. First of all Dr. Friedrich Heitger form the CSEM, who made this research possible. My thesis adviser, Prof. Wulfram Gerstner, steered me towards a wonderful research topic and has been patient, kind, and a fount of wisdom during the past three and a half years. Special thanks to my coadviser Prof. Sabine Süsstrunk for the help during valuable discussions and for all the motivation in moments of slow progress. The collaboration with Dr. Pierre-Yves Burgi and Dr. Friedrich Heitger from the CSEM was very productive and opened my eyes for the real vision world.

I also wish to thank Prof. Martin Hasler for presiding the thesis committee, and Dr. Pierre-Yves Burgi, Prof. Pascal Fua, and Prof. Bernhard Schölkopf for eminently filling out its ranks.

I am indebted to my colleagues past and present at LCN, former MANTRA, LAMI, and LAP. Hearty thanks to my different office mates and colleagues playing the PhD game with me. Special thanks to all members of the coffee circle Alix, Angelo, Fabrizio, Pascal, Ricardo, Thomas and Werner. It is surprising how many creative ideas emerge when sitting around some cups of coffee. Thanks to Edgar for importing tasteful Cuban coffee. Special thanks to Thomas and Pierre-Edouard for making the computer more my friend than my enemy.

Particular thanks to our secretaries Brigitte Ramuz, Monique Dubois, and Marie-Jo Pellaud, the powers behind the thrones, whose agreeable competence makes life in the lab not only possible but smooth. I wish to thank my family and friends for their longstanding patience and encouragement.

Notation

When we define variables or functions $v_1, v_2, ...v_m$ we will often talk of the variables v_i, which means all the variables $v_i, i = 1, .., m$, but for convenience we suppress the values of the index if it is clear form the context which set is meant.

We will define algorithms with respect to example data $\{x_1, ..., x_l\}$. Mathematically correct would be to denote by $X_1, ..., X_l$ random variables to define the algorithm and then to denote samples by $x_1, ..., x_l$ in experiments. Again, for ease of notation, we will use the same symbols $x_1, ..., x_l$.

The following symbols will be used in the thesis. If not otherwise stated, their meaning is as indicated:

X	a nonempty set
x, y, x_i	elements of X
S	a training set
l	the size of the trainings set
Y	a set of labels
c, c_i	labels, elements of Y
m	the number of labels in Y
k	a kernel
H	a Hilbert space
$\cdot, <\cdot, \cdot>_H$	scalar products
ϕ	a feature map
$[\cdot]_+$	the positive part function, it is the identity if the argument is positive, otherwise zero
$[\cdot]_-$	the negative part function, it is zero if the argument is positive, otherwise minus the argument
$\lceil \cdot \rceil$	the smallest integer larger or equal to the argument
$\lfloor \cdot \rfloor$	the largest integer smaller or equal to the argument

Chapter 1

Introduction

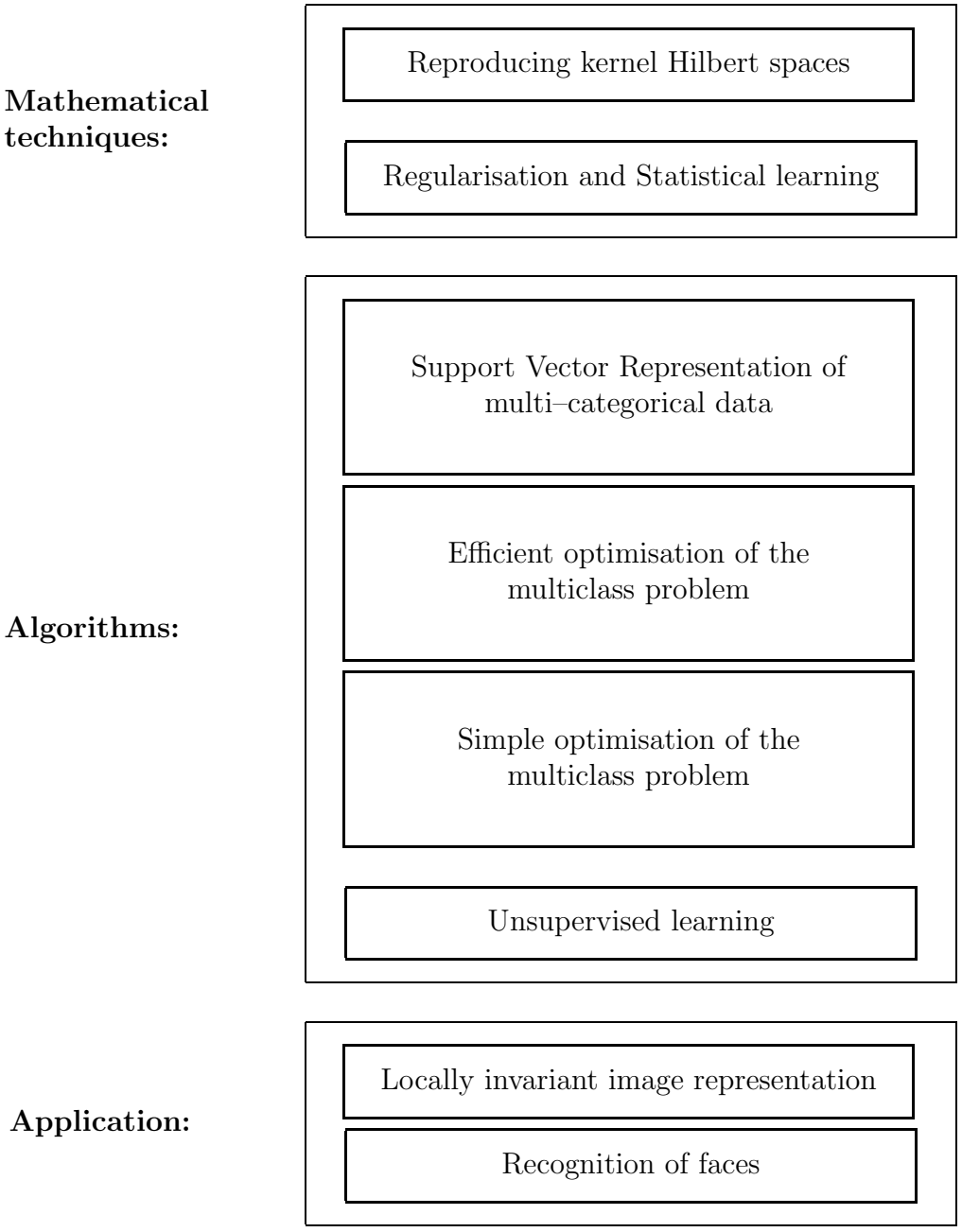
Learning from data instead of designing by hand is an appealing methodology. *Categorisation of data* is a classical application of this learning strategy. The goal is to learn from a set of labelled data, a representation of the classes. We are interested in *support vector–type learning algorithms*. During the learning phase, these algorithms select from a large set of examples a characteristic subset and estimate an elaborate combination thereof, which then serves to define a decision function and to possibly classify new examples. In this thesis we develop and apply support vector–type algorithms. Our algorithms are then applied to *object recognition*. Our objects are the *faces* of a small number of individuals.

The key question we ask in this thesis is *how to build efficiently support vector representations of multiple objects*. In addition to obvious demands like efficiency and robustness, there are practical requirements. In a vision application, sampling in a representative manner the whole space of visual input is difficult. Hence, learning should be done from positive examples, that is, faces only.

1.1 Organisation of the thesis

The thesis is organised as shown on page 2. In chapters 2 and 3 we start by reviewing important mathematical techniques used throughout the thesis. In chapter 2, we discuss Reproducing Kernel Hilbert spaces. These spaces serve as our feature spaces. We look at the structure induced by a kernel and show an application of how geometrical considerations can be used to increase the performance of support vector machines. Chapter 3 is a brief overview of regularisation and statistical learning theory to motivate the development of support vector type algorithms. After that, the chapters 4 to 7 are devoted to the development and analysis of algorithms and implementations. These are algorithms closely related to the well–known support vector machines. In chapter 4, we designed a new optimisation problem for the categorisation of data into multiple classes. In chapter 5, we propose an efficient implementation to solve the optimisation problem. Chapter 6 is devoted to an online learning rule, and in chapter 7 we use a modified version of our cost function for unsupervised learning.

Chapters 8 and 9 present an application. We use our algorithms to recognise faces. For this, we develop a suitable representation of images acquired with the CSEM vision sensor. In chapter 9 we learn to detect and recognise faces. Finally, there is a conclusion, where we review the main results achieved in this thesis and point out future developments.


Mathematical techniques:
Reproducing kernel Hilbert spaces
Regularisation and Statistical learning
Algorithms:
Support Vector Representation of multi–categorical data
Efficient optimisation of the multiclass problem
Simple optimisation of the multiclass problem
Unsupervised learning
Application:
Locally invariant image representation
Recognition of faces

Chapter 2

Reproducing kernel Hilbert spaces

In this chapter, we review mathematical concepts and notations that will be central for the thesis. That is why we have chosen a more formal style for the chapter. In section 2.4 we use these concepts for optimising the decision hyperplane in a normalised feature space.

Throughout this thesis we assume the following situation. We are given data $x_1, ..., x_l$ and possibly some additional information. For example, the data points could be labelled, that is, for each example x_i we are given a label c_i in form of a positive integer. We assume for the moment that these data points belong to $\mathbb{R}^n$ for a positive integer n. The goal is to estimate one or multiple functions f of the form

$$f = \sum_i \alpha_i <x_i, \cdot> + \rho, \tag{2.1}$$

with real coefficients α_i and a margin $\rho \in \mathbb{R}$. Here $<\cdot, \cdot>$ denotes the scalar product of $\mathbb{R}^n$. These functions f represent the data, in a sense to be defined later. To allow for more general representations, we will replace the scalar product $<\cdot, \cdot>$ through a kernel, a symmetric real–valued function, which can be nonlinear in each argument. Then, we are interested in estimating functions of the form

$$f = \sum_i \alpha_i k(x_i, \cdot) + \rho, \tag{2.2}$$

for the kernel k. The kernel k can be interpreted as a measure of similarity. If $k(x, y)$ takes a large value, then x and y are considered to be similar. On the other hand, if $k(x, y)$ has a small value, x and y are considered to be dissimilar. In the rest of this chapter, these concepts will become clearer. In section 2.1 we define a class of kernels, the positive definite kernels, and show the connections between (2.1) and (2.2). In section 2.2 we describe how kernels can be constructed and in section 2.3 we indicate the relation of a kernel and the metric properties of the underlying space.

2.1 Reproducing kernel Hilbert spaces

In this section we present the theory of reproducing kernels, which goes back to Aronszajn [5]. It was introduced to machine learning in the sixties [1], and was used extensively in other areas as approximation and regularisation theory [76] or signal processing [73]. More recently, the potential of the approach for machine learning has been recognised. Today, it can be found in textbooks on machine learning [23, 63].

Let X be a nonempty set, which contains our data $x_1, ..., x_l$ and potentially many other points. For example, X could be the $\mathbb{R}^n$.

Definition 1 *Given a function $k : X \times X \to \mathbb{R}$ and points $x_1, ..., x_l \in X$, then the $l \times l$ matrix K with elements*

$$K_{ij} = k(x_i, x_j) \tag{2.3}$$

is called the **Gram matrix** *or* **kernel matrix** *of k with respect to $x_1, ..., x_l$.*

If we think of a kernel as a measure of similarity, the kernel matrix consists of the pairwise similarity of the data points $x_1, ..., x_l$.

Definition 2 *A symmetric matrix $K \in \mathbb{R}^{l \times l}$ is called* **positive definite** *if*

$$\sum_{i,j} c_i c_j K_{ij} \geq 0, \tag{2.4}$$

for all $c_i, c_j \in \mathbb{R}$.

Now we can define the class of kernels we are interested in.

Definition 3 *A symmetric function $k : X \times X \to \mathbb{R}$ which for any m points $x_1, ..., x_l \in X$ gives rise to a positive definite Gram matrix is called a* **positive definite kernel**.

For ease of notation in the remaining chapters, we will often talk of a kernel, but if not otherwise stated, a positive definite kernel is meant.

Let now k be a positive definite kernel on the set X. The idea to relate (2.1) to (2.2) is the following. We want to construct a Hilbert space H of features, together with a mapping $\phi : X \to H$, such that for points $x, y \in X$ we have

$$<\phi(x), \phi(y)>_H = k(x, y), \tag{2.5}$$

where $<\cdot, \cdot>_H$ denotes the scalar product in H. In words, the similarity of two points measured with the kernel is the scalar product of the two points mapped into the Hilbert space H. Then, estimating a function of the form (2.2) in X is equivalent to estimating a function of the form (2.1) in H, with the x_i replaced by $\phi(x_i)$. To construct the desired Hilbert space, we set H' to be the vector space of all linear combinations of the form

$$\sum_{i=1}^{m} \alpha_i k(x_i, \cdot), \tag{2.6}$$

for $m \in \mathbb{N}$, points $x_1, ..., x_m \in X$, and $\alpha_i \in \mathbb{R}$. For

$$f = \sum_{i=1}^{l} \alpha_i k(x_i, \cdot)$$

and another function with potentially different points $y_1, ..., y_m$,

$$g = \sum_{j=1}^{n} \beta_j k(y_j, \cdot)$$

we define

$$<f, g>_{H'} = \sum_{i=1}^{l} \sum_{j=1}^{n} \alpha_i \beta_j k(x_i, y_j). \tag{2.7}$$

Then, it is straightforward to verify that $<\cdot, \cdot>_{H'}$ is an inner product on H'. This inner product defines a metric d on H' by

$$d(f, g) = (<f - g, f - g>_{H'})^{1/2} \tag{2.8}$$

and we let H be the completion of H' with respect to this metric. Then H together with the scalar product $<\cdot, \cdot>_H$ defined with (2.7) is a Hilbert space. For $f \in H$ we find that k is the *representer of evaluation*,

$$<f, k(x, \cdot)>_H = f(x) \tag{2.9}$$

and especially

$$<k(x, \cdot), k(y, \cdot)>_H = k(x, y). \tag{2.10}$$

Because of these properties, k is called a *reproducing kernel* and H a *reproducing kernel Hilbert space*, RKHS for short. Later, if it is clear from the context which scalar product is meant, we write $<\cdot, \cdot>$ or just a $\cdot$ for short. Furthermore, we define the feature map $\phi : X \to H$ by

$$\phi(x) = k(x, \cdot). \tag{2.11}$$

With (2.10) we find

$$<\phi(x), \phi(y)>_H = k(x, y), \tag{2.12}$$

as desired.

2.2 Constructing kernels

Let us take a look at how we can construct new kernels from known kernels. Up to now, we know that inner products in Hilbert spaces are kernels. The next proposition will show several methods to construct new kernels.

Proposition 4 *Let k_1 and k_2 be kernels over $X \times X$, with X a nonempty set, $a \in \mathbb{R}^+$, f a real–valued function on X, $\phi : X \to H$, and k_3 a kernel over H. Then the following functions are kernels:*

1. $k(x, y) = k_1(x, y) + k_2(x, y)$,
2. $k(x, y) = a k_1(x, y)$,

3. $k(x,y) = f(x)f(y)$,

4. $k(x,y) = k_3(\phi(x), \phi(y))$,

5. $k(x,y) = \exp(k_1(x,y))$,

6. $k(x,y) = \exp(-||x-y||^2/a)$,

where in 6. the set X is a subset of $\mathbb{R}^n$.

The proof can be found in [23]. The kernel in 6. is called *Gaussian kernel*, and we will use it in combination with these construction techniques later on to build appropriate kernels for face detection and recognition. Note that these and other techniques to construct kernels can be found in many papers, for example [37] and [77] constructed kernels on discrete structures such as text. Many other methods can be found in [63]. We will see in chapter 9 examples of how to construct kernels for computer vision applications.

2.3 The metric induced by a kernel

An advantage of kernel algorithms is, that the only structure needed is the kernel itself. From data points $\{x_1, ..., x_l\} \subset X$, where X is any nonempty set, we can learn properties of the data with a kernel algorithm if we find a suitable kernel on $X \times X$. In fact, apart from the kernel, no other structure in X is needed. This is contrary to other statistical methods. For example, an underlying vector space structure is needed to estimate the mean of a set of samples. On the other hand, if the kernel induces all the needed structure, it is no surprise that it is by no means a trivial problem to choose a suitable kernel for a given problem.

Let us have a closer look at structures induced by the kernel. To do this, let X be a nonempty set, $k : X \times X \to H$ a kernel, and let us denote by ϕ the feature map from (2.11). Then, the kernel k defines a metric d_X on X by

$$d_X(x,y) = d(\phi(x), \phi(y)) = \big(k(x,x) + k(y,y) - 2k(x,y)\big)^{1/2}. \tag{2.13}$$

For the second equality sign we have used (2.8), that is,

$$\begin{aligned} d(\phi(x), \phi(y)) &= \big(<\phi(x) - \phi(y), \phi(x) - \phi(y)>_H\big)^{1/2} \\ &= \big(<\phi(x), \phi(x)>_H + <\phi(y), \phi(y)>_H - 2 <\phi(x), \phi(y)>_H\big)^{1/2}, \end{aligned} \tag{2.14}$$

together with (2.12). The metric d_X is called the pull–back of the metric d from (2.8) by ϕ. If the feature map is "sufficiently nice" such that $\phi(X)$ is a submanifold of H, then there is another natural metric on $\phi(X)$, namely the intrinsic metric, where distances are measured along the submanifold. Again, this metric can be pulled back to a metric g on X. To discuss this, let us suppress the technical details. We refer the reader to [70] for the geometric framework. We suppose in addition, that our data lives in $X \subset \mathbb{R}^n$ and the feature mapping $\phi : X \to H$ maps the data to the submanifold $\phi(X) \subset H$. The scalar

product in H defines a Riemannian metric in the submanifold $\phi(X)$. This Riemannian metric can be pulled back to a new metric g in the input space given by

$$g_x(v, w) = < D_x\phi(v), D_x\phi(w) >_H, \tag{2.15}$$

for $x \in X$. Here D denotes the derivative. Note that measuring the distance of two points $x, y \in X$ with g is equivalent to measuring the distance of the points $\phi(x), \phi(y)$ in the feature space along the submanifold. This geometric interpretation has been used to analyse and improve the performance of support vector machines in [16, 4].

2.4 Application: Maximal margin classification in a normalised feature space

After this review of mathematical concepts, we use a kernel induced metric for the optimisation of the decision hyperplane in a normalised feature space. This was a first, short part of my thesis research and led to a publication [32]. The idea is to normalise the data and use the metric g from (2.15) to readjust the decision function of support vector machines to improve their performance. In more detail, suppose our data is in $X \subset \mathbb{R}^n$. The feature map is the projection on the unit sphere,

$$\phi_n : x \to x/||x||, \tag{2.16}$$

which normalises the data. Hence the distance of two points x and y measured by g is the distance of $\phi_n(x)$ and $\phi_n(y)$ measured along the unit sphere, which is the angle between the two vectors x and y, see figure 2.1.A. Suppose we are given data examples belonging to two different classes. We use this data to train a support vector machine on the normalised data. As solution, we find two parallel margin hyperplanes, separating the data of the two classes maximally apart. Then, we set the decision hyperplane to be in the middle of the margins measured along the surface. Hence, the hyperplane is sitting at half the angle, see figure 2.1.B. In contrast to this, a standard support vector machine puts the decision hyperplane at half the distance of the two margin hyperplanes measured along a line orthogonal to the planes.

As a second step, the normalisation can be done on the data mapped in the feature space H. Let $\phi : X \to H$ be a feature map, which maps the data from X into the Hilbert space H. We normalise the data mapped into H. Our combined feature map is then

$$\phi_n \circ \phi : X \to S_H, \quad x \mapsto \frac{\phi(x)}{||\phi(x)||_H}, \tag{2.17}$$

where S_H denotes the unit sphere in H. If we chose two points $x, y \in X$, the scalar product of the mapped points $(\phi_n \circ \phi)(x)$ and $(\phi_n \circ \phi)(y)$ can be calculated by

$$\frac{k(x, y)}{\sqrt{k(x, x)k(y, y)}}. \tag{2.18}$$

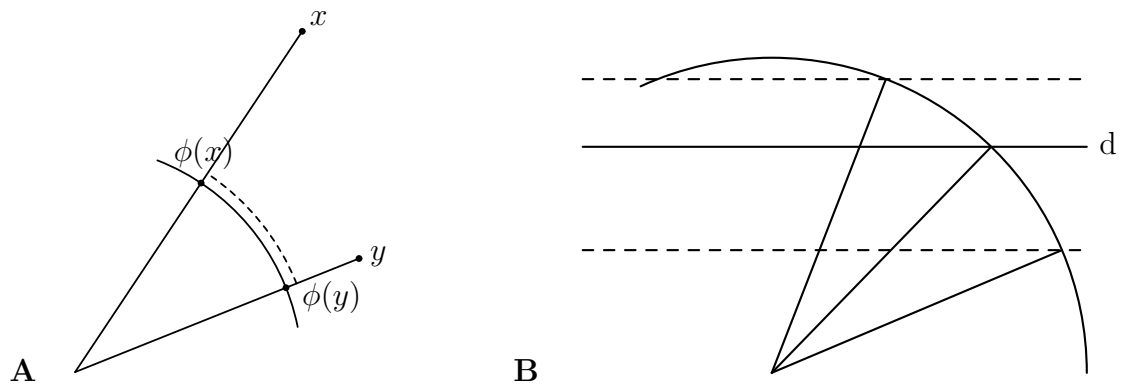


Figure 2.1: Readjusting the separating hyperplane: In **A** the distance between x and y using the metric g is measured along the unit hyper sphere (dotted line) and is equal to the angle between the vectors. In **B** the decision hyperplane d is readjusted to be at half angle between the margin hyperplanes (dotted lines). Note that now the distance of the decision hyperplane to the two margins, measured along an orthogonal vector, is not the same.

Here, we used the kernel $k(x, y) = <\phi(x), \phi(y)>_H$. Therefore, we can normalise the data implicitly in feature space.

The margin hyperplanes are learnt using a support vector machine with kernel (2.18), thus separating the data, which sits on the unit sphere S_H. Let us denote by (w, b) the solution of the support vector machine. Hence, the margin hyperplanes are given by

$$\{f \in H | <w, f>_H + b \pm 1 = 0\}. \tag{2.19}$$

Let us denote by m_+ and m_- the distance from the margin hyperplanes to the origin,

$$m_\pm = \frac{b \pm 1}{||w||_H}. \tag{2.20}$$

A short calculation shows, that the decision hyperplane, sitting in between the two margin hyperplanes, measured along the sphere S_H is

$$\{f \in H | <w, f>_H + ||w|| \bar{b} = 0\}. \tag{2.21}$$

Here, $\bar{b}$ is given by

$$\bar{b} = \cos\Big(\frac{1}{2}\big(\arccos(m_+) + \arccos(m_-)\big)\Big). \tag{2.22}$$

One can see that the decision hyperplane can be calculated using the kernel evaluations only, without explicitly calculating points $\phi(x)$ in feature space.

Let us make a remark concerning standard choices of kernels. For the Gaussian kernel (6) the data is automatically normalised in feature space, as $k(x, x) = 1$ for all x. Thus,

when using a Gaussian kernel, the decision hyperplane can be calculated as described above.

For a homogeneous polynomial kernel, a kernel of the form $k(x, y) = (< x, y >)^p$, normalisation in feature space is equivalent to the normalisation in the space X. Hence, for a homogeneous polynomial kernel, the above procedure makes sense even if the data is normalised in the initial space X.

In [32] it was shown that in an object recognition task, normalising the data in feature space and setting the decision hyperplane as described above, increases the performance of support vector machines.

2.5 Discussion

In this chapter we reviewed how the kernel induces geometrical structures. We proposed a method to use this geometrical structure to optimise the decision hyperplane of a support vector machine in a normalised feature space to increase performance.

Chapter 3

Regularisation and Statistical learning

To learn to categorise data can be a difficult task, as the determination of a suitable optimisation problem is non–trivial. Not only should one estimate a representation of the data that generalises well, but it should be possible to solve the corresponding optimisation problem efficiently. In this chapter we review very briefly two theoretical branches, regularisation and statistical learning theory, which can help to determine suitable optimisation problems and determine their properties. We start by introducing the basic concepts of loss and risk. We show how this is connected to regularisation theory and finally, how we can use results from statistical learning theory to motivate the algorithms developed in the following chapters. More details can be found in the books [74, 63, 23].

In classification, the general setting as follows. Given a set X, the set of all possible data examples, and a finite set $Y = \{1, ..., m\}$ of labels. Furthermore the set $X \times Y$ is equipped with a probability measure P. From P we can derive conditional distributions, for example, $P(\cdot|x)$. Intuitively, for $c \in Y$ we interpret $P(c|x)$ as the probability that a data example x belongs to class c. Our goal is to predict, for a new point $(x, c) \in X \times Y$, the class label c by knowing x. We do this with a classification rule. A *classification rule*, or *classifier* for short, is a function $f : X \to Y$, assigning to each x a class label c. To estimate the classification rule, we are given a set of labelled data

$$S = \{(x_1, c_1), ..., (x_l, c_l)\}, \tag{3.1}$$

the *training data*. The set S consists of independent samples drawn from X according to P. To quantify the quality of the classification rule, we specify a loss function L, assigning a real value $L(x, c, f(x))$ to each classifier f and labelled example (x, c). A classifier with a small value of the loss function is a good predictor of the labels.

As example, we take a look at binary classification. Here, $Y = \{0, 1\}$ and a standard loss function is the $0 - 1$–loss, counting the number of misclassifications,

$$L(x, c, f(x)) = \begin{cases} 0 & \text{if} f(x) = c \\ 1 & \text{otherwise.} \end{cases} \tag{3.2}$$

For a *test set* $\{(x'_1, c'_1), ..., (x'_{l'}, c'_{l'})\}$ the *test error* or risk on the test set is the expected loss on the test set,

$$R_{test}(f) = \frac{1}{l'} \sum_{i=1}^{l'} \sum_{c \in Y} L(x'_i, c, f(x'_i)) P(c|x'_i). \tag{3.3}$$

If the test set is known, this is the quantity we are interested in minimising. In other words, we try to find a strategy to construct from the training sample a classifier f minimising $R_{test}(f)$. If we have no knowledge about the test set, we should minimise the *expected loss* over all possible test patterns. Thus, we should minimise the *risk*

$$R(f) = \int_{X \times Y} L(x, c, f(x)) dP(x, c). \tag{3.4}$$

In general, direct minimisation of the risk is not a tractable strategy. For a training set S we estimate the risk by the *empirical risk* of f given by

$$R_{emp}(f) = \frac{1}{l} \sum_{i=1}^{l} L(x_i, c_i, f(x_i)), \tag{3.5}$$

which is the risk of the classifier f with respect to the empirical probability measure. As is well known, minimisation of the empirical risk can be critical, if no measures against overfitting are taken.

To illustrate this, we look at a classical example from binary classification, see (3.2). A minimiser of the empirical risk is given by the function

$$f(x) = \begin{cases} c_i & \text{if } x = x_i \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

but in general this is not a good prediction rule.

To avoid overfitting, one can choose a suitable restricted set of classifiers $\mathcal{F}$, and minimisation of the empirical risk is performed over the set $\mathcal{F}$. From a computational point of view it is often more efficient to consider a larger class of functions, but to restrict the minimum to a restricted set by adding a regularisation term $\Omega : \mathcal{F} \to \mathbb{R}$ to the optimisation. In summary, the optimisation problem is

$$\min_{f \in \mathcal{F}} \quad \Omega(f) + R_{emp}(f). \tag{3.7}$$

We call $\Omega + R_{emp}$ a *regularised risk functional*. Later, we discuss various optimisation problems. They are all of the general form of (3.7). The question of how to choose a good regularisation term is not an easy one. The regularisation terms we use are motivated mainly by the geometric interpretation of our optimisation problems.

The following questions arise naturally. Which regularisers should be chosen ? How good is the approximation of the minimum of the risk by the minimum of the regularised risk functional. In statistical learning theory, one tries to bound the risk in terms of the empirical risk and a dependency of a measure of capacity of the set of functions $\mathcal{F}$. These bounds are typically of the form

$$R(f) \leq R_{emp}(f) + C(\mathcal{F}), \tag{3.8}$$

and hold with probability $1 - \epsilon$. Here the constant ϵ can be chosen arbitrarily, but ϵ appears in the term $C(\mathcal{F})$ on the right hand side. As one can expect, measuring the

capacity of the set $\mathcal{F}$ is a nontrivial thing and different measures exist. These measures can serve as a motivation to choose a regularisation term. In machine learning, a very well known measure of the complexity of $\mathcal{F}$ is the VC–dimension [74]. Multiple other bounds can be found in [23, 63] and the references therein. Some of these bounds serve as a motivation for the maximal margin regulariser in the support vector machines. Below we discuss a specific measure together with the corresponding term C of (3.8). Even though it is designed for a slightly different situation, it serves as an illustration and motivation for the algorithm discussed in chapter 4.

The measure of capacity in our example is the cardinality of the set $\mathcal{F}$. This is very attractive, because in contrast to the VC–dimension, which can be very difficult to calculate or even estimate, cardinality is easily determined. The following result is of the form of inequality 3.8, where $C(\mathcal{F})$ depends essentially on the cardinality of $\mathcal{F}$.

Consider a binary classification problem with loss function (3.2). Let X be any nonempty set and $Y = \{0, 1\}$. We denote by $S = ((x_1, c_1), ..., (x_l, c_l))$ a labelled training set, and by $S' = ((x'_1, c'_1), ..., (x_l, c'_l))$ a test set. Both consist of independent, identically distributed samples from a probability measure P on $X \times Y$. We are allowed to use the trainings set S and the test points $x'_1, ..., x'_l$. The goal is to predict the test labels $c'_1, ..., c'_l$. The set of hypothesis is $\mathcal{F} = \{f_1, ..., f_M\}$, the set of possible decision functions. It can either be chosen in advance, without looking at the data, or it can be chosen in a data dependent way. But if we chose the set $\mathcal{F}$ in a data dependent way, we have to do it by looking only at the data points $x_1, ..., x_l, x'_1, ..., x'_l$. Furthermore, each f_i must depend in an *exchangeable* way on S and S', that is, by exchanging x_i with x'_i the function f_i should not change. For example, we can define a hypothesis depending on the mean of $x_1, ..., x_l, x'_1, ..., x'_l$. Another example is the following. For each pair x_i, x'_i we can define a hypothesis f_i in the following way. First, we sort the x_i, x'_i in lexicographic order and take for f_i the classifier responding 0 on all points closer to the first one and 1 on all the points closer to the second one.

We consider the x_i, x'_i as random variables on X. Hence, the empirical risk and the risk on the test set are random variables. The result of Catoni [18] states that with probability $1 - \epsilon$ the following bound holds

$$R_{test}(\hat{f}) \leq 3R_{emp}(\hat{f}) + 2\frac{1}{l}\log\frac{M}{\epsilon}, \tag{3.9}$$

with

$$\hat{f} = \arg\min_{f \in \mathcal{F}} R_{emp}(f). \tag{3.10}$$

In words, with probability $1 - \epsilon$, the risk on the test set is bounded by three times the empirical risk plus a term depending on the cardinality M of the set $\mathcal{F}$, the size l of the training and test set and on ϵ, the confidence, with which the bound holds. This is a so–called union bound [63], hence the dependency on M/ϵ. Even though the situation here is different from our situation in chapter 4, this result can serve us as motivation. We will see in chapter 5 that our classification algorithm uses finite accuracy calculations. We will choose finite accuracy computations for efficiency reasons. As a consequence, the

set of hypothesis $\mathcal{F}$ is finite. According to the above result, having a small cardinality can limit the capacity and thus gives a smaller risk.

Chapter 4

Support Vector Representation of Multi–categorical Data

In this chapter, we propose a new optimisation problem for the categorisation of data into multiple classes. Our approach can be viewed as a generalisation of the well known support vector machines to multiple classes and includes the two–class problem as a special case. For one class only it reduces to a single–class support vector machine, whose solution can be seen as an estimate of the support of a distribution. We discuss the properties of our strategy and relate it to other multi–class approaches. In chapters 5 and 6 we will propose efficient algorithms to solve the optimisation problem. The algorithm in chapter 5 is directed to find the global optimum, whereas the algorithm of chapter 6 uses a stochastic gradient descent method to find an approximate solution.

4.1 Introduction

Given a set of examples belonging to different categories or classes, the problem is to learn from these examples a decision function which assigns to new examples one of the classes or possibly rejects them. At present one of the best performing methods are the support vector machines [14]. They were originally designed for binary decision problems with two classes only. In the case of more than two categories, there exist many different strategies to combine several support vector machines. Examples are the combination of one–versus–rest trained machines or of pairwise trained ones for all pairs [45, 29], or of pairwise trained ones, using a directed acyclic graph [54]. Furthermore, one can use error correcting output codes [24, 3] to combine the output of several binary classifiers. From a conceptual point of view, it would be more satisfying to be able to train directly a classifier for multiple classes. Different generalisations of support vector machines have been proposed so far [74, 78, 15, 46, 33]. Unfortunately, these generalisations need to estimate for each training example at least m parameters simultaneously, where m is the number of classes. They are therefore in practical applications often too complex to be used. Another multi-categorical generalisation was proposed by Cramer and Singer [21]. As above, they estimate m parameters for each example, but the advantage of their formulation is that it allows a more efficient implementation.

We present a simple algorithm, which can be seen as a generalisation of the support vector machine to multi–categorical data. It was proposed in [12]. This algorithm needs to learn only one parameter for each training example, as explained in subsection 4.2.5. For only one class it reduces to a one-class support vector machine [62], which yields an estimate of the support of a distribution.

The idea is to represent each class c by a vector w_c, which can be expressed as a

linear combination of the data in *its* class only. Then a decision function for each class is defined in the following way: The value of the decision function of class c for a data point x is the scalar product of x with the representative vector w_c minus the scalar product with the average $\bar{w}_c$ of the other representatives, and this is compared to a learnt margin, see Figure 4.1.A–C. If it is larger than the margin, the data point x is accepted to be of this class, and otherwise not. To allow for nonlinear separations, we apply the well-known kernel trick reviewed in chapter 2, that is, we map the data in a reproducing kernel Hilbert space [49, 5] and perform the linear separations there. Then the preimages of these separations are possibly nonlinear separations in the space of the data. We will call the family of decision functions a representation of the multi–categorical data.

We want to point out two conceptual differences between our approach and the existing generalisations. First, we have a parameter that controls the influence of other classes on a given one. In the extreme, the representation of each class is learnt independently of the other classes, that is, our method reduces to support estimation [62]. Furthermore in our approach, we will not assume that each data point belongs exactly to one class. Data points may belong to several classes or to none at all. However, in the standard approaches mentioned at the beginning of this section [74, 78, 15, 46, 33], the goal is the assignment of each example to a single class. This is achieved in two steps. First, just as in our approach, there is a decision function for each class. The final decision, however, is made according to the decision function, whose value is *maximal* for a data point. Therefore, every point is assigned to exactly one class, except the rare event of having multiple decision functions giving exactly the same value to a data point.

4.2 The optimisation problem

We start this section by introducing some useful notation to describe the multi–class learning problem. Then, in subsection 4.2.2, we introduce a cost function, whose minimum will be the desired representation of the multi–categorical data. In subsection 4.2.3, we show how our cost function reduces for certain choices of parameters to other well–known cost functions. Especially, we will see that in the case of two classes only, our approach reduces to a support vector machine. Because there are already approaches to the generalisation of the binary support vector machine, we discuss their relation to our approach. In subsections 4.2.5 –4.2.7, the dual problem will be calculated, an equivalent optimisation problem of simpler form, and its properties will be analysed. We then conclude the section by illustrating the categorisation performance for a two–dimensional toy problem.

4.2.1 Representation of multi–categorical data

Given is a set X and a positive integer m. In this set, we are given subsets $S_c \subset X$ for $c = 1, .., m$, called classes or categories. Note that we make no assumption that the subsets are pairwise disjoint, even though this will often be the case. Later, we will simply refer to class c for the subset S_c. Normally, these subsets are unknown, but we are given

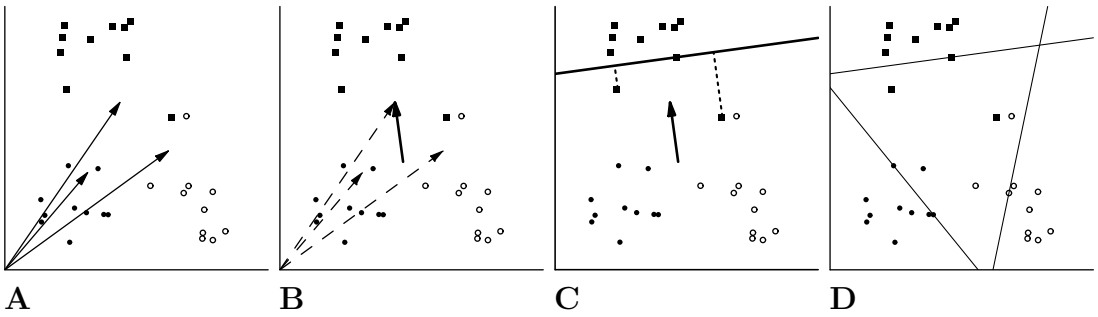


Figure 4.1: The idea of the decision functions and our cost function for three classes ∘, •, ■ and the parameter $\gamma = 1$. Plot **A** shows the vector w_c of each class. Plot **B** shows the vectors w_c for each class (dashed arrows), and $w_c - \bar{w}_c$ for the class ■ (solid arrow). In **C**, we see the vector $w_c - \bar{w}_c$ together with the separating hyperplane, which is the zero level of the function d_c, for the class ■. The length of the dashed lines are proportional to the empirical risk $R_{emp,c}$ for this class, see (4.2). In **D**, we see the separating hyperplanes for all three classes. Points inside the triangle spanned by the three lines are categorised as not belonging to any class.

examples of each subset, in the form of labelled training examples $(x_1, c_1), ..., (x_l, c_l)$. Here x_i is an element of the set X and the label c_i is an element of the set $\{1, ..., m\}$, the class to which x_i belongs, that is $x_i \in S_{c_i}$. Let us denote by l_c the number of training examples in class c, and, as before, by m the number of classes. The learning problem consists of finding a representation of the subsets $S_1, ..., S_m$, that is a family of real–valued functions $(d_1, ..., d_m)$, called decision functions, such that for a new point x the values of these functions at x give information about the class of x.

4.2.2 The cost function

Let $k : X \times X \to \mathbb{R}$ be a positive definite kernel and H the associated reproducing kernel Hilbert space (RKHS) [5]. We recall that, in particular, $k(x, \cdot) \in H$ and that the inner product is defined by linear continuation of the fundamental equation $k(x, \cdot) \cdot k(y, \cdot) = k(x, y)$, where $\cdot$ denotes the scalar product in H. We denote by $\phi : X \to H$ the mapping $x \mapsto k(x, \cdot)$, therefore $\phi(x) \cdot \phi(y) = k(x, y)$.

Let us now apply these concepts to our multi–class problem. For each class c we seek a vector $w_c \in H$ and a real value ρ_c determining a decision function of the form

$$d_c(x) = (w_c - \gamma \bar{w}_c) \cdot \phi(x) - \rho_c, \tag{4.1}$$

where $\bar{w}_c = \frac{1}{m-1} \sum_{d \neq c} w_d$. Note that for $m = 1$ the sum is indexed by an empty set and therefore $\bar{w}_c$ is defined to be zero. The parameter $\gamma \in [0, 1]$ lets us adjust the influence of the other classes relative to class c. The sign of $d_c(x)$ decides if example x is accepted to be an element of class S_c or is rejected, that is, is not accepted to be of class S_c, see Figure 4.1.A, B. We will use the shorthand notations $d = (d_1, ..., d_m)$ for the m decision functions, $w = (w_1, ..., w_m)$ for the m representative vectors, $\rho = (\rho_1, ..., \rho_m)$ for the margins, and

so on. For the training examples $\{(x_j, c)|x_j \in S_c\}$ of class c and a parameter $\nu \in (0, 1]$, the empirical risk of d_c is

$$R_{emp,c}(d_c) = \frac{1}{\nu l_c} \sum_{x_i \in S_c} [d_c(x_i)]_-, \tag{4.2}$$

where the negative part $[\cdot]_-$ maps a real argument $x < 0$ to $-x$ and $x \geq 0$ to zero, see Figure 4.1.C. The constant $1/\nu$ was chosen because ν has a direct interpretation, see subsection 4.2.6.

For a parameter $\gamma \in [0, 1]$ we define the regulariser

$$\Omega(d) = \sum_c \big(\frac{1}{2}||w_c||^2 - \frac{1}{2}\gamma w_c \cdot \bar{w}_c - \rho_c\big), \tag{4.3}$$

where $d = (d_1, ..., d_m)$ is defined in (4.1). Intuitively, the regulariser is small if the vectors w_c all have a small norm, and the margins ρ_c are large, which tries to separate one class as much as possible from the origin. This is as in the single class support vector machine. The term $w_c \cdot \bar{w}_c$ is specific to our approach, and later we will see that this term in the regulariser puts more weight on the discriminative examples, see Figure 4.4.B-D. In summary, we arrive at the optimisation problem

$$\text{minimise} \quad \Omega(d_1, ..., d_m) + \sum_{c=1}^{m} R_{emp,c}(d_c). \tag{4.4}$$

We are faced with the problem of finding m vectors $w_1, ..., w_m$ in the possibly very high dimensional feature space together with the margins $\rho_1, ..., \rho_m$. A priori, this is a very difficult problem, but we will show next that each of the vectors w_c can be expressed as a linear combination of the training data mapped into the feature space. This reduces the problem of finding the optimal $w_1, ..., w_m$ to determining their expansion coefficients.

Lemma 1 (Representer Theorem) *Let $d = (d_1, ..., d_m)$ be a tuple minimising the regularised risk functional*

$$\sum_{c=1}^{m} R_{emp,c}(d_c) + \Omega(d). \tag{4.5}$$

with d_c given by (4.1). Then each w_c admits a representation of the form

$$w_c = \sum_{i=1}^{l} \lambda_{c,i} \phi(x_i), \qquad c = 1, .., m, \tag{4.6}$$

for some real coefficients $\lambda_{c,i}$.

The proof can be found in section 4.4 at the end of this chapter. The idea of the lemma goes back to Kimeldorf and Wahba [42]. Our formulation is close in spirit to the form of Schölkopf and Smola [63] but differs from the above in that we look for multiple

elements in H. Note that from lemma 1 we can not conclude that the vectors w_c are expressible in terms of the data of their class only. But later we will see that this is the case. In contrast to the above lemma, which has counterparts for a wide class of objective functionals, the next lemma is more specific.

Lemma 2 *The function* $\Omega(d_1, ..., d_m) + \sum_{c=1}^{m} R_{emp,c}(d_c)$ *is bounded from below and convex.*

Again, the proof can be found in the appendix.

For the discussion in the remainder of the chapter, it is useful to rewrite the optimisation problem (4.4) as a constrained optimisation problem

$$\begin{aligned} \text{minimise} \quad & \Omega(d_1, ..., d_m) + \sum_{i=1}^{l} \frac{1}{\nu l_{c_i}} \xi_i \\ \text{subject to} \quad & (w_{c_i} - \gamma \bar{w}_{c_i}) \cdot \phi(x_i) \geq \rho_{c_i} - \xi_i \\ & \xi_i \geq 0 \ . \end{aligned} \tag{4.7}$$

The constraints tell us that if a training example is not assigned to its class by (4.1), then the corresponding ξ_i is positive and this is penalised in the cost function. In addition, the cost function favours vectors w_c of small norm and large values of ρ_c. With γ close to 1, the second term in the regulariser Ω moves the vectors w_c towards the most discriminant examples. We mention that for $\gamma = 0$ we have m independent problems. With the parameter ν the number of outliers can be controlled, see subsection 4.2.6.

By lemma 2, instead of optimising (4.7) we can equivalently optimise the dual problem [10], as indicated in section 4.2.5. Before we do so, let us discuss the relation of our approach to other well known paradigms.

4.2.3 Special Cases

Let us note the following special cases.

Estimating the Support of a Distribution:

If we have only one class, $m = 1$, problem (4.7) reduces to

$$\begin{aligned} \text{minimise} \quad & \frac{1}{2}||w||^2 - \rho + \frac{1}{\nu l} \sum_i \xi_i \\ \text{subject to} \quad & w \cdot \phi(x_i) \geq \rho - \xi_i, \ \ \xi_i \geq 0 \ , \end{aligned} \tag{4.8}$$

which tries to separate the data in feature space from zero [62].

Support Vector Machine:

In the case of two classes and $\gamma = 1$, we define $w = w_1 - w_2$, $b = \frac{1}{2}(\rho_1 - \rho_2)$ and $\rho = \frac{1}{2}(\rho_1 + \rho_2)$. Furthermore, we define labels $y_i = 1$ if the example x_i is in the first class, and $y_i = -1$ otherwise. Then (4.7) reduces to

$$\begin{aligned} \text{minimise} \quad & \frac{1}{2}||w||^2 - \rho + \frac{1}{\nu}\sum_i \frac{1}{l_{c_i}}\xi_i \\ \text{subject to} \quad & y_i(w \cdot \phi(x_i) - b) \geq \rho - \xi_i, \;\; \xi_i \geq 0 \;\; . \end{aligned} \tag{4.9}$$

This is the well-known ν–SVM of [65] with the following three differences: First, there is no restriction on ρ to be positive. Therefore, if we find a solution with a positive ρ, we would find the same solution with a ν–SVM, otherwise not. Note that if we find a negative value of ρ, the classes are not well separated. Dropping the positivity condition for ρ can in our case still result in a solution with few support vectors. The second difference is that the constant $1/\nu$ is in front of the last term in the cost function instead of a constant ν in front of ρ. See exercise 7.16 in the book of Schölkopf and Smola [63], showing that this yields the same decision function. Third, the penalty ξ_i for each outlier is weighted by a factor one over the number of examples in its class, instead of the total number of examples. This is necessary for the objective function to be bounded, otherwise the value for ρ can grow infinitely large. This can be seen in one dimension, for example, for one class consisting of one data point at $+1$ and many data points of the other class at 0. Now setting $\nu = 1$, fixing the vector $w = 1$, varying ρ, and setting $b = \rho$, one can observe that the empirical risk grows slower than ρ does, and therefor the cost function is not bounded from below.

Reformulation of the regulariser for $\gamma = 1$:

For $\gamma = 1$ a direct calculation shows

$$\sum_c ||w_c - \bar{w}_c||^2 = m\Omega. \tag{4.10}$$

Therefore, in this case the solution of the optimisation problem separates the data of any class c maximally from the origin in direction $w_c - \bar{w}_c$.

Translation invariance for $\gamma = 1$:

For $\gamma = 1$, both the decision function (4.1) and the objective function (4.4) are translation invariant with respect to the vectors w_c. In other words, let $v \in H$ be a fixed vector and denote $w'_c = w_c + v$ and $d'_c(x) = (w'_c - \bar{w}'_c) \cdot \phi(x) - \rho_c$. Then $d'_c = d_c$ and therefore $R_{emp,c}(d'_c) = R_{emp,c}(d_c)$. Furthermore, from (4.10) follows that $\Omega(d'_1, ..., d'_m) = \Omega(d_1, ..., d_m)$ holds.

4.2.4 Difference to other multi–class approaches

As mentioned in 4.1 there exist already different multi-class approaches for Support Vector Machines. We will recall the associated optimisation problem and state the differences to our approach. Vapnik [74], Weston and Watkins [78] learn a decision function

$$d(x) = \arg\max_c w_c \cdot \phi(x) + b_c, \quad c = 1, .., m, \tag{4.11}$$

and minimise the cost function

$$\Omega + C \sum_{d \neq c} \sum_{i=1}^{l} [(w_c - w_d) \cdot \phi(x_i) + b_c - b_d + 2]_-, \tag{4.12}$$

with a constant C, and a regulariser $\Omega = \frac{1}{2} \sum_c ||w_c||^2$. Each of the vectors w_c is expressed as a linear combination of the whole training data and therefore they need to estimate ml expansion coefficients, compared to l in our approach, see subsection 4.2.5 below. Furthermore, their decision function aims at a situation where every new example is assigned to exactly one of the classes, contrary to our approach. A similar optimisation problem as (4.12) is used in Bredensteiner and Bennet [15] and Guermeur [33] but with different regularisers $\Omega = \frac{1}{2} \sum_{c<d} ||w_c - w_d||^2 + \frac{1}{2} \sum_c ||w_c||^2$ and $\Omega = \frac{1}{2} \sum_{c<d} ||w_c - w_d||^2$, respectively.

The approach from Cramer and Singer [21] follows the above ones, with a modification of the empirical error term, and without bias terms b_c. This results in a more 'compact' quadratic optimisation problem, for which they present in a clear manner the details of an efficient implementation. But again, one has to estimate ml expansion coefficients.

In [46] a decision function (4.11) is used with the regulariser $\Omega = \frac{1}{2} \sum_c ||w_c||^2$, but with a different empirical risk term. Their empirical risk is for each example the term $\sum_{c' \neq c_i} [-1/(m-1) - d_{c'}(x_i)]_-$ with the decision functions $d_c(x) = w_c \cdot \phi(x) - b_c$ for each class. They put a further constraint on these functions to sum to zero. This constraint couples the otherwise independent optimisation problems for d_c to one problem. Again, the whole training data appears in the expressions for each decision function d_c and thus they need ml parameters for each class.

In general, our approach differs from the above in that the false positives do not appear in our empirical error term. This results in putting fewer constraints and therefore yields fewer parameters to be learnt. This puts our approach in a different context. We do not assume that each data point belongs to exactly one class.

4.2.5 The Dual Problem

To determine the dual of (4.7) we set $f = \Omega + \sum_{c=1}^{m} R_{emp,c}$ and consider the Lagrangian

$$L(w, \rho, \xi) = f(w, \rho, \xi) - \sum_i \alpha_i \big[(w_{c_i} - \gamma \bar{w}_{c_i}) \cdot \phi(x_i) - \rho_{c_i} + \xi_i \big] - \sum_i \beta_i \xi_i \tag{4.13}$$

with multipliers $\alpha_i, \beta_i \geq 0$. Setting the derivatives with respect to w_c, ρ_c, ξ_i equal to zero yields a system of linear equations, which is solved by

$$w_c = \sum_{i \in c} \alpha_i \phi(x_i) \ , \tag{4.14}$$

$$\alpha_i = \frac{1}{\nu l_{c_i}} - \beta_i \leq \frac{1}{\nu l_{c_i}} \ , \quad \sum_{i \in c} \alpha_i = 1 \ , \tag{4.15}$$

where we used $\beta_i \geq 0$ for the inequality in (4.15). Equation (4.14) shows that the vector w_c representing class c can be expressed as a linear combination of examples in class c only. Note that for $\gamma = 1$ the solution for w_c is not unique, as we know from the paragraph on translation invariance in 4.2.3. However, the cost function of all these solutions has the same value so that we can work with the solution (4.14). The examples x_i with $\alpha_i \neq 0$ are called *support vectors*. Substitution of (4.14) and (4.15) in L and using $k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ yields the dual problem

$$\begin{aligned} \text{maximise} \quad & -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_{ij} k(x_i, x_j) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq \frac{1}{\nu l_{c_i}} \ , \quad \sum_{i \in c_i} \alpha_i = 1 \end{aligned} \tag{4.16}$$

with

$$y_{ij} = \begin{cases} 1 & \text{if } c_i = c_j \\ -\gamma \frac{1}{m-1} & \text{if } c_i \neq c_j \end{cases} .$$

We recall that the dual is a concave function [10]. When solving the dual problem we find the optimal values for the multipliers α_i. Note that these are l real values to be determined. Then with (4.14) we find the optimal vectors w_c. We will see in the next subsection, when we discuss the so called ν-property, how we can find the optimal values for ρ_c and ξ_i given the optimal $w_1, ..., w_m$. We note that the constraint $\sum_{i \in c_i} \alpha_i = 1$ means geometrically that the vector w_c lies in the convex hull of the data points $x_i \in S_c$.

4.2.6 The ν-property

We will show that our representation has the ν-property [65]. For this, let us fix $w_1, ..., w_m$. Given w, the optimal values for $\rho_1, ..., \rho_m$ are easily obtained from (4.7). First, we note that these optimal values can be found for each class separately. Consider a class c and set $a_i = (w_c - \gamma \bar{w}_c) \cdot \phi(x_i)$ for all x_i in class c, see Figure 4.2.A. We want to

$$\text{minimise} \quad -\rho_c + \frac{1}{\nu l_c} \sum_{i: x_i \in S_c} \xi_i \qquad \text{subject to} \quad a_i \geq \rho_{c_i} - \xi_i, \quad \xi_i \geq 0. \tag{4.17}$$

For fixed ρ_c the optimal value of ξ_i is the positive part of $\rho_c - a_i$. If we change ρ_c in the negative direction, the term $\frac{1}{\nu l_c} \sum \xi_i$ will change proportionally to the relative number of

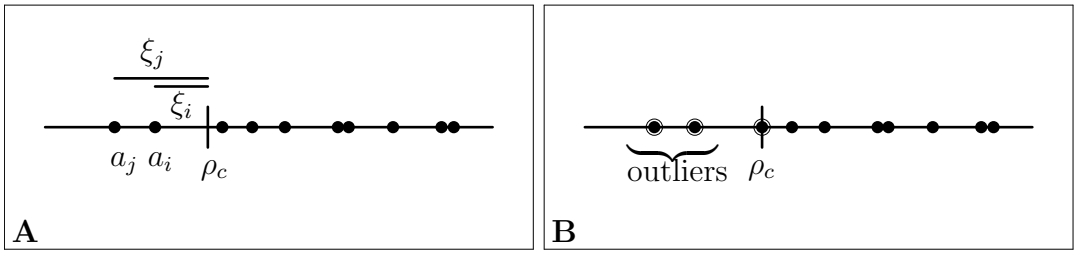


Figure 4.2: The ν-property: The dots show the points $a_i(x_i)$. It is a one dimensional problem to determine the optimal value for ρ_c. Figure **A** shows how the empirical risk changes when changing ρ_c. Figure **B** shows the optimal value for ρ_c with $\nu = 0.3$. The points a_i corresponding to support vectors are marked with a circle, $\circ$, around the dot.

points x_i that have a nonzero ξ_i, that is, points with $a_i < \rho_c$, called *outliers*. Changing ρ_c in the positive direction changes the sum proportional to the number of points with $a_i \leq \rho_c$. Let us now sort the set $\{a_i | i : x_i \in S_c\}$ according to magnitude,

$$a_{i_1} \leq a_{i_2} \leq ... \leq a_{i_{l_c}}.$$

We denote by i_* the index i_n with $n = \lceil \nu l_c \rceil$, where $\lceil \cdot \rceil$ denotes the smallest integer greater or equal to the argument. Then the optimal value for ρ_c is

$$\rho_c = a_{i_*}. \tag{4.18}$$

In other words, consider the set $\{a_i | a_i = w_c \cdot x_i, x_i \in S_c\}$. Hence, ρ_c is equal to the ν–quantile of the empirical distribution over the set of all a_i, see figure 4.3. Note that when νl_c is an integer, the values ρ_c and ξ_i are not necessarily unique.

If we denote by n_c^{SV} the number of support vectors and by n_c^{OL} the number of outliers, we have proved that

$$n_c^{\mathrm{OL}} \leq \nu l_c \leq n_c^{\mathrm{SV}}. \tag{4.19}$$

Let us see what happens if we have more and more examples, that is $l_c \to \infty$. We make the assumption that the training data of each class $\{x_i | x_i \in S_c\}$ be i.i.d. samples from a distribution P_c. If the kernel k and the distribution are such that for any decision function of the form (4.1) the zero level ${d_c}^{-1}(\{0\})$ has measure zero under P_c, then in the limit $l_c \to \infty$ the equalities

$$n_c^{\mathrm{OL}}/l_c = \nu = n_c^{\mathrm{SV}}/l_c \tag{4.20}$$

hold. To see this, consider the set of examples that lie exactly on the margin, $\{x_i \in S_c | d_c(x_i) = 0\}$ and let n_c^{OM} be the number of those points. Then $n_c^{\mathrm{SV}} = n_c^{\mathrm{OM}} + n_c^{\mathrm{OL}}$ and by the assumptions, $n_c^{\mathrm{OM}}/l_c \to 0$.

Let us take a look at the implications of the ν-property for the dual variables $\alpha_1, ..., \alpha_l$. The values (α_i) are optimal values if and only if the Karush–Kuhn–Tucker conditions

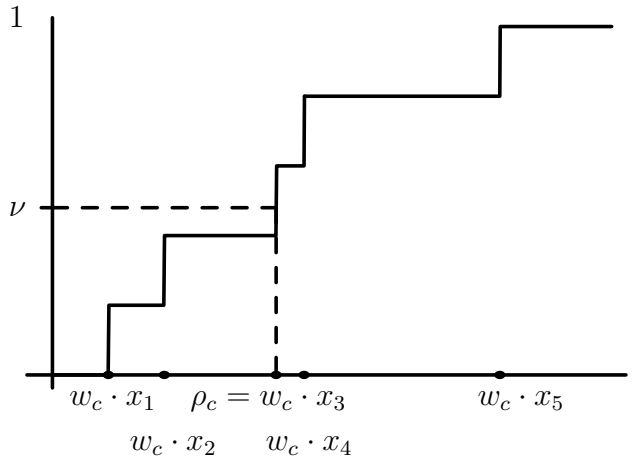


Figure 4.3: Setting the margin at the ν–quantile: shown is the representative vector w_c and the data examples x_i of class c, for $i = 1, .., 5$. The function is the empirical cumulative distribution, and the margin ρ_c is set to the ν–quantile.

(KKT)

$$d_c(x_i) > 0 \Longrightarrow \alpha_i = 0 \tag{4.21}$$

$$d_c(x_i) < 0 \Longrightarrow \alpha_i = \frac{1}{\nu l_{c_i}} \tag{4.22}$$

are satisfied [10], see Figure 4.2.B. If we look at (4.20) we can see that in the limit of $l_c \to \infty$, the relative number of support vectors x_i whose multipliers α_i are not at bound is zero, and therefore the representative vector w_c of class c is the mean of the outliers,

$$\frac{1}{\nu l_c} \sum_{x_i : d_c(x_i) < 0} \phi(x_i) \longrightarrow w_c. \tag{4.23}$$

We will use this fact in section 5.1 to develop a fast algorithm to solve the optimisation problem.

Note that we could choose a different parameter ν for each class. For ease of presentation we did not do so, but imagine a situation in which we had some prior knowledge on the expected number of outliers of each class. If they were very different, it would be suitable to use a different constant ν for each class. Lin et al. [47] investigated these so called non–standard situations for support vector machines.

4.2.7 A parametrised family of representations

Let us have a look at the dual problem (4.16) and let $F(\alpha, \gamma)$ denote its cost function, as a function of the vector $\alpha = (\alpha_1, ..., \alpha_l)$ and the parameter $\gamma \in [0, 1]$. Let us choose a

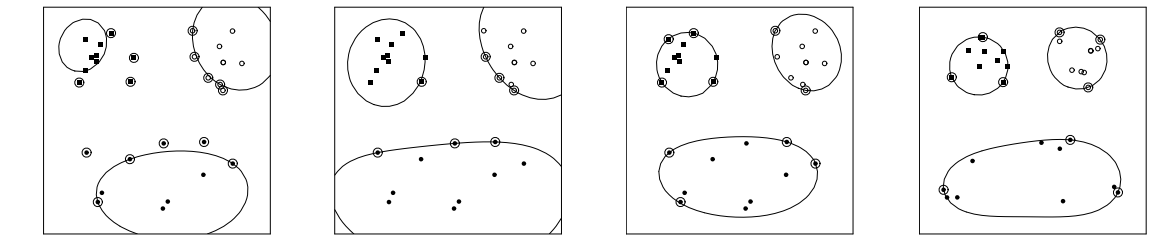


Figure 4.4: A toy example with the three classes $\circ, \bullet$, and $\blacksquare$. For each class, the line shows the zero level of the decision function (4.1) for the solution. We used a Gaussian kernel and maximal coupling $\gamma = 1$ for **A,B**, $\gamma = 0.5$ for **C**, and $\gamma = 0$ for **D**. Plot **A** shows the solution with $\nu = 0.4$ and **B,C,D** with $\nu = 0.1$. The support vectors are marked with a $\bigcirc$. We can see how the parameter ν controls the number of outliers in **A,B**. From **B** to **D** the parameter γ is decreasing and a change in the selection of the support vectors can be seen.

strictly positive definite kernel k. Then for each fixed $\gamma \in [0, 1)$, the function $F(\cdot, \gamma)$ is strictly concave and the solution of (4.16),

$$\alpha_{\max} = \arg\max_{\alpha} F(\alpha, \gamma), \tag{4.24}$$

where the maximisation is done over all α fulfilling the constraints, is unique. Therefore, the mapping $\gamma \mapsto \alpha_{\max}$ is well defined. Because each $\alpha_{\max}$ defines a representation of our data, γ parametrises a family of representations. When we look at the toy example in Figure 4.4, we can see how the support vectors and therefore their multipliers $\alpha_1, ..., \alpha_l$ change with a change in γ. By subsection 4.2.6 above, as the number l of examples grows, the relative number of multipliers not at bound goes to zero. Informally speaking, we expect to have most of the $\alpha_1, ..., \alpha_l$ at bound, either zero or $1/\nu l$, and the only multipliers not at bound belong to support vectors sitting exactly on the margin, that is, are mapped to zero by the decision function. So one can wonder if the support vectors suddenly appear or disappear as we change γ. This is not the case, that is, by changing γ continuously none of the optimal multipliers $(\alpha_1, ..., \alpha_l)$ jumps, but they all change continuously as well.

Lemma 3 *With the above notations, the mapping $\gamma \mapsto \alpha_{\max}$ is continuous.*

The details of the argumentation and the proof of the lemma can be found in the appendix.

4.2.8 Toy Experiment

We tested the algorithm on artificial data in two dimensions, see figure 4.4. Each of the three clusters represents a class and we plotted the lines where the decision functions (4.1) are zero. One can easily check the ν–property. In 4.4.A with $\nu = 0.4$, we should

have at least 4 support vector of the 10 vectors in each class, and maximally 4 outliers. If we look at B to D, there the parameter γ is decreasing from 1 to 0, and therefore the influence of the other two classes to a given one is reduced. One can see the tendency of the support vectors to be closer to the other classes for large γ. Note that if γ is equal to zero, the representation of each class is learnt independently of the other classes, just as in Schölkopf et al. [62].

4.3 Discussion

We proposed a new supervised algorithm to learn representations of multi–categorical data. The main inspiration for our algorithm came from the resemblance of the support vector algorithms for one and two classes. We have seen that in the case of two classes, there is a difference in our algorithm to the standard support vector machine, as we put no constraint on the margin ρ to be positive. This has the advantage that even if the classes are not well separated, we can arrive at a solution with few support vectors.

A possible generalisation of our algorithm to the so called non–standard situations is possible, either by choosing a different ν_c for different classes c to weight the whole classes differently, or to include a weighting factor in front of each summand in the empirical risk, as in the support vector machine case [47]. We did not do any experiments in this direction.

4.4 Details and Proofs $*$

Here, we will give the proofs of the lemmata in the text.

Proof of lemma 1: Let us write $w = (w_1, ..., w_m)$ and define $g_c(w) = ||w_c||^2 - \gamma w_c \cdot \bar{w}_c$. For $m > 1$ a short calculation shows

$$\sum_c g_c(w) = \frac{m}{m-1}\big((m-1+\gamma)\frac{1}{m}\sum_c ||w_c||^2 - \gamma m||\frac{1}{m}\sum_c w_c||^2\big) \ .$$

With $m-1+\gamma \geq \gamma(m-1)+\gamma = \gamma m$, for $\gamma \in [0,1]$, and Jensen's inequality [8] it follows

$$\sum_c g_c(w) \geq 0 \ . \tag{4.25}$$

Note that strict inequality holds for $\gamma < 1$.

Now we proof the representer theorem. Let us write for $c = 1, ..., m$,

$$w_c = w_c^{||} + w_c^{\perp} = \sum_i \lambda_{c,i}\phi(x_i) + w_c^{\perp},$$

with $w_c^{\perp}$in the orthogonal complement of the span of $\phi(x_1), ..., \phi(x_l)$. Now

$$\begin{aligned} d_c(x_j) &= (w_c - \gamma \frac{1}{m-1} \sum_{d \neq c} w_d) \cdot \phi(x_j) - \rho_c \\ &= (w_c^{||} - \gamma \frac{1}{m-1} \sum_{d \neq c} w_d^{||}) \cdot \phi(x_j) - \rho_c, \end{aligned}$$

because $w_c^{\perp} \cdot \phi(x_j) = 0$ for all $j = 1, ..., l$. Thus the value of $R_{emp,c}(d_c)$ is independent of $w_c^{\perp}$. Let us write the dependence of the decision function d_c on w explicitly as $d_c(w)$ and accordingly we use $d(w)$. To proof the lemma, we will show that $\Omega(d(w)) \geq \Omega(d(w^{||}))$. We have

$$\Omega(d(w)) = \sum_c (g_c(w^{||} + w^{\perp}) - \rho_c) = \sum_c (g_c(w^{||}) + g_c(w^{\perp}) - \rho_c).$$

But (4.25) implies $\sum_c g_c(w^{\perp}) \geq 0$ and the claim follows. ■

Now let us show lemma 2.

Proof of lemma 2: To see that the cost function (4.4) is bounded from below we argue as follows. The empirical risk is a positive function. We write for the regulariser $\Omega = \sum_c (g_c - \rho_c)$. By (4.25) above, for Ω to take arbitrary large negative values, ρ_c must take arbitrary large values. Now look what happens for $\rho_c > \max w_c \cdot \phi(x_i)$. Increasing ρ_c by $\Delta\rho_c$ yields a decrease $\Delta\rho_c$ in Ω but an increase $l_c/(\nu l_c)\Delta\rho_c \geq \Delta\rho_c$ in the empirical risk term. Therefore the cost function is bounded. To see the convexity choose $t \in [0, 1]$ and another vector $v = (v_1, ..., v_m)$. One calculates

$$t g_c(w) + (1-t) g_c(v) - g_c(tw + (1-t)v) = t(1-t) g_c(w - v) \ , \tag{4.26}$$

and because of (4.25) the sum over all classes of (4.26) is greater or equal to zero, thus $\sum_c g_c$ is convex. Therefore Ω as the sum of a convex function and the linear function $\sum_c \rho_c$ is convex and because the empirical risk is convex, the sum of them as well. ■

Proof of lemma 3: To proof that the mapping $\gamma \mapsto \alpha_{\max}(\gamma)$ is continuous let us choose a value $\bar{\gamma} \in [0, 1)$ and a sequence (γ_n) converging to $\bar{\gamma}$, in short, $(\gamma_n) \to \bar{\gamma}$. We will show that the induced sequence $(\alpha_{\max}(\gamma_n))$ converges to $\alpha_{\max}(\bar{\gamma})$ by contradiction. Let us write shortly α^n for $\alpha_{\max}(\gamma_n)$. Because the sequence α^n is bounded, there is a convergent subsequence and therefore without loss of generality we assume that (α^n) converges to $\bar{\alpha}$. Consider the combined sequences $(\alpha_{\max}(\bar{\gamma}), \gamma_n) \to (\alpha_{\max}(\bar{\gamma}), \bar{\gamma})$ and $(\alpha_n, \gamma_n) \to (\bar{\alpha}, \bar{\gamma})$. Because the function $(\alpha, \gamma) \mapsto F(\alpha, \gamma)$ is a polynomial, it is continuous and therefore $F(\alpha_{\max}(\bar{\gamma}), \gamma_n) \to F(\alpha_{\max}(\bar{\gamma}), \bar{\gamma})$ and $F(\alpha_n, \gamma_n) \to F(\bar{\alpha}, \bar{\gamma})$. If now $\bar{\alpha} \neq \alpha_{\max}(\bar{\gamma})$ then by the strict convexity of $F(\cdot, \gamma)$ we have $F(\alpha_{\max}(\bar{\gamma}), \gamma_n) > F(\alpha_n, \gamma_n)$ and therefore $F(\alpha_{\max}(\bar{\gamma}), \bar{\gamma}) \geq F(\bar{\alpha}, \bar{\gamma})$. But by definition of $\alpha_{\max}(\bar{\gamma})$, we have $F(\alpha_{\max}(\bar{\gamma}), \bar{\gamma}) < F(\bar{\alpha}, \bar{\gamma})$, a contradiction. ■

Chapter 5

Efficient optimisation of the multiclass problem

In this chapter we propose an algorithm to find an exact solution of our multi-class optimisation problem 4.2 and test the performance on a real–world task. In detail, we start in section 5.1 by introducing our algorithm, which is an adaptation of the Sequential Minimal Optimisation algorithm [53, 62]. We will show that the algorithm can be efficiently implemented, using finite accuracy computations to avoid a large number of small but time consuming updates of the variables. We prove termination of the algorithm after a finite number of steps and give a bound on the accuracy needed to find a solution. In section 5.2 we test the recognition performance of our algorithm on a standard handritten digit recognition task and show how our finite accuracy implementation speeds up optimisation.

5.1 The algorithm

The quadratic problem (4.16) can be solved in a standard way. We choose a small subset of variables and try to find their optimal values while keeping the other variables fixed. Then we repeat this step with different subsets until the algorithm converges. Our algorithm is based on the Sequential Minimal Optimisation algorithm [53, 62], which we have adapted to our problem. The idea is to solve the smallest possible subproblem. Because of the constraint

$$\sum_{i \in c} \alpha_i = 1,$$

a change of only one variable is not possible. The smallest possible subproblem is to modify two multipliers simultaneously, where these multipliers belong to examples of the *same* category. In order to make this elementary step for one class c, it is enough to know the mean vector $\bar{w}_c = \sum_{j \notin c} \alpha_j k(x_i, \cdot)$ of all the other classes. The algorithm proceeds by iterating this elementary step and observing optimality conditions to check if the optimum is reached. The values (α_i) are optimal values if the Karush–Kuhn–Tucker conditions (4.21) and (4.22) are satisfied. Figure 5.1 gives an overview of the algorithm, which is described in detail in the text.

Let us first describe the elementary step, the optimisation of two multipliers of the same category. Then we will describe how we iterate this elementary step to find the solution. Finally, we will describe how one can deal with finite accuracy calculations, which leads to the modified algorithm in Figure 5.2. We start similar to [62].

5.1.1 The elementary step

Suppose we want to optimise the variables α_i and α_j while keeping the other multipliers fixed. Therefore, we have to maximise the cost function (4.16) as a function of α_i and α_j. Let us write F for minus the cost function,

$$F(\alpha_i, \alpha_j) = \frac{1}{2} \sum_{i',j'} \alpha_{i'} \alpha_{j'} K_{i'j'}, \tag{5.1}$$

with $K_{ij} = k(x_i, x_j) y_{ij}$. Let us denote by α_l^{new} the value of the multiplier α_l after the elementary step. At the optimum, the gradient of F is perpendicular to the 1-dimensional subspace, spanned by the equality constraints

$$\alpha_i + \alpha_j = \Delta, \quad \text{with} \quad \Delta = 1 - \sum_{l \neq i,j}^{l_c} \alpha_l.$$

Therefore, $\partial_{\alpha_i} F - \partial_{\alpha_j} F = 0$ at $(\alpha_i^{new}, \alpha_j^{new})$. With $\partial_{\alpha_i} F(\alpha_i^{new}, \alpha_j^{new}) = \sum_l \alpha_l^{new} K_{il}$ we get

$$\alpha_i^{new} = \alpha_i + \frac{d_c(x_j) - d_c(x_i)}{K_{11} + K_{22} - 2K_{12}}, \tag{5.2}$$

where we used $\sum_l \alpha_l (K_{jl} - K_{il}) = d_c(x_j) - d_c(x_i)$. If α_i^{new} is outside the feasible interval $[0, 1/\nu l_c]$, we take the closest value in this interval for α_i^{new}. From this we find $\alpha_j^{new} = \Delta - \alpha_i^{new}$. Again, it is possible that α_j is not in the feasible interval. So we project α_j^{new} in the feasible region and recompute α_i. Now, we found the optimal values for the two points, keeping all the other points fixed.

5.1.2 Optimisation algorithm

We initialise the algorithm by choosing randomly a fraction ν of multipliers of each category c and setting them to $1/\nu l_c$. If νl_c is not an integer, one more multiplier is chosen and set to a value in $(0, 1/\nu l_c)$ to ensure that the multipliers of each class sum to one.

We proceed by iterating over all categories. For each category c we calculate $d_c(x_i)$, for all $x_i \in S_c$, choose two multipliers α_i, α_j, and perform an elementary step to optimise them. For the choice of the multipliers we use the KKT conditions. In practical cases, we compute with a finite accuracy, so we should replace the comparisons in equation (4.21) and (4.22) with ϵ tolerant comparisons. We do this, but rewrite the conditions in a slightly different form,

$$d_{c_i}(x_i) \alpha_i < \epsilon \tag{5.3}$$

$$d_{c_i}(x_i)(\alpha_i - \frac{1}{\nu l_{c_i}}) < \epsilon, \tag{5.4}$$

which we call the relaxed conditions or ϵ–KKT conditions. It is easily verified that for $\epsilon = 0$ these two conditions are equivalent to (4.21) and (4.22). This form is useful because it

choose γ, ν, ϵ
initialise all α_i
while the ϵ–KKT conditions are not satisfied [(5.3),(5.4)]
    repeat for all classes c
        compute $d_c(x_i)$ for all $x_i \in S_c$, and compute ρ_c [(4.1),(4.18)]
        repeat multiple times
            choose α_i, with $c_i = c$, a violator of ϵ–KKT [(5.3),(5.4)]
            choose α_j with $c_j = c$
            perform the elementary step with α_i, α_j [(5.2)]
            update $d_c(x_i)$ for all $x_i \in S_c$, ρ_c [(5.5),(4.18)]
        end repeat
    end repeat
end while

Figure 5.1: The standard version of the main algorithm, as described in the text. In square brackets at the end of each line are the relevant equations to perform the described actions.

measures the KKT–Gap [10] and therefore bounds the difference of the objective function to a solution of the relaxed conditions (5.3) and (5.4) to the true minimum by $l\epsilon$.

We scan over all multipliers of class c until a violator α_i of (5.3) or (5.4) is found. When one is found, choose a second multiplier of the same class. Then, do an elementary optimisation step with α_i, α_j. Now, the values $d_c(x_m), x_m \in S_c$ are updated,

$$d_c(x_m) \leftarrow d_c(x_m) + \sum_{l=i,j} (\alpha_l^{new} - \alpha_l^{old}) k(x_l, x_m) \tag{5.5}$$

where α_l^{old} denotes the value of α_l before and α_l^{new} its value after the elementary step. Next, the margin ρ_c is recomputed according to (4.18). After we have scanned over all multipliers α_i of class c, we go to the next class. We iterate until there are no violations of the relaxed conditions. In practice, we perform multiple iterations over the same class before going to the next one. This is possible, because the multipliers of one class depend on the other classes only through the mean of all other w_c, and therefore changing the multipliers of one class has a small influence on the other classes.

Up to now, we have described the basic algorithm to solve the optimisation problem. Unfortunately, in practice this does not yet yield an efficient implementation. We need a good choice heuristics for the pairs α_i, α_j we want to optimise at each elementary step. Otherwise, it can happen that we make a lot of small changes instead of a few large ones, which dramatically slows down optimisation. For support vector machines, there already exist many good heuristics, see Platt [53], Joachims [40], Keerthi et al. [41], or the overview in Schölkopf and Smola [63]. We will deal in the next section with finite accuracy computations and see how that can circumvent in a natural manner the problem of performing small changes.

5.1.3 Finite accuracy

Recall from subsection 4.2.6 that for a large training set we expect to have very few points that lie exactly on the margin. But according to the optimality conditions (4.21) and (4.22), these are the only points x_i whose multipliers α_i can have a value not at bound. All the other multipliers α_i are saturated at one of the bounds, either 0 or $1/(\nu l_{c_i})$. Therefore, it is natural to ask if we can find a good approximation to the solution with each α_i taking only one of the two values $\{0, 1/(\nu l_{c_i})\}$. We will see in the next section that the answer is no, at least in the general case. An illustration is given in Table 5.2, where we compare the performance of this binary algorithm with the exact solution on a handwritten digit recognition task. Nevertheless, it is possible to start the algorithm in an initial phase with binary weights. The idea is to allow the multipliers at the beginning of optimisation to take only the two extreme values, therefore preventing small changes, and then during optimisation to allow more and more intermediate values.

Recall from subsection 4.2.6 that we can choose a different parameter ν, called ν_c, for each class c. We will do this in the following way. For a given ν, we choose the closest ν_c, separately for each category, such that $\nu_c l_c$ is an integer. This is for the sake of convenience only; for large datasets, the values of ν and ν_c will be very close. For each class c the possible values for $\alpha_i, i \in c$ are in the interval $[0, \frac{1}{\nu_c l_c}]$. Let us choose an integer p. We define the accuracy for the multipliers of each class to be

$$\delta_c = 1/(p\nu_c l_c) \tag{5.6}$$

and choose as possible approximations of α_i the $p+1$ values $\{0, \delta_c, ..., p\delta_c\}$. Next, we have to modify the elementary step from 5.1.1 such that the multipliers always take one of the allowed values. Suppose we want to do this new elementary step for the multipliers α_i, α_j. We will use the symbol $\alpha_l^{new}, l = i, j$ to denote their new values. We know that $\alpha_i^{new} + \alpha_j^{new} = \alpha_i + \alpha_j$, therefore once α_i^{new} is found we use this equation to find α_j^{new}. We first calculate the optimal value of α_i according to (5.2). Let us denote this value by α_i^{opt}. We determine an integer q such that

$$q\delta_c \leq \alpha_i^{opt} < (q+1)\delta_c. \tag{5.7}$$

Now either $q\delta_c = \alpha_i^{opt}$, and then α_i^{opt} is the new value of the multiplier, or we have to round α_i^{opt} to one of the two bounds of (5.7). We take the better one by checking the value of the dual objective function. Concretely, with the notations of 5.1.1, we test if

$$F(q\delta_c, \alpha_i + \alpha_j - q\delta_c) < F((q+1)\delta_c, \alpha_i + \alpha_j - (q+1)\delta_c) \tag{5.8}$$

and if so, $\alpha_i^{opt} = q\delta_c$ otherwise $\alpha_i^{opt} = (q+1)\delta_c$.

The whole algorithm with finite accuracy is as follows, see Figure 5.2. We start with $p = 1$. Then, the multipliers can take only the values 0 and $1/(\nu_c l_c)$. Now we are running the loop, see Figure 5.1. The computational costly part of the algorithm is the update of the values $d_{c_i}(x_i)$ after a change in the multipliers. As we allow the multipliers to have only one of the two values $\{0, 1/(\nu l_{c_i})\}$ we do few and large changes. We run the algorithm

choose γ, ν, ϵ
set $p = 1$
initialise all $\alpha_i \in \{0, 1\}$
while the ϵ–KKT conditions are not satisfied [(5.3),(5.4)]
  repeat for all classes c
    compute $d_c(x_i)$ for all $x_i \in S_c$, and compute ρ_c [(4.1),(4.18)]
    repeat multiple times
      if p does not exceed the bound [(5.9)]
        choose α_i, α_j, with $c_i = c, c_j = c$, with maximum heuristic [(5.10)]
      else
        choose α_i, with $c_i = c$, a violator of ϵ–KKT [(5.3),(5.4)]
        choose α_j with $c_j = c$
      end if
        perform the elementary step with α_i, α_j [(5.2),(5.7),(5.8)]
      update $d_c(x_i)$ for all $x_i \in S_c$, ρ_c [(5.5),(4.18)]
    end repeat
  end repeat
  if none of the α_i was changed
    $p \leftarrow p \cdot 100$
  end if
end while

Figure 5.2: The final version of the algorithm, which uses finite accuracy computations and the maximum heuristic (5.10) to choose a pair of multipliers. Again, the relevant equations to perform the described actions are in square brackets at the end of each line. The vertical lines mark the difference to the standard algorithm in Figure 5.1. The thin lines mark the changes for the finite accuracy computations. The changes needed to include the maximum heuristic are indicated by the bold line.

until there are no KKT violations or until no positive progress is made. Because there is only a finite number of possible changes, we will come to this point after a finite number of steps. If the relaxed KKT conditions are not yet satisfied we change the value of p to a larger value, for example we multiply p by 100, and redo the optimisation steps until the relaxed KKT–conditions are satisfied. We will state next that this procedure comes to an end.

Proposition 5 *The finite accuracy algorithm terminates after a finite number of steps. More precisely, if $\epsilon > 0$ is the relaxation of the KKT–conditions, it will stop if*

$$p > 2M\frac{1}{\epsilon}\max_c\Big(\frac{1}{\nu_c l_c}\Big)^2, \tag{5.9}$$

with $M = \max_i ||\phi(x_i)||$.

Proof : Let us say that the multipliers $(\alpha_1, ..., \alpha_l)$ are *p–accurate* if the algorithm with an accuracy $\delta_c = 1/(p\nu_c l_c)$ for each c either stops or does not yield any positive progress. We will show that for any $\epsilon > 0$ we can choose an integer p such that any p–accurate multipliers fulfil the relaxed KKT–conditions. Let us fix a class c. Define

$$\begin{aligned} i &= \arg\max_{l:x_l\in S_c} \quad \alpha_l d_c(x_l) \\ j &= \arg\max_{l:x_l\in S_c} \quad \big(\alpha_l - \frac{1}{\nu_c l_c}\big) d_c(x_l). \end{aligned} \tag{5.10}$$

Thus, if there are violators of the relaxed KKT–conditions (5.3) and (5.4), then x_i, x_j yield a maximal violation. By the definition of ρ_c in (4.18) and the constraints in (4.16) on α_l we have at any time of the algorithm

$$d_c(x_i) \geq 0, \quad d_c(x_j) \leq 0.$$

Because the multipliers are p–accurate, no elementary step (5.2) is made, thus

$$d_c(x_i) - d_c(x_j) < \delta_c\big(k(x_i, x_i) + k(x_j, x_j) - 2k(x_i, x_j)\big).$$

Therefore,

$$\begin{aligned} \alpha_i d_c(x_i) &\leq \alpha_i\big(d_c(x_i) - d_c(x_j)\big) < \alpha_i\delta_c\big(k(x_i, x_i) + k(x_j, x_j) - 2k(x_i, x_j)\big) \\ &\leq \frac{1}{\nu_c l_c}\frac{1}{p\nu_c l_c}2M. \end{aligned}$$

If we choose p according to (5.9), there will be no violators of (5.3). An analogous argument shows that there will be no violators of the second condition (5.4). ■

One may wonder if we could use (5.10) as a heuristic for the choice of the two multipliers for an elementary step. Note that we did not prove that with such a choice the algorithm can find the optimal solution. But we still can use (5.10) as a heuristic in the following way. We run the algorithm, with a pair of multipliers for an elementary step

	false accept	false reject	number SV
$\gamma = 1$	75	245	580
$\gamma = 0.5$	188	248	569
$\gamma = 0$	223	685	515
6×SVM	71	241	2530

Table 5.1: Results on the USPS handwritten digits dataset: The algorithm was trained on the digits 0 to 5 and tested on all the digits 0 to 9. For all the results we chose a parameter $\nu = 0.05$. The algorithm was trained with different parameters γ, given in the first column. The second and third column show the number of false accepted and rejected test examples, respectively. The fourth column gives the number of Support Vectors. The results of our algorithm are compared with a ν–support vector machine, trained one–against–all. There, the decision function is given by an adjusted margin as described in the text.

chosen according to (5.10). We iterate until the relaxed KKT–conditions are satisfied or until the accuracy is such that p fulfils (5.9). If the second is true, and if we cannot fulfil the KKT–conditions with this maximum heuristic (5.10), we just look for a pair of multipliers according to our more general choice, see Figure 5.1. Now with lemma 5, we will find a solution of the relaxed conditions without further increasing the accuracy. In Figure 5.2 we show the final version of the algorithm. The differences to the algorithm in Figure 5.1 are highlighted by the vertical lines.

Figure 5.3 shows the convergence of the algorithm with and without finite accuracy computations, and with and without the heuristic (5.10). We can see that the combination of the two yields a very efficient algorithm. See section 5.2 for a detailed discussion and the exact parameters of the experiment.

5.2 Performance test

In this section we present different experiments with our algorithm. Recall that for $m = 1$ and $m = 2$ all the results of the one class and the ν–support vector machine with $\rho \geq 0$ can be reproduced. In the case of three classes we have already seen various toy examples. Figure 4.1.**C** shows the solution found by our algorithm for a linear kernel and $\gamma = 1$, and in Figure 4.4 the dependence of the solution with a Gaussian kernel on the parameters ν, γ is illustrated. In the next subsection we will perform experiments on real–world data.

5.2.1 Real–world data

We test the algorithm on the US postal service database of handwritten digits, a standard dataset for multi–categorical classification. The digits run from 0 to 9 and are images of size 16×16. This can be considered as a 'closed' data base: Each image necessarily

belongs to one of the ten classes. In real–world applications, however, data comes from an 'open' scenario. The classifier trained to recognise digits must also respond to non–digit images, such as characters, lines, drawings, etc.. In order to mimic such an open situation we trained the algorithm on the digits 0 to 5 only. The test set, however, contains also samples from digits $6-9$. Ideally these additional test examples should be classified as belonging to none of the classes. There are 4896 examples in the training set. The data was normalised, such that all components of a data vector lie in the interval $[-1/16, 1/16]$. For optimisation, we used a Gaussian kernel of variance 0.1, which is a suitable value for this task [60]. We tested the model on 2007 examples of the digits 0 to 9. Remember that we did not present examples of 6 to 9 to the algorithm during learning. We say that an example of class c is accepted, if the decision function (4.1) is positive, and otherwise it is rejected. Note that one example can be falsely accepted by more than one class. The results are summarised in Table 5.1.
The results are compared with a ν-support vector machine, trained one–against–all on the examples of 0 to 5. For the ν–support vector machine we used the libsvm package [20]. To accept or reject examples, one needs a threshold value for each support vector machine. Let us denote by (v_c, b_c, μ_c) the solution of support vector machine c, where v_c is the normal vector of the separating hyperplane, b_c is the offset and μ_c the margin. We found that neither the hyperplane nor the margin are suitable thresholds, in other words the test $v_c \cdot \phi(x) - b_c \lesseqgtr 0$ accepted too many wrong examples (337 false acceptances, 66 false rejections), whereas the test $v_c \cdot \phi(x) - b_c \lesseqgtr \mu_c$ reject too many positive examples (13 false acceptances, 368 false rejections). Therefore, we defined a threshold θ_c for each classifier to be the νl_c–smallest value of the set $\{v_c \cdot \phi(x_i) | x_i \in S_c\}$, where l_i is the number of examples in class i, and the test for acceptance or rejection is $v_c \cdot \phi(x) \lesseqgtr \theta_c$.

The same parameters are used for the SVM as for our algorithm, in particular the kernel, the parameter ν, and the accuracy of the solution. It should be pointed out that the SVM needs six times more parameters because one has to learn a multiplier for each example and each of the six classifiers. Noted in the table are the median values over five runs, with a high accuracy of 10^{-6} for the relaxed KKT conditions, therefore the number of false acceptance and false rejections respectively did not change across different runs by more than one (except in the case $\gamma = 0$, where we found a maximal difference of 4 examples). One can see that for higher values of γ the performance is better. This is expected, as for $\gamma = 0$ the representation is learnt for each class separately, and the higher the value of γ, the more information of the other classes is used to estimate the vector w_c of class c. We calculated the lower bound of the number of support vectors according to subsection 4.2.6, which is 241. The actual solution uses 500-600 support vectors. This indicates that we are still far from the limit, where the inequality (4.19) turns into an equality. More interesting is that the number of support vectors is small compared to the total of 2530 support vectors of all the six SVMs together. Note that a support vector machine trained on all digits 0-9 achieves an error of about 80-90 examples, but uses the additional knowledge that each shown new example belongs exactly to one class [60].

	false accept	false reject	number SV
standard algorithm	75	245	580
binary algorithm	176(4)	252(21)	241

Table 5.2: Comparison of the true solution with an finite accuracy solution, where each multiplier α_i can only take either of the two values $\{0, 1/(\nu l_{c_i})\}$. The numbers for the standard algorithm are the ones taken form table 5.1, for the binary algorithm they are the mean over five runs, with standard deviation in parenthesis.

5.2.2 Testing finite accuracy

Let us investigate the influence of the finite accuracy computations on the behaviour of the algorithm. We do the same digit experiment as above with the same parameters. We choose an initial value $p = 1$. First, we wanted to test if we can find a good approximation of the solution with each multiplier α_i taking one of the two values $\{0, 1/(\nu l_{c_i})\}$ only, that is, for each class the vector w_c will be the mean of the outliers. Because we kept a very small value $\epsilon = 10^{-6}$ for the relaxation of the KKT conditions, the algorithm was not able to fulfil them and therefore we stopped when no positive progress was made. We tested the solution found on the test set and compared it with the precise solution, see Table 5.2. One can see that for the reasonable choice of $\nu = 0.05$ the approximate solution has a lot more false acceptances than the true one. When comparing the number of support vectors, we can see that the precise solution has many more, and therefore many support vectors sitting exactly on the margin.

Next, we repeated the same experiment. However, when there was no positive progress, we multiplied p by 100 and continued the optimisation. We repeated this until the relaxed KKT–conditions were satisfied, as described in Figure 5.2. In Figure 5.3 we show the value of the objective function as a function of the number of changes made by an elementary step. This is an indicator for the speed of the algorithm, because after a change by an elementary step, we recompute the values $d_c(x_i)$ which is the time consuming part of it. The solid lines show the cases where we used the finite accuracy modification, whereas for the dashed lines we did not. Note that to choose a pair of multipliers, we either scanned in a double loop over all pairs of multipliers belonging to the same class, where the outer loop checks for a KKT–violation (thin line), or we used the maximum heuristic (5.10) to choose a pair (bold line). Note that in the first case, we made ten times the outer loop for each class, before changing to the next class, whereas in the second case, we made 100 elementary steps for the same class before changing to the next.
For the finite accuracy modifications, it occurred twice that there was no positive progress and therefore we augmented the accuracy by multiplying p with 100 to a final value of $p = 10^4$. The theoretical bound of lemma 5 is $p > 2 * 10^6 * 1/28^2 \approx 2600$. The vertical arrows mark this increase in accuracy for the fastest algorithm using finite accuracy and the maximum heuristic. One can see that an increase in accuracy yields a smaller slope of the objective function, as expected, because smaller steps are made. Because

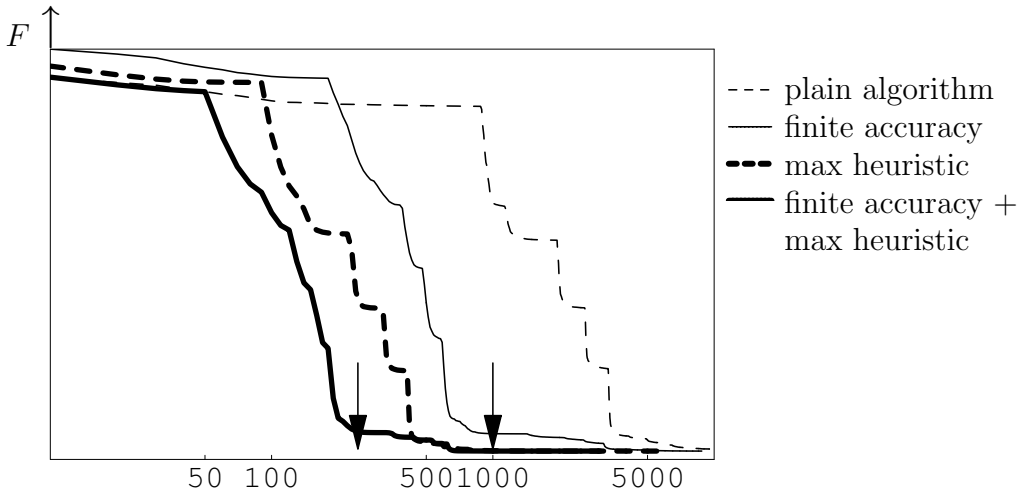


Figure 5.3: The log–log plot shows the value of the objective function F from (5.1) with respect to the number of changes made by an elementary step of the algorithm. For the solid lines we used the finite accuracy calculations, but not for the dashed lines. For the bold lines we used the maximum heuristic, but not for the thin lines. The vertical arrows mark the moment of an increase in accuracy for the bold line, that is, when using finite accuracy and the maximum heuristic. See the text for more details.

the moment of convergence is not well visible in figure 5.3, we give an average number of changes made by an elementary step for the different algorithms in Table 5.3. We can see that the combined version of finite accuracy and maximum heuristic is the most efficient method.
Note that by using the maximum heuristic, but without finite accuracy computations, it is not sure that the algorithm finds a solution. In our example it always did.

When we evaluate the kernel only when needed and keep these values in memory, we need for the best version about $6 * 10^6$ kernel evaluations. This is few compared to the about $25 * 10^6$ entries of the kernel matrix. To have an idea of training time, our Java implementation took about $30 - 40$ seconds to find the solution on a Sun–Blade (running

	algorithm	number of changes in elementary step
- -	plain algorithm	$\gg$50'000
–	finite accuracy	9810 (390)
- -	max heuristic	5640 (610)
—	finite accuracy + max heuristic	3320 (160)

Table 5.3: Average number of changes made by an elementary step until convergence with its variance in parenthesis. See Figure 5.3 and text for details.

at 500MHz), having enough memory to store all needed kernel evaluations.

5.3 Discussion

In this chapter we showed how to solve the optimisation problem of section 4.2 efficiently. We proved that our finite accuracy algorithm terminates after a finite number of steps, but we did not tell how long one has to wait for that. One can see that the proof is actually constructive in the sense that it could be used to calculate a bound on the number of iterations. This is a subject for future work.

Eventually, we want to point out that there exist more possibilities to further speed up the algorithm. For example, one could do several elementary steps before updating the decision function and the margin. Many of these possibilities are discussed elsewhere [40, 53, 21, 19], and could be combined with the proposed approach.

The experiments showed a good performance of our algorithm on the USPS dataset. They confirmed that our approach uses less parameters and less support vectors than a standard one–versus–all approach using binary support vector machines. We recall, that we did not use the standard closed scenario, in which one knows, that every test example belongs to one of the learnt classes. But we mention here that we tried our algorithm on the closed scenario. In more detail, we used all the ten digits to train the algorithm and then classified the test examples. The class c is attributed to the test example x if the decision function d_c of class c takes a maximal value $d_c(x)$ among the decision functions of all classes. In this situation, we noted a significant decrease in performance with respect to a standard one–versus–all approach using binary support vector machines. This comes with no surprise, as in our empirical risk term only positive examples lying on the wrong side of the margin are penalised, and therefore, our approach is not designed for this situation.

Chapter 6

Simple optimisation of the multiclass problem

In this chapter we derive a stochastic gradient descent algorithm to solve the optimisation problem 4.4. Intuitively, this type of algorithm is called an *online* algorithm, because at each iteration of the algorithm, only a fraction of the training data is available, and is used immediately by the algorithm to optimise the parameters. This is in contrast to the so-called *batch* algorithms, where at any time the whole training data can be used for optimisation. The algorithm in chapter 5 was an example of a batch algorithm. Even though at each elementary step only two samples are used, we need access to all examples to select the two most relevant ones. This is different from the online algorithm in this section, where at each step we are not free to choose the examples.
We will start by deriving the learning rule in section 6.2, that is, the update rule of the parameters at each iteration. Then, in section 6.3, we will apply the online algorithm to a toy problem and to the handwritten digit problem of section 5.2. Finally, we will discuss our results.

6.1 Introduction

The classical perceptron algorithm by Rosenblatt [56] is one of the simplest online algorithms for learning a separating hyperplane. At each instant, a new example is shown and if it is misclassified, the hyperplane is moved toward the misclassified point, otherwise it is kept unchanged. If the data is linearly separable, the perceptron algorithm converges, but it is not sure that the resulting hyperplane has a large margin. Although, margins were used right from the beginning, their size was fixed beforehand, see [25] and the references therein. A large margin perceptron algorithm was proposed by Freund and Schapire [28], where the margin is optimised during training. The idea is to learn multiple perceptrons, each with a weight factor. Then, the decision rule is given by the sign of the weighted perceptrons. An online maximum margin perceptron rule was proposed by Kowalczyk [44]. The idea is to generate a solution by approximating the closest points of the convex hulls of the data points of both classes. A modification of the perceptron algorithm to multiple classes was proposed by [22].

Our approach relies on the idea of Kivinen et al. [43] and uses their framework. The idea is to use a gradient descent method, but instead of using all the data to calculate the empirical risk term, we use only one example of each class to estimate its value at a given time step. The resulting algorithm is of the same form as the perceptron algorithm: at each instant a few examples are shown, tested, and an update is performed depending on the outcome of the test.

6.2 Learning rule

We will consider the following situation. At each time step t we have an estimate $w_t = (w_{t,1}, ..., w_{t,m})$ and $\rho_t = (\rho_{t,1}, ..., \rho_{t,m})$ of the minimum of (4.4). Furthermore, some new examples are shown and used for an update of the parameters. We concentrate on the case of $\gamma = 1$ and additionally suppose that at each time t one example of each class, denoted by $x_{t,c}$, is available. Thus, our estimate of the empirical risk (4.2) at moment t is

$$\hat{R}_{emp}(t) = \frac{1}{\nu} \sum_c [d_c(x_{t,c})]_- \tag{6.1}$$

We perform a stochastic gradient descent, with updates

$$(w_{t+1}, \rho_{t+1}) \leftarrow (w_t, \rho_t) - (\partial_w, \partial_\rho)(\Omega + \hat{R}_{emp}(t)). \tag{6.2}$$

Calculating the partial derivatives ∂_{w_c} and ∂_{ρ_c} is straightforward, but we will discuss the derivative ∂_{w_c} in more detail. We get

$$\partial_{w_c}\Omega = w_c - \frac{1}{m-1} \sum_{d \neq c} w_d \tag{6.3}$$

and for the derivative of the estimate of the empirical risk

$$\partial_{w_c}\hat{R}_{emp}(t) = \frac{1}{\nu}\big(\theta_c(t)k(x_{t,c}, \cdot) - \frac{1}{m-1} \sum_{d \neq c} \theta_d(t)k(x_{t,d}, \cdot)\big), \tag{6.4}$$

with the function θ_c given by

$$\theta_c(t) = \begin{cases} 1 & \text{if } d_c(x_{t,c}) < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6.5}$$

We note that the negative part function $[\cdot]_-$, which takes the value $-x$ if the argument $x < 0$ and zero otherwise, is not differentiable in 0. But let us for the moment suppress this detail. From section 4.2.5 we know that the final solution has the advantage that each vector w_c is expressible as a linear combination of the data in its class only. But the components of the gradient vector at a non optimal point do not share this property, as one can see from (6.3) and (6.4). We profit now from the restriction to the case $\gamma = 1$ and recall the translation invariance 4.2.3 of the decision function and the regulariser. Accordingly, we can add the same vector v to each of the w_c without changing the decision function nor the value of the cost function. Therefore, if at each update (6.2) we add

$$v = \frac{1}{m-1} \sum_d \big(w_{t,d} + \theta_d(t)k(x_{t,d}, \cdot)\big) \tag{6.6}$$

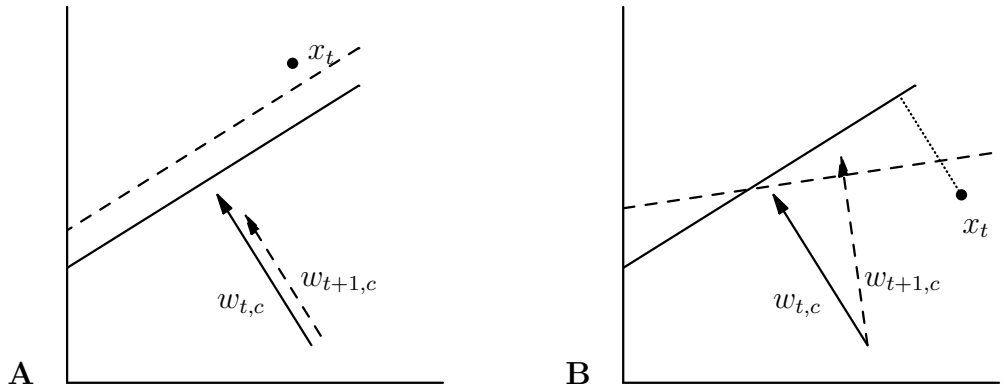


Figure 6.1: Illustration of the learning rule, shown is the estimate at time t of the representative vector $w_{t,c}$ of class c (solid arrow) and the zero level of the decision function d_c (solid line). In **A** the new point x_t is correctly classified, and the updated representative vector $w_{t+1,c}$ and zero level are shown (dashed). In **B** the new point is on the wrong side of the margin, as indicated by the dotted line. Therefore $w_{t+1,c}$ is turned in direction of x_t and the margin is reduced (dashed line).

to each $w_{t+1,c}$, the representative vectors remain expressible as a linear combination of the data of their class only and we derive the following simple update rule:

$$w_c \leftarrow (1 - \eta\tfrac{m}{m-1})w_c + \begin{cases} \eta\tfrac{1}{\nu}\tfrac{m}{m-1}k(x_{t,c}, \cdot) & \text{if } d_c(x_t) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.7}$$

$$\rho_c \leftarrow \rho_c + \eta - \begin{cases} \eta\tfrac{1}{\nu} & \text{if } d_c(x_t) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.8}$$

with $w_c = \sum_s \alpha_s k(x_{s,c}, \cdot)$. The above learning rule has a the following interpretation, see figure 6.1. The first term on the right hand side of (6.7), and the first two terms of (6.8), respectively, decreases $w_{t,c}$ and increase $\rho_{t,c}$ at each iteration. These terms correspond to the partial derivatives of the regulariser and will increase the separation of the classes. On the other hand, the last terms correspond to the empirical risk. They are zero, if at time t example $x_{t,c}$ is classified correctly. If not, then we move the vector $w_{t,c}$ in direction of $x_{t,c}$ and decrease the margin $\rho_{t,c}$. Note how the ν–property is reflected in the update rule (6.8). At each step we increase $\rho_{t,c}$ by the learning rate η, but if an example if rejected by error, then the margin is decreased by $\eta 1/\nu$. Therefore, we will always find approximately a fraction of ν of outliers.

Let us take a look at our estimate $w_{t,c}$ of the representative vector at time t. We can write it as an expansion

$$w_{t,c} = \sum_s \alpha_s k(x_s, \cdot). \tag{6.9}$$

From section 4.2.5, we know that the coefficients should sum to one, $\sum_s \alpha_s = 1$. We argue that this is the case for our online algorithm, too. Let us define a random variable

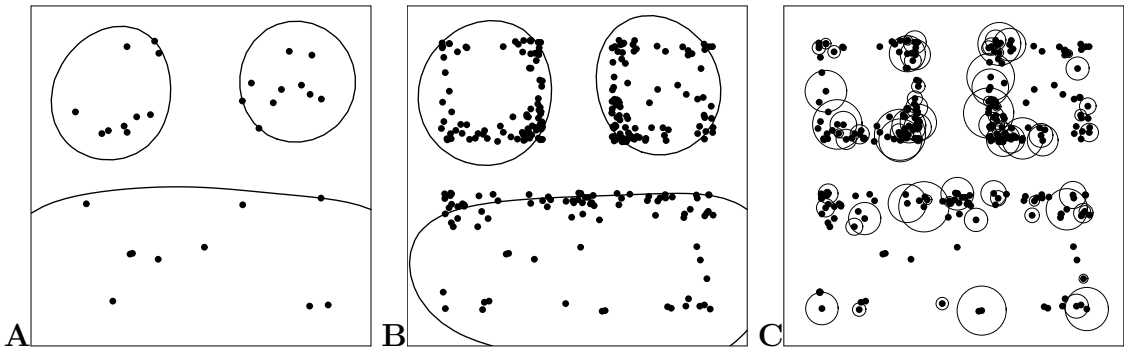


Figure 6.2: Toy experiment: **A** shows the initial situation. In **B**, the estimates of the zero level of the decision function d_c of each class are shown by the solid line after 800 iterations. In **C**, the support vectors are shown. The circles indicate the value of the corresponding multiplier. Parameters: $\eta = 0.005$, $\nu = 0.1$.

S_t, which is the sum of the expansion coefficients of $w_{t,c}$ at time t,

$$S_t = \sum_s \alpha_s, \tag{6.10}$$

with α_s given by (6.9). Suppose now that $S_t = 1$, and we show that the expected value for S_{t+1} is equal to one, under the assumption that all points are drawn from a fixed distribution. Because the online algorithm fulfils the ν–property, a new point of class c is on the correct side of the margin with probability $1 - \nu$. This will decrease S_t by a factor $(1 - \eta\, m/(m-1))$, by (6.7). On the other hand, the new point is with probability ν on the wrong side of the margin and this decreases S_t as above, but adds a new coefficient with value $\eta 1/\nu\, m/(m-1)$. Together, the expectation of S_{t+1} is given by

$$E(S_{t+1}) = (1-\nu)(1 - \eta\frac{m}{m-1})S_t + \nu(1 - \eta\frac{m}{m-1})S_t + \nu\eta\frac{1}{\nu}\frac{m}{m-1} = 1. \tag{6.11}$$

One can see that if an example $x_{t,c}$ has $d_{t,c}(x_{t,c}) < 0$, then it will be and remain a support vector. Therefore, we expect the number of support vectors to be considerably higher than the number of support vectors of the exact solution. Even worse, if we continue to show new data points, the expansion continues to grow. A possible solution is to truncate the expansion after a fixed number of terms, as proposed in [44], where the truncation error can be estimated by a direct calculation. Note further that the computation cost for each iteration is at least the prediction cost, as we have to predict the class of the examples available at that time step.

6.3 Experiments

We start with a toy example in two dimensions. The examples of each class are drawn from a uniform distribution with rectangular support. We used a Gaussian kernel. Figure

	false accept	false reject	number SV
exact solution	75	245	580
stoch grad	229	179	1027

Table 6.1: The same experiment on the USPS handwritten digits dataset as described in section 5.2. The result from table 5.1 of the exact solution (first line) is compared to a stochastic gradient solution (second line), where we ran 10^3 iterations with $\eta = 10^{-4}$. The second and third column show the number of false accepted and rejected test examples, respectively. The fourth column gives the number of Support Vectors. For more details see the text.

6.2.A shows the initial situation, with each w_c initialised to the mean of ten samples. Then we run our online algorithm for 800 iterations, and after that the estimate of the solution is shown in figure 6.2.B. The solid line shows the zero level of the decision function of each class. For each of the closed solid lines, points inside are attributed to belong to the class, where points outside not. The points show the support vectors. One can easily see, how the support vector tend to be close to the border. Note that most of the multipliers corresponding to the support vectors are actually close to zero, which can be seen in C. Here, the size of the circles around each support vector is proportional to the size of the corresponding multiplier.

Next we use the dataset of handwritten digits and perform the same experiment as described in section 5.2. We initialise each representative vector $w_{0,c}$ to be the mean of 100 randomly chosen examples of class c. At each iteration, we show one example of each class, randomly chosen among all examples of the class. Table 6.1 shows the results and compares them to the exact solution of section 5.2. We can see that the error is higher than the one of the exact solution and, as expected, has considerably more support vectors.

6.4 Discussion

In this chapter we derived a stochastic gradient decent rule to solve the multi–class optimisation problem 4.2. As expected, we found more support vectors as for the exact solution. To circumvent this problem, the solution could be truncated [43]. On the other hand, the algorithm is appealing, because it has a direct interpretation and is very easily implemented.

Chapter 7

Unsupervised Learning

In unsupervised learning one is concerned with the problem of finding structure in given data. In this chapter we modify our algorithm to categorise labelled data into an algorithm that estimates clusters of the data. The data is equipped with a kernel measuring the pairwise similarity of data examples. We start the chapter by reviewing related unsupervised learning algorithms and clustering strategies. We will focus on approaches close to ours, especially on algorithms using kernels. In section 7.1 we propose the initial idea of our kernel clustering algorithm. We will see that we have to modify our first approach to get a useful cost function. We propose two possible modifications in sections 7.2 and 7.3 and show an example application.

7.1 Introduction

The kernel trick, reviewed in chapter 2, paved the way to use nonlinear kernels and turn algorithms that extract linear structure in data into algorithms that detect nonlinear structure. Examples of this are kernel Principal Component Analysis [64], kernel Independent Component Analysis [6], or the estimation of the support of a distribution [62]. Different kernel clustering algorithms can be found in the literature [30, 31, 9]. In [30], each point is mapped into a feature space and clusters c are estimated by estimating their centres m_c. The centres are estimated in feature space, such that the average squared norm, averaged over all clusters and points belonging to the cluster, is minimised. The clusters in feature space can be learnt implicitly by using a kernel. An other kernel clustering algorithm was proposed by Graepel and Obermayer [31]. It is similar to the above mentioned algorithm, but a further neighbourhood relation is introduced among the clusters. This leads to a topographic arrangement of the clusters.

In this chapter we propose two unsupervised algorithms. Both are modifications of an initial optimisation problem. They can be interpreted as kernel clustering algorithms. The main idea was published in [13]. The difference to the above algorithms is that not every point belongs to one of the clusters. The idea is to estimate multiple vector and margin pairs, each separating a fraction of the data from the rest and thus defining a cluster. The cost function penalises two such vectors if they point in a similar direction, but we put no constraint on orthogonality of these vectors, unlike for example in Principal Component Analysis. In more detail, we are given a set of data examples

$$S = \{x_1, .., x_l\} \tag{7.1}$$

from a nonempty set X. Let k denote a kernel on $X \times X$ and ϕ denotes the mapping from

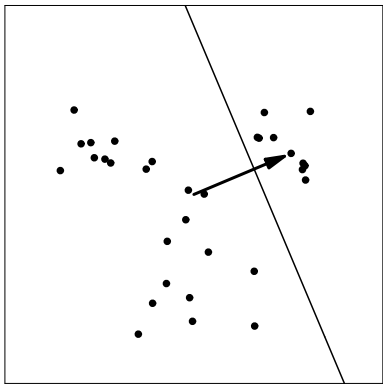


Figure 7.1: Extracting one linear feature: Shown are the data points, the vector w_1 and the hyperplane $w_1 \cdot x - \rho_1 = 0$, which separates one cluster from the rest of the data.

X into the Hilbert H space induced by k. We recall the relation $\phi(x_i) \cdot \phi(x_j) = k(x_i, x_j)$. Here, the $\cdot$ denotes the scalar product in the Hilbert space H. The general idea is the following. Consider a vector w_c and a margin $\rho_c \in \mathbb{R}$. Then this pair defines a cluster, namely all points for which the function

$$d_c(x) = w_c \cdot \phi(x) - \rho \tag{7.2}$$

is larger than zero. Figure 7.1 shows one vector with its margin separating one cluster from the rest of the data. A candidate to learn such a pair of vector and margin is the cost function of the one–class support vector machine [62], which tries to separate the data as far as possible from the origin, with a fraction ν of outliers. Let us denote this cost function by

$$f_c(w_c, \rho_c) = \frac{1}{2}||w_c||^2 - \rho_c + \frac{1}{\nu l}\sum_i [d_c(x_i)]_-. \tag{7.3}$$

Now, the following two questions arise.
Question 1: This cost function allows to learn only one pair of vector and margin. We have to find a way to combine several of these cost functions to learn multiple vector and margin pairs.
Question 2: As shown in figure 7.1, we would like to separate a fraction of the data only to form a cluster. Even though we can adjust the number of outliers by the parameter ν in the last term of (7.3), they are penalised by this term. Therefore, it is not sure if by choosing a large fraction ν of outliers we can learn a meaningful cluster.

Let us start by tackling the first question. We can learn multiple pairs by just adding multiple cost functions. But by adding the cost functions, we have independent optimisation problems for each pair and thus, we will find the same solution for each of the

pairs (w_c, ρ_c). To find different pairs, we add a penalty term

$$G(w_1, ..., w_m) = \sum_{c,d \neq c} w_c \cdot w_d, \tag{7.4}$$

penalising pairwise vectors pointing in the same direction. Together, the optimisation problem reads

$$\text{minimise} \quad \sum_c f_c + \gamma G. \tag{7.5}$$

The minimisation is done over all $((w_1, \rho_1), ..., (w_m, \rho_m)) \in (H \times \mathbb{R})^m$. The parameter γ weighs the penalty term G.

To answer the second question, we analyse the optimisation problem (7.5). Let us denote by m the number of vector and margin pairs we would like to learn. Furthermore, we denote by

$$F : (H \times \mathbb{R})^m \to \mathbb{R} \tag{7.6}$$

the cost function of (7.5). We show in 7.5 at the end of the chapter that for a suitable choice of the parameter γ the cost function F is bounded from below and convex. Let us choose such a value for γ. Suppose we have found a solution $(w_1, \rho_1), ..., (w_m, \rho_m)$ of (7.5). We choose two indices, i, j and interchange the pairs (w_i, ρ_i) and (w_j, ρ_j) along a straight line $\tau : [0, 1] \to (H \times \mathbb{R})^m$,

$$\begin{aligned} \tau(t) = (1-t)((w_1, \rho_1), ..., (w_i, \rho_i), ..., (w_j, \rho_j), ..., (w_m, \rho_m)) + \\ t((w_1, \rho_1), ..., (w_j, \rho_j), ..., (w_i, \rho_i), ..., (w_m, \rho_m)). \end{aligned} \tag{7.7}$$

We note that the cost function is invariant to the interchange of two vector and margin pairs. This implies $F(\tau(0)) = F(\tau(1))$. But because F is convex, F takes the same value at all points sitting on the straight line between these two points, that is,

$$F(\tau(t)) = F(\tau(0)), \quad \text{for all } t \in [0, 1]. \tag{7.8}$$

Let us denote by $(\bar{w}, \bar{\rho})$ the mean of all the pairs (w_l, ρ_l) from our solution. Continuing with the same argumentation, we can show that the value of F on our solution is equal to the value of the cost function at the point $((\bar{w}, \bar{\rho}), ..., (\bar{w}, \bar{\rho}))$. Therefore, by solving (7.5) we can always find a solution in which all the vectors and margins are equal. But such a solution is not very meaningful.

To solve this problem, we propose two modifications. In section 7.2 we modify the empirical risk term, the last term in (7.5), such that only a fraction of points are penalised by the empirical risk of each cluster. The second proposed solution is then discussed in section 7.3, where we calculate and modify the dual problem of (7.5). In addition, a third modification is used in chapter 9. Because the situation is slightly different, we explain the details there.

7.2 Modifying the cost function

In this section we propose a modification of our initial cost function (7.5). The idea is very simple. We constrain each point to belong to maximally one cluster. We do this by introducing a binary value $\lambda_{c,i} \in \{0,1\}$ indicating if the point x_i belongs to cluster c or not. We denote by l_c the number of points in cluster c. Not every point belongs to one of the clusters, and the fraction of points not belonging to any of the clusters is controlled by the parameter $\nu_{tot} \in [0,1]$. For simplicity, we choose equally many points in each of the clusters. In summary, we have the following constraints:

$$\lambda_{c,i} \in \{0,1\}, \qquad \sum_i \lambda_{c,i} = l_c, \quad \text{and} \quad l_c = \Big\lfloor \frac{(1-\nu_{tot})l}{m} \Big\rfloor, \tag{7.9}$$

with $\lfloor \cdot \rfloor$ denoting the largest integer smaller or equal to the argument. As in our initial idea, a cluster is defined by a separating hyperplane allowing a fraction ν of outliers. Using the indicators $\lambda = (\lambda_{c,i})_{c,i} \in \{0,1\}^{ml}$ and using the shorthand $(w,\rho) = ((w_1,\rho_1),...,(w_m,\rho_m))$ the empirical risk term reads

$$R_{emp}(w,\rho,\lambda) = \sum_c \frac{1}{\nu l_c} \sum_i \lambda_{c,i}[w_c \cdot \phi(x_i) - \rho_c]_-. \tag{7.10}$$

In words, a point x_i yields a positive empirical risk if it belongs to cluster c, that is, $\lambda_{c,i} = 1$, but is on the wrong side of the margin, $w_c \cdot x_i < \rho$. We use a regulariser, which is just the remaining part of (7.5) but with a small modification. We penalise two vectors w_c, w_d only if the angle between them is less than $\pi/2$. This makes the optimisation problem slightly more complicated. However, as we will use a gradient method, it is still tractable. Explicitly, the regulariser is

$$\Omega(w,\rho) = \frac{1}{2} \sum_c ||w_c||^2 + \gamma \sum_{c,d \neq c} [w_c \cdot w_d]_+ - \sum_c \rho_c. \tag{7.11}$$

The interpretation of the regularising term is as follows. We seek vectors w_c of small norm and large margins ρ_c, the first and the last term in (7.11). Together they try to separate the clusters as far away from the origin as possible. The middle term puts a penalty on vectors pointing in a similar direction and helps to get clusters sitting in different directions of the data space. In summary, the optimisation problem reads

$$\begin{aligned} &\text{minimise} && \Omega + R_{emp} \\ &\text{subject to} && \lambda_{c,i} \in \{0,1\}, \qquad \sum_i \lambda_{c,i} = l_c. \end{aligned} \tag{7.12}$$

The minimisation is done over $(w,\rho,\lambda) \in (H \times \mathbb{R})^m \times \{0,1\}^{ml}$. In the next subsection, we describe how we find an approximate solution of the optimisation problem (7.12).

7.2.1 Two–step optimisation

We propose a two–step procedure to optimise (7.12). The algorithm proceeds by alternating the two steps, until it has converged or a maximal number of iterations was performed. We recall that the variables are the cluster indicators $\lambda_{c,i}$, the vectors w_c and the margins ρ_c. Our method is similar in spirit to the k–means algorithm [25]. Mathematically speaking, it is a coordinate descent: In the first step, we fix the variables $\lambda_{c,i}$ and ρ_c and update the variables w_c with a stochastic gradient descent, as in section 6.2. In the second step, we fix the variables w_c and update the variables $\lambda_{c,i}$ and ρ_c. Because the variables $\lambda_{c,i}$ are binary, we do not use a gradient method but rather we will see that we can directly guess a close to optimal point. In detail, the two steps are the following.

Step one: Let the variables $\lambda_{c,i}$ and ρ_c be fixed. We will compute the partial derivative of the cost function (7.12) and perform an update

$$w_c \leftarrow w_c - \eta \partial_{w_c}(\Omega + R_{emp}), \tag{7.13}$$

with learning rate η. For efficiency, we do not use all the data to estimate the empirical risk, but rather use only a subset of s examples. We draw these examples randomly from our dataset. Let us denote the set of chosen examples at step t by S_t. A short calculation gives the update rule

$$w_c \leftarrow (1-\eta)w_c - \eta\gamma \sum_{d:w_c \cdot w_d > 0} w_d + \eta \frac{m}{\nu(1-\nu_{tot})s} \sum_{x_i \in S_t} \theta_{c,i}\lambda_{c,i}\phi(x_i). \tag{7.14}$$

Here $\theta_{c,i} \in \{0,1\}$ is zero if $w_c \cdot x_i > \rho_c$ and 1 otherwise. We recall that ν_{tot} is the relative number of examples, which do not belong to any cluster. The value ν is the relative number of points in each cluster, which are on the wrong side of the margin. The integer s is the number of examples in S_t. The factor $m/\nu(1-\nu_{tot})s$ in front of the last term in (7.14) is just one over the expected number of examples in S_t, which belong to cluster c and are on the wrong side of the margin.

We want to mention a technical detail. The negative part function $[\cdot]_-$, which assigns x the value $-x$ if $x < 0$ and zero otherwise, is not differentiable at 0. By our algorithm described below there are points x_i with $w_c \cdot x_i = \rho_c$. In rare cases, for example when $\lceil \nu l_c \rceil = 1$ this yields problems. Therefore, in these cases the negative part function has to be replaced in a small neighbourhood of zero, such that it is differentiable.

Step two: In this step we fix the variables w_c and optimise with respect to the variables $\lambda_{c,i}$ and ρ_c. The algorithm will proceed by first choosing good values for the indicators $\lambda_{c,i}$ and then given these values, determine the optimal margins ρ_c.

To motivate the choice of good values for the indicators, we first look at the optimal value given the binary values $\lambda_{c,i}$. Thus, suppose each point x_i is assigned to one cluster according to (7.9). Setting the optimal margin ρ_c is now an independent problem for each cluster c. In analogy to the ν–property discussed in section 4.2.6, the optimal values of

the margins are set as follows. For a cluster c define $a_{c,i} = w_c \cdot x_i$ for all x_i belonging to c and sort the values $a_{c,i}$ in ascending order,

$$a_{c,j_1} \leq a_{c,j_2} \leq ... \leq a_{c,j_{l_c}}. \tag{7.15}$$

Then we set

$$\rho_c = a_{c,j_n}, \quad \text{with} \quad n = \lceil \nu l_c \rceil. \tag{7.16}$$

Hence, ρ_c is sitting at the ν–quantile of the empirical distribution of $\{a_{c,j_1}, ..., a_{c,j_{l_c}}\}$, see figure 4.3.

From the above we see that suitable values for the indicators $\lambda_{c,i}$ are values that allow large values of ρ_c, because the margins have to be maximised by (7.11). According to (7.16), we should choose points x_i to belong to cluster c if their dot product with w_c is large. For this we chose the simplest possible strategy. We iterate over all clusters. For the first cluster we calculate the values $a_{c,i} = w_1 \cdot x_i$ for all points x_i and select l_c points with a maximal value of $a_{c,i}$. For these points we set the indicator $\lambda_{1,i} = 1$. Now suppose the indicators for clusters 1 to $c-1$ are set. Then for cluster c we calculate the values $a_{c,i}$ for all points x_i, which do not already belong to one of the previous clusters. Again, we set the indicators $\lambda_{c,i} = 1$ for the points x_i with maximal values $a_{c,i}$. After the indicators are set for all clusters, the optimal margins ρ_c are determined according to (7.16) and step two is finished.

From the point of view of efficiency, we can adjust the complexity of the first step by choosing a suitable fraction of the data to estimate the empirical risk term. In step two, all the scalar products of the data with the vectors w_c have to be computed. Therefore, the complexity of one iteration is of the order of ml.

7.2.2 Example

We show a toy problem in two dimensions. We sampled 40 points from four uniform distributions with a rectangular support. The points are shown as the dots in figure 7.2. The data has zero mean. Suppose we are given this data, but do not know its structure. We try to find three clusters, which give us information about the structure of the data. We use the parameters $\gamma = 1/4$, $\nu_{tot} = 0.4$ and $\nu = 0.3$. Hence, 60% of the data belong to one of the cluster and in each cluster 30% are on the wrong side of the margin. The learning rate was $\eta = 0.01$. We run 200 iterations. The vectors w_c found by optimisation are shown in the figure by the three arrows. The perpendicular lines are the hyperplanes $\{x | w_c \cdot x = \rho_c\}$. The circles indicate the points belonging to one of the clusters.

7.3 Modifying the dual problem

In this section we show a second modification of our initial optimisation problem (7.5). For this, we calculate the dual problem, which can be easily interpreted. The interpretation will show us that a change of the sign in the dual formulation can resolve the problems

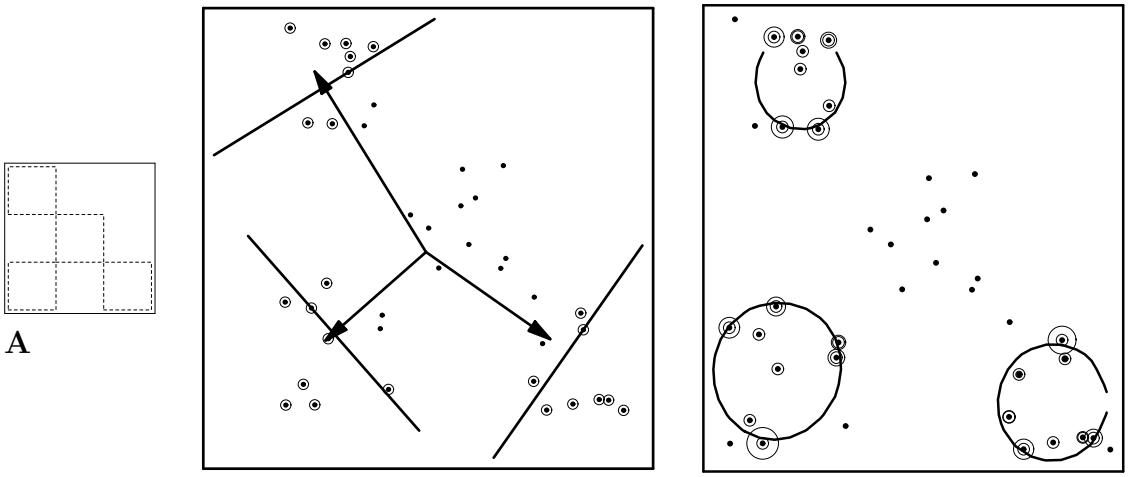


Figure 7.2: A toy problem: We defined four uniform distributions with supports as indicated in **A**. The dots shown in **B** and **C** are the data points consisting of ten samples of each distribution. We plotted the solution for three clusters. In **B** with the standard scalar product as kernel. Each cluster is represented by its vector w_c (arrow) and the margin. In **C** the solution for a Gaussian kernel is shown. The lines are the preimages of the margin hyperplanes. The circles indicate the points belonging to a cluster. The second larger circles in **C** are proportional to the size of the coefficients, used in the expansion of the solution. See the text for details.

mentioned above in questions 1 and 2 on page 48. In section 7.5 we derive the dual problem of (7.5), which is

$$\begin{aligned} \underset{\alpha_{c,i}}{\text{maximise}} \quad & -\frac{1}{2}\sum_{c,d,i,j}\alpha_{c,i}\alpha_{d,j}y_{cd}k(x_i,x_j) \\ \text{subject to} \quad & 0 \le \alpha_{c,i} \le \frac{1}{\nu l} \ , \quad \sum_i \alpha_{c,i} = 1, \end{aligned} \tag{7.17}$$

with

$$y_{cd} = \begin{cases} \gamma_- & \text{if } c = d \\ \gamma_+ & \text{if } c \neq d \quad . \end{cases} \tag{7.18}$$

Here, the parameters γ_-, γ_+ are related to γ according to (7.32) in section 7.5. The dual variables $\alpha_{c,i}$ are related to the primal variables w_c by

$$w_c = \sum_{d,i}\alpha_{c,i}y_{cd}\phi(x_i), \tag{7.19}$$

see (7.33) in section 7.5, and each margin ρ_c is set to be the $\lceil \nu l \rceil$–smallest element of $\{w_c \cdot \phi(x_i) | i = 1, ..., l\}$.

Lemma 5 in section 7.5 implies, that the dual problem is equivalent to the primal problem (7.5) for $0 < \gamma < 1$. With lemma 6 in section 7.5, we derive $\gamma_- > 0$ and $\gamma_+ < 0$. In this case, maximising (7.17) means the following: For a cluster c we have to choose values $\alpha_{c,1}, ..., \alpha_{c,l}$ such that their sum is equal to one, and such that

$$\sum_{i,j} \alpha_{c,i}\alpha_{c,j}k(x_i, x_j) \tag{7.20}$$

is small. On the other hand, for two clusters c, d with $c \neq d$ the values $\alpha_{c,i}, \alpha_{d,j}$ should be chosen such that

$$\sum_{i,j} \alpha_{c,i}\alpha_{d,j}k(x_i, x_j) \tag{7.21}$$

is large. Therefore, at an optimal solution, two multipliers $\alpha_{c,i}, \alpha_{c,j}$ of the same cluster should be large if the points x_i, x_j are dissimilar. On the other hand, two multipliers $\alpha_{c,i}, \alpha_{d,j}$ of two different clusters should be large if the points are similar. One possible solution is that for a given index i all the multipliers $\alpha_{1,i}, ..., \alpha_{m,i}$ belonging to this point are equal. This implies that all the vectors $w_1, .., w_m$ are equal, exactly as we found in 7.8.

This is exactly the opposite of what a clustering algorithm should do. We would expect that the vector w_c of cluster c is expressed as a linear combination of similar points. Thus, the multipliers $\alpha_{c,i}, \alpha_{c,j}$ should be large if the points x_i, x_j are similar. On the other hand, another vector w_d should be expressed as a linear combination of points, dissimilar to the points used to express w_c. To do so, the easiest modification of (7.17) is to change the sign of the constants γ_- and γ_+, which changes the sign of the optimisation problem. Thus, in the following we will chose $\gamma_- < 0$ and $\gamma_+ > 0$. Furthermore, we will later translate the data in feature space so that it has a mean of zero. This allows the clusters to sit in different directions in feature space.

Let us take a look at what happens by changing the sign of the constants γ_-, γ_+. We consider the $ml \times ml$ matrix $\tilde{K}$ with entries

$$\tilde{K}_{cd,ij} = y_{cd}k(x_i, x_j). \tag{7.22}$$

This is the matrix appearing in the dual problem (7.17).

Lemma 4 *Fix $\gamma_- < 0$. Choose $\gamma_+ \in [0, -1/(m-1)\gamma_-]$. Then the matrix $\tilde{K}$ in (7.22) is negative definite.*

The proof can be found in section 7.5. From this lemma we conclude that by choosing the parameters $\gamma_- < 0$ and $\gamma_+ \in [0, -1/(m-1)\gamma_-]$, the cost function in the dual problem (7.17) is convex. Thus we try to maximise a convex function. This implies that the optimal values of the multipliers $\alpha_{c,i}$ lie at the border of the range allowed by the constraints. In other words, at optimum each of the multipliers is either zero or equal to $\frac{1}{\nu l}$. Note that strictly speaking, this is only true if νl is an integer, otherwise there might be some exceptions. This observation will lead us to a simple algorithm to optimise the dual problem. Note that in the above mentioned parameter range, we still have the link

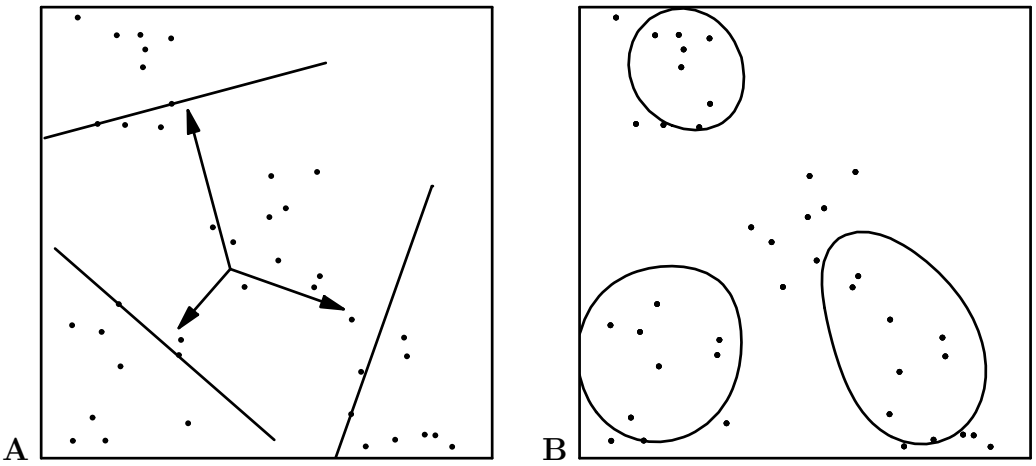


Figure 7.3: Toy example with $m = 3$ clusters and the same data as in figure 7.2. In **A** for $\gamma_- = -2$ and $\gamma_+ = 1$ we find three different vector and margin pairs. **B** shows the solution found for a Gaussian kernel. The three lines shown are the preimages of the three hyperplanes $w_c \cdot x - \rho_c = 0, \quad c = 1, 2, 3$ in the Hilbert Space H. We used $\nu = 0.8$.

to the primal variables 7.19, but we are optimising the modified dual problem, which is different from the initial problem (7.5).

7.3.1 Finding a local maximum

In this section we describe an algorithm to find a local maximum of the cost function (7.17). As we argued above, we are faced with solving a binary quadratic program. In general, the maximisation of an unconstrained quadratic program in zero–one variables

$$\begin{aligned} \underset{\alpha}{\text{maximise}} \quad & \alpha^T Q \alpha \\ \text{subject to} \quad & \alpha \in \{0,1\}^n, \end{aligned} \tag{7.23}$$

is a classical NP–hard combinatorial optimisation problem, even if the matrix Q is positive definite [2]. We will not try to find a global maximum, but present a simple algorithm to find a local maximum. In our experiments, we found that different local maxima are qualitatively equivalent and we found no need to find the best of them.

Parameters: We start by choosing suitable parameters. The parameter $\nu \in [0, 1]$ controls the fraction of data separated by each vector and margin pair. Let us choose for convenience a value ν such that νl is an integer. Then we choose m the number of vector and margin pairs to be estimated and finally we choose $\gamma_- < 0$ and $\gamma_+ \in [0, -1/(m-1)\gamma_-]$.

Mean zero in feature space: To evaluate the cost function, we have to compute the kernel evaluations $k(x_i, x_j)$. Note that we want the data to have zero mean in feature

space. Therefore, if the data is not already centred in feature space, we do this implicitly by replacing the kernel evaluation $k(x_i, x_j)$ by

$$k'(x_i, x_j) = k(x_i, x_j) - \frac{1}{l}\sum_{j'} k(x_i, x_{j'}) - \frac{1}{l}\sum_{i'} k(x_{i'}, x_j) + \frac{1}{l^2}\sum_{i',j'} k(x_{i'}, x_{j'}), \qquad (7.24)$$

see [64]. If the matrix $(k'(x_i, x_j))_{i,j}$ is too large to be stocked in memory, we store the sums $\sum_{j'} k(x_i, x_{j'})$ and $\sum_{i'} k(x_{i'}, x_j)$ and recompute (7.24) when needed.

Initialisation: We initialise the algorithm by choosing for each $c \in \{1, ..., m\}$ a fraction ν of the data randomly and setting their multipliers $\alpha_{c,i}$ to the maximal value $\frac{1}{\nu l}$ allowed by the constraints.

Optimisation loop: In order to optimise (7.17) we proceed in a way similar to the Sequential Minimal Optimisation algorithm [53]. At each iteration of the algorithm we choose two different pairs of indices $(c, i), (d, j) \in \{1, ..., m\} \times \{1, ..., l\}$ and try to find the optimal values for $\alpha_{c,i}, \alpha_{d,j}$ while keeping all the other $\alpha_{c',i'}$ fixed. If the two values $\alpha_{c,i}, \alpha_{d,j}$ are both zero or at maximum, we cannot change anything without violating the constraints. Otherwise, we check if the cost function takes a higher value with the actual choices of $\alpha_{c,i}, \alpha_{d,j}$ or with the values interchanged, and we change the values of $\alpha_{c,i}, \alpha_{d,j}$ accordingly. We iterate this step until no changes are made anymore.

In that case we have found values $\alpha_{c,i}$, which lead to a high value of the cost function. Unfortunately, we are not sure to have found the global maximum.

Computing w_c and ρ_c: With the values of $\alpha_{c,i}$ we can compute the corresponding vectors w_c by using (7.31) and (7.33). Then we compute the scalar product of each data point mapped into feature space $\phi(x_i)$ with each of the vectors w_c and set ρ_c to be equal to the νl-smallest value of the set $\{\phi(x_i) \cdot w_c | i \in \{1, ..., l\}\}$, that is, to the ν–quantile, see figure 4.3.

From the description of the optimisation loop we can derive the complexity of one loop. For each cluster we have to check all pairs of multipliers, such that one of them is zero and the other one not. Therefore, in one loop of optimisation we check $m(\nu l)(l - \nu l)$ many pairs. The complexity of one loop is of the order of ml^2.

7.3.2 Experiments

We look first at a toy example in two dimensions. Figure 7.3 shows the solution found for three clusters, in A with the standard scalar product as kernel and in B with a Gaussian kernel.

Next, we tested the algorithm on the US postal service database of handwritten digits. The digits are images of size 16×16. For convenience, we used only a subset of the whole database consisting of 100 randomly chosen examples of each of the digits 0,1 and 2. The data was normalised as in chapter 5. Now suppose one did not know that the database

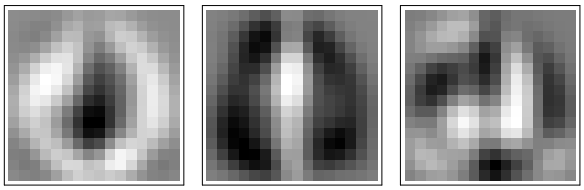


Figure 7.4: The three vectors w_c found with a bilinear kernel, and the parameters $m = 3$, $\gamma_- = -2$, $\gamma_+ = 1$ and $\nu = 0.8$.

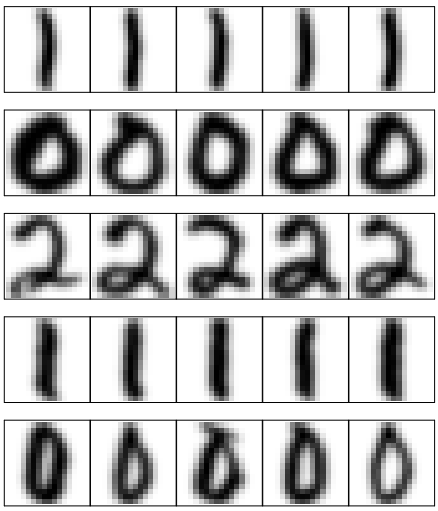


Figure 7.5: Each line shows the elements of a cluster found with a Gaussian kernel with $\sigma^2 = 0.1$, $m = 5$,$\gamma_- = -2$,$\gamma_+ = 1$ and $\nu = 0.98$.

consisted of the digits 0,1 and 2. We wanted to know if our algorithm could detect this structure in the database. We first choose the kernel to be the standard scalar product in $\mathbb{R}^n$. We set $m = 3$, and $\nu = 0.8$, that is, each vector and margin pair separates one cluster, consisting of 20% of the data from the rest. Figure 7.4 shows the three vectors w_c found by our algorithm. Note that black stands for negative values, white for positive ones and zero is a grey level in between. One immediately sees that the first vector is a detector for zeros. Forming the scalar product with a digit 0 and w_1 yields large values, while the black part makes the product with the digits 1 and 2 as small as possible. In analogy, the second vector detects digit 1 and the third digit 2. Thus, from the solution found we can see the structure of the dataset.

In the second experiment, we used a Gaussian kernel with $\sigma^2 = 0.1$. We choose a large value for ν, such that each vector and margin pair cuts off 5 digits from the whole dataset. Furthermore, we choose $m = 5$ to have five of these small clusters. In figure 7.5 each line shows the five vectors forming a cluster. We see that from these clusters we can learn a lot about the structure of the dataset. In each of the clusters are only the same digits. Even more, the clusters consists of the same digits of the same style, for example,

in the second row are bold and large zeros, whereas in the last row are thin and slim zeros.

7.4 Conclusion

In this chapter we proposed two new unsupervised kernel algorithms, which can be interpreted as clustering algorithms. To optimise the cost function of the former, we developed a two–step algorithm similar in spirit to the classical k–means algorithm. The second algorithm optimises a binary quadratic program. Solving a binary quadratic program is a classical combinatorial optimisation problem. Typical to our approach are the constraints set on the variables. The optimisation procedure is appealing by its simplicity, but for large datasets it is not efficient enough.

7.5 Details and Proofs $*$

We will give the proofs of the lemmata in the text:

Lemma 5 *For $0 < \gamma \leq 1$ the function $\sum_c g_c(w)$ is bounded from below and convex*

Proof : We calculate

$$\begin{aligned} \sum_c g_c(w) &= \sum_c ||w_c||^2 + \gamma \sum_c w_c \cdot (\sum_d w_d - w_c) \\ &= (1-\gamma) \sum_c ||w_c||^2 + \gamma || \sum_c w_c||^2. \end{aligned} \tag{7.25}$$

From this we see that $\sum_c g_c$ is bounded from below if $1-\gamma$ is positive. For convexity choose $t \in [0,1]$ and another vector $v = (v_1, ..., v_m)$. One calculates

$$t g_c(w) + (1-t) g_c(v) - g_c(tw + (1-t)v) = t(1-t) g_c(w-v) \tag{7.26}$$

and because of (7.25) the sum over all c of (7.26) is greater or equal to zero for $1-\gamma \geq 0$. ■

To determine the dual of (4.7) we consider the Lagrangian

$$\begin{aligned} L(w, \rho, \xi) =& \frac{1}{2} \sum_c f_c + \frac{1}{2} \gamma \sum_{c, d \neq c} w_c \cdot w_d \\ & - \sum_{c,i} \alpha_{c,i} (w_c \cdot \phi(x_i) - \rho_c + \xi_{c,i}) - \sum_{c,i} \beta_{c,i} \xi_{c,i} \end{aligned} \tag{7.27}$$

with multipliers $\alpha_{c,i}, \beta_{c,i} \geq 0$. Setting the derivatives with respect to $w_c, \rho_c, \xi_{c,i}$ equal to zero yields

$$v_c = w_c + \gamma_+ \sum_{d \neq c} w_d \tag{7.28}$$

$$\alpha_{c,i} \leq \frac{1}{\nu l} - \beta_{c,i} \leq \frac{1}{\nu l} \tag{7.29}$$

$$\sum_i \alpha_{c,i} = 1, \tag{7.30}$$

with

$$v_c = \sum_i \alpha_{c,i} \phi(x_i). \tag{7.31}$$

We want to solve (7.28) for w_c and then use this with (7.29) and (7.30) to eliminate the primal variables w, ρ, ξ from the Lagrangian. Let $m > 1$ and let us denote by $M(1, \gamma) \in \mathbb{R}^{m \times m}$ the matrix with 1's on the diagonal and entries γ everywhere else. Then (7.28) reads $v = M(1, \gamma)w$. Define $\delta = (1 - \gamma)(1 + (m - 1)\gamma)$. Then a direct calculation shows the

Lemma 6 *The matrix $M(1, \gamma) \in \mathbb{R}^{m \times m}$ is invertible if and only if $\delta \neq 0$. Furthermore, if the matrix $M(1, \gamma)$ is invertible, then $M(1, \gamma)^{-1} = M(\gamma_-, \gamma_+)$ with*

$$\gamma_- = \frac{1}{\delta}(1 + (m - 2)\gamma), \qquad \gamma_+ = -\frac{1}{\delta}\gamma. \tag{7.32}$$

From lemma 6 it follows that

$$w = M(\gamma_-, \gamma_+)v. \tag{7.33}$$

Substitution of (7.28), (7.29) and (7.30) in the Lagrangian (7.27) and using $k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ yields the dual problem

Let us poof Lemma 4 from the text.

Proof of lemma 4: We start by decomposing the matrix $\tilde{K}$ in

$$\tilde{K} = \sum_{c<d} J^{cd} \tag{7.34}$$

with the matrix J^{cd} given by

$$J^{cd}_{ab,ij} = \begin{cases} \frac{1}{m-1}\tilde{K}_{aa,ij} & \text{if } a = b \text{ and } (\ a = c \text{ or } a = d) \\ \tilde{K}_{ab,ij} & \text{if } a \neq b \text{ and } (a = c, b = d \text{ or } a = d, b = c) \\ 0 & \text{otherwise.} \end{cases} \tag{7.35}$$

It is enough to show that each of the matrices J^{cd} is negative definite. By the choice of kernel, the matrix K with entries $K_{ij} = k(x_i, x_j)$ is positive definite. A direct calculation shows that for $\gamma_- < 0$ and $\gamma_+ = 0$ or $\gamma_+ = -1/(m - 1)\gamma_-$ the matrix J^{cd} is negative definite. Because the set of negative definite matrices is a convex cone, the matrix J^{cd} is negative definite for all the choices of $\gamma_- < 0$ and $\gamma_+ \in [0, -1/(m - 1)\gamma_-]$. ■

Chapter 8

Locally invariant image representation

Up to now, we have discussed various algorithms for supervised and unsupervised learning. In the rest of the thesis we discuss a real–world application of our algorithms. We developed a face recognition system in collaboration with the Centre Suisse d'Electronique et Microtechnique (CSEM) in Neuchâtel. The system consists of two main parts. First, the camera, developed by the CSEM, which uses an adaptive vision sensor [7, 58]. Second, our algorithmic strategies to learn to recognise faces. The CSEM set up has several attractive features. The computation of local contrast and illumination gradient are implemented in analog hardware on the sensor chip. Therefore, they are computed in real time. In addition, the camera has a high dynamic range.

In this chapter we explain how the sensor processes its visual input and how we post-process the sensor output to have a representation with local invariances. The next chapter uses this representation and explains how our algorithms are applied to face recognition. In section 8.1 we describe the main functional principles of this sensor. Then we describe how to use the sensor output to construct locally invariant image representations. For this, we introduce a key tool to construct local representations in section 8.2, which we apply in section 8.3 to the output of the vision sensor. Finally, in section 8.4 we conclude by relating our image processing to biology, that is, to processing in the visual pathway. We will see that even though the resources of biology are very different from our hardware and software processing tools, we can find interesting functional parallels.

8.1 The CSEM adaptive vision sensor

We start by describing the vision sensor, following [58]. The vision sensor is the heart of the CSEM camera. The main part of it is an array of 128×128 pixels. Each pixel has the structure shown in figure 8.1. Let us explain the acquisition of a frame, that is, the acquisition of an image, by focusing on what happens at the pixel level. Consider a pixel sitting at position C. At the beginning of a frame acquisition the capacitors C_C, C_L, C_R, C_T, C_B are reset and the transistors M_L, M_R, M_T, M_B are turned on. The voltage V_C, resulting from the integration of the photocurrent at the photodiode PD, is dispatched to the four neighbours at the top, the bottom, to the left and to the right. At the same time, the voltages V_L, V_R, V_T, V_B resulting from the integration of the photocurrent at the four neighbouring positions are dispatched to pixel C. During integration of the photocurrent, the voltage V_C increases until it reaches a fixed voltage level V_{REF}. Let us call this time t_C. At time t_C the transistors M_L, M_R, M_T, M_B are turned off so that the four voltages $V_L(t_C), V_R(t_C), V_T(t_C), V_B(t_C)$ of the neighbouring

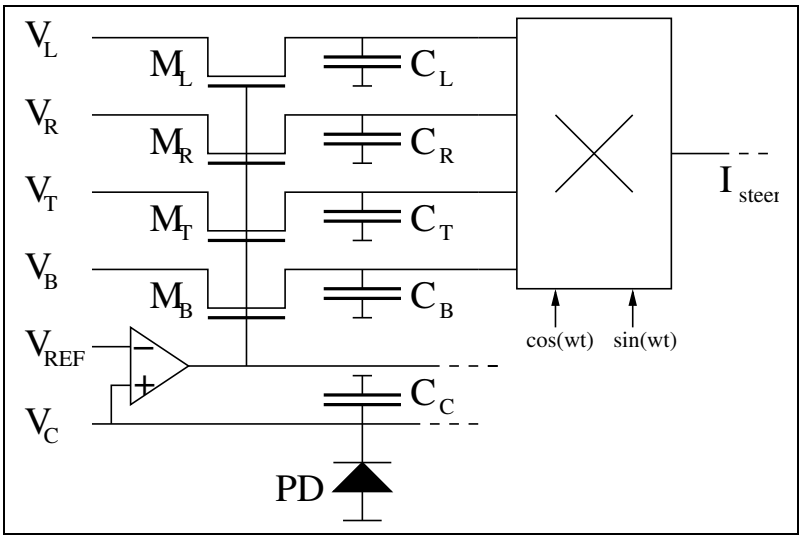


Figure 8.1: The pixel structure of the CSEM adaptive vision sensor.

pixels are hold by the capacitors C_L, C_R, C_T, C_B. The normalised illumination gradient ∇J_C at pixel C is approximated by

$$\nabla J_C \approx (\frac{1}{2}(V_T - V_B), \frac{1}{2}(V_L - V_R)). \tag{8.1}$$

It is called normalised because the voltages are taken at time t_C. We call the magnitude of ∇J_C the *local contrast*.

On the chip, the gradient is time encoded in the following way. Consider the time dependent unit vector $u(t)$ rotating around the origin and defined by components $(\cos \omega t, \sin \omega t)$ for a fixed value of ω. The components are called steering functions, because at each time t their value determines the direction of u. The scalar product of u with the gradient ∇J_C,

$$\nabla J_C \cdot u(t) \approx \frac{1}{2}(V_T - V_B) \cos \omega t + \frac{1}{2}(V_L - V_R) \sin \omega t \tag{8.2}$$

is a sinusoidal function, whose amplitude is equal to the magnitude of the gradient and the phase encodes the gradient direction. In the computer vision literature, the linear operation that maps J_C to $\nabla J_C \cdot u(t)$ is called a steerable filter [27]. On the chip this time encoding of ∇J_C is performed by applying a sine and a cosine function to the multiplier shown on the top right. The details can be found in [7]. The output of the multiplier is the steering current I_{steer}, which is proportional to $\nabla J_C \cdot u(t)$.

Up to now, we have explained how the illumination gradient is normalised with respect to the local contrast and how it is transformed into the steering current. Now, we will explain how the steering current is transformed into a series of pulses emitted on a communication bus, which is the output of the sensor. We will only explain the idea and not the hardware implementation, which can be found in [7, 58].

The idea is show in figure 8.2. The horizontal axis is the time axis, starting at t_{max}, the maximally allowed integration time for the photocurrent integration, and stopping

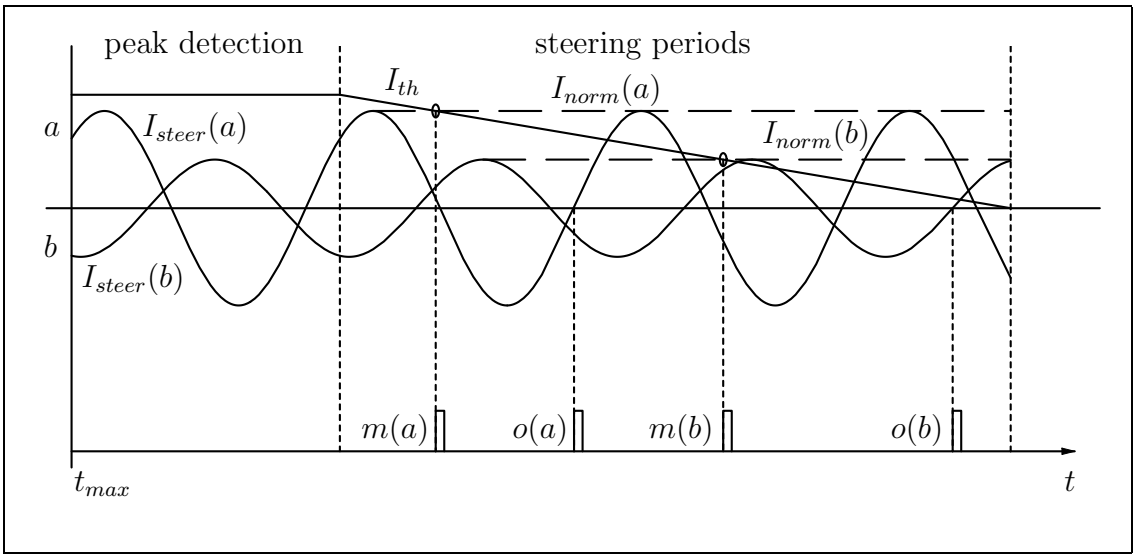


Figure 8.2: Encoding the steering current in a sequence of pulses: the two sine functions are the steering currents I_{steer} of two pixels a and b. Each is encoded in two pulses, coding the magnitude (m) and the orientation (o) of the gradient at the pixel location, see the text for more details.

after the total time used to acquire one frame. This time interval is divided into two parts, the peak detection period, whose length is one period of the steering functions, and the steering periods. Shown are the steering currents of two pixels a and b. During the peak detection period, the amplitude of the steering current of each pixel is detected and stored in I_{norm}. Then, during the subsequent steering periods, a reference threshold current I_{th}, which is monotonically decreasing, is compared to each of the I_{norm}. When for example the threshold I_{th} reaches I_{norm} of pixel a, a pulse signal encoding the address of pixel a is emitted on the communications bus. The timing of the pulse encodes the magnitude of the gradient, that is, the amplitude of the steering current. The gradient direction at the pixel a is encoded by emitting a pulse at the first zero crossing with a positive slope of the steering current I_{steer} of pixel a, following the emission of the magnitude information. The pulses for magnitude (m) and direction (o) are shown in the figure for the two pixels a and b. In summary, the sensor output is a sequence of pulses, encoding magnitude and direction of the image gradient, normalised according to the local illumination. Furthermore, the sequence of pulses is ordered in time according to the local contrast. The communication bus is made such that from the pulses the location of the pixel can be recovered.

The sensors output is accessible through a software interface in form of a stream of events. The stream corresponds to a sequence of images, called *frames* in this context. Each event in the stream is of the form

(*frame number, gradient magnitude, gradient direction, x coordinate, y coordinate*), (8.3)

where the first number is the number of the frame in the sequence, the second number

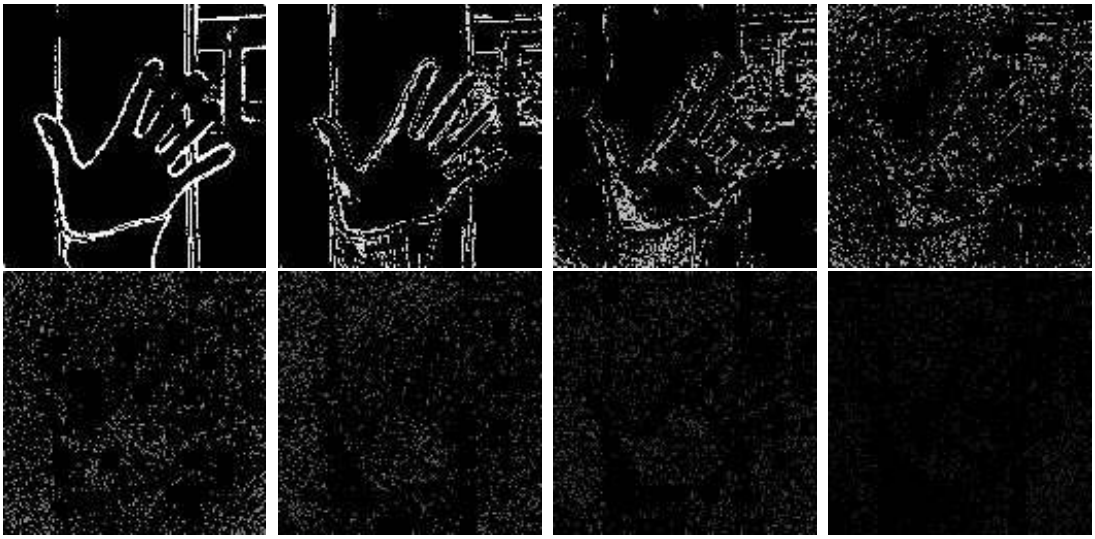


Figure 8.3: Example frame acquired with the vision camera: each image shows all events, whose magnitude of the gradient falls in the same bin of the histogram, starting from the largest magnitude to the smallest.

is the value of the local contrast, the third number is the direction of the gradient, followed by the coordinates of the occurrence of the event. Recall that the events are ordered according to descending local contrast. We will us this information to construct histograms of gradient magnitude. Specifically, consider a frame from the vision sensor camera, that is, a sequence of events $e = (e_1, ..., e_l)$. For an event e_i from the sequence we denote by $e_{i,m}$ the magnitude of the gradient, $e_{i,d}$ the direction of the gradient, $e_{i,x}$ the x–coordinate and $e_{i,y}$ the y–coordinate of the event. We set the parameters of the camera such that the sequence is complete, that is, for each point in the pixel array of the sensor there is exactly one event in the sequence, hence it has length $l = 128 * 128$.

Below, we define histograms over contrast, but instead of considering absolute contrast values, we rather use the feature that our sequences of events are ordered, that is, pixels with the largest values are transmitted first. This allows us to round off and remap the local contrast value of an event with respect to its position in the list. Let us choose $q+1$ values $m_0, ..., m_q$ between zero and the length l of the sequence. The values m_i divide the list into q parts or bins, and within each part pixels are considered to have the same contrast. In particular, event e_i has the local contrast value $e_{i,m} = r$ if the event is in the rth part of the list,

$$e_{i,m} = r \quad \text{if and only if} \quad i \in [m_r, m_{r+1}). \tag{8.4}$$

Later, we will use

$$q = 8 \quad \text{and} \quad m_i = il/10. \tag{8.5}$$

Note that the frequency of each contrast value is equal and therefore the histogram over the local contrast is flat. Furthermore, note that we neglect the last fifth of the event sequence.

Figure 8.3 shows an example frame. Each image shows all events of the same local contrast r, for $r = 0, ..., 7$. The brighter the points shown the stronger the contrast.

In the next section we will describe a tool to construct a decomposition, which is local in space and in orientation and allows us to compute local histograms as features.

8.2 Decomposition into local parts

In this section we describe the partition of unity, which is used to decompose functions into local parts. This is a common tool in analysis [57] as well as in wavelet theory [48]. We construct a partition of unity composed of B–splines, which was used extensively by [73].

A *partition of unity* on $\mathbb{R}$ is a family of real–valued functions (ψ_i) on $\mathbb{R}$, such that

$$\sum_i \psi_i = 1.$$

We will construct a partition of unity with the *cubic B–spline* $b : \mathbb{R} \to \mathbb{R}$ defined by

$$b(x) = \begin{cases} \frac{2}{3} - x^2 + \frac{1}{2}|x|^3 & \text{if } |x| < 1 \\ \frac{1}{6}(2 - |x|)^3 & \text{if } 1 \leq |x| < 2 \\ 0 & \text{if } |x| \geq 2 \end{cases}. \tag{8.6}$$

Note that by definition

$$||b||_1 = 1. \tag{8.7}$$

One can easily verify that the family $(b(\cdot - i))_i$ of translated B–splines is a partition of unity, see figure 8.4.A. Note that we can easily dilate the partition of unity by a factor $s \in \mathbb{R}$ by considering the family $(b_s(\cdot - si))_i$ defined by

$$b_s(x - si) = b(\frac{x}{s} - i). \tag{8.8}$$

By defining $b(x, y) = b(x)b(y)$, the family $(b(\cdot - i, \cdot - j)_{i,j}$ is a partition of unity on $\mathbb{R}^2$. Figure 8.4.B shows a density plot of b in two dimensions. Similarly, we can lift the family $(b(\cdot - i))_{i=0,...,7}$ with the invertible function

$$\varphi : [0, 8) \xrightarrow{\cdot 1/8} [0, 1) \xrightarrow{\exp(2\pi i \cdot)} S^1 \tag{8.9}$$

to a partition of unity on the unit circle $S^1 \subset \mathbb{R}^2$, consisting of the eight functions

$$b_\varphi(x - \varphi(i)) = b(\varphi^{-1}(x) - i), \quad i = 0, ..., 7. \tag{8.10}$$

In the next section we will use this partition on S^1 to decompose the image gradient in eight orientation directions.

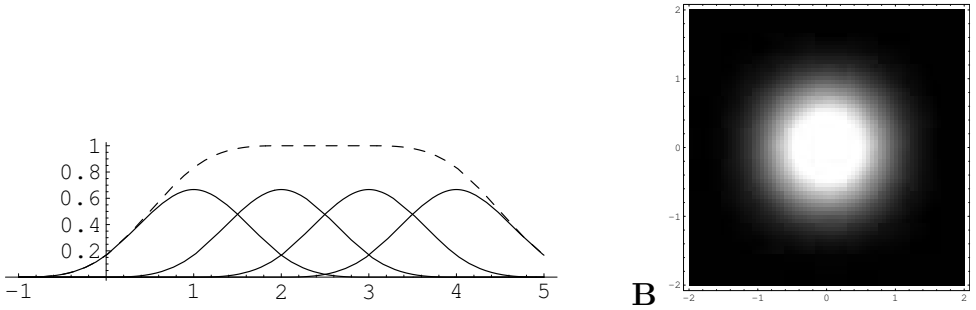


Figure 8.4: The solid lines in plot **A** show four cubic B–spline functions, each shifted by one. The dashed line is the sum of the four splines, which is equal to one in the interval $[2, 3]$ and reflects the partition of unity condition. Plot **B** shows a density plot of the two dimensional cubic B–spline. One can see that it is nearly isotropic.

8.3 Local histogram

First, we will illustrate with a simple example how we construct local histograms. Consider a sequence $f : \{0, ..., p\} \to \{0, ..., q-1\}$. Then the *histogram* h of f is the function

$$h[f] : \{0, ..., q-1\} \to \mathbb{R}, \quad h[f](r) = \sum_{i=0}^{p} \delta_{f(i)-r}, \tag{8.11}$$

which counts the frequency of occurrence of a value r in the sequence f. Here $\delta_x = 1$ for $x = 0$ and zero otherwise. To simplify the notation, we will often suppress the explicit dependence on the function f if it is clear from the context which function is meant. The *local histogram* of f at position $u \in \mathbb{R}$ with respect to the cubic B–spline b is the function

$$h_u[f] : \{0, ..., q-1\} \to \mathbb{R}, \quad h_u[f](r) = \sum_{i=0}^{p} b(i-u)\delta_{f(i)-r}, \tag{8.12}$$

which counts the frequency of occurrence of the value r in the sequence f weighted by a factor depending on the position of occurrence. Because the family of shifted B–splines is a partition of unity, we have

$$h = \sum_{u} h_u. \tag{8.13}$$

From equation (8.13) we can see that a partition of unity allows us to decompose a global feature, the histogram, into a sum of local features, the local histograms.

More specifically, consider a frame from the vision sensor camera, that is, a sequence of events $e = (e_1, ..., e_l)$ of length $l = 128 * 128$. For the decomposition of a frame in local histograms, we choose a partition of unity consisting of translated B–splines with centres on a rectangular regular lattice $L \subset \mathbb{R}^2$. The lattice has a step size s, which is the distance between two nodes in horizontal or vertical direction. Furthermore, we denote by M the lattice consisting of the eight points $e^{2\pi ij/8}, j = 0, ..., 7$, on the unit circle S^1, see figure

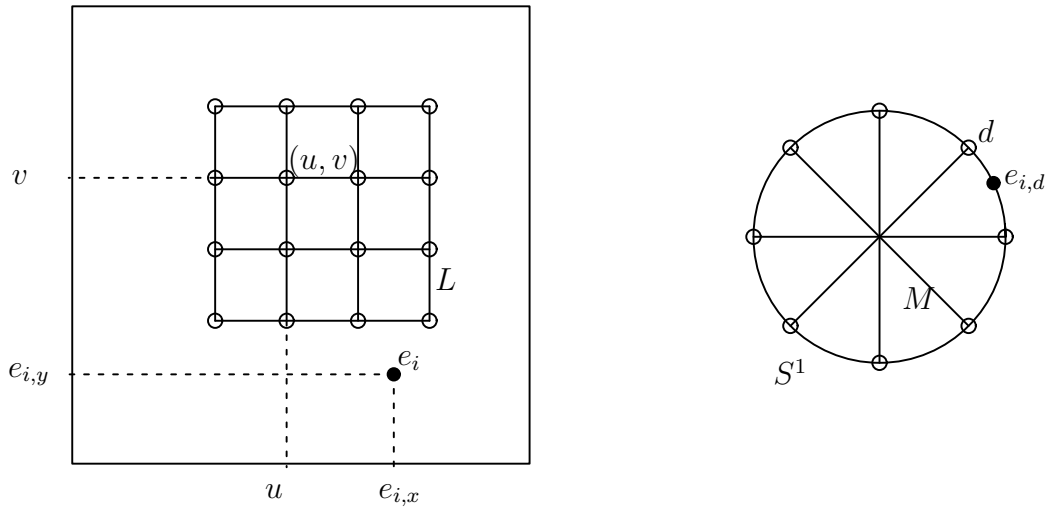


Figure 8.5: Example lattices for a local decomposition: The plot on the left shows the size of a frame (rectangle) and the lattice L with the point (u, v) of the lattice, which has x–coordinate u and y–coordinate v. The black point shows the location of event e_i. The right plot shows the unit circle S^1 together with the lattice M and one of its points d. The point $e_{i,d}$ shows the direction of the gradient of event e_i.

8.5 These points are the centre points of the cubic B–splines used in the partition of unity on S^1, see (8.10). For a point (u, v, d) in the total lattice $L \times M$ the *local histogram* of the sequence of events $e = (e_1, ..., e_l)$ is the function $h_{(u,v,d)}[e] : \{0, ..., q-1\} \to \mathbb{R}$ defined by

$$h_{(u,v,d)}[e](r) = \sum_{i=1}^{l} w(e_i)\delta_{e_{i,m}-r}. \tag{8.14}$$

with weight

$$w(e_i) = b_s(e_{i,x} - u)b_s(e_{i,y} - v)b_\varphi(e_{i,d} - d), \tag{8.15}$$

see (8.8) and (8.10). The local histogram $h_{u,v,d}[e](r)$ counts the occurrence of events e_i with gradient magnitude r, and each occurrence is weighted with respect to position and orientation of the event e_i. Because the sequence of events is ordered with respect to the local contrast we have

$$h_{(u,v,d)}[e](r) = \sum_{i=m_r}^{m_{r+1}-1} w(e_i), \tag{8.16}$$

with the values m_r defined in (8.5).

Figure 8.6 shows an example frame. The frame is decomposed into eight directions of the gradient, using the partition of unity on S^1 from (8.10), with centres of the cubic B–splines sitting on the lattice M. Note that the intensity of each event is computed using the quantised magnitude of the gradient, quantised to $q = 8$ values using the points m_i from (8.5) and multiplied by a weight factor $b(e_{i,d} - d)$.

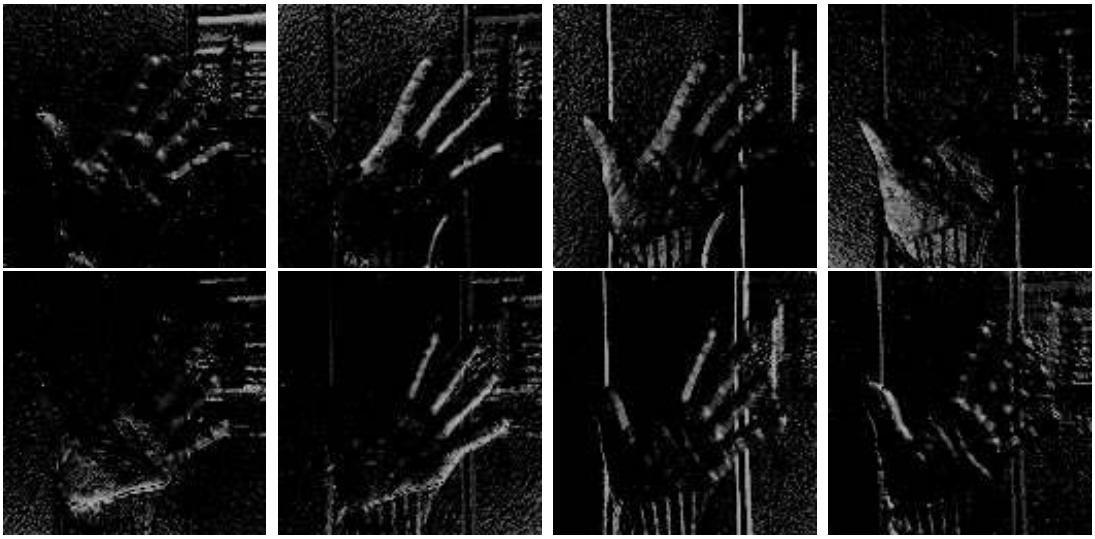


Figure 8.6: Example frame decomposed into eight directions of the gradient, see the text for further details.

8.3.1 Properties:

Let us note some immediate consequences of the definition of the local histograms. From (8.7) we know that $||b||_1 = 1$. Hence, for all (u, v, d) the following inequalities hold,

$$0 \leq h_{(u,v,d)}(r) \leq 1, \qquad \sum_r \sum_d h_{(u,v,d)}(r) \leq 1 \tag{8.17}$$

where on the right hand side equality holds if we choose the bin limits at the beginning, $m_0 = 0$, and at the end of the event list, $m_q = 128 * 128$. In that case, the sum over all directions of the histograms at a given position (u, v) on the lattice L sit on a hyperplane,

$$\sum_d h_{(u,v,d)} \in \{(a_1, .., a_q) | \sum_i a_i = 1\}. \tag{8.18}$$

The situation is illustrated in figure 9.4.A. For ease of presentation only two bins and two directions are shown.

Local translation and rotation invariance: The main motivation for decomposing the image into local histograms are their local invariance properties. The cubic B–spline window used for the weighting of each event in the histogram depends in a differentiable manner on a translation of the event. Therefore, translating the image by a small amount gives rise to a small change in the local histograms. Furthermore, the representation has a local rotational invariance. Note that even if the two dimensional spline window we use is nearly isotropic, as shown in figure 8.4.B, the lattice is not. But still, a small rotation of the image yields a spatial translation of the event location and a rotation of the gradient direction of the events. Because we decompose with a differentiable window, spatially and in orientation, the representation has a local rotational invariance.

Local scale invariance: Our representation has one more local invariance, it is the invariance to small changes in scale. For ease of presentation we look at a simple example. Let us suppose that there is an object in front of the camera, and this object fills the whole image. We denote by f the light intensity distribution falling on the sensor. Now, if we approach the object to the camera, the distribution changes to

$$f_s(x) = f(sx), \tag{8.19}$$

a dilated version of f, with a scaling factor $s \in [0, 1]$. If we compute the gradient

$$\nabla f_s(x) = s \nabla f(sx), \tag{8.20}$$

we can see that first, the position of the events has changed, and second, that the magnitude of the gradient is different, namely scaled by a factor s. But the ordering of the event list has not changed, the strongest event is still the strongest event, only its position has changed by a factor s. If this change in position stays inside the window, the local histograms change only slightly.

In summary, we can say that by decomposing the event list into equal parts defines an intensity histogram which is scale invariant. Furthermore, if we decompose the event list into local histograms, the representation is locally scale invariant, because our B-spline windows have a local translation invariance.

8.4 Discussion and links to biology

There are interesting functional parallels between biological processing in the visual system and the computations of our vision system. The vision sensor transmits information in an order of decreasing importance. First, the events corresponding to a high local contrast are transmitted and later the low contrast events. From this ordered sequence of events, we compute the contrast histograms. Therefore, the information we retain is contained in the ordering and not in the absolute value of the local contrast. This is a rank order coding strategy. It has been suggested that the visual system might use rank order codes [71]. Furthermore, features computed from rank order codes have been successfully used in computer vision [68]. A second parallel is our decomposition in local histograms. Locality can be interpreted as a receptive field in position and orientation. Similarly, cells in the primary visual cortex show selectivity to edges, in a certain interval of position and orientation [38, 39].

An other very important property of sensory systems in biology is adaptation and sensitivity regulation. At the sensor level, the computation of the local contrast can be seen as an adaptation to different illumination conditions. In more detail, as described in section 8.1 a pixel P_C determines the integration time for the four neighbours, which are then used to calculate the gradient (8.1). Therefore, the magnitude of the gradient, which is proportional to the integration time, is proportional to the illumination at the centre pixel.

In biological sensory systems, adaptation is probably used at every processing step. In the visual pathway adaptation starts at the level of the photoreceptors [59] and continues the way along to higher visual areas [17]. In our processing, a second adaptation step is used by decomposing the long list of events in equally long parts used to fill the bins of the histogram. Therefore, the total histogram over the whole image is flat, regardless of the input. This yields an invariance of our representation to the absolute value of the gradient and therefore an independence to the various parameters the gradient depends on. Furthermore, this decomposition is important for the local scale invariance, as discussed in section 8.3.

Chapter 9

Recognition of faces

Our goal is to recognise faces captured by the vision sensor described in the previous chapter. A lot of research has been done on face recognition, and we will start this chapter with discussing selected approaches in section 9.1 and relate these strategies to our recognition system. In section 9.2 we describe how we collected the database of faces used for learning and recognising. In our online setting it is not sure that at every moment there is a face in front of the camera, and if there is a face, its position is not known. Therefore, the first processing step is the detection of a face discussed in sections 9.3-9.6. More precisely, section 9.3 describes the class of features used for detection and defines a kernel measuring the similarity of face features. In section 9.4 we show how to learn characteristic features in a data dependent way, and in section 9.5 we describe a strategy to learn more complex features from simpler ones. Eventually, section 9.6 closes the part on face detection by showing the achieved detection results. In sections 9.7-9.9 we turn to face recognition. In section 9.7 we define again a kernel. This kernel is more sensitively tuned, so as to allow to distinguish between the faces of different persons. Face recognition is now a straightforward application of our algorithm for multi–categorical data classification from chapter 4. Section 9.8 discusses the results from detection and recognition combined. Finally, section 9.9 concludes with a review of the chapter from a more general point of view.

9.1 Introduction

Face recognition has a long tradition and is still a topic of active research and general interest, as a look in today's newspapers confirms[1]. In the next subsection we take a closer look at a few selected examples of face detection and recognition systems. In subsection 9.1.2 we describe their relation to our approach.

9.1.1 State of the art

Many different strategies are used and they can be classified according to a variety of criteria. If we look at the type of features used, we can distinguish between local and global features. We say a feature is local if we can compute the feature or part of it from a fraction or part of the image only. Note that this is more an intuitive notion than a mathematical definition. Thus, on one end of the scale, there are approaches using

[1] Neue Zürcher Zeitung, Dreidimensionale Gesichtserkennung, march 28th, 2003; The New York Times, Face–Recognition Technology Improves, march 14th, 2003.

a global structure to match with the face to recognise. Examples are Eigenfaces [72], deformable templates [80], or an elastic graph matching [79]. We recall that the latter uses features calculated from Gabor filter responses evaluated at the nodes of the elastic graph. But to determine the position of the graph relative to the face, the whole image is needed. On the other end are approaches using local features. Bichsel [11] uses a system which first detects the face by a saccadic focusing on local invariants, and then identifies it by calculating a similarity on the basis of facial feature shapes and positions. Whereas in Bichsel [11] the saccadic focusing strategy is along a multi–resolution chain, Smeraldi and Bigun [69] studied a more biologically motivated saccadic strategy. A retinotopic sampling grid is moved in saccades over the image, and the next location is determined by the activity of Gabor filters evaluated at the grid positions. Once the eyes and the mouth are detected, an authentication stage follows. For this, a support vector machine is used with a kernel calculating the similarity of the test face and the given example faces. The similarity measure uses Gabor filter responses at the retinotopic sampling grid positions centred at the eyes and the mouth. This approach showed a very high performance. Other strategies using support vector machines for the detection of faces are reported in the literature; Osuna et al. [50] use as features preprocessed face images. Because no edge filters are applied, various preprocessing steps are performed to compensate for different illuminations. Histogram equalisation proved to be an important step. Papageorgiou and Poggio [51] used support vector machines to detect faces, using wavelet coefficients as features. A disadvantage of the support vector machine approach with nonlinear kernels, as in [50, 51], is that the evaluation of the decision function needs many kernel evaluations, which is computationally costly. To speed this up, reduced set methods were developed [55]. In contrast to these approaches, the more descriptive features of Smeraldi and Bigun [69] allow the use of a linear support vector machine, which is fast, but the more complex features have to be calculated each time the sampling grid is moved. Of course, there is a tradeoff between computational complexity of the features and of the decision function. To combine computationally simple features, but in a data dependent way, using only the necessary number of features proved to be a very efficient strategy for face detection [26, 75] and gender classification [66].

9.1.2 Our approach

The main difference of our approach compared to the methods mentioned above stems from the different input we have. While the algorithms described above rely on images from a standard camera, our data comes from the vision sensor delivering gradient and local contrast information with a high dynamic range. On the other hand, the resolution of the sensor is not as good as in other recognition systems [52], which might limit the capabilities to distinguish between a large number of different faces. Therefore, our system is not targeted at tasks like an automatic portier [11], where many people have to be recognised or identified with high accuracy. We want to focus on a different scenario. The goal is to recognise a small number of persons with as few as possible false positives. In addition the system should be simple, that is, it should have few tunable parameters

Figure 9.1: A subject in front of the CSEM adaptive vision sensor camera. All images are acquired in a cluttered office environment.

and ideally it should be computationally efficient. Furthermore, our scenario requires that a person performs small movements in front of the camera, while the algorithm tries to recognise the face. Therefore, the false negative rate is less important, as multiple images of a person are available during the recognition process. The heart of our recognition system is the multi–class support vector machine algorithm developed in chapter 4. As our algorithm is simpler than other multi–class support vector approaches, not only learning but recognition is computationally more efficient.

Because the size of the face is only a fraction of the whole image seen by the camera, we would like to have an efficient method to determine if at a given position in the image there is possibly a face or not. Scanning over the whole image and using at every position the recognition model to try to recognise a face is a simple but unfortunately not an efficient strategy. We propose a computationally more efficient strategy to test for a face at a given position. The main idea stems on one hand from the elegant face detection algorithm of Fleuret and Geman [26] and on the other hand from our unsupervised algorithm described in chapter 7. From [26] we use the idea of constructing complex characteristic features from simple ones, consisting of products of local features. To select characteristic features from a large set we optimise a cost function of the same form as the cost function (7.5) used in our unsupervised algorithm, whereas some adaptations to the specific problem are made.

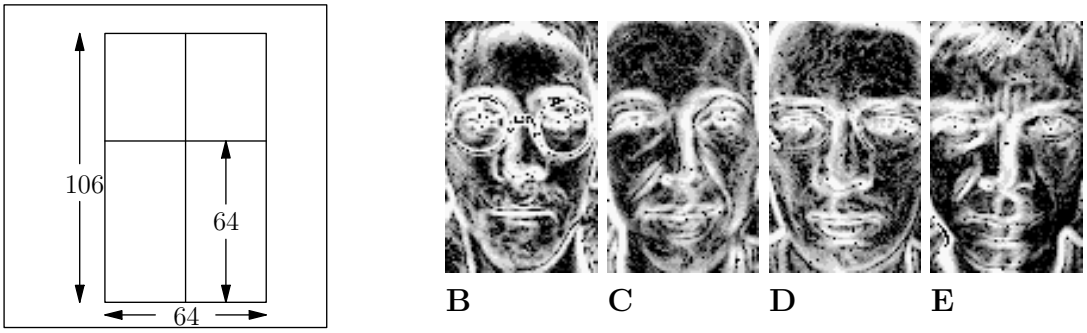


Figure 9.2: Image **A** shows the dimensions of the rectangular region we used to cut out the faces. Images in **B**-**E** are face examples of the four persons.

9.2 The database of faces

To learn to recognise faces we need, of course, a database consisting of example faces. In this section we describe how we collected the data. We created a first database consisting of images acquired by the vision sensor where in most images there is a face. The images without a face are for test purposes only. From the images with faces of this first database, we created a second one by defining a rectangular subimage of each frame centred on the face. We will call such a reduced image a *subimage* or *subframe*. In more detail we proceeded as follows. We created a first database consisting of face images of four persons, which should later be recognised, as well as a few images without a face. Figure 9.1 illustrates the arrangement of a person in front of the camera. The camera was positioned at different angles with respect to the window, the only light source. The axes of the camera was either parallel or perpendicular to the window. This was done in order to have a certain degree of invariance to illumination in the database. The distance of the face to the camera was adjusted to be about 35cm, thus the height of the face is nearly the height of the image. The subject was advised to perform small movements in front of the camera. Then, images were acquired in sequences of 20 frames with a frame rate of about 5 frames per second. We took a minimal number of 60 frames of each person to be recognised.

From each frame we cut a subimage containing the face. This subimage was defined such that the middle point between the eyes was always at the same position. Within each frame we use subimages of dimension 106×64, see figure 9.2.A. Let us recall that the total frame has 128×128 pixels. Note that we did not rescale the images, as our control of the distance of the face to the camera was precise enough to have only small scale variations of about 5% in the database. Figure 9.2.B-E shows an example face of each person we used to train the algorithms.

This set of cut faces is our *training database*. We denote this dataset of subimages or subframes by

$$S = \{x_1, ..., x_l\}. \tag{9.1}$$

All the parameters we learn are estimated using this data. Note that we use only examples

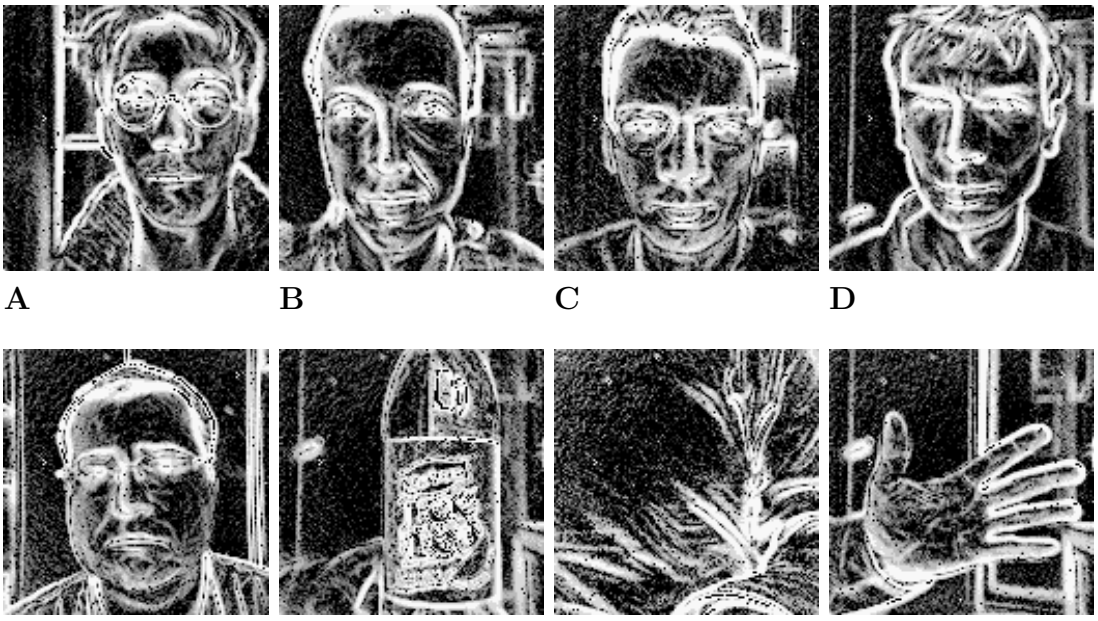


Figure 9.3: The images in **A** to **E** show examples of frames that contain a face, the faces in **A** to **D** are used for training, **E** shows an example used only for testing. The images in **C** to **H** are non–face images.

containing faces in the training database. This is desirable as it is difficult to collect a representative set of negative, that is, non–face examples, which represent all possible non–face images. Because of the two sizes of images in the database, we will need two sizes of lattices for the decomposition in local histograms. To distinguish between the two, we introduce the following notation. We denote by $x, x_i, ...$ cut images, and by $\hat{x}, \hat{x}_i, ...$ images of full size. In analogy, we denote by L a lattice over a cut image, and by $\hat{L}$ one over a full image. In the subsequent sections we will develop different models to detect and recognise faces. All these models take as argument a cut image x or a local histogram decomposition $h_{(\cdot)}[x]$ of a cut image, respectively. We can lift these models on $h_{(\cdot)}[x]$ to models on the decomposition $h_{(\cdot)}[\hat{x}]$ of the whole image $\hat{x}$ by scanning over the whole lattice $\hat{L}$ and cutting all possible sublattices of the same size as L.

For the local decompositions we set the centres of the B–spline windows on the nodes of regular lattices. We use regular lattices L and $\hat{L}$ with a difference of 5 pixels between two nodes, in vertical and horizontal direction. The value of 5 was chosen as follows. A strong edge has the width of about 5 pixels, which should be captured in a single window of the local decomposition. Therefore, 5 is a good first guess. On the other hand, if the step size is too large, we lose too much information (and 5 could already be too large). Ideally, the representation should have a translation invariance of at least half the step size, such that it is independent of the exact position of the lattice relative to the image. For each subimage x we produced four local decompositions. One by centring the lattice L with respect to the midpoint between the eyes, and the others by translating the lattice

by 2 pixels in horizontal, vertical, or in horizontal and vertical directions. This is done in order to have an invariance in the position of the lattice. The lattice L is of dimension 20×12, and the lattice $\hat{L}$ has 24×24 nodes. The lattice $\hat{L}$ is always centred with respect to the image and has $5 * 13 = 65$ sublattices of the same dimension as L.

9.3 The detection kernel

We are in the following situation. We are given a set $\{x_1, ..., x_l\}$ of example images of faces acquired by the vision sensor. Since we are working with the cut subframes each face fills the whole image and all the images are of the same size. Furthermore, the face is centred in the image, as described in the previous section. The goal is to learn a set of tests, which test for face features. More precisely, we want to learn tests, which take as argument local histograms $h_{(\cdot)}[x]$ defined with respect to the lattice L. Once the tests are learnt, we apply them to an image $\hat{x}$ of the vision sensor. This is done by first cutting all possible sublattices of the lattice $\hat{L}$ of the same size as the lattice L. Second, by applying the tests on the local histograms defined with respect to these sublattices. The result of the tests applied to the local histograms at the nodes of the sublattice gives us information if there is possibly a face or not.

Consider the total lattice $L \times M$ consisting of the lattice L in position and the lattice M in orientation. For a subset $A \subset \{(u, v, d) | (u, v) \in L, d \in M\}$ all the local histograms $h_{(u,v,d)}[x]$ are elements of $\mathbb{R}^q$, with q denoting the number of bins. For a face image x we define the component–wise product of histograms, $h_A[x] \in \mathbb{R}^q$, over the set A to be

$$h_A[x](r) = \prod_{(u,v,d) \in A} h_{(u,v,d)}[x](r). \tag{9.2}$$

The product can be interpreted as an "and" operation. For example for $r = 0$ and a subset A of four lattice points a large value of $h_A[x](r)$ requires that we have a strong contrast at all 4 lattice points.

For a set of examples $\{x_1, ..., x_l\}$ we want to learn affine linear models on these product histograms. We denote by ϕ the feature map

$$\phi(x, A) = h_A[x], \tag{9.3}$$

which maps a pair consisting of an image x and a set of positions in the total lattice $L \times M$ to the component–wise product of the local histogram at the given positions. The feature $\phi(x, A)$ is an element of $\mathbb{R}^q$. As above, q is the number of bins in the local histograms. Then for frames x_i, x_j and a subset A we define the *detection kernel*

$$k((x_i, A), (x_j, A)) = <\phi(x_i, A), \phi(x_j, A)>_\lambda, \tag{9.4}$$

with $<\cdot, \cdot>_\lambda$ denoting the weighted scalar product, defined by

$$<u, v>_\lambda = \sum_{r=0}^{q-1} \lambda^r u_r v_r, \tag{9.5}$$

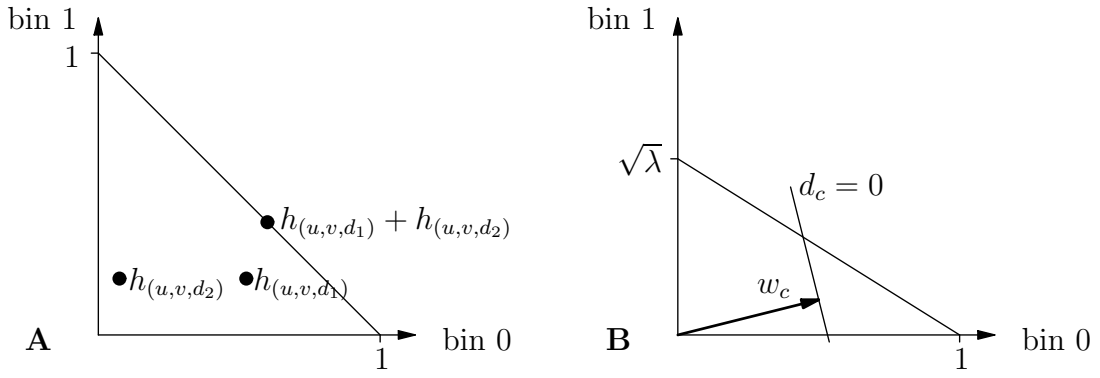


Figure 9.4: Shown in plot **A** are histograms of a local decomposition with only two bins. The dots show two histograms at location (u, v) with orientation d_1 and d_2 respectively and their sum. Plot **B** shows a feature detector. Shown are the vector w_c and the zero level of d_c. The detector has a positive value on all points in the triangle on the right of the zero level.

with $\lambda \in \mathbb{R}$. With a value of $\lambda \in (0, 1)$ we can put more weight on the bins consisting of events with a strong gradient magnitude, that is, small values of r, see (9.2). We used $\lambda = 4/5$ for our experiments. By construction the kernel k is positive definite. It is important to note that our detection kernel first computes products or "and" conditions at different positions of one example and then compares the result with that of a second example, in contrast to ANOVA [74] or R–convolution kernels [37].

9.4 Learning multiple features

Let us now show how we can learn characteristic features of our data. The feature vector of a subimage is composed of local histograms. In each histogram, a bin with low bin number but high value means that an edge of a certain orientation and intensity is present in the image. A test for a feature is then a measure of the presence or absence of edges, where the local histograms allow us to distinguish between orientation, intensity and position. More precisely, for a tuple (w_c, ρ_c, A_c) we denote by d_c the function

$$d_c(x) = <w_c, h_{A_c}[x]>_\lambda - \rho_c, \tag{9.6}$$

which tests if the feature is present, $d_c(x) \geq 0$, or absent, $d_c(x) < 0$. We call d_c a *feature detector* for (w_c, ρ_c, A_c). If $\rho_c = 0$ we call the feature detector trivial, as the test $d_c(x) \geq 0$ is true for all x.

Figure 9.4 shows the situation. In plot A the space $\mathbb{R}^q$ of the local histograms is shown. The dots represent some example histograms, their position is shown according to (8.17) and (8.18). Plot B shows a feature detector. Using the weighted scalar product of (9.5) is equivalent to using the standard scalar product and rescaling the n-th bin by a factor $(\sqrt{\lambda})^n$, as shown in the figure.

We recall that in chapter 7 we developed an unsupervised algorithm by starting from

an optimisation problem of the form

$$\min_{(w_1,\rho_1),...,(w_m,\rho_m)} \sum_c F(w_c, \rho_c) + G(w_1, ..., w_m). \tag{9.7}$$

Minimisation was done over m pairs of vector and margin. We will use a similar optimisation problem here to learn multiple feature detectors. Note that a feature detector is defined not only by a pair but a triple consisting of a vector w_c, a margin ρ_c and in addition, a subset A_c of lattice nodes. As in chapter 7, we learn feature detectors with vectors $w_c, w_{c'}$ expressible as a linear combination of feature vectors,

$$w_c = \sum_i \alpha_i \phi(x_i, A_c), \qquad w_{c'} = \sum_i \alpha'_i \phi(x_i, A_{c'}), \tag{9.8}$$

with possibly different subsets $A_c, A_{c'}$. As in (9.7), the cost function consists of two terms, but each term is adapted to the problem as follows. For fixed A, the function $F(\cdot, \cdot, A)$ is the cost function of a one–class support vector machine [62], as before, with kernel k from (9.4). Explicitly,

$$F(w_c, \rho_c, A_c) = \frac{1}{2} <w_c, w_c>_\lambda - \rho_c + \frac{1}{\nu l} \sum_i [d_c(x_i)]_-. \tag{9.9}$$

We used a parameter $\nu = 2/3$. To get a diversity of feature detectors, we want to put a penalty on pairs of detectors, which are in a sense too similar. This is the role of the function G appearing in (9.7). Again, G measures similarity by computing an inner product, but this time it is not the inner product between two vectors $w_c, w_{c'}$ but an inner product depending on the subsets $A_c, A_{c'}$. Therefore, two feature detectors are similar if they detect features at close positions. In detail, we define a mapping ψ, which assigns each set A of nodes of the position lattice L a linear combination of two dimensional B-splines centred at these nodes,

$$\psi(A) = \sum_{(u,v,d)\in A} b_s(\cdot - u) b_s(\cdot - v), \tag{9.10}$$

with s denoting the step size of the lattice and b_s is a dilated B–spline defined in (8.8). Note that $\psi(A)$ is an element of $L_2(\mathbb{R}^2)$. Then we set

$$G(A_1, ..., A_m) = \sum_{c,c'\neq c} <\psi(A_c), \psi(A_{c'})>_{L_2} . \tag{9.11}$$

The function G computes an L_2 inner product, but as the arguments are linear combinations of cubic B–splines translated on a regular lattice, equation (9.10), the only values needed are the inner products

$$<b, b(\cdot - i)>_{L_2}, \quad i = 0, .., 3, \tag{9.12}$$

which can be computed in advance and stored in a table. In summary, our optimisation problem takes the following form,

$$\min_{(w_1,\rho_1,A_1),...,(w_m,\rho_m,A_m)} \sum_c F(w_c,\rho_c,A_c) + G(A_1,...,A_m). \tag{9.13}$$

As in our unsupervised algorithm of chapter 7, the function G couples the otherwise independent optimisation problems for the triples (w_c,ρ_c,A_c) together. Because of the special form of the optimisation problem, we use the following two–step procedure to find an approximate solution. We first choose a possibly large collection of subsets $\{A_1,...,A_n\}$ and then we optimise

$$\min_{(w_c,\rho_c)} F(w_c,\rho_c,A_c) \tag{9.14}$$

for each A_c among $\{A_1,...,A_n\}$ independently. By this, we find detectors $d_1,...,d_n$ for the tuples (w_1,ρ_1,A_1),...,(w_n,ρ_n,A_n). Second, we select iteratively maximally m detectors among the n solutions as follows. The first detector we choose is the one with a maximal margin ρ_{i_1} as an approximation to the one with minimal value of F. Next, suppose that we have already chosen the detectors $d_{i_1},...,d_{i_l}$ for $l \geq 1$. It remains to explain how we choose the $l+1^{st}$. We add detector d_{l+1} to the selected ones if it has the largest margin among all not yet selected detectors d_j with

$$\max_{i\in\{i_1,...,i_l\}} <\psi(A_i),\psi(A_j)>_{L_2} < \theta \tag{9.15}$$

for a parameter θ. In words, we concentrate on subsets A_j that are dissimilar to all previous subsets $A_{i_1},...,A_{i_l}$. Equation (9.15) can be seen as an approximation to (9.11). In our experiments we chose $\theta = (l+1)/2$. We continue until the we have chosen m detectors, or the remaining ones are either trivial or violate condition (9.15).

9.4.1 Adjusting the margins

During learning and selection of the models, the margins are set to a large value in order to find characteristic features of the data. Once the detection models are learnt and the non–trivial models are selected, we readjust the margins such that for every face from the data set there is at least one positive test. If for a face example x all tests fail, that is, for all c the feature test $d_c(x) < 0$, then we will select a test d_c and readjust the margin ρ_c as follows. For all c choose $\mu_c \in [0,1]$ with

$$<w_c, h_{A_c}[x]>_\lambda = \mu_c\rho_c. \tag{9.16}$$

Then, we perform an update

$$\rho_c \leftarrow \mu_c\rho_c \quad \text{for} \quad c = \arg\max_{c'} \mu_{c'}. \tag{9.17}$$

In words, we select the test that needs the smallest readjustment factor for the margin to make the result of the test positive. After scanning over all examples and adjusting of the margins, it is again possible that some of our feature detectors are trivial, and again we select only the non–trivial ones.

9.4.2 Setting the detection threshold

Given the selected detectors $d_1, ..., d_n$, we test for each subframe x_i in S how many times $d_c(x_i)$ is positive. We denote the number of positive detection tests for example x_i by m_i. Then we define a detection threshold ϑ. We set ϑ to be the $0.2 * |S|$–smallest value of the set $\{m_1, ..., m_{|S|}\}$, where $|S|$ denotes the cardinality of S. A face is detected on a new subframe x if the number of positive tests on x is larger or equal to ϑ. By definition of ϑ, we do not necessarily detect all faces in the training set S but accept 20% of false negatives. We do this for robustness, because by setting ϑ to the smallest number $\min_i m_i$, a wrong example in the training set would lead to a very low threshold and possibly to a detector which yields too many positive responses.

Until now we did not explain how to select suitable subsets A. This is the subject of the next section, where we explain an iterative strategy to select the subsets A.

9.5 Learning a hierarchy

In this section we explain how we can learn more and more complex feature detectors from simple ones. We do this by learning a *hierarchy* of subsets A. For each *level* l in the hierarchy we consider only sets

$$A^{(l)} \subset \{(u, v, d) | (u, v) \in L, d \in M\} \tag{9.18}$$

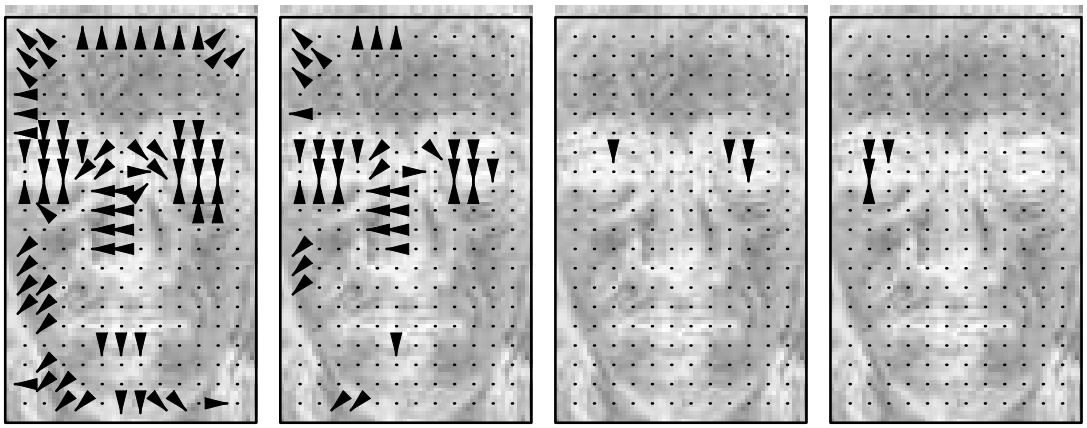
of cardinality $l + 1$. Subsets at a higher level in the hierarchy are unions of subsets at lower levels. The subsets are constructed from the lowest up to the highest level. We start for $l = 0$, considering all subsets with exactly one element, let us denote them by $A_i^{(0)}, i = 1, ..., m^{(0)}$. Then we train and select feature detectors on these subsets, as described in section 9.4. Next, we consider only the remaining subsets $A_j^{(0)}$ corresponding to a selected feature detector. From these subsets we get a set of subsets of level 1 by joining two non equal subsets. Now suppose we have constructed in this manner subsets $A_i^{(l)}, i = 1, ..., m^{(l)}$ of level l and we explain how we construct the subsets of level $l + 1$. Again, we learn feature detectors on the products $h_{A_i^{(l)}}$ of the local histograms, select a number of them, and consider the remaining subsets $A_i^{(l)}$ corresponding to one of the selected feature detectors. For such a subset $A_i^{(l)}$ of level l and a subset $A_j^{(0)}$ of level 0 we create a set of level $l + 1$,

$$A^{(l+1)} = A_i^{(l)} \cup A_j^{(0)} \quad \text{if} \quad A_i^{(l)} \cap A_j^{(0)} = \emptyset. \tag{9.19}$$

We do this for all pairs of subsets of level l and level 0. We put the constraint of joining only disjoint subsets in order to ensure that the features for higher levels depend on products at many different positions in the lattices L, M.

9.5.1 Estimating a normalisation factor

In this subsection we want to discuss a more technical, but important issue. By learning more and more complex features, we compute products of higher and higher order. But



A **B** **C** **D**

Figure 9.5: The detection tests: Each triangle indicates the position and the direction used for one of the detection tests. In figure **A** we plotted the total for all tests at level 0, in **B** the total at level 3. We plotted in the background the mean of the faces from figure 9.2. We can see how the tests are concentrated around the eyes, nose, mouth, and the contours of the face. Figure **C** and **D** show the first two selected tests of level 3.

by optimising (9.14) we have to specify a convergence tolerance of our algorithm. In order to speed up runtime, we would like to have a large tolerance. But on the other hand, if the values of the features are not close to one, by computing products it can happen that we end up with very small or very large numbers, and that an overall tolerance value is either to small or too large. We circumvent this problem by renormalising the kernel evaluation at each level, concretely we replace

$$k((x_i, A^{(l)}), (x_j, A^{(l)})) \to \kappa^{(l)} k((x_i, A^{(l)}), (x_j, A^{(l)})), \tag{9.20}$$

where $\kappa^{(l)}$ is the normalisation factor at level l. We use the previously learnt detection models at level $l-1$ to estimate $\kappa^{(l)}$. At level $l-1$ we estimate the order of magnitude of kernel evaluations $k((x_i, A^{(l-1)}), (x_j, A^{(l-1)}))$ by the mean margin $\bar{\rho}^{(l-1)}$, where $\bar{\rho}^{(l-1)}$ denotes the mean of all margins at level $l-1$. Therefore, if the data is suitably normalised we start with a renormalisation factor $\kappa^{(0)} = 1$ for level 0. Then, inductively we define

$$\kappa^{(l)} = \frac{1}{\bar{\rho}^{(l-1)}} \kappa^{(l-1)} \tag{9.21}$$

at level l.

9.6 Detection results

Our training set S consists of 1374 examples of the four persons. To learn the feature detectors we removed 80 examples of each person that are used for testing. We choose the 80 examples such that they are the translated versions of a single frame sequence

level	0	1	2	3	4	5	6	7	8
detectors	80	80	18	13	11	8	7	5	5
threshold	22	33	6	4	3	3	1	1	1
correct accept	79	59	80	80	80	80	80	80	80
above threshold	286	265	395	270	436	434	590	602	687

Table 9.1: The learnt feature detectors at the levels 0 to 8 (first row) and the result on the test data: the second row shows the number of detection tests at each level with the detection threshold (third row). The last two rows show the result on the test data, consisting of 80 face examples.

of 20 examples, and in a way that the training set still consists of examples of both illumination conditions. We started at level 0 and learnt 9 levels. Therefore, the last level computes products of order 9. Note that at level 0 we learn $12 * 20 * 8 = 1920$ feature detectors. We recall that the lattice L has $12 * 20$ nodes and the lattice M has 8 nodes for the directions. Because at every location only 1 or 2 of the 8 local histograms will allow a feature detector with large margin, and not at all locations we expect to have strong edges, only a fraction of them are really needed. Furthermore, we would like to keep the number of detectors at level 0 small, because we use their one point subsets $A_i^{(0)}$ to construct the subsets at higher levels, see (9.19), in order to learn only few models at higher levels. Therefore, we set the maximal number of detectors at each level to be 80. The first line in table 9.1 shows the number of non trivial detectors at each level. One can see that it is rapidly decreasing, which is a consequence of our constraint (9.15). This is desirable, because it allows us to use few tests of higher order for detection. The second line shows the detection threshold, that is, the number of tests that have to be positive in order to detect a face. As the tests of higher order extract more complex features, the threshold is decreasing.

Figure 9.5 illustrates the estimated detection tests. Shown in A are the tests at level zero. Each triangle indicates a point in the lattice, the position stands for the position in lattice L and the orientation of the triangle represents the node in the lattice M. We can see the influence of the illumination conditions during the collection of our dataset. Because subjects were either looking in direction of the window — the only light source — or the window was on their right, there is an asymmetry in the dataset. Therefore, more tests are positioned to the right of the face than to the left. In B the total of all tests at level 3 are shown. Each feature test consists of products of order four. Note that from the figure it is not clear which one is multiplied with which other, but the distribution of all test over the lattices can be seen. As the tests of level three are constructed from tests at level 0, the triangles in B are a subset of the triangles in A. In C, D the first two selected tests at level 3 are shown.

After learning, we tested the detectors on the before unseen face examples. The test was done on the entire images $\hat{x}$. We recall that we have one sequence consisting of 20 images of each of the four persons. Hence, in total we have 80 test images containing a

face. The fourth line in table 9.1 gives the number of detected faces. Except for the first two levels, all faces were detected. The last line shows the number of times a face in a subframe was detected. For example at level 3, on average, in an image with a face, three subframes pass the detection test, whereas at level 7 between seven and eight locations in an image pass the test. This is first a consequence of the local translation invariance of our representation and second, because a large fraction of detected features are edges, which are longer than our local window that cuts the image into local parts. We note that even if multiple subframes of an image pass the detection test, they are never displaced more than two nodes from the optimal position. Furthermore, we tested the detectors on non–face images. We used a few images of different objects with a cluttered office background, figure 9.3 shows some examples. We found that there were very few false detections, but we did not systematically collect non–face images to quantify the fraction of false detections. In a further test session we saw that the detector is tolerant enough that even persons unseen during training are detected.

9.7 The recognition kernel

We are in the following situation. We are given a set $\{x_1, ..., x_l\}$ of example images of faces. Each image belongs to one of the persons, thus defining a label for each example. The idea is to run our support vector algorithm for multicategorical data to learn a representation of the data. Once the representation is learnt, we can try to recognise one of the persons from the training set in the following way. For a new image $\hat{x}$ from the vision sensor we apply the face detection step discussed above. This then selects a sublattice of the lattice $\hat{L}$ defined on $\hat{x}$ corresponding possibly to a face region. Then the vector v composed of the local histograms $h_{(.)}[\hat{x}]$ with respect to this sublattice is of the same dimension as the vectors composed of the local histograms $h_{(.)}[x_i]$ over the lattice L. We apply our decision functions d_c of the multiclass representation to v to test if in the given region a face of one of the persons is present, that is, the decision function will attribute the vector to its class, or not.

In order to run our algorithm from chapter 4 we need a measure of similarity, a kernel. In section 9.3 we defined a kernel for detecting faces. The kernel was constructed in order to find similarities among faces, in particular, among faces of different persons. On the other hand, the kernel for recognition has a different task, it should detect dissimilarities between persons. Therefore we define a new kernel for the recognition task. For two images x_i, x_j we measure the dissimilarity between the two features $\phi(x_i, \{(u, v, d)\})$ and $\phi(x_j, \{(u, v, d)\})$ by

$$\Delta_{(u,v,d)} = \phi(x_i, \{(u, v, d)\}) - \phi(x_j, \{(u, v, d)\}), \tag{9.22}$$

with ϕ defined in (9.3). Note that $\Delta \in \mathbb{R}^q$ where q is the number of bins. The recognition kernel is then

$$k_{rec}(x_i, x_j) = \prod_{(u,v,d)\in L\times M} e^{-||\Delta_{(u,v,d)}||^2_\lambda/2\sigma^2}, \tag{9.23}$$

$\nu\backslash\sigma^2$	1	2	5	10	20	50	100
0.05	107	99	91	87	85	83	84
0.1	111	108	100	98	97	96	95
0.2	123	120	116	113	111	112	112

Table 9.2: Cross validation results: The number of false rejections out of the 240 shown test examples of the 3 persons for each value of the parameters ν (first column) and σ^2 (first row). There was no false acceptance of examples of the three seen and the fourth unseen person.

where $||\cdot||_\lambda$ is the norm induced by the scalar product $<\cdot,\cdot>_\lambda$. The kernel k_{rec} is a positive definite kernel by (6). We used the function $e^{-\cdot}$ to turn the dissimilarity measure $||\Delta_{(u,v,d)}||^2_\lambda$ in a similarity measure. Note that there is a general relation between the two [61].

9.7.1 Estimating the parameters

To estimate the kernel parameter σ^2 and the parameter ν we use a cross validation procedure. We split the data multiple times in a training and a test set. Each training set consists of the examples of three persons, but we left out 80 examples of each person, which we put in the test set together with the examples of the fourth person. As for detection, we choose the 80 examples such that they are the translated versions of a frame sequence of 20 frames, and in a way that the training set still consists of examples of both illumination conditions. In summary, we used eight different splits of our data in training and test set. Now, for each pair of values for the parameters σ^2 and ν we train and test our algorithm on these eight splits and store the result. To make this procedure not too time consuming, we need some prior guesses of the possible values of the two parameters. By the ν–property 4.2.6, we have a clear interpretation leading to a prior for ν. First, ν smaller than $\min_c 1/l_c$, where l_c denotes the number of training examples of class c, gives the same solution as ν equal to this minimum. In our case $\min_c l_c = 160$. Because the persons were advised to perform small movements in front of the camera, we expect to have some outliers due to too large movements. Therefore, the minimum value for ν is expected to be too small, and we use a minimum of 0.05. On the other hand, we tried to be careful by acquiring the face images. Therefore, we do not expect to have too many outliers, and we fix the maximal value for ν to be 0.2. Furthermore, we choose an intermediate value, in total we choose $\nu \in \{0.05, 0.1, 0.2\}$. Choosing a set of reasonable values for σ^2 is more difficult. The inequalities (8.17) give us some indications. By using the weighted scalar product (9.5), the kernel (9.23) is most sensitive to differences in the bin of low numbers, that is, strong edges. Let us consider images in which we have at each location either a very strong edge in one direction only or none at all. Therefore, at each position in the lattice we have a maximal squared distance to another image of this type of about 1. Thus, we expect σ^2 not to be larger than the number of nodes in the

level	ν	σ^2	# ex. of seen	# ex. of unseen	# false reject	# false accept
3	0.1	50	240	> 240	113	0

Table 9.3: Result of detection and recognition combined. The first column denotes the level of the detection model used. Then, ν is the parameter of the multi–class algorithm controlling the number of outliers, and σ^2 is the kernel parameter. Then, next are the number of test examples of persons seen during training and not seen, respectively. At the end are the number of false rejections (images containing a face of a person seen during training, but the person is not recognised), and false acceptances (images containing a face of a person not seen during training, but the person is falsely identified).

lattice, which is $20 \times 12 = 240$. In summary, we choose $\sigma^2 \in \{0.5, 1, 2, 5, 10, 20, 50, 100\}$.

Table 9.2 shows the number of false rejections, averaged over the eight runs. Note that there was no false acceptance. As expected from section 4.2.6, by increasing ν the number of false rejections increases. The results show that the solution is not too sensitive to the choice of the parameter σ^2. From these results we chose a value of $\sigma^2 = 50$ and the more conservative value $\nu = 0.1$ to train a final model. The final model is trained using all the training data and is tested during another session.

9.8 Recognition results

The goal is to recognise the persons whose face is in our dataset. But recognition should be possible in a new session, days or weeks after storing the face images even when the environment has changed. Especially changes in the illumination conditions are difficult to deal with [52]. But before doing so, we first put the detection and recognition model together, to test recognition not only on subframes, but on the whole image. This is the subject of the next subsection.

9.8.1 Detection and recognition on training session

Let us test the model learnt on subimages on the whole images. We test this by scanning over the whole local histogram decomposition and applying our detection and recognition model on all possible subdecompositions. Note that this is far more difficult, because we fix the lattice $\hat{L}$ on the whole image and then consider subdecompositions. Thus, in general the lattice L at the position used during training is not a sublattice of $\hat{L}$. Table 9.3 shows the models we chose and the result, which is the mean over two runs of the four different splits of the data. We chose the examples as described in subsection 9.7.1. As desired, there was no false acceptance. On the other hand, nearly every second frame containing a known face was rejected. This might seem a lot, but we remind the reader that persons perform small movements in front of the camera, therefore, not every face in a frame can and needs to be recognised.

# persons in training set	4
# unseen persons in test	5
# persons detected	all
# persons recognised	all, using multiple frames
# false acceptance	0

Table 9.4: Summary of the results on the test session.

9.8.2 Results on the test session

Now having trained the face detectors and the recognition model, we tested their performance. This was about two weeks after acquiring the dataset of faces. We recall that all the data used for training was acquired in a single session. Without changing any parameter, three of the four persons were immediately recognised, whereas the fourth person was not. It was already seen by inspection of the training data that this person performed larger movements than the other persons, and a couple of examples had to be eliminated from the training database. Note that the illumination conditions were different for the test session than for the training session, as it was snowing, while during collection of the data it was a sunny day. During the test session, we stored some more examples of the fourth not recognised person. Then, we proceeded as follows. We added this new data to our dataset and retrained the parameters of the representative vector w_4 corresponding to the fourth person, while keeping all the other parameters fixed, which is efficient compared to retraining the whole model. We recall that for performing the elementary steps 5.1.1 of optimisation we need to show two examples of the same class only. Therefore, for retraining the representative vector of one class keeping the others fixed, we perform elementary steps for examples of this class, as long as there are violators of the KKT–conditions. Then, when there are no more violators, we readjust the margins of all classes.

Three weeks later another test was carried out, with the retrained recognition model. Now, all four persons were correctly recognised. Furthermore, five persons not seen before were sitting in front of the camera. Interestingly, the faces of all five persons were detected. There was no false acceptance, all the five subjects were correctly rejected. Table 9.4 summarises the results.

Let us note the runtime of detection and recognition. To speed up recognition, we tested for each frame maximally at one location to recognise a face. In other words, even if a face was detected at multiple locations, we selected only the location which passed a maximal number of detection tests. This did not affect performance, but did speed up processing, as the recognition part is time consuming. We used a notebook with a PPC processor running at 400MHz. Running the interface for the camera, detection of a face and then trying to recognise it took about 1 second. If no face is detected, multiple frames can be processed in 1 second. By construction, our representation is invariant to translation and it has a local invariance to rotation and scale. Therefore, a person

seen during training is very quickly recognised. By performing small movements in front of the camera, 5 frames are normally enough for recognition. We note that this is an estimation and not a precise measurement. We did not do precise measurements, as this depends largely on how precise the person is positioned in front of the camera before one starts counting. We note that the processing routines could be further optimised. The interface of the camera was developed independently from the detection and recognition part, and therefore some frame processing is done twice.

9.9 Discussion

For the detection and recognition of faces, we did not use "hand designed" features, such as putting more weight on certain regions of the face. All we did was cutting a rectangular region containing the face. We did this for two reasons. First, for practical reasons, because our images are gradient images of low resolution, and it is sometimes hardly possible to exactly locate the eyes by hand. Second, from an abstract point of view, we need little prior knowledge about a face, which makes our approach more general in that it could be used to recognise other objects as well. Other face recognition systems use hand designed features like the eye and mouth region [69], or in addition the nose [11]. Experiments in psychology show that the most important features are the head contour, the eye region, the mouth region and the nose [35]. But feature saliency varies considerably among different individuals [36] and different persons use different strategies [34]. Other studies show that there is evidence that the facial features appropriate for discriminating among dark faces are different from those appropriate for white faces [67]. Therefore, ideally we should learn the features from the data.

In contrast to recognition, for face detection we learn the features from the data. We recall that our detection kernel takes as input an example face x together with the set A of lattice locations, at which products are computed. We chose this in a way that it fits into the framework of our unsupervised algorithm. But we could define different kernels, namely for each subset A a kernel

$$k_A(x_i, x_j) = k((x_i, A), (x_j, A)),$$

which calculates products over that subset. Even though this reformulation is trivial, the interpretation is not because we can interpret the iterative strategy to determine the sets A, as a strategy to learn the kernel.

By estimating the parameters of the detector we use a dataset of a few persons only, and therefore the resulting detector is not necessarily general enough to detect any face. It is a detector of the faces of the persons in our database. Still, it detected the other test persons, but as there were few of them, it is not enough to compare the performance with other detection systems. In addition, we can detect faces only in a limited range of scales and poses.

The application of our multiclass support vector algorithm showed us the following. The algorithm is general and can be used for other applications. On the other hand,

the specially designed processing of the images is specific but important for the face recognition application. The algorithm has few tunable parameters. The most important ones are the fraction ν of outliers and the width σ of the Gaussian kernel. Hence, already in the first test session our system recognised three of the four persons, without fine tuning of parameters.

Let us note the following limitation. During a test session we made the following experiment. One of the persons, whose face images are in the training set, wears glasses with a characteristic frame. During a test session, another person, not seen during training, wore these glasses. Sitting in front of the camera and performing small movements, this person was nearly recognised as the owner of the glasses. This shows us the following limitations: Learning from a small group of individuals only is on one hand an easier task, as there are less possibilities for confusion. On the other hand, it is more difficult to estimate characteristic features from only a few sample faces. In term of the above example, as there were not different persons with these special glasses in the training set, the glasses served as an important feature to identify that person. This is not a weakness of the algorithm, but shows the fact that our training sample was small.

Of course, there are many things that could be further improved. We want to point out the following points. First, there is the fact discussed above that we did not learn the recognition kernel. Even though our simple kernel performed good enough to recognise faces, it would be good to optimise the kernel to be more efficient. For example, by using only a subset of all the histograms. Second, we treat each frame as being an independent image. We could us the detection result of the preceding image as a prior for the next frame, in order to speed up detection, as used in [66]. Third, our local image representation has an important feature which we did not use. The B–splines fulfil a two–scale relation, that is, the B–spline b dilated by a factor 2 is a linear combination of translated B–splines,

$$b\Big(\frac{x}{2}\Big) = \sum_i \lambda_i b(x-i), \tag{9.24}$$

for suitable chosen coefficients λ_i. Therefore, the local histograms computed with a lattice of step size $2n$ can be efficiently computed by a weighted sum of local histograms at step size n. A whole hierarchy of local histograms can be computed in this way. This could be used for more efficient processing, as one uses a multi–resolution chain in wavelet theory. In summary, there is still room for improvements.

Chapter 10

Conclusion

Finding appropriate learning strategies is not an easy task as there are multiple constraints. The class of functions that are possible solutions of the algorithm has to be sufficiently rich. But if the class is too large, the complexity of learning from examples can become prohibitive. In addition, for face recognition, the representation of the data should be robust as there are several sources of noise. Illumination, the environment, and the pose of a person in front of the camera are continually changing.

Motivated by the close resemblance of the one– and two–class support vector machine, we developed a suitable class of functions and an optimisation algorithm to learn the categorisation of data into multiple classes. This algorithm proved to be well suited for the recognition of faces. For a fast processing, a detection step is used. Detection consists of testing for occurrences of characteristic features. These features are estimated from the face data available for training.

For face recognition, various strategies exist in the literature. Obviously, our approach is different as we use image sequences acquired with the CSEM adaptive vision sensor and not with a standard camera. In addition, our objective was different. Learning from a small group of individuals only is on one hand an easier task, as there are less possibilities for confusion. On the other hand, it is more difficult to estimate characteristic features from only a few sample faces. Representing an image as a collection of local histograms weighed by a spline kernel is a compact robust feature containing all the necessary information. Furthermore, the processing is well adapted to the sensor output.

10.1 Main results

In chapter 2 we reviewed how the kernel induces geometrical structures. We proposed a method to use this geometrical structure to optimise the decision hyperplane of a support vector machine in a normalised feature space to increase performance.

Chapters 4 to 6 are devoted to a new support vector type optimisation problem for the categorisation of data belonging to multiple classes. Our approach can be seen as a generalisation of the support vector machine algorithms for one and two classes to the case of more than two classes. It is conceptually different from other multi–class support vector approaches. In our case, we allow to reject examples, that is, they are classified as belonging to none of the classes from the training set. Of course, there is a tradeoff for an algorithm between performance and complexity. We demonstrated that our algorithm is simpler in training and the solution is more compact. As expected, we pay for this by a small decrease in performance.

We developed in chapter 5 an efficient implementation to find a minimum of our multi–categorical optimisation problem. The algorithm is an adaptation of the Sequential Minimal Optimisation algorithm to our case. In addition, we showed that optimisation efficiency can be improved using finite accuracy calculations to avoid a large number of small but time consuming updates. We proved the termination of our algorithm after a finite number of steps. Finally, we explored learning in an online setting.

In chapter 7 we showed possibilities to use similar versions of our multi–class optimisation problem for unsupervised learning.

We then turned to an application of our algorithms to real–time object recognition. We constructed a new representation of images in chapter 8. As the sensor output is ordered with respect to decreasing local contrast, it was natural to use a histogram representation over the contrast. Our histograms are local in position and direction of the gradient. Therefore, they allow to distinguish between position and direction of the gradient, and between different local contrasts. Furthermore, our representation has local translation, scale, and rotation invariance.

Eventually, in chapter 9 we designed an optimisation procedure to estimate characteristic features of a face. These features were used to detect potential locations of a face in an image. They were estimated from a training set of face images from a small number of persons. We applied our categorisation algorithm to this same training set. With the estimated parameters we could recognise these persons later, even under different illumination conditions. Faces of persons, unseen during training, were correctly rejected.

10.2 Perspectives

Much of the research effort has been spent to develop a classification algorithm with a suitable tradeoff between efficiency and performance, and to have a robust image representation. While these objective have been achieved, there are other aspects which could further be improved. The sensor output consists of information from a sequence of images. Of course, this sequence has a temporal coherence. In our recognition strategy, we only use the possibility to choose an image from multiple images, in which a person can be recognised. But we could use the temporal coherence to predict information of a future image from past images to reduce the amount of data processing. This could increase efficiency.

We did not explore the possibility of learning online. Having enough data to detect faces accurately enough, we could learn new persons while they are sitting in front of the camera.

Furthermore, we did not use all the features of the camera and our representation. Our representation allows easily to construct a hierarchy of local histograms. First, a hierarchy with respect to the location of the histogram. From histograms over small regions, histograms over lager regions can be obtained by summing the former ones. Second, one could construct a hierarchy with respect to the number of bins in the histogram. By taking only the high local contrast information, we can construct histograms with

fewer bins. Such hierarchies would allow a faster processing, as multiresolution chains do in wavelet theory.

Bibliography

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

[2] K. Allemand, K. Fukuda, T. M. Liebling, and E. Steiner. A polynomial case of unconstrained zero–one quadratic optimization. *Math. Program., Ser. A*, 91:49–52, 2001.

[3] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *Journal of Machine Learning Research*, pages 113–141, 2000.

[4] S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789, 1999.

[5] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337 – 404, 1950.

[6] F. R. Bach and M. I. Jordan. Kernel independent component analysis. *Journal of Machine Learning Research*, 3:1–48, 2002.

[7] M. Barbaro, P.-Y. Burgi, A. Mortara, P. Nussbaum, and F. Heitger. A 100x100 pixel silicon retina for gradient extraction with steering filter capabilities and temporal output coding. *IEEE Journal of Solid–State Circuits*, 37(2):160–172, 2002.

[8] H. Bauer. *Probability theory.* de Gruyter, 1996.

[9] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–137, 2001.

[10] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 1995.

[11] M. Bichsel. *Strategies of robust object recognition for the automatic identification of human faces.* PhD thesis, Eidgenössische Technische Hochschule Zürich (ETHZ), 1991.

[12] S. Borer and W. Gerstner. Support vector representation of multi–categorical data. In J. R. Dorronsoro, editor, *Artificial Neural Networks — ICANN 2002*, pages 733–738. Springer Lecture Notes in Computer Science, Vol. 2415, 2002.

[13] S. Borer and W. Gerstner. A new kernel clustering algorithm. In L. Wang et. al., editor, *Neural Information Processing— ICONIP 2002*. IEEE Proceedings, in print, 2003.

[14] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning theory*, 1992.

[15] E. J. Bredensteiner and K. P. Bennet. Multicategory classification by support vector machines. *Computational Optimizations and Applications*, 12:53–79, 1999.

[16] C.J.C. Burges. Geometry and invariance in kernel based methods. In B. Schölkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods: Support Vector Learning*. MIT Press, 1999.

[17] P. Buser and M. Imbert. *Vision*. MIT Press, 1992.

[18] O. Catoni. Catoni's simple proof of a simple inequality, 2002. Available http://www.proba.jussieu.fr/users/catoni/homepage.

[19] C.-C. Chang and C.-J. Lin. Training ν-support vector classifiers: Theory and algorithms. *Neural Computation*, 13(9):2119–2147, 2001.

[20] C.-C. Chang and C.-J. Lin. Libsvm – a library for support vector machines, 2002. Available http://www.csie.ntu.edu.tw/∼cjlin/libsvm/.

[21] K. Cramer and Y. Singer. On the learnability and desing of output codes for multiclass problems. *Computational Learning Theory*, pages 35–46, 2000.

[22] K. Cramer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *COLT-01*, 2001.

[23] N. Cristianini and J. Shawe-Taylor. *Support Vector Machines*. Cambridge University Press, 2000.

[24] T. G. Dietterich and G. Bakiri. Solving multi–class learning problems via error–correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[25] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[26] F. Fleuret and D. Geman. Coarse-to-fine visual selection. *International Journal of Computer Vision*, 41(1/2):85–107, 2001.

[27] W. T. Freeman and E. L. Adelson. The design and use of steerable filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13:891–906, 1991.

[28] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37:277–296, 1999.

[29] J. H. Friedman. Another approach to polychotomous classification. Technical report, Departement of Statistics and Standford Linear Accelerator Center, Standford University, 1996.

[30] M. Girolami. Mercer kernel–based clustering in feature space. *IEEE Transactions on Neural Networks*, 9(6):780–784, 2002.

[31] T. Graepel and K. Obermayer. Fuzzy topographic kernel clustering. In W. Brauer, editor, *Proceedings of the 5th GI Workshop Fuzzy Neuro Systems '98*, pages 90–97, 1998.

[32] A. B. A. Graf, A. J. Smola, and S. Borer. Classification in a normalized feature space using support vector machines. *IEEE Transactions on Neural Networks*, 14 (3):597–605, 2003.

[33] Y. Guermeur. Combining discriminant models with new multi–class svms. *Pattern Analysis and Applications*, 5:168–179, 2002.

[34] N. D. Haig. How faces differ — a new comparative technique. *Perception*, 14:601–615, 1985.

[35] N. D. Haig. Exploring recognition with interchanged facial features. *Perception*, 15: 235–247, 1986.

[36] N. D. Haig. High–resolution facial feature saliency mapping. *Perception*, 15:373–386, 1986.

[37] D. Haussler. Convolution kernels on discrete structures. Technical report ucsc-crl-99-10, University of Santa Cruz, 1999.

[38] D. Hubel and T. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology*, 195:215–243, 1968.

[39] D. H. Hubel. *Eye, Brain and Vision*. W. H. Freeman, 1988.

[40] T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

[41] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13:637–649, 2001.

[42] G. Kimeldorf and G. Wahba. Some results on tchebycheffian spline functions. *Journal of Mathathematical Analysis and Applications*, 33:82–95, 1971.

[43] J. Kivinen, A. J. Smola, and R. C. Wiliamson. Online learning with kernels. In *NIPS*, 2002.

[44] A. Kowalczyk. Maximal margin perceptron. In *Advances in Large Margin Classifiers*, pages 75–113. MIT Press, 2000.

[45] U. Kressel. Pairwise classification and support vector machines. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 255–268. 1999.

[46] Y. Lee, Y. Lin, and G. Wahba. Multicategory support vector machines. In *Proceedings of the 33rd Symposium on the Interface*, 2001.

[47] Y. Lin, Y. Lee, and G. Wahba. Support vector machines for classification in nonstandard situations. *Machine Learning*, 46:191–202, 2002.

[48] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1998.

[49] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London*, A 209:415–446, 1909.

[50] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proc. Computer Vision and Pattern Recognition '97*, pages 130–136, 1997.

[51] C. Papageorgiou and T. Poggio. A trainable system for object detection. *International Journal of Computer Vision*, 38(1):15–33, 2000.

[52] P. J. Phillips, P. Grother, R. J. Micheals, D. M. Blackburn, E. Tabassi, and J. M. Bone. Face recognition vendor test, 2003. Available http://www.frvt.org/FRVT2002/documents.htm.

[53] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in kernel methods – Support vector learning*, pages 185–208. 1999.

[54] J. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin dags for multiclass classification. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12, pages 547–553, Cambridge, MA, 2000. MIT Press.

[55] S. Romddhani, P. Torr, B. Schölkopf, and A. Blake. Computationally efficient face detection. In *Proceedings of the International Conference on Computer Vision*, pages 695–700, 2001.

[56] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[57] W. Rudin. *Real and Complex Analysis*. McGraw–Hill, 1987.

[58] Pierre-Francois Rüedi, Pascal Heim, Francois Kaess, Eric Grenet, Friedrich Heitger, Pierre-Yves Burgi, Stève Gyger, and Pascal Nussbaum. A 128x128 pixel 120db dynamic ranage vision sensor chip for image contrast and orientation extraction. In *IEEE International Solid–State Circuits Conference (ISSCC 2003)*, pages 226–227. IEEE, Vol. 46, 2003.

[59] J. L. Schnapf, B. J. Nunn, M Meister, and D A. Baylor. Visual transduction in cones of the monkey *macaca fascicularis*. *Journal of Physiology*, 1990.

[60] B. Schölkopf. *Support Vector Learning*. R. Oldenbourg Verlag, Munich, 1997.

[61] B. Schölkopf. The kernel trick for distances. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13, Cambridge, MA, 2001. MIT Press.

[62] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high–dimensional distribution. *Neural Computation*, 13 (7), 2001.

[63] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.

[64] B. Schölkopf, A. J. Smola, and K.-R. Müller. Kernel principal component analysis. *Neural Computation*, 10, 1998.

[65] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12(5):1207–1245, 2000.

[66] G. Shakhnarovich, P. A. Viola, and B. Moghaddam. A unified learning framework for real time face detection and classification. In *Proceedings of the Int. Conf. on Automatic Face and Gesture Recognition*, 2002.

[67] J. Shepherd. Social factors in face recognition. In G. Davies, H. D. Ellis, and J. Shepherd, editors, *Percieving and Remembering Faces*. 1981.

[68] F. Smeraldi. Ranklets: orientation selective non-parametric features applied to face detection. In *Proceedings of the Symposium of the Swedish Society for Automated Image Analysis, Lund (Sweden)*, pages 1–4. Lund University, March 2002.

[69] F. Smeraldi and J. Bigun. Retinal vision applied to facial features detection and face authentication. *Pattern Recognition Letters*, 23:463–475, 2002.

[70] M. Spivak. *A Comprehensive Introdution to Differential Geometry*. Brandeis University, 1970.

[71] S. J. Thorpe, D. Fize, and C. Marlot. Speed of processing in the human visual system. *Nature*, 381:520–522, 1996.

[72] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[73] M. Unser. Sampling – 50 years after Shannon. *Proceedings of the IEEE*, 88(4): 569–587, 2000.

[74] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[75] P. Viola and M. J. Jones. Robust real–time object detection. In *Proceedings of IEEE Workshop on Statistical and Computational Theories of Vision*, 2001.

[76] G. Wahba. *Spline Models for Observational Data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, 1990.

[77] C. Watkins. Dynamic alignment kernels. In A. J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.

[78] J. Weston and C. Watkins. Support vector machines for multi-class pattern recognition. In *Proceedings of the 6th European Symposium on Artificial Neural Networks (ESANN)*, 1999.

[79] L. Wiskott, J.-M. Fellous, N. Krüger, and Ch. von der Malsburg. Face recognition by elastic bunch graph matching. *IEEE Transactions on Pattern Analysis and Machine Intellingence*, 19(7):775–779, 1997.

[80] A. L. Yuille. Deformable templates for face recognition. *Journal of Cognitive Neuroscience*, 3(1):59–70, 1991.

Curriculum vitae

Silvio Borer

Avenue de l'Avant–Poste 3, 1005 Lausanne
email: silvio.borer@epfl.ch
Phone: 021 693 3910
Born November 21, 1973 in Dornach, Switzerland

Education:

2003:	PhD degree from the Swiss Federal Institute of Technology (EPFL) in Lausanne Assistant in the Laboratory of Computational Neuroscience Adviser: Wulfram Gerstner Co-Adviser: Sabine Süsstrunk
1999:	Diploma in mathematics and physics from the University of Zurich
1993:	Matura in scientific direction

Languages:

German: native
English: fluent, written and spoken
French: fluent, written and spoken
Spanish: comprehension and possibility for conversation

Publications:

S. Borer and W. Gerstner. Support vector representation of multi–categorical data. *Journal of Machine Learning Research*, submitted.

A. B. A. Graf, A. J. Smola, and S. Borer. Classification in a normalized feature space using support vector machines. *IEEE Transactions on Neural Networks*, 14(3):597–605, 2003.

S. Borer and W. Gerstner. A new kernel clustering algorithm. In L. Wang et. al., editor, *Neural Information Processing— ICONIP 2002*. IEEE Proceedings, in print, 2003.

S. Borer and W. Gerstner. Learning multiple categories from sequences of examples. In

L. Wang et. al., editor, *Neural Information Processing— ICONIP 2002*. IEEE Proceedings, in print, 2003.

S. Borer and W. Gerstner. Support vector representation of multi–categorical data. In J. R. Dorronsoro, editor, *Artificial Neural Networks — ICANN 2002*, pages 733–738. Springer Lecture Notes in Computer Science, Vol. 2415, 2002.

S. Borer and S. Süsstrunk. Opponent Color Space Motivated by Retinal Processing. *Colour in Graphics, Imaging and Vision— IS&T CGIV 2002*, pages 187-189. IS&T, 2002.