Support Vector Learning

vorgelegt von
Diplom-Physiker, M.Sc. (Mathematics)
Bernhard Schölkopf
aus Stuttgart

Vom Fachbereich 13 — Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. K. Obermayer
Berichter: Prof. Dr. S. Jähnichen
Berichter: Prof. Dr. V. Vapnik

Tag der wissenschaftlichen Aussprache: 30. September 1997

Berlin 1997

D 83

Support Vector Learning

Bernhard Schölkopf

Dissertation zum Dr. rer. nat. — Zusammenfassung der wesentlichen Ergebnisse

Inhalt der Arbeit ist das Lernen von Mustererkennung als statistisches Problem. Eine Lernmaschine extrahiert aus einer Menge von Trainingsmustern Strukturen, die ihr die Klassifikation neuer Beispiele erlauben. Die Arbeit behandelt folgende Fragen:

- *Welche "Merkmale" sollte man aus den einzelnen Trainingsmustern extrahieren?* — Zum Studium dieser Frage wurde eine neue Form von nichtlinearer Hauptkomponentenanalyse ("Kernel PCA") entwickelt. Durch die Benutzung von Integraloperatorkernen kann in Merkmalsräumen sehr hoher Dimensionalität (z.B. im 10^{10}-dimensionalen Raum aller Produkte von 5 Pixeln in 16×16-dimensionalen Bildern) eine lineare Hauptkomponentenanalyse durchgeführt werden. Im Ursprungsraum betrachtet, führt dies zu nichtlinearen Merkmalsextraktoren. Der Algorithmus besteht in der Lösung eines Eigenwertproblemes, in dem die Wahl verschiedener Kerne die Verwendung einer großen Klasse verschiedener Nichtlinearitäten gestattet.

- *Welche der Trainingsmuster enthalten am meisten Information über die zu konstruierende Entscheidungsfunktion?* — Diese Frage, wie auch die folgende, wurde anhand des vor wenigen Jahren von Vapnik vorgeschlagenen "Support-Vektor-Algorithmus" innerhalb des von Vapnik und Chervonenkis entwickelten statistischen Paradigmas des *Lernens aus Beispielen* untersucht. Durch die Wahl verschiedener Integraloperatorkerne ermöglicht dieser Algorithmus die Konstruktion einer Klasse von Entscheidungsregeln, die als Spezialfälle Neuronale Netze, Polynomiale Klassifikatoren und Radialbasisfunktionennetze enthält. Für Bilder von 3-D-Objektmodellen und handgeschriebenen Ziffern konnte gezeigt werden, dass die verschiedenen Entscheidungsregeln in ihrer Klassifikationsgenauigkeit den besten bisher bekannten Verfahren ebenbürtig sind, und dass ihre Konstruktion lediglich eine kleine, von der speziellen der Kerne weitgehend unabhängige Teilmenge (in den betrachteten Beispielen 1% – 10%) der Trainingsmenge verwendet.

- *Wie kann man am besten "A-Priori"-Information verwenden, die zusätzlich zu den Trainingsmustenr vorhanden ist?* (beispielsweise Information über die Invarianz einer Klasse von Bildern unter Translationen) — die Arbeit schlägt drei Verfahren vor, die alle zu deutlichen Verbesserungen der Klassifikationsgenauigkeit führen. Zwei der Verfahren bestehen in der Konstruktion von speziellen, dem Problem angepassten Integraloperatorkernen. Das dritte Verfahren verwendet Invarianztransformationen, um aus der oben genannten Teilmenge (der "Support-Vektor-Menge") aller Trainingsmuster zusätzliche künstliche Trainingsbeispiele zu generieren.

genehmigt: Prof. Jähnichen

Foreword

The Support Vector Machine has recently been introduced as a new technique for solving various function estimation problems, including the pattern recognition problem. To develop such a technique, it was necessary to first extract factors responsible for future generalization, to obtain bounds on generalization that depend on these factors, and lastly to develop a technique that constructively minimizes these bounds.

The subject of this book are methods based on combining advanced branches of statistics and functional analysis, developing these theories into practical algorithms that perform better than existing heuristic approaches. The book provides a comprehensive analysis of what can be done using Support Vector Machines, achieving record results in real-life pattern recognition problems. In addition, it proposes a new form of nonlinear Principal Component Analysis using Support Vector kernel techniques, which I consider as the most natural and elegant way for generalization of classical Principal Component Analysis.

In many ways the Support Vector machine became so popular thanks to works of Bernhard Schölkopf. The work, submitted for the title of Doktor der Naturwissenschaften, appears as excellent. It is a substantial contribution to Machine Learning technology.

Vladimir N. Vapnik, Member of Technical Staff, AT&T Labs Research
Professor, Royal Holloway and Bedford College, London

Vorwort

Interessant an der Arbeit von Herrn Schölkopf sind nicht nur die fachlichen Aspekte, sondern auch die unterschiedlichen und sehr intensiven Kontakte zu internationalen Forschungseinrichtungen. Sie zeigen, daß der Autor sowohl in der Lage ist, seine Ergebnisse im wissenschaftlichen Spitzenfeld zu präsentieren und zu plazieren, als auch aus Arbeiten der "Community" heraus seine Ergebnisse zu entwickeln. Aus dieser Sicht läßt sich auch die fachliche Qualität der Arbeit ersehen.

Herr Schölkopf untersucht zwei Grundprobleme der Klassifikation großer Datenmengen. Zum einen ist dies die Extraktion weniger aber relevanter starker Merkmale zur Reduktion der Informationsflut, und zum anderen die Beschreibung von Datenbeispielen, die charakteristisch für ein gegebenes Klassifikationsproblem sind. Beide Probleme werden von Herrn Schölkopf sowohl theoretisch als auch in Experimenten ausgiebig und erschöpfend untersucht. Sowohl die in der Arbeit entwickelte, sehr elegante Methode der nichtlinearen Merkmalsextraktion (kernel PCA), als auch die vorgeschlagenen Weiterentwicklungen der Support-Vektor-Maschine benutzen *schwache* Merkmale und setzen sich damit konzeptuell von der oben beschriebenen Philosophie der *starken* Merkmale ab. Somit spiegelt sich in der Arbeit gewissermaßen ein Paradigmenwechsel in der Klassifikation und Merkmalsextraktion wider.

Herr Schölkopf war während seiner Dissertation ein gern gesehener Gast der GMD FIRST Berlin, und es war eine Freude, seine Arbeit zu lesen und zu betreuen. Insbesondere freue ich mich, daß Herr Schölkopf seine Forschung in seiner neuen Position bei GMD FIRST weiterführen wird.

Stefan Jähnichen, Direktor, GMD FIRST
Professor, Technische Universität Berlin

Contents

Acknowledgements

First of all, I would like to express my gratitude to Prof. H. Bülthoff, Prof. S. Jähnichen, and Prof. V. Vapnik for supervising the present dissertation, and to Prof. K. Obermayer for chairing the committee in the "Wissenschaftliche Aussprache". I am grateful to Vladimir Vapnik for introducing me to the world of statistical learning theory during numerous extended discussions in his office at AT&T Bell Laboratories. I have deep respect for the completeness and depth of the body of theoretical work that he and his co-workers have created over the last 30 years. To Heinrich Bülthoff, I am grateful for introducing me to the world of biological information processing, during my work on the Diplom and the doctoral dissertation. He created a unique research atmosphere in his group at the Max-Planck-Institut für biologische Kybernetik, and provided excellent facilities without which the present work would not have been possible. I would like to thank Stefan Jähnichen for his advice, and for hosting me at the GMD Berlin during several research visits. A significant amount of the reported work was influenced and carried out during these stays, where I closely collaborated with A. Smola and K.-R. Müller.

Thanks for financial support in the form of grants go to the Studienstiftung des deutschen Volkes and the Max-Planck-Gesellschaft. In addition, it was the Studienstiftung that made it possible in the first place that I got to know Vladimir Vapnik at AT&T in 1994, and that helped in getting A. Smola join the team in 1995.

A number of people contributed to this dissertation in one way or another. Let me start with V. Blanz, C. Burges, M. Franz, D. Herrmann, K.-R. Müller, and A. Smola, who helped at the very end, in proofreading the manuscript, leading to many improvements of the exposition.

The work for this thesis was done at several places, and each of the groups that I was working in deserves substantial credit. More than half of the time was spent at the Max-Planck-Institut für biologische Kybernetik, and I would like to thank all members of the group for providing a stimulating interdisciplinary research atmosphere, and for bearing with me when I maltreated their computers at night with my simulations. Special thanks go to the people in the Object Recognition group for a number of lively discussions, and to the small group of theoreticians at the MPI, who helped me in various ways over the years.

Almost one year of the time of my thesis work was spend in the adaptive systems group at AT&T and Bell Laboratories. I learnt a lot about machine learning as applied to real-world problems from all members of this excellent group. In particular, I would like to express my thanks to C. Burges, L. Bottou, C. Cortes, and I. Guyon for helping me understand Support Vectors, and to L. Jackel, Y. LeCun, and C. Nohl for making my stays possible in the first place. In addition to their scientific advice, E. Cosatto, P. Haffner, E. Säckinger, P. Simard and C. Watkins have helped me through their friendship. Finally, I want to express my gratitude for the possibility to use code and databases developed and assembled by these people and their co-workers. A substantial part of this thesis would not have been possible without this.

During my time in the USA, I also had the opportunity to spend a month at the Center for Biological and Computational Learning (Massachusetts Institute of Technology), hosted by T. Poggio. I would like to thank him, as well as G. Geiger, F. Girosi, P. Niyogi, P. Sinha, and K. Sung, for hospitality and fruitful discussions.

At the GMD, I had the possibility to interact with the local connectionists group, which (in addition to those mentioned already) included J. Kohlmorgen, N. Murata, and G. Rätsch. The present work profited a great deal from my stays in Berlin.

When starting to do research on one's own, one cannot help but noticing that the more specialized the field of work is, the more international and widespread seems to be the group of people interested in it. Out of the scientists working on machine learning and perception, I want to thank J. Buhmann, S. Canu, A. Gammerman, J. Held, D. Kersten, J. Lemm, D. Leopold, P. Mamassian, G. Roth, S. Solla, F. Wichmann, and A. Yuille for stimulating discussions and advice.

Without first studying science, it is hard to become a scientist. Studying science predominantly means arguing about scientific problems. During my education, I was in the favourable position to have enough people for scientific discussions. With many of these friends and teachers, there is still contact and exchange of ideas. I would like to thank all of them, and in particular G. Alli, C. Becker, V. Blanz, D. Corfield, H. Fischer, M. Franz, D. Henke, D. Herrmann, D. Janzing, U. Kappler, D. Köpf, F. Lutz, A. Rieckers, M. Schramm, and G. Sewell.

Finally, without my parents, I would not even have studied anything in the first place. Many thanks to them.

Summary

Learning how to recognize patterns from examples gives rise to challenging theoretical problems: given a set of observations,

- *which of the observations should be used to construct the decision boundary?*
- *which features should be extracted from each observation?*
- *how can additional information about the decision function be incorporated in the learning process?*

The present work is devoted to the above issues, studying *Support Vectors* in high-dimensional *feature spaces*, and *Kernel PCA feature extraction.*

The material is organized as follows. We start with an introduction to the problem of pattern recognition, to concepts of statistical learing theory, and to feature spaces nonlinearly related to input space (Chapter 1). The paradigm for learning from examples which is studied in this thesis, the Support Vector algorithm, is described in Chapter 2, including empirical results obtained on realistic pattern recognition problems. The latter in particular includes the finding that the set of Support Vectors extracted, i.e. those examples crucial for solving a given task, is largely independent of the type of Support Vector machine used. One specific topic in the development of Support Vector learning, the incorporation of prior knowledge, is studied in some detail in Chapter 4: we describe three methods for improving classifier accuracies by making use of transformation invariances and the local structure of images. Intertwined between these two chapters, we propose a novel method for nonlinear feature extraction (Chapter 3), which works in the same types of features spaces as Support Vector machines, and which forms the basis of some developments of Chapter 4. Finally, Chapter 5 gives a conclusion. As such, it partly reiterates what has just been said, and the reader who still remembers the present summary when arriving at Chapter 5 may find it amusing to contemplate whether the conclusion coincides with what had been evoked in their mind by the summary that they have just finished reading.

Disclaimer. This thesis was written in an interdisciplinary research environment, and it was supervised by a statistician, a biologist, and a computer scientist. Accordingly, it attempts to be of interest for rather different audiences. If your interests fall into one of these categories exclusively, please bear with me: whenever you encounter a section which you find utterly useless, boring, or incomprehensible, there is the theoretical possibility that it is of interest to somebody else. Accordingly, please feel free to ignore all these parts.

“To see a thing one has to comprehend it. An armchair presupposes the human body, its joints and limbs; a pair of scissors, the act of cutting. What can be said of a lamp or a car? The savage cannot comprehend the missionary’s Bible; the passenger does not see the same rigging as the sailors. If we really saw the world, maybe we would understand it.”

J. L. Borges, There are more things. In: *The Book of Sand*, 1979, Penguin, London.

Chapter 1

Introduction and Preliminaries

The present work studies visual recognition problems from the point of view of learning theory. This first chapter sets the scene for the main part of the thesis. It gives a brief introduction of the problem of Learning Pattern Recognition from examples. The two main contributions of this thesis are motivated in the conceptual part of the chapter. Section 1.1 discusses ***prior knowledge*** *that might be available in addition to the set of training examples, and introduces the problem of extracting useful* ***features*** *from individual examples. The technical part of the chapter, Sec. 1.2, gives a concise description of some mathematical concepts of* ***Statistical Learning Theory*** (Vapnik, *1995b). This theory describes learning from examples as a problem of limited sample size statistics and provides the basis for the* ***Support Vector algorithm****. Finally, Sec. 1.3 introduces mathematical concepts of feature spaces, which will be of central importance to all following chapters.*

1.1 Learning Pattern Recognition

Let us think of a *pattern* as an abstraction, defined by a collection of possible instances such as sample images. When trying to learn how to recognize a pattern, we face the problem that we will often be unable to see all instances during learning, yet we want to be able to *recognize* as many as possible. The *extensive* notion of a pattern that we just introduced already suggests a specific approach to the problem of pattern recognition: a statistician tries to collect a large number of instances, and use inductive methods to learn how to recognize them.

For an alternative point of view, consider a pattern as something observable which is generated by an underlying physical *entity*, as for instance the 2-D views of a 3-D object. To recognize a pattern of this nature, a physicist would try to understand the laws governing the entity, and the mechanisms by which the pattern is brought about.

In this process, it may turn out that different observables, or functions thereof, contain different amounts of information for understanding the underlying entity, i.e. it may be the case that from the initial raw observations, we have to extract useful *features* ourselves.

The current work is located in the intersection of the aspects sketched in the above three paragraphs. It studies an inductive learning algorithm which has been developed in the framework of statistical learning theory, and it tries to enhance it by incorporating prior knowledge about a recognition task at hand. Finally, it studies the extraction of features for the purpose of recognition.

Even though pattern recognition is not limited to the visual domain, we shall focus on visual recognition. Much of what is said in this thesis, however, would equally apply to the recognition of acoustic patterns, say.

In the remainder of this section, we introduce the terminology which is used in discussing different aspects of visual recognition problems: these are, in turn, the data, the tasks, and the methods for recognition.

Data. Different types of pattern recognition problems make different types of assumptions about the underlying causes generating the patterns. Nevertheless, it is possible to discuss them in a common framework which we try to describe presently. It draws from machine learning terminology; as such, it will differ from psychological usage of the relevant terms in some respects.[1]

Observers visually perceive *views*. Sets of views constitute *classes*. Sometimes, classes have a structure that goes beyond being mere collections of views. For instance, the class of all views of rainbows has the property that if a specific view belongs to it, then so do all views which are generated by translating it, parallel to the horizon. *Objects* are specific classes, with a rich class structure, containing for instance all view transformations corresponding to rigid 3-D transformations of a specific underlying physical entity. Some of these transformations are shared by all objects, for instance translations; others, like deformations, are object-specific.

More radically, and fundamentally *view-based*, we could give up the notion of priority of the underlying physical entities, and think of an object only as a collection of views, with a specific class structure. On a practical level, this is the approach pursued in the current work. The distinction between objects and other classes then becomes a distinction between different types of transformation invariances. For instance, a rainbow would not be an object, as we cannot possibly see it from above, not even with a spacecraft. The class of handwritten digits '6' would not be an object for similar reasons; in fact, an image plane rotation by 180° would even take us into the class '9'. As an aside, we note that mathematics and physics have already undergone a paradigm shift away from the notion of objects as "things in the world", towards studying their transformation properties. In mathematics, this is exemplified in Felix Klein's *Erlanger Programm* (Klein, 1872) which shifts geometry away from points and lines towards transformation groups; in physics, an example is the modern definition of elementary particles as transformation group representations (e.g. Primas, 1983). Kac and Ulam (1968) refer to this as

> "[...] the immensely powerful and fruitful idea that much can be learned

[1]The ideas put forward in the following were influenced by discussions with people in the MPI's object recognition group, in particular with V. Blanz.

> about the *structure* of certain objects by merely studying their *behaviour under the action of certain groups*."

Later in the thesis, the reader will encounter methods for improving visual recognition systems by taking into account transformation properties of handwritten characters and 3-D objects (Sec. 4.2).

Prior Knowledge. The statistical approach of learning from examples in its pure form neglects the additional knowledge of class structure described above. However, the latter, referred to as *Prior Knowledge*, can be of great practical relevance in recognition problems.

Suppose we were given temporal sequences of detailed observations (including spectra) of double star systems, and we would like to predict whether, eventually, one of the stars will collapse into a black hole. Given a small set of observations of different double star systems, including target values indicating the eventual outcome (supposing these were available), a purely statistical approach of learning from examples would probably have difficulties extracting the desired dependency. A physicist, on the other hand, would infer the star masses from the spectra's periodicity and Doppler shifts, and use the theory of general relativity to predict the eventual fate of the stars.

Of course, one could argue that the physicist's model of the situation is based on a huge body of prior examples of situations and phenomena which are related to the above in one way or another. This, however, is exactly how the term *prior* knowledge should be understood in the present context. It does not refer to a Kantian *a priori*, as prior to all experience, but to what is prior to a given problem of learning from examples.

What do we do, however, if we do not have a dynamical model of what is happening behind the scenes? In this case, which for instance applies whenever the underlying dynamics is too complicated, the strengths of the purely statistical approach become apparent. Let us consider the case of handwritten character recognition. When a human writer decides to write the letter 'A', the actual outcome is the result of a series of complicated processes, which in their entirety cannot be modelled comprehensively. The intensity of the lines depends on chemical properties of ink and paper, their shape on the friction between pencil and the paper, on the dynamics of the writer's joints, and on motor programmes initiated in the brain, these in turn are based on what the writer has learnt at school — the chain could be continued ad infinitum. Accordingly, nobody tries to recognize characters by completely modelling their generation.

However, the lack of a complete dynamical model does not mean that there is *no* prior knowledge in handwritten digit recognition. For instance, we know that handwritten digits do not change their class membership if they are translated on a page, or if their line thickness is slightly changed. This type of knowledge can be used to augment the purely statistical approach (Sec. 4.2). More abstract prior knowledge in many visual recognition tasks includes the fact that the correlations between nearby image locations are often more reliable features for recognition than those with larger distances (Sec. 4.3).

Features. Before we proceed to the tasks that can be performed, depending on the available data, we need to introduce a concept widely used in both statistics and in the analysis of human perception. In its general form, a *feature detector* or *feature extractor* is a function which assigns a (typically scalar) value to each raw observation. Often, a number of different such functions are applied to the observations in a *feature extraction process*, leading to a preprocessed vector *representation* of the data. The goal of extracting features is to improve subsequent stages of processing, be it by improving accuracies in a recognition task, or by reducing storage requirements or processing time.

The feature functions serving this purpose can either be specified in advance, for instance in a way such that they incorporate prior knowledge about a problem at hand, or computed from the set of observations. Both approaches, as well as combinations thereof, shall be addressed in this thesis (Chapter 3, Sec. 4.2.2, Sec. 4.3).

The actual term *feature* is used with different meanings. In vision research and psychophysics, it is mainly used for the optimal stimulus of a corresponding feature detector. However, note that given a nonlinear feature detector, it may be practically impossible to determine this optimal stimulus. In statistics, on the other hand, the term feature mostly refers to the *feature values*, i.e. the outputs of feature detectors, or to the feature detector itself. Possibly, this ambiguity arose from the fact that, in some cases, the different meanings coincide: in the case where the feature detector consists of a dot product with a weight vector, as in linear neural network model receptive fields, the optimal stimulus is aligned with the weight vector, and thus the two can be identified.

Tasks. Suppose we are only given solitary views, and neither nontrivial classes nor objects (which were structured collections of views). Then out of the *tasks* of discrimination, classification, and identification, only *discrimination* can be carried out on these views. This does, however, not prevent the term discrimination from being used also in the context of classes and objects. Discrimination, the mere detection of a difference, can be preceded by feature extraction processes; in these cases, results will depend on the extraction process used.

Classification consists of attributing views to classes, and thus requires the existence of classes. These can be specified abstractly — by describing features, or Gibsonian affordances ("something to sit on"), e.g. — or provided (approximately) by a sample of training views. This definition of classes by training sets is widespread in machine learning; it will also be the paradigm that we are going to use in this thesis. One talks about *yes-no* and *old-new* classification tasks (one specified class) or *naming* tasks (several classes). Pattern recognition problems like Handwritten Digit Recognition are examples of classification.

Similarly, *identification* consists in determining to which object a presented view belongs. As objects are special types of classes, we again have the possibility for the above tasks. Identification makes sense only for objects: for instance, it is meaningless to ask whether the rainbow we see today is a view of the same (object as a) rainbow we saw last year.

In this thesis, we study classification and identification. Often, both of these tasks are referred to as *recognition*, the term which we shall mostly employ. Indeed, when classes are given by training sets, the question whether there is an underlying object producing the observed views becomes secondary. It is then only of relevance insofar as it determines the type of prior knowledge available.

Human Object Recognition. The position that object recognition is not about recovering physical 3-D entities, but about learning their views, and potentially also their transformation properties, can be supported by biological and psychological evidence. Bülthoff and Edelman (1992) have shown that when recognizing unfamiliar objects, observers exhibit viewpoint effects which suggest that they do not recover the 3-D shape of objects, but rather build a representation based on the actual training views (cf. also Logothetis, Pauls, and Poggio, 1995). They thought of this representation as an interpolation mechanism (cf. Poggio and Girosi, 1990), but one could of course conceive of more sophisticated mechanisms for combining information contained in the training views. In the above terminology, one might argue that due to their unfamiliarity, the wire frame objects of Bülthoff and Edelman (1992) make it very hard to use the transformations which form the structure of the underlying class of views. Ullman (1996) has put forward a multiple-view variant of his theory of "recognition by alignment", where objects are recognized by aligning them with stored view templates. The alignment process can make use of certain transformations specific to the object in question. The results of Troje and Bülthoff (1996) have shown that these transformations in some cases directly operate on 2-D views, and that they are much simpler than transformations using an underlying 3-D model: in experiments probing face recognition under varying poses, observers performed better on views which were obtained by simply applying a mirror reversal transformation to a previously seen view, rather than by rotating the head in depth to generate the true view of the other side. Rao and Ballard (1997) recently proposed a model in which the "what" and the "where" pathway (Mishkin, Ungerleider, and Macko, 1983) in the visual system are conceived of as estimating object identity and transformations, respectively. Using a collection of patches taken from natural images, they construct a generative model for the data which learns, transforms and linearly combines simple basis functions. Their model, however, does not directly make use of the valuable information contained in the *temporal* stream of visual data: comparing subsequent images, e.g. by optic flow techniques, would give a more direct means of constructing processing elements encoding transformations. Indeed, in the dorsal stream (the "where" pathway), neurons have been found coding for various types of large-field transformations (Duffy and Wurtz, 1991). Of somewhat related interest are the large-field neurons in the fly's visual system, coding for specific flow fields which are generated by the fly's movement in the environment (Krapp and Hengstenberg, 1996).

Representations and Processes. The above illustrates that the question of how, given a recognition problem, the actual processing can be performed, is intimately related to underlying *representations* (of classes or objects), computed by some feature extraction process. A representation should satisfy certain constraints in terms of storage cost, computational cost and accuracy.

General classes without structure are not compressible (except for a separate compression of the individual images). Classes with some internal structure can be compressed, hence a smaller representation is possible, which in turn makes generalization to novel views possible (cf. Kolmogorov, 1965; Rissanen, 1978). This is the underlying computational reason for the constructive nature of perception.

If we can generate a class (e.g. an object) from some prototype views using a specified set of transformations, we can represent it as the set of prototypes plus transformations. The more prototypes we store, the less complex are the transformations that we need to remember. In this sense, there is a continuum of different view-based approaches. In principle, further compression is conceivable if we allow for the construction of a suitable underlying representation. E.g., Ullman's approach of storing a 3-D model plus the set of 3-D transformations (Ullman, 1989) is cheap in terms of storage: storing these transformations is almost for free, and storing one 3-D model is reasonably cheap. Constructing this representation, however, may be computationally quite expensive. Reading out and matching 2-D views, on the other hand, is computationally rather cheap if done in parallel neural architecture. The type of representation to be used should thus depend on the task, e.g. on speed constraints. Indeed, proponents of view-based object recognition theories are mainly concerned with *fast* recognition tasks (Bülthoff and Edelman, 1992). Moreover, the storage cost strongly depends on the task and the type of feature extraction applied to the raw data.

In some cases, we can extract features from views which allow reasonably high recognition accuracies while enabling us to work with much simpler sets of transformations. For instance, if there exists a diagnostic object feature which is visible from all viewpoints, we only need to store the feature (e.g. the colour), the extraction process (which can be thought of as a specific image transformation which needs to be stored), and the fact that it may occur anywhere in the view (i.e. the set of all image plane translations).

Clearly, the set of features which are extracted from views influences all further processing. Applied to our setting, *constructing* a feature representation consists of two parts: the features have to be extracted from a possibly large set of views, and the transformations which connect features belonging to views of the same class have to be computed. This may require a trade-off: for some feature representations, the extraction process is difficult (e.g. using correspondence methods, Beymer and Poggio, 1996; Vetter and Troje, 1997), whereas the computations of transformations might be simple. A similar trade-off exists in *utilizing* such a representation: for a recognition task done by matching, e.g., we would have to extract features from test views, and transform them to match stored ones. Put in machine learning language, features should be used which allow solving a given task within specified limits on training time, testing speed, error rate, and memory requirements.

Implementations. So far, not much has been said about actual implementations of recognition systems. The present work focuses on algorithmic questions rather than on questions of implementation, both with respect to the computational side and with respect to the biological side of the recognition problem. The former normally need not be justified: in statistics, scientific studies of mere algorithms, without discussion of implementation details, are abundant. In biology, which is the main focus of interest in the group where much of the present work was carried out, the type of abstraction presented here is much less common. Indeed, the relevance of this thesis to biological pattern recognition is on the level of statistical properties of problems and algorithms — not more, and not less. In our hope that this type of theoretical work should be of interest to people studying the brain, we concur with Barlow (1995):

> "If artificial neural nets, designed to imitate cognitive functions of the brain, are truly performing tasks that are best formulated in statistical terms, then is this not likely also to be true of cognitive function in general? The idea that the brain is an accomplished statistical decision-making organ agrees well with notions to be sketched in the last section of this [Barlow's, the author] article."

To study object recognition from a *statistical* point of view, we shall in the following section briefly review some of the basic concepts and results of statistical learning theory.

1.2 Statistical Learning Theory

Out of the considerable body of theory that has been developed in statistical learning theory by Vapnik and others (e.g. Vapnik and Chervonenkis, 1968, 1974; Vapnik, 1979, 1995a,b), we briefly review a few concepts and results which are necessary in order to be able to appreciate the Support Vector learning algorithm, which will be used in a substantial part of the thesis.[2]

For the case of two-class pattern recognition, the task of *learning from examples* can be formulated in the following way: we are given a set of functions

$$\{f_\alpha : \alpha \in \Lambda\}, \quad f_\alpha : \mathbf{R}^N \to \{\pm 1\} \tag{1.1}$$

and a set of *examples*, i.e. pairs of *patterns* $\mathbf{x}_i$ and *labels* y_i,

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell) \in \mathbf{R}^N \times \{\pm 1\}, \tag{1.2}$$

[2] A high-level summary is given in (Schölkopf, 1996).

each one of them generated from an unknown probability distribution $P(\mathbf{x}, y)$ containing the underlying dependency. (Here and below, bold face characters denote vectors.) We want to *learn* a function f_{α^*} which provides the smallest possible value for the average error committed on independent examples randomly drawn from the same distribution P, called the *risk*

$$R(\alpha) = \int \frac{1}{2}|f_\alpha(\mathbf{x}) - y| \; dP(\mathbf{x}, y). \tag{1.3}$$

The problem is that $R(\alpha)$ is unknown, since $P(\mathbf{x}, y)$ is unknown. Therefore an *induction principle* for risk minimization is necessary.

The straightforward approach to minimize the *empirical risk*

$$R_{emp}(\alpha) = \frac{1}{\ell}\sum_{i=1}^{\ell}\frac{1}{2}|f_\alpha(\mathbf{x}_i) - y_i| \tag{1.4}$$

turns out not to guarantee a small actual risk, if the number ℓ of training examples is limited. In other words: a small error on the training set does not necessarily imply a high *generalization* ability (i.e. a small error on an independent *test* set). This phenomenon is often referred to as *overfitting* (e.g. Bishop, 1995). To make the most out of a limited amount of data, novel statistical techniques have been developed during the last 30 years. The *Structural Risk Minimization* principle (Vapnik, 1979) is based on the fact that for the above learning problem, for any $\alpha \in \Lambda$ and $\ell > h$, with a probability of at least $1 - \eta$, the bound

$$R(\alpha) \leq R_{emp}(\alpha) + \phi\left(\frac{h}{\ell}, \frac{\log(\eta)}{\ell}\right) \tag{1.5}$$

holds, where the *confidence term* ϕ is defined as

$$\phi\left(\frac{h}{\ell}, \frac{\log(\eta)}{\ell}\right) = \sqrt{\frac{h\left(\log\frac{2\ell}{h} + 1\right) - \log(\eta/4)}{\ell}}. \tag{1.6}$$

The parameter h is called the *VC(Vapnik-Chervonenkis)-dimension* of a set of functions. It describes the *capacity* of a set of functions. For binary classification, h is the maximal number of points which can be separated into two classes in all possible 2^h ways by using functions of the learning machine; i.e. for each possible separation there exists a function which takes the value 1 on one class and -1 on the other class.

A *learning machine* can be thought of as a set of functions (that the machine has at its disposal), an induction principle, and an algorithmic procedure for implementing the induction principle on the given set of functions. Often, the term learning machine is used to refer to its set of functions — in this sense, we talk about the capacity or VC-dimension of learning machines.

The bound (1.5), which forms part of the theoretical basis for Support Vector learning, deserves some further explanatory remarks.

Suppose we wanted to learn a "dependency" where $P(\mathbf{x}, y) = P(\mathbf{x})\cdot P(y)$, i.e. where the pattern $\mathbf{x}$ contains no information about the label y, with uniform $P(y)$. Given a training sample of fixed size, we can then surely come up with a learning machine which achieves zero training error. However, in order to reproduce the random labellings, this machine will necessarily require a VC-dimension which is large compared to the sample size. Thus, the confidence term (1.6), increasing monotonically with h/ℓ, will be large, and the bound (1.5) will *not* support possible hopes that due to the small training error, we should expect a small test error. This makes it understandable how (1.5) can hold independent of assumptions about the underlying distribution $P(\mathbf{x}, y)$: it always holds, but it does not always make a nontrivial prediction — a bound on an error rate becomes void if it is larger than the maximum error rate. In order to get nontrivial predictions from (1.5), the function space must be restricted such that the VC-dimension is small enough (in relation to the available amount of data).[3]

According to (1.5), given a fixed number ℓ of training examples, one can control the risk by controlling two quantities: $R_{emp}(\alpha)$ and $h(\{f_\alpha : \alpha \in \Lambda'\})$, Λ' denoting some subset of the index set Λ. The empirical risk depends on the *function* chosen by the learning machine (i.e. on α), and it can be controlled by picking the right α. The VC-dimension h depends on the *set of functions* $\{f_\alpha : \alpha \in \Lambda'\}$ which the learning machine can implement. To control h, one introduces a structure of nested subsets $S_n := \{f_\alpha : \alpha \in \Lambda_n\}$ of $\{f_\alpha : \alpha \in \Lambda\}$,

$$S_1 \subset S_2 \subset \ldots \subset S_n \subset \ldots, \tag{1.8}$$

whose VC-dimensions, as a result, satisfy

$$h_1 \leq h_2 \leq \ldots \leq h_n \leq \ldots \tag{1.9}$$

For a given set of observations $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell)$ the *Structural Risk Minimization principle* chooses the function $f_{\alpha_\ell^n}$ in the subset $\{f_\alpha : \alpha \in \Lambda_n\}$ for which the guaranteed

[3]The bound (1.5), formulated in terms of the VC-dimension, is only the last element of a series of tighter bounds which are formulated in terms of other concepts. This is due to the inequalities

$$H^\Lambda(\ell) \leq H^\Lambda_{ann}(\ell) \leq G^\Lambda(\ell) \leq h\left(\log\frac{2\ell}{h} + 1\right), \quad (\ell > h). \tag{1.7}$$

The VC-dimension h is probably the most-used and best-known concept in this row. However, the other ones lead to tighter bounds, and also play important roles in the conceptual part of statistical learning theory: the *VC-entropy* H^Λ and the *Annealed VC-entropy* H^Λ_{ann} are used to formulate conditions for the consistency of the empirical risk minimization principle, and for a fast rate of convergence, respectively. The *Growth function* G^Λ provides both of the above, independently of the underlying probability measure P, i.e. independently of the data. The VC-dimension h, finally, provides a constructive upper bound on the Growth function, which can be used to design learning machines (for details, see Vapnik, 1995b).

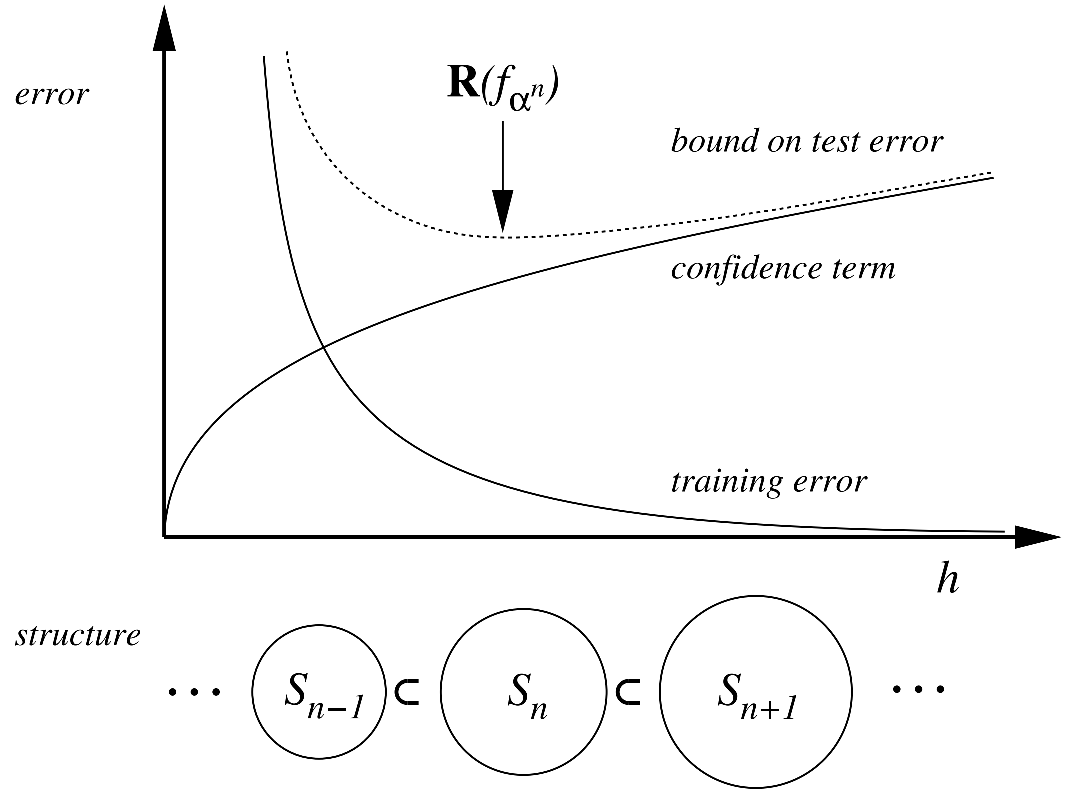

FIGURE 1.1: Graphical depiction of (1.5), for fixed ℓ. A learning machine with larger complexity, i.e. a larger set of functions S_n, allows for a smaller training error; a less complex learning machine, with a smaller S_i, has smaller VC-dimension and thus provides a smaller confidence term ϕ (cf. (1.6)). Structural Risk Minimization picks a trade-off in between these two cases by choosing the function of the learning machine f_{α^n} such that the risk bound (1.5) is minimal.

risk bound (the right hand side of (1.5)) is minimal (cf. Fig. 1.1). The procedure of selecting the right subset for a given amount of observations is referred to as *capacity control*.

We conclude this section by noting that analyses in other branches of learning theory have led to similar insights in the trade-off between reducing the training error and limiting model complexity, for instance as described by regularization theory (Tikhonov and Arsenin, 1977), Minimum Description Length (Rissanen, 1978; Kolmogorov, 1965), or the Bias-Variance Dilemma (Geman, Bienenstock, and Doursat, 1992). Haykin (1994); Ripley (1996) give overviews in the context of Neural Networks.

1.3 Feature Space Mathematics

The present section summarizes some mathematical preliminaries which are essential for both Support Vector machines (Chapter 2) and nonlinear Kernel Principal Component Analysis (Chapter 3).

1.3.1 Product Features

Suppose we are given patterns $\mathbf{x} \in \mathbf{R}^N$ where most information is contained in the d-th order products (monomials) of entries x_j of $\mathbf{x}$,

$$x_{j_1} \cdot \ldots \cdot x_{j_d}, \tag{1.10}$$

where $j_1, \ldots, j_d \in \{1, \ldots, N\}$. In that case, we might prefer to *extract* these product features first, and work in the feature space F of all products of d entries. In visual recognition problems, e.g., this would amount to extracting features which are products of individual pixels.

For instance, in $\mathbf{R}^2$, we can collect all monomial feature extractors of degree 2 in the nonlinear map

$$\Phi : \mathbf{R}^2 \to F = \mathbf{R}^3 \tag{1.11}$$

$$(x_1, x_2) \mapsto (x_1^2, x_2^2, x_1 x_2). \tag{1.12}$$

This approach works fine for small toy examples, but it fails for realistically sized problems: for N-dimensional input patterns, there exist

$$N_F = \frac{(N+d-1)!}{d!(N-1)!} \tag{1.13}$$

different monomials (1.10), comprising a feature space F of dimensionality N_F. Already 16×16 pixel input images and a monomial degree $d = 5$ yield a dimensionality of 10^{10}.

In certain cases described below, there exists, however, a way of *computing dot products* in these high-dimensional feature spaces without explicitely mapping into them: by means of nonlinear kernels in input space $\mathbf{R}^N$. Thus, if the subsequent processing can be carried out using dot products exclusively, we are able to deal with the high dimensionality.

The following section describes how dot products in polynomial feature spaces can be computed efficiently, followed by a section which discusses more general feature spaces.

1.3.2 Polynomial Feature Spaces Induced by Kernels

In order to compute dot products of the form $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$, we employ kernel representations of the form

$$k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})), \tag{1.14}$$

which allow us to compute the value of the dot product in F without having to carry out the map Φ. This method was used by Boser, Guyon, and Vapnik (1992) to extend the *Generalized Portrait* hyperplane classifier of Vapnik and Chervonenkis (1974) to nonlinear Support Vector machines (Sec. 2.1). Aizerman, Braverman, and Rozonoer (1964) call F the *linearization space*, and use it in the context of the potential function classification method to express the dot product between elements of F in terms of elements of the input space. They also consider the possibility of choosing k a priori, without being directly concerned with the corresponding mapping Φ into F. A specific choice of k might then correspond to a dot product between patterns mapped with a suitable Φ.

What does k look like for the case of polynomial features? We start by giving an example (Vapnik, 1995b) for $N = d = 2$. For the map

$$C_2 : (x_1, x_2) \mapsto (x_1^2, x_2^2, x_1x_2, x_2x_1), \tag{1.15}$$

dot products in F take the form

$$(C_2(\mathbf{x}) \cdot C_2(\mathbf{y})) = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2 = (\mathbf{x} \cdot \mathbf{y})^2, \tag{1.16}$$

i.e. the desired kernel k is simply the square of the dot product in input space. Boser, Guyon, and Vapnik (1992) note that the same works for arbitrary $N, d \in \mathbf{N}$: as a straightforward generalization of a result proved in the context of polynomial approximation (Poggio, 1975, Lemma 2.1), we have:

Proposition 1.3.1 *Define C_d to map $\mathbf{x} \in \mathbf{R}^N$ to the vector $C_d(\mathbf{x})$ whose entries are all possible d-th degree ordered products of the entries of $\mathbf{x}$. Then the corresponding kernel computing the dot product of vectors mapped by C_d is*

$$k(\mathbf{x}, \mathbf{y}) = (C_d(\mathbf{x}) \cdot C_d(\mathbf{y})) = (\mathbf{x} \cdot \mathbf{y})^d. \tag{1.17}$$

Proof. We directly compute

$$\begin{aligned}(C_d(\mathbf{x}) \cdot C_d(\mathbf{y})) &= \sum_{j_1,\ldots,j_d=1}^{N} x_{j_1} \cdot \ldots \cdot x_{j_d} \cdot y_{j_1} \cdot \ldots \cdot y_{j_d} &(1.18)\\ &= \left(\sum_{j=1}^{N} x_j \cdot y_j\right)^d = (\mathbf{x} \cdot \mathbf{y})^d. &(1.19)\end{aligned}$$

$\square$

Instead of ordered products, we can use unordered ones to obtain a map Φ_d which yields the same value of the dot product. To this end, we have to compensate for the multiple occurence of certain monomials in C_d by scaling the respective monomial entries of Φ_d with the square roots of their numbers of occurence. Then, by this definition of Φ_d, and (1.17),

$$(\Phi_d(\mathbf{x}) \cdot \Phi_d(\mathbf{y})) = (C_d(\mathbf{x}) \cdot C_d(\mathbf{y})) = (\mathbf{x} \cdot \mathbf{y})^d. \tag{1.20}$$

For instance, if n of the j_i in (1.10) are equal, and the remaining ones are different, then the coefficient in the corresponding component of Φ_d is $\sqrt{(d-n+1)!}$ (for the general case, cf. Smola, Schölkopf, and Müller, 1997). For Φ_2, this simply means that (Vapnik, 1995b)

$$\Phi_2(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}\, x_1 x_2). \tag{1.21}$$

If $\mathbf{x}$ represents an image with the entries being pixel values, we can use the kernel $(\mathbf{x} \cdot \mathbf{y})^d$ to work in the space spanned by products of any d pixels — provided that we are able to do our work solely in terms of dot products, without any explicit usage of a mapped pattern $\Phi_d(\mathbf{x})$. Using kernels of the form (1.17), we take into account higher-order statistics without the combinatorial explosion (cf. (1.13)) of time and memory complexity which goes along already with moderately high N and d.

To conclude this section, note that it is possible to modify (1.17) such that it maps into the space of all monomials *up to* degree d, defining (Vapnik, 1995b)

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^d. \tag{1.22}$$

1.3.3 Feature Spaces Induced by Mercer Kernels

The question which function k does correspond to a dot product in some space F has been discussed by Boser, Guyon, and Vapnik (1992); Vapnik (1995b). To construct a map Φ induced by a kernel k, i.e. a map Φ such that k computes the dot product in the space that Φ maps to, they use Mercer's theorem of functional analysis (Courant and Hilbert, 1953):

Proposition 1.3.2 *If k is a continuous symmetric kernel of a positive[4] integral operator K, i.e.*

$$(Kf)(\mathbf{y}) = \int_C k(\mathbf{x}, \mathbf{y}) f(\mathbf{x})\, d\mathbf{x} \tag{1.23}$$

with

$$\int_{C \times C} k(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y})\, d\mathbf{x}\, d\mathbf{y} \geq 0 \tag{1.24}$$

for all $f \in L^2(C)$ (C being a compact subset of $\mathbf{R}^N$), it can be expanded in a uniformly

[4] When referring to operators, the term *positive* is always meant in the sense stated here. If we talk about positive *definite* operators, we will express this explicitely.

convergent series (on $C \times C$) in terms of Eigenfunctions ψ_j and positive Eigenvalues λ_j,

$$k(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{N_F} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{y}), \tag{1.25}$$

where $N_F \leq \infty$.

Note that originally proven for the case where $C = [a, b]$, this Proposition also holds true for general compact spaces (Dunford and Schwartz, 1963).

For the converse of Proposition 1.3.2, cf. Appendix D.1.

From (1.25), it is straightforward to construct a map Φ, mapping into a potentially infinite-dimensional l^2 space, which does the job. For instance, we may use

$$\Phi : \mathbf{x} \mapsto (\sqrt{\lambda_1}\psi_1(\mathbf{x}), \sqrt{\lambda_2}\psi_2(\mathbf{x}), \ldots). \tag{1.26}$$

We thus have the following result (Boser, Guyon, and Vapnik, 1992):[5]

Proposition 1.3.3 *If k is a continuous kernel of a positive integral operator (conditions as in Proposition 1.3.2), one can construct a mapping Φ into a space where k acts as a dot product,*

$$(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})) = k(\mathbf{x}, \mathbf{y}). \tag{1.27}$$

Besides (1.17), Boser, Guyon, and Vapnik (1992) and Vapnik (1995b) suggest the usage of Gaussian radial basis function kernels (Aizerman, Braverman, and Rozonoer, 1964)

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\,\sigma^2}\right) \tag{1.28}$$

and sigmoid kernels

$$k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + \Theta). \tag{1.29}$$

Note that all these kernels have the convenient property of unitary invariance, i.e. $k(\mathbf{x}, \mathbf{y}) = k(U\mathbf{x}, U\mathbf{y})$ if $U^\top = U^{-1}$ (if we consider complex numbers, then U^* instead of $U^\top$ has to be used). The radial basis function kernel additionally is translation invariant.

[5] In order to identify k with a dot product in another space, it would be sufficient to have pointwise convergence of (1.25). Uniform convergence lets us make an assertion which goes further: given an accuracy level $\epsilon > 0$, there exists an $n \in \mathbf{N}$ such that even if the range of Φ is infinite-dimensional, k can be approximated within accuracy ϵ as a dot product in $\mathbf{R}^n$, between images of

$$\Phi_n : \mathbf{x} \mapsto (\sqrt{\lambda_1}\psi_1(\mathbf{x}), \ldots, \sqrt{\lambda_n}\psi_n(\mathbf{x})).$$

1.3.4 The Connection to Reproducing Kernel Hilbert Spaces

The feature space that Φ maps into is a *reproducing kernel Hilbert space* (RKHS). To see this, we follow Wahba (1973) and recall that a RKHS is a Hilbert space of functions f on some set C such that all evaluation functionals $f \mapsto f(\mathbf{y})$ $(\mathbf{y} \in C)$ are continuous. In that case, by the Riesz representation theorem (e.g. Reed and Simon, 1980), for each $\mathbf{y} \in C$ there exists a unique function of $\mathbf{x}$, call it $k(\mathbf{x}, \mathbf{y})$, such that

$$f(\mathbf{y}) = \langle f, k(., \mathbf{y}) \rangle \tag{1.30}$$

(here, $k(., \mathbf{y})$ is the function on C obtained by fixing the second argument of k to $\mathbf{y}$, and $\langle ., . \rangle$ is the dot product of the RKHS). In view of this property, k is called a *reproducing kernel*.

Note that by (1.30), $\langle f, k(., \mathbf{y}) \rangle = 0$ for all $\mathbf{y}$ implies that f is identically zero. Hence the set of functions $\{k(., \mathbf{y}) : \mathbf{y} \in C\}$ spans the whole RKHS. The dot product on the RKHS thus only needs to be defined on $\{k(., \mathbf{y}) : \mathbf{y} \in C\}$ and can then be extended to the whole RKHS by linearity and continuity. From (1.30), it follows that in particular

$$\langle k(., \mathbf{x}), k(., \mathbf{y}) \rangle = k(\mathbf{y}, \mathbf{x}) \tag{1.31}$$

for all $\mathbf{x}, \mathbf{y} \in C$ (this implies that k is symmetric). Note that this means that any reproducing kernel k corresponds to a dot product in another space.

To establish a connection to the dot product in a feature space F, we next assume that k is a Mercer kernel (cf. Proposition 1.3.2). First note that it is possible to construct a dot product such that k becomes a reproducing kernel for a Hilbert space of functions

$$f(\mathbf{x}) = \sum_{i=1}^{\infty} a_i k(\mathbf{x}, \mathbf{x}_i) = \sum_{i=1}^{\infty} a_i \sum_{j=1}^{N_F} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{x}_i). \tag{1.32}$$

Using only linearity, which holds for any dot product $\langle ., . \rangle$, we have

$$\langle f, k(., \mathbf{y}) \rangle = \sum_{i=1}^{\infty} a_i \sum_{j,n=1}^{N_F} \lambda_j \psi_j(\mathbf{x}_i) \langle \psi_j, \psi_n \rangle \lambda_n \psi_n(\mathbf{y}). \tag{1.33}$$

Since k is a symmetric kernel, the ψ_i $(i = 1, \ldots, N_F)$ can be chosen to be orthogonal with respect to the dot product in $L^2(C)$. Hence it is straightforward to construct a dot product $\langle ., . \rangle$ such that

$$\langle \psi_j, \psi_n \rangle = \delta_{jn} / \lambda_j \tag{1.34}$$

(using the Kronecker symbol δ_{jn}), in which case (1.33) reduces to the reproducing kernel property (1.30) (using (1.32)).

To write $\langle .,. \rangle$ as a dot product of coordinate vectors, we thus only need to express the functions of the RKHS in the basis $(\sqrt{\lambda_n}\psi_n)_{n=1,\ldots,N_F}$, which is orthonormal with respect to $\langle .,. \rangle$, i.e.

$$f(\mathbf{x}) = \sum_{n=1}^{N_F} \alpha_n \sqrt{\lambda_n}\psi_n(\mathbf{x}). \tag{1.35}$$

To obtain the coordinates α_n, we compute, using (1.34),

$$\alpha_n = \langle f, \sqrt{\lambda_n}\psi_n \rangle = \langle \sum_{i=1}^{\infty} a_i \sum_{j=1}^{N_F} \lambda_j \psi_j(\mathbf{x}_i)\psi_j, \sqrt{\lambda_n}\psi_n \rangle = \sqrt{\lambda_n} \sum_{i=1}^{\infty} a_i \psi_n(\mathbf{x}_i). \tag{1.36}$$

Comparing (1.35) and (1.26), we see that F has the structure of a RKHS in the sense that for f and g given by (1.35) and

$$g(\mathbf{x}) = \sum_{j=1}^{N_F} \beta_j \sqrt{\lambda_j}\psi_j(\mathbf{x}), \tag{1.37}$$

we have

$$(\boldsymbol{\alpha} \cdot \boldsymbol{\beta}) = \langle f, g \rangle. \tag{1.38}$$

Note, moreover, that due to (1.35), we have $f(\mathbf{x}) = (\boldsymbol{\alpha} \cdot \Phi(\mathbf{x}))$ in F. Comparing to (1.30), this shows that $\Phi(\mathbf{x})$ is nothing but the coordinate representation of the kernel as a function of one argument (cf. also (1.27)).

To conclude the brief detour into RKHS theory, note that in (1.30), k does not have to be linear in its arguments; however, its action as an evaluation functional in Hilbert space is linear — this is the underlying reason why Mercer kernels compute bilinear dot products in Hilbert spaces: the dot product is obtained by combining two evaluations of a possibly nonlinear function in a suitable Hilbert space.

1.3.5 Kernel Values as Pairwise Similarities

In practice, we are given a finite amount of data $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$. The following simple observation shows that even if we do not want to (or are unable to) analyse a given kernel k analytically, we can still compute a map Φ such that k corresponds to a dot product in the linear span of the $\Phi(\mathbf{x}_i)$:

Proposition 1.3.4 *Suppose the data $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$ and the kernel k are such that the matrix*

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \tag{1.39}$$

is positive. Then it is possible to construct a map Φ into a feature space F such that

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)). \tag{1.40}$$

Conversely, for a map Φ into some feature space F, the matrix $K_{ij} = (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$ is positive.

Proof. Being positive, K can be diagonalized as

$$K = SDS^\top \tag{1.41}$$

with an orthogonal matrix S and a diagonal matrix D with nonnegative entries. Then

$$\begin{aligned} k(\mathbf{x}_i, \mathbf{x}_j) &= (SDS^\top)_{ij} && (1.42) \\ &= \sum_{k=1}^{\ell} S_{ik} D_{kk} \left(S^\top\right)_{kj} && (1.43) \\ &= \sum_{k=1}^{\ell} S_{ik} D_{kk} S_{jk} && (1.44) \\ &= (\mathbf{s}_i \cdot D\mathbf{s}_j), && (1.45) \end{aligned}$$

where we have defined the $\mathbf{s}_i$ as the rows of S (note that the columns of S would be K's Eigenvectors). Therefore, K is the dot product matrix (or *Gram matrix*) of the vectors $\sqrt{D_{kk}} \cdot \mathbf{s}_i$.[6] Hence the map Φ, defined on the $\mathbf{x}_i$ by

$$\Phi : \mathbf{x}_i \mapsto \sqrt{D_{kk}} \cdot \mathbf{s}_i, \tag{1.46}$$

does the job (cf. (1.40)).

Note that if the $\mathbf{x}_i$ are linearly dependent, it will typically not be the case that Φ can be extended to a linear map.

For the converse, assume an arbitrary $\boldsymbol{\alpha} \in \mathbf{R}^\ell$, and compute

$$\sum_{i,j=1}^{\ell} \alpha_i \alpha_j K_{ij} = \left(\sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i) \cdot \sum_{j=1}^{\ell} \alpha_j \Phi(\mathbf{x}_j) \right) \geq 0. \tag{1.47}$$

□

In particular, this result implies that given data $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$, and a kernel k which gives rise to a positive matrix K, it is always possible to construct a feature space F of dimensionality $\leq \ell$ that we are implicitely working in when using kernels.

If we perform an algorithm which requires k to correspond to a dot product in some other space (as for instance the Support Vector algorithm to be described below), it could happen that even though k does not satisfy Mercer's conditions in general, it still gives rise to a positive matrix K for the given training data. In that case,

[6] The fact that every positive matrix is the Gram matrix of some set of vectors is well-known in linear algebra (see e.g. Bhatia, 1997, Exercise I.5.10).

Proposition 1.3.4 tells us that nothing will go wrong during training when we work with these data. Moreover, if a kernel leads to a matrix K with some small negative Eigenvalues, we can add a small multiple of some positive definite kernel to obtain a positive matrix.[7]

Note, finally, that Proposition 1.3.4 does not require the $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$ to be elements of a vector space. They could be any set of objects which, for some function k (which could be thought of as a similarity measure for the objects), gives rise to a positive matrix $(k(\mathbf{x}_i, \mathbf{x}_j))_{ij}$. Methods based on pairwise distances or similarities have recently attracted attention (Hofmann and Buhmann, 1997). They have the advantage of being applicable also in cases where it is hard to come up with a sensible vector representation of the data (e.g. in text clustering).

[7] For instance, for the hyperbolic tangent kernel (1.29), Mercer's conditions have not been verified. It does not satisfy them in general: in a series of experiments with 2-D toy data, we noticed that the dot product matrix in K had some negative Eigenvalues, for most choices of Θ that we investigated (except for large negative values). Nevertheless, this kernel has successfully been used in Support Vector learning (cf. Sec. 2.3). To understand the latter, note that by shifting the kernel (i.e. choosing different values of Θ), one can approximate the shape of the polynomial kernel (which is known to be positive), as a function of $(\mathbf{x} \cdot \mathbf{y})$ (within a certain range), up to a vertical offset. This offset is irrelevant in SV learning: due to (2.15), adding a constant to all elements of the dot product matrix does not change the solution.

Chapter 2

Support Vector Machines

*This chapter discusses theoretical and empirical issues related to the **Support Vector (SV) algorithm**. This algorithm, reviewed in Sec. 2.1, is based on the results of learning theory outlined in Sec. 1.2. Via the use of kernel functions (Sec. 1.3), it gives rise to a number of different types of pattern classifiers (Vapnik and Chervonenkis, 1974; Boser, Guyon, and Vapnik, 1992; Cortes and Vapnik, 1995; Vapnik, 1995b).*

*The original contribution of the present chapter is largely empirical. Using object and digit recognition tasks, we show that the algorithm allows us to construct high-accuracy polynomial classifiers, radial basis function classifiers, and perceptrons (Sections 2.2 and 2.3), relying on almost identical subsets of the training set, their **Support Vector sets** (Sec. 2.4). These Support Vector Sets are shown to contain all the information necessary to solve a given classification task. To understand the relationship between SV methods and classical techniques, we then describe a study comparing SV machines with Gaussian kernels to classical radial basis function networks, with results favouring the SV approach. Following this, Sec. 2.6 shows that one can utilize the error bounds of learning theory to select values for free parameters in the SV algorithm, as for instance the degree of the polynomial kernel which will perform best on a test set (Schölkopf, Burges, and Vapnik, 1995; Blanz, Schölkopf, Bülthoff, Burges, Vapnik, and Vetter, 1996; Schölkopf, Sung, Burges, Girosi, Niyogi, Poggio, and Vapnik, 1996c). Finally, at the end of the chapter, we summarize various ways of understanding and interpreting the high generalization performance of SV machines (Sec. 2.7).*

2.1 The Support Vector Algorithm

As a basis for the material in the following section, we first need to describe the SV algorithm in some detail. The original treatments are due to Vapnik and Chervonenkis (1974), Boser, Guyon, and Vapnik (1992), Guyon, Boser, and Vapnik (1993), Cortes and Vapnik (1995), and Vapnik (1995b).

We describe the SV algorithm in four steps. In Sec. 2.1.1, a structure of decision functions is described which is sufficiently simple to admit the formulation of a bound on their VC-dimension. Based on this result, the *optimal margin* algorithm minimizes

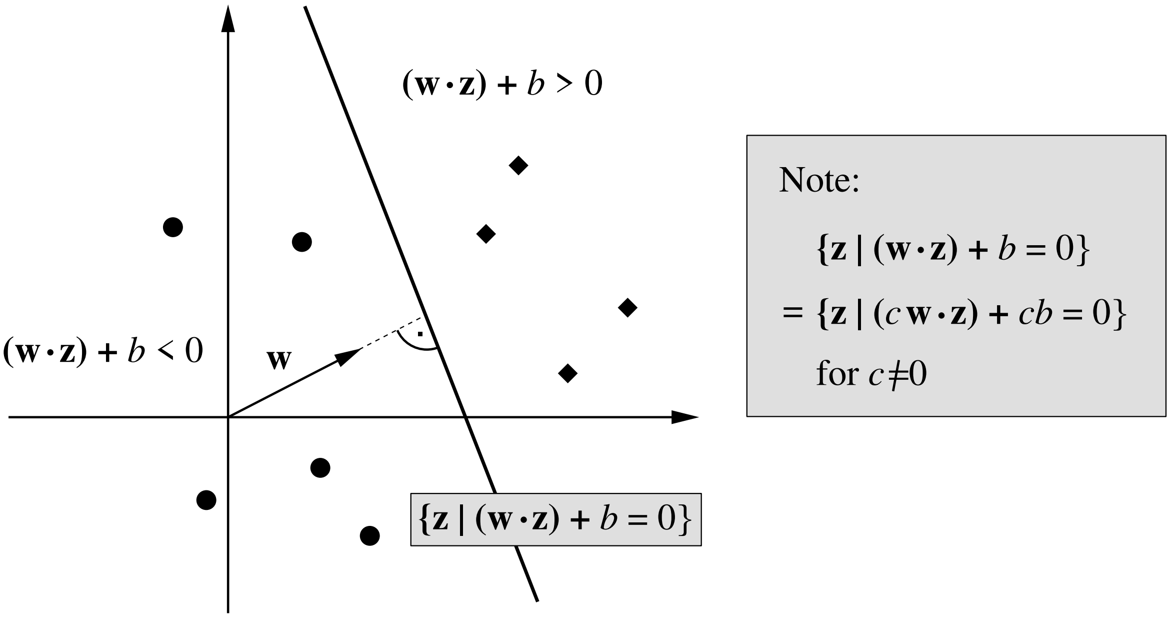

FIGURE 2.1: A separating hyperplane, written in terms of a weight vector $\mathbf{w}$ and a threshold b. Note that by multiplying both $\mathbf{w}$ and b with the same nonzero constant, we obtain the same hyperplane, represented in terms of different parameters. Fig. 2.2 shows how to eliminate this scaling freedom.

the VC-dimension for this class of decision functions (Sec. 2.1.2). This algorithm is then generalized in two steps in order to obtain SV machines: nonseparable classification problems are dealt with in Sec. 2.1.3, and nonlinear decision functions, retaining the VC-dimension bound, are described in Sec. 2.1.4.

To be able to utilize the results of Sec. 1.3, we shall formulate the algorithm in terms of dot products in some space F. Initially, we think of F as the input space. In Sec. 2.1.4, we will substitute kernels for dot products, in which case F becomes a feature space nonlinearly related to input space.

2.1.1 A Structure on the Set of Hyperplanes

Each particular choice of a structure (1.8) gives rise to a learning algorithm, consisting of performing Structural Risk Minimization in the given structure of sets of functions. The SV algorithm is based on a structure on the set of separating hyperplanes.

To describe it, first note that given a dot product space F and a set of pattern vectors $\mathbf{z}_1, \ldots, \mathbf{z}_r \in F$, any hyperplane can be written as

$$\{\mathbf{z} \in F : (\mathbf{w} \cdot \mathbf{z}) + b = 0\}. \tag{2.1}$$

In this formulation, we still have the freedom to multiply $\mathbf{w}$ and b with the same nonzero constant (Fig. 2.1). However, the hyperplane corresponds to a *canonical* pair

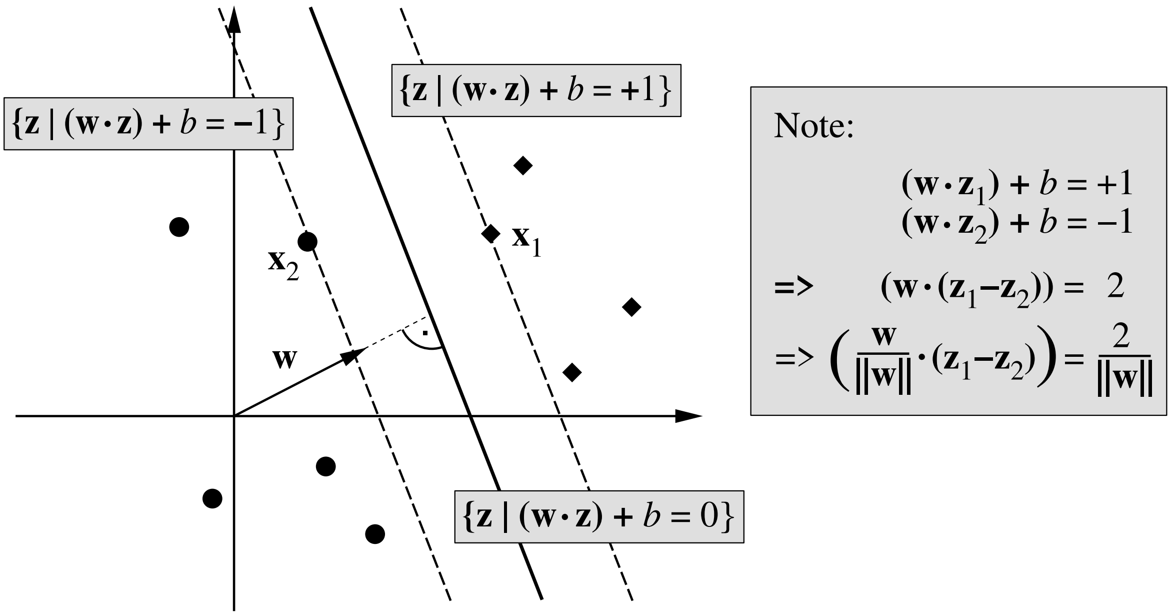

FIGURE 2.2: By requiring the scaling of $\mathbf{w}$ and b to be such that the point(s) closest to the hyperplane satisfy $|(\mathbf{w}\cdot\mathbf{z}_i)+b| = 1$, we obtain a *canonical* form $(\mathbf{w}, b)$ of a hyperplane (cf. Fig. 2.1). Note that in this case, the margin, measured perpendicularly to the hyperplane, equals $2/\|\mathbf{w}\|$, which can be seen by considering two opposite points which precisely satisfy $|(\mathbf{w}\cdot\mathbf{z}_i) + b| = 1$.

$(\mathbf{w}, b) \in F \times \mathbf{R}$ if we additionally require

$$\min_{i=1,\dots,r} |(\mathbf{w}\cdot\mathbf{z}_i) + b| = 1, \tag{2.2}$$

i.e. that the scaling of $\mathbf{w}$ and b be such that the point closest to the hyperplane has a distance of $1/\|\mathbf{w}\|$ (Fig. 2.2).[1] Thus, the margin between the two classes, measured perpendicular to the hyperplane, is at least $2/\|\mathbf{w}\|$. The possibility of introducing a structure on the set of hyperplanes is based on the following result (Vapnik, 1995b):

Proposition 2.1.1 *Let R be the radius of the smallest ball $B_R(\mathbf{a}) = \{\mathbf{z} \in F : \|\mathbf{z} - \mathbf{a}\| < R\}$ $(\mathbf{a} \in F)$ containing the points $\mathbf{z}_1, \dots, \mathbf{z}_r$, and let*

$$f_{\mathbf{w},b} = \operatorname{sgn}((\mathbf{w}\cdot\mathbf{z}) + b) \tag{2.3}$$

be canonical hyperplane decision functions defined on these points. Then the set $\{f_{\mathbf{w},b} : \|\mathbf{w}\| \le A\}$ has a VC-dimension h satisfying

$$h < R^2A^2 + 1. \tag{2.4}$$

[1] The condition (2.2) still allows two such pairs: given a canonical hyperplane $(\mathbf{w}, b)$, another one satisfying (2.2) is given by $(-\mathbf{w}, -b)$. However, we do not mind this remaining ambiguity: first, the following Proposition only makes use of $\|\mathbf{w}\|$, which coincides in both cases, and second, these two hyperplanes correspond to different decision functions $\operatorname{sgn}((\mathbf{w}\cdot\mathbf{z}) + b)$.

Note. Dropping the condition $\|\mathbf{w}\| \leq A$ leads to a set of functions whose VC-dimension equals $N_F + 1$, where N_F is the dimensionality of F. Due to $\|\mathbf{w}\| \leq A$, we can get VC-dimensions which are much smaller than N_F, enabling us to work in very high dimensional spaces — remember that the risk bound (1.5) does not explicitely depend upon N_F, but on the VC-dimension.

To make Proposition 2.1.1 intuitively plausible, note that due to the inverse proportionality of margin and $\|\mathbf{w}\|$, (2.4) essentially states that by requiring a large lower bound on the margin (i.e. a small A), we obtain a small VC-dimension. Conversely, by allowing for separations with small margin, we can potentially separate a much larger class of problems (i.e. a larger class of possible labellings of the training data, cf. the definition of the VC-dimension, following (1.6)).

Recalling that (1.5) tells us to keep both the training error and the VC-dimension small in order to achieve high generalization ability, we conclude that hyperplane decision functions should be constructed such that they maximize the margin, and at the same time separate the training data with as few exceptions as possible. Sections 2.1.2 and 2.1.3 will deal with these two issues, respectively.

2.1.2 Optimal Margin Hyperplanes

Suppose we are given a set of examples $(\mathbf{z}_1, y_1), \ldots, (\mathbf{z}_\ell, y_\ell)$, $\mathbf{z}_i \in F, y_i \in \{\pm 1\}$, and we want to find a decision function $f_{\mathbf{w},b} = \operatorname{sgn}((\mathbf{w} \cdot \mathbf{z}) + b)$ with the property

$$f_{\mathbf{w},b}(\mathbf{z}_i) = y_i, \quad i = 1, \ldots, \ell. \tag{2.5}$$

If this function exists (the nonseparable case shall be dealt with in the next section), canonicality (2.2) implies

$$y_i \cdot ((\mathbf{z}_i \cdot \mathbf{w}) + b) \geq 1, \quad i = 1, \ldots, \ell. \tag{2.6}$$

As an aside, note that out of the two canonical forms of the same hyperplane $(\mathbf{w}, b)$, $(-\mathbf{w}, -b)$, only one will satisfy equations (2.5) and (2.6). The existence of class labels thus allows to distinguish two orientations of a hyperplane.

Following Proposition 2.1.1, a separating hyperplane which generalizes well can thus be found by minimizing

$$\tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 \tag{2.7}$$

subject to (2.6). To solve this convex optimization problem, one introduces a Lagrangian

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{\ell} \alpha_i \left(y_i((\mathbf{z}_i \cdot \mathbf{w}) + b) - 1\right) \tag{2.8}$$

with multipliers $\alpha_i \geq 0$. The Lagrangian L has to be maximized with respect to α_i

and minimized with respect to $\mathbf{w}$ and b. The condition that at the saddle point, the derivatives of L with respect to the primal variables must vanish,

$$\frac{\partial}{\partial b}L(\mathbf{w},b,\boldsymbol{\alpha})=0, \quad \frac{\partial}{\partial \mathbf{w}}L(\mathbf{w},b,\boldsymbol{\alpha})=0, \tag{2.9}$$

leads to

$$\sum_{i=1}^{\ell}\alpha_i y_i = 0 \tag{2.10}$$

and

$$\mathbf{w}=\sum_{i=1}^{\ell}\alpha_i y_i \mathbf{z}_i. \tag{2.11}$$

The solution vector thus has an expansion in terms of training examples. Note that although the solution $\mathbf{w}$ is unique (due to the strict convexity of (2.7), and the convexity of (2.6)), the coefficients α_i need not be.

According to the Kuhn-Tucker theorem of optimization theory (e.g. Bertsekas, 1995), at the saddle point only those Lagrange multipliers α_i can be nonzero which correspond to constraints (2.6) which are precisely met, i.e.

$$\alpha_i\cdot[y_i((\mathbf{z}_i\cdot\mathbf{w})+b)-1]=0, \qquad i=1,\ldots,\ell. \tag{2.12}$$

The patterns $\mathbf{z}_i$ for which $\alpha_i>0$ are called *Support Vectors*.[2]

According to (2.12), they lie exactly at the margin.[3] All remaining examples of the training set are irrelevant: their constraint (2.6) is satisfied automatically, and they do not appear in the expansion (2.11).[4]

This leads directly to an upper bound on the generalization ability of optimal margin hyperplanes: suppose we use the leave-one-out method to estimate the expected test error (e.g. Vapnik, 1979). If we leave out a pattern $\mathbf{z}_{i^*}$ and construct the solution from the remaining patterns, there are several possibilities (cf. (2.6)):

---

[2]This terminology is related to corresponding terms in the theory of convex sets, relevant to convex optimization (e.g. Luenberger, 1973; Bertsekas, 1995). Given any boundary point of a convex set C, there always exists a hyperplane separating the point from the interior of the set. This is called a *supporting hyperplane*.

SVs do lie on the boundary of the convex hulls of the two classes, thus they possess supporting hyperplanes. The SV optimal hyperplane is the hyperplane which lies in the middle of the two parallel supporting hyperplanes (of the two classes) with maximum distance.

Vice versa, from the optimal hyperplane one can obtain supporting hyperplanes for all SVs of both classes by shifting it by $1/\|\mathbf{w}\|$ in both directions.

[3]Note that this implies that the solution $(\mathbf{w}, b)$, where b is computed using the fact that $y_i((\mathbf{w}\cdot\mathbf{z}_i)+b)=1$ for SVs, *is* in canonical form with respect to the training data. (This makes use of the reasonable assumption that the training set contains both positive and negative examples.)

[4]In a statistical mechanics framework, Anlauf and Biehl (1989) have put forward a similar argument for the optimal stability perceptron, also computed by contrained optimization.

1. $y_{i^*} \cdot ((\mathbf{z}_{i^*} \cdot \mathbf{w}) + b) > 1$, i.e. the pattern is classified correctly and does not lie on the margin. These are patterns that would not have become Support Vectors anyway.

2. $y_{i^*} \cdot ((\mathbf{z}_{i^*} \cdot \mathbf{w}) + b) = 1$, i.e. $\mathbf{z}_{i^*}$ exactly meets the constraint (2.6). In that case, the solution $\mathbf{w}$ does not change, even though the coefficients α_i in the dual formulation of the optimization problem might change: namely, if $\mathbf{z}_{i^*}$ might have become a Support Vector (i.e. $\alpha_{i^*} > 0$) if it had been kept in the training set. In that case, the fact that the solution is the same no matter whether $\mathbf{z}_{i^*}$ is in the training set or not means that $\mathbf{z}_{i^*}$ can be written as $\sum_{\mathrm{SVs}} \beta_i y_i \mathbf{z}_i$ with $\beta_i \geq 0$. Note that this is *not* equivalent to saying that $\mathbf{z}_{i^*}$ can be written as some linear combination of the remaining Support Vectors: since the sign of the coefficients in the linear combination is determined by the class of the respective pattern, not any linear combination will do. Strictly speaking, $\mathbf{z}_{i^*}$ must lie in the cone spanned by the $y_i\mathbf{z}_i$, where $\mathbf{z}_i$ are all Support Vectors.[5]

3. $1 > y_{i^*} \cdot ((\mathbf{z}_{i^*} \cdot \mathbf{w}) + b) > 0$, i.e. $\mathbf{z}_{i^*}$ lies within the margin, but still on the correct side of the decision boundary. In that case, the solution looks different from the one obtained if $\mathbf{z}_{i^*}$ was in the training set (for, in that case, $\mathbf{z}_{i^*}$ would satisfy (2.6) after training), but classification is nevertheless correct.

4. $0 > y_{i^*} \cdot ((\mathbf{z}_{i^*} \cdot \mathbf{w}) + b)$. In that case, $\mathbf{z}_{i^*}$ will be classified incorrectly.

Note that the cases 3 and 4 necessarily correspond to examples which would have become SVs if kept in the training set; case 2 potentially includes such cases. However, only case 4 leads to an error in the leave-one-out procedure. Consequently, we have the following result on the generalization error of optimal margin classifiers (Vapnik and Chervonenkis, 1974):[6]

Proposition 2.1.2 *The expectation of the number of Support Vectors obtained during training on a training set of size ℓ, divided by $\ell - 1$, is an* upper bound *on the expected probability of test error.*

A sharper bound can be formulated by making a further distinction in case 2, between SVs that must occur in the solution, and those that can be expressed in terms of the other SVs (Vapnik and Chervonenkis, 1974).

Substituting the conditions for the extremum, (2.10) and (2.11), into the Lagrangian (2.8), one derives the dual form of the optimization problem: maximize

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j (\mathbf{z}_i \cdot \mathbf{z}_j) \tag{2.13}$$

[5]Possible non-uniquenesses of the solution's expansion in terms of SVs are related to zero Eigenvalues of $K_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$, cf. Proposition 1.3.4. Note, however, the above caveat on the distinction between linear combinations and linear combinations with coefficients of fixed sign.

[6]It also holds for the generalized versions of optimal margin classifiers explained in the following sections.

subject to the constraints

$$\alpha_i \geq 0, \quad i = 1, \ldots, \ell, \tag{2.14}$$

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0. \tag{2.15}$$

On substitution of the expansion (2.11) into the decision function (2.3), we obtain an expression which can be evaluated in terms of dot products between the pattern to be classified and the Support Vectors,

$$f(\mathbf{z}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} \alpha_i y_i (\mathbf{z} \cdot \mathbf{z}_i) + b\right). \tag{2.16}$$

It is interesting to note that the solution has a simple physical interpretation (Burges and Schölkopf, 1997). If we assume that each Support Vector $\mathbf{z}_j$ exerts a perpendicular force of size α_j and sign y_j on a solid plane sheet lying along the hyperplane $\mathbf{w} \cdot \mathbf{z} + b = 0$, then the solution satisfies the requirements of mechanical stability. The constraint (2.15) translates into the forces on the sheet summing to zero; and (2.11) implies that the torques $\mathbf{z}_i \times \alpha_i y_i \mathbf{w}/\|\mathbf{w}\|$ also sum to zero. This mechanical analogy illustrates the physical meaning of the term *Support* Vector.

2.1.3 Soft Margin Hyperplanes

In practice, a separating hyperplane often does not exist. To allow for the possibility of examples violating (2.6), Cortes and Vapnik (1995) introduce slack variables

$$\xi_i \geq 0, \quad i = 1, \ldots, \ell, \tag{2.17}$$

and use relaxed separation constraints (cf. (2.6))

$$y_i((\mathbf{z}_i \cdot \mathbf{w}) + b) \geq 1 - \xi_i, \quad i = 1, \ldots, \ell. \tag{2.18}$$

The SV approach to minimizing the guaranteed risk bound (1.5) consists of the following: minimize

$$\tau(\mathbf{w}, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + \gamma \sum_{i=1}^{\ell} \xi_i \tag{2.19}$$

subject to the constraints (2.17) and (2.18) (cf. (2.7)). Due to (2.4), minimizing the first term is related to minimizing the VC-dimension of the considered class of learning machines, thereby minimizing the second term of the bound (1.5) (it also amounts to maximizing the separation margin, cf. the remark following (2.2), and Fig. 2.2). The term $\sum_{i=1}^{\ell} \xi_i$, on the other hand, is an upper bound on the number of misclassifications on the training set (cf. (2.18)) — this controls the empirical risk term in (1.5). For a suitable positive constant γ, this approach therefore constitutes a practical implementation of Structural Risk Minimization on the given set of functions.[7] Note, however, that $\sum_{i=1}^{\ell} \xi_i$ is significantly larger than the number of errors if many of the ξ_i attain large values, i.e. if the classes to be separated strongly overlap, for instance due to noise. In these cases, there is no guarantee that the hyperplane will generalize well.

As in the separable case (2.11), the solution can be shown to have an expansion

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{z}_i, \tag{2.20}$$

where nonzero coefficients α_i can only occur if the corresponding example $(\mathbf{z}_i, y_i)$ precisely meets the constraint (2.18). The coefficients α_i are found by solving the following quadratic programming problem: maximize

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j (\mathbf{z}_i \cdot \mathbf{z}_j) \tag{2.21}$$

subject to the constraints

$$0 \leq \alpha_i \leq \gamma, \quad i = 1, \ldots, \ell, \tag{2.22}$$

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0. \tag{2.23}$$

2.1.4 Nonlinear Support Vector Machines

Although we have already introduced the concept of Support Vectors, one crucial ingredient of SV machines in their full generality is still missing: to allow for much more general decision surfaces, one can first nonlinearly transform a set of input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$ into a high-dimensional feature space by a map $\Phi : \mathbf{x}_i \mapsto \mathbf{z}_i$ and then do a linear separation there.

Note that in all of the above, we made no assumptions on the dimensionality of F. We only required F to be equipped with a dot product. The patterns $\mathbf{z}_i$ that we talked about in the previous sections thus need not coincide with the input patterns. They can equally well be the results of mapping the original input patterns $\mathbf{x}_i$ into a high-dimensional feature space.

Maximizing the target function (2.21) and evaluating the decision function (2.16) then requires the computation of dot products $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_i))$ in a high-dimensional space. Under Mercer's conditions, given in Proposition 1.3.2, these expensive calculations can be reduced significantly by using a suitable function k such that

$$(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_i)) = k(\mathbf{x}, \mathbf{x}_i), \tag{2.24}$$

[7]It slightly deviates from the Structural Risk Minimization (SRM) Principle in that (a) it does not use the bound (1.5), but a related quantity (2.19) which can be minimized efficiently, and (b) the SRM Principle strictly speaking requires the structure of sets of functions to be fixed a priori. For more details, cf. Vapnik (1995b); Shawe-Taylor, Bartlett, Williamson, and Anthony (1996).

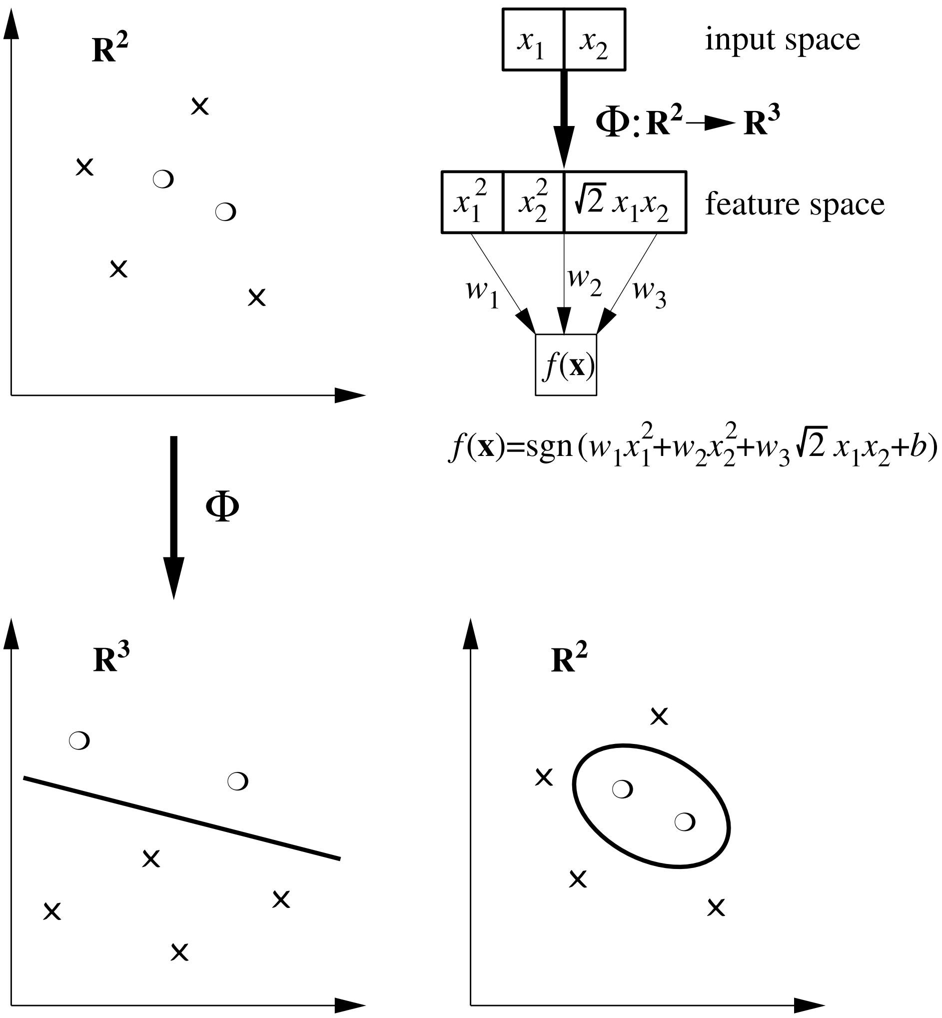

FIGURE 2.3: By mapping the input data *(top left)* nonlinearly (via Φ) into a higher-dimensional feature space F (here: $\mathbf{R}^3$), and constructing a separating hyperplane there *(bottom left)*, an SV machine *(top right)* corresponds to a nonlinear decision surface in input space (here: $\mathbf{R}^2$, *bottom right*).

leading to decision functions of the form

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} y_i \alpha_i \cdot k(\mathbf{x}, \mathbf{x}_i) + b\right). \tag{2.25}$$

Consequently, everything that has been said about the linear case also applies to nonlinear cases obtained by using a suitable kernel k instead of the Euclidean dot product (Fig. 2.3). By using different kernel functions, the SV algorithm can construct a variety of learning machines (Fig. 2.4), some of which coincide with classical architectures:

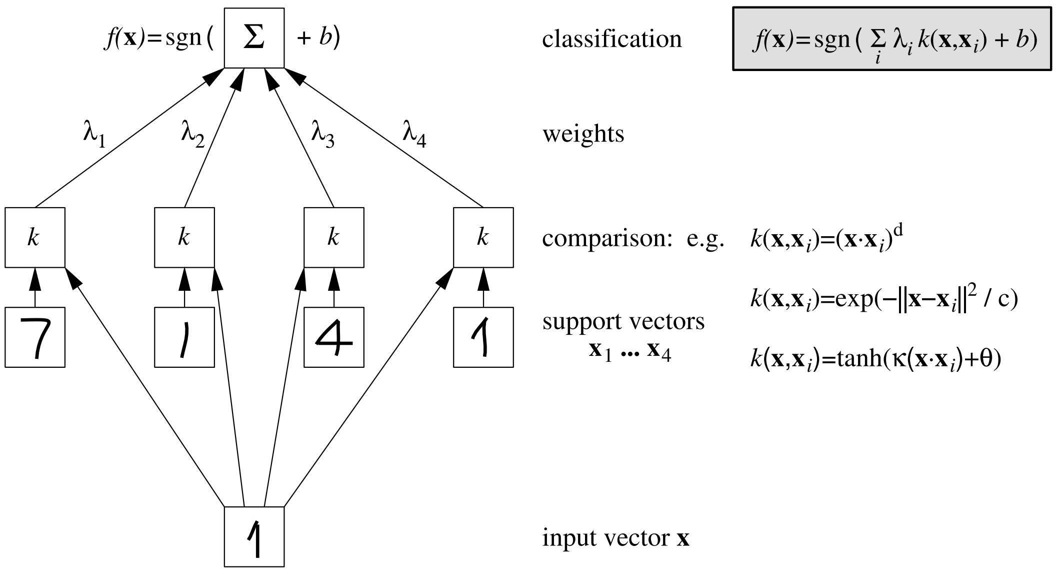

FIGURE 2.4: Architecture of SV machines. The kernel function k is chosen a priori; it determines the type of classifier (e.g. polynomial classifier, radial basis function classifier, or neural network). All other parameters (number of hidden units, weights, threshold b) are found during training by solving a quadratic programming problem. The first layer weights $\mathbf{x}_i$ are a subset of the training set (the Support Vectors); the second layer weights $\lambda_i = y_i\alpha_i$ are computed from the Lagrange multipliers (cf. (2.25)).

Polynomial classifiers of degree d:

$$k(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x} \cdot \mathbf{x}_i)^d \tag{2.26}$$

Radial basis function classifiers:

$$k(\mathbf{x}, \mathbf{x}_i) = \exp\left(-\|\mathbf{x} - \mathbf{x}_i\|^2 / c\right) \tag{2.27}$$

Neural networks:

$$k(\mathbf{x}, \mathbf{x}_i) = \tanh(\kappa \cdot (\mathbf{x} \cdot \mathbf{x}_i) + \Theta) \tag{2.28}$$

To find the decision function (2.25), we maximize (cf. (2.21))

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{2.29}$$

subject to the constraints (2.22) and (2.23). Since k is required to satisfy Mercer's conditions, it corresponds to a dot product in another space (2.24), thus $K_{ij} := (y_i y_j k(\mathbf{x}_i, \mathbf{x}_j))_{ij}$ is a positive matrix, providing us with a problem that can be solved efficiently. To see this, note that (cf. Proposition 1.3.4)

$$\sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{i=1}^{\ell} \alpha_i y_i \Phi(\mathbf{x}_i) \cdot \sum_{j=1}^{\ell} \alpha_j y_j \Phi(\mathbf{x}_j) \right) \geq 0 \tag{2.30}$$

for all $\boldsymbol{\alpha} \in \mathbf{R}^{\ell}$.

To compute the threshold b, one takes into account that due to (2.18), for Support Vectors $\mathbf{x}_j$ for which $\xi_j = 0$, we have

$$\sum_{i=1}^{\ell} y_i \alpha_i \cdot k(\mathbf{x}_j, \mathbf{x}_i) + b = y_j. \tag{2.31}$$

Thus, the threshold can for instance be obtained by averaging

$$b = y_j - \sum_{i=1}^{\ell} y_i \alpha_i \cdot k(\mathbf{x}_j, \mathbf{x}_i) \tag{2.32}$$

over all Support Vectors $\mathbf{x}_j$ (i.e. $0 < \alpha_j$) with $\alpha_j < \gamma$.

Figure 2.5 shows how a simple binary toy problem is solved by a Support Vector machine with a radial basis function kernel (2.27).

2.1.5 SV Regression Estimation

This thesis is primarily concerned with pattern recognition. Nevertheless, we briefly mention the case of SV regression (Vapnik, 1995b; Smola, 1996; Vapnik, Golowich, and Smola, 1997). To estimate a linear regression (Fig. 2.6)

$$f(\mathbf{z}) = (\mathbf{w} \cdot \mathbf{z}) + b \tag{2.33}$$

with precision ε, one minimizes

$$\tau(\mathbf{w}, \zeta, \zeta^*) = \frac{1}{2}\|\mathbf{w}\|^2 + \gamma \sum_{i=1}^{\ell} (\zeta_i + \zeta_i^*) \tag{2.34}$$

subject to

$$((\mathbf{w} \cdot \mathbf{z}_i) + b) - y_i \leq \varepsilon + \zeta_i \tag{2.35}$$

$$y_i - ((\mathbf{w} \cdot \mathbf{z}_i) + b) \leq \varepsilon + \zeta_i^* \tag{2.36}$$

$$\zeta_i, \zeta_i^* \geq 0 \tag{2.37}$$

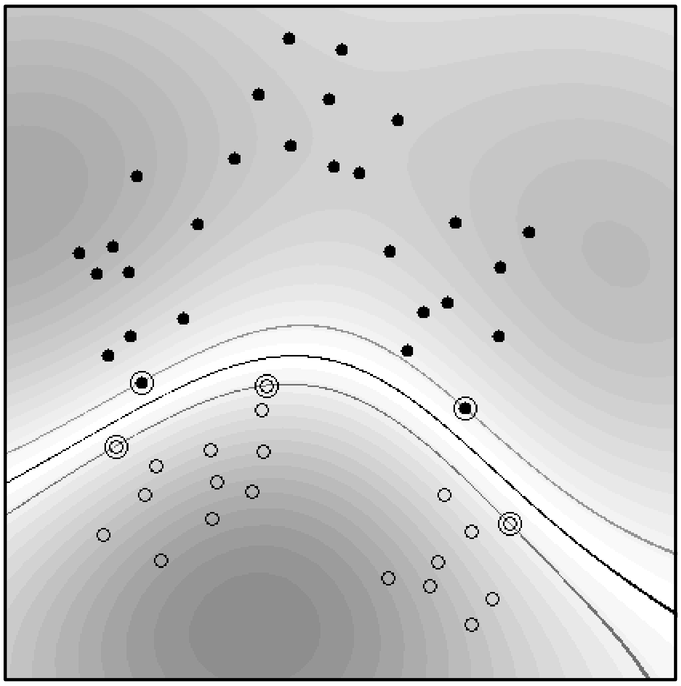

FIGURE 2.5: Example of a Support Vector classifier found by using a radial basis function kernel $k(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2)$. Both coordinate axes range from -1 to +1. Circles and disks are two classes of training examples; the middle line is the decision surface; the outer lines precisely meet the constraint (2.6). Note that the Support Vectors found by the algorithm (marked by extra circles) are not centers of clusters, but examples which are critical for the given classification task (cf. Sec. 2.5). Grey values code the modulus of the argument $\sum_{i=1}^{\ell} y_i \alpha_i \cdot k(\mathbf{x}, \mathbf{x}_i) + b$ of the decision function (2.25). (From Schölkopf, Burges, and Vapnik (1996a).)

for all $i = 1, \ldots, \ell$.

Generalization to nonlinear regression estimation is carried out using kernel functions, in complete analogy to the case of pattern recognition. A suitable choice of the kernel function then allows the construction of multi-dimensional splines (Vapnik, Golowich, and Smola, 1997).

Different types of loss functions can be utilized to cope with different types of noise in the data (Müller, Smola, Rätsch, Schölkopf, Kohlmorgen, and Vapnik, 1997; Smola and Schölkopf, 1997b).

2.1.6 Multi-Class Classification

To get k-*class classifiers*, we construct a set of binary classifiers $f^1, \ldots, f^k$, each trained to separate one class from the rest, and combine them by doing the multi-class classification according to the maximal output before applying the sgn function, i.e. by

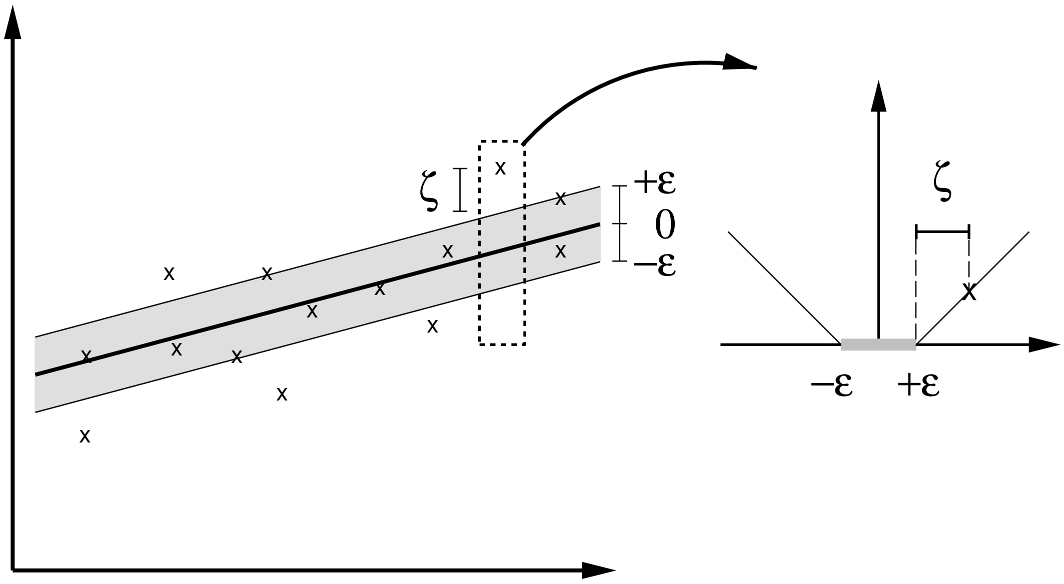

FIGURE 2.6: In SV regression, a desired accuracy ε is specified a priori. It is then attempted to fit a tube with radius ε to the data. The trade-off between model complexity and points lying outside of the tube (with positive slack variables ζ) is determined by minimizing (2.34).

taking

$$\operatorname{argmax}_{j=1,\ldots,k} g^j(\mathbf{x}), \quad \text{where } g^j(\mathbf{x}) = \sum_{i=1}^{\ell} y_i \alpha_i^j \cdot k(\mathbf{x}, \mathbf{x}_i) + b^j \tag{2.38}$$

(note that $f^j(\mathbf{x}) = \operatorname{sgn}(g^j(\mathbf{x}))$, cf. (2.25)). The values $g^j(\mathbf{x})$ can also be used for reject decisions (e.g. Bottou et al., 1994), for instance by considering the difference between the maximum and the second highest value as a measure of confidence in the classification.

In the following sections, we shall report experimental results obtained with the SV algorithm. We used the Support Vector algorithm with standard quadratic programming techniques[8] to construct polynomial, radial basis function and neural network classifiers. This was done by choosing the kernels (2.26), (2.27), (2.28) in the decision function (2.25) and in the function (2.29) to be maximized under the constraints (2.22) and (2.23). We shall start with object recognition experiments (Sec. 2.2), and then move to handwritten digit recognition (Sec. 2.3).

[8]An existing implementation at AT&T Bell Labs was used, largely programmed by L. Bottou, C. Burges, and C. Cortes.

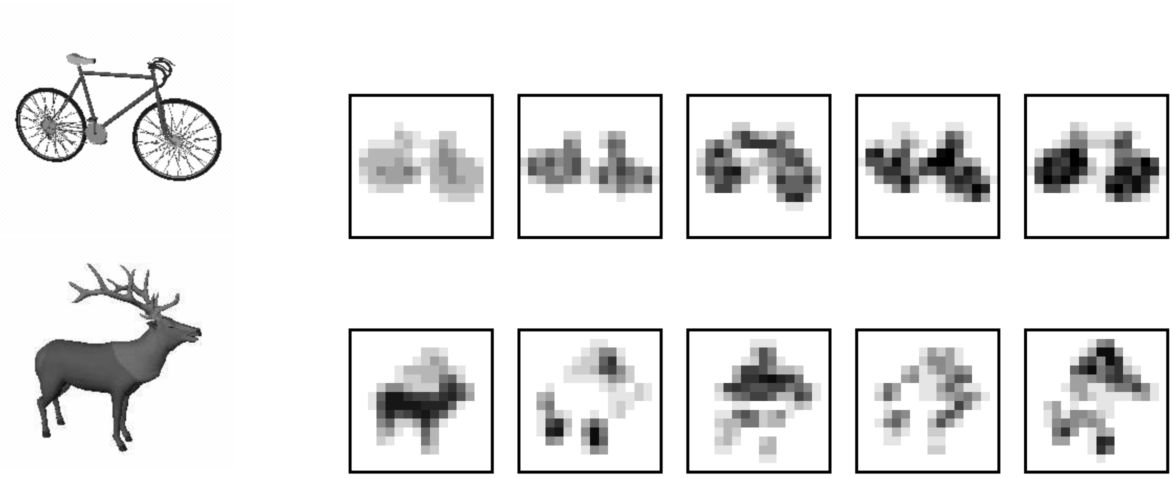

FIGURE 2.7: Examples from the entry level *(top)* and animal *(bottom)* databases. *Left*: rendered views of two 3-D models; *right*: 16×16 downsampled images, and four 16×16 downsampled edge detection patterns.

2.2 Object Recognition Results

2.2.1 Entry-Level and Animal Recognition

For purposes of psychophysical and computational studies, the object recognition group at the Max-Planck-Institut für biologische Kybernetik has compiled three databases of rendered 3D CAD models. The *entry level database* (see Appendix A for snapshots and further description) comprises views of 25 3-D object models, which in psychophysical experiments were found to belong to different entry level categories (Liter et al., 1997). Objects tend to get identified by humans first at a particular level of abstraction which is neither the most general nor the most specific, e.g. an object might be identified first as an apple, rather than as a piece of fruit or as a cox orange. For a discussion of this concept, referred to as entry (or basic) level, see (Jolicoeur, Gluck, and Kosslyn, 1984; Rosch, Mervis, Gray, Johnson, and Boyes-Braem, 1976). In *subordinate level* recognition, on the other hand, finer distinctions between objects sharing the same entry level become relevant, as for instance those between different types of birds contained in the second database, the *animal database* (Appendix A). It should be noted, however, that the animal database does not pose a purely subordinate level recognition task, since many of its animals are also distinct on the entry level. The third MPI database, containing 25 chairs, however, can be considered a subordinate level database. We will use this one in Sec. 2.2.2.

In order to recognize the objects from all orientations of the upper viewing hemisphere, a fairly complex decision surface in high-dimensional space must be learnt. The objects were realistically rendered and then downsampled. Compared to many real-world databases, the database should be considered as containing relatively little noise; in particular, they do not contain wrongly labeled patterns. Under these circumstances, we reasoned that it should be possible to separate the data with zero training error even with moderate classifier complexity, and we decided to determine the value of the constant γ (cf. (2.19)) by the following heuristic: out of all values 10^n, with integer n, we chose the smallest one which made the problem separable. On the entry level databases, this led to $\gamma = 1000$, on the animal databases, to $\gamma = 100$. Of both databases, we used 12 variants, obtained by

- choosing one of three database sizes: 25, 89, (regularly spaced) or 100 (random, uniformly distributed) views per object;
- choosing either grey-scale images or binarized silhouette images (both in downsampled versions); and
- using just 16×16 resolution images, obtained from the original images by downsampling, or additionally four more 16×16 patterns, containing downsampled versions of edge detection results obtained from the original images. Note that in the latter case, the resulting 1280-dimensional vectors contain information which is *not* contained in the 16×16 images, since the edge detection, involving a (nonlinear) modulus operation, is done *before* downsampling (cf. Blanz, Schölkopf, Bülthoff, Burges, Vapnik, and Vetter, 1996).

For more details on the databases, see (Liter et al., 1997), and Appendix A. Example images of the original models, and of the downsampled images and edge detection patterns for the entry level and the animal databases are given Fig. 2.7.

We trained polynomial SV machines on these 25-class recognition tasks, and obtained accuracies which in some cases exceeded 99% (see Table 2.1). A few aspects of the results deserve being pointed out:

Performance. The highest recognition rates were obtained using polynomial SV classifiers of degrees around 20; however, we found no pronounced minimum. Generally, *all* of the higher degrees afforded high accuracies. The regularly spaced 89-view-per-object set led to higher accuracies than the random 100-view-per-object set. This suggests that regular spacing of the views on the viewing sphere corresponds to a useful spacing of the knots (or centers) of the approximating functions in $\mathbf{R}^N$. Edge detection information significantly cuts errors rates, in many cases by a factor of two or more. Generally, accuracies were higher for grey-scale images than they were for silhouettes. The differences, however, were not large: high accuracies were also obtained for silhouettes. To understand this, we have to note that the thresholding operation used to produce silhouettes was applied to the original high-resolution images, and not to the downsampled versions. After downsampling, this yields *grey-scale* images whose grey values do not code grey values in the original image, however, they do still code useful information on the *high-resolution* object silhouettes.

TABLE 2.1: Object recognition test error rates on different databases of 25 objects, using polynomial SV classifiers of varying degrees. The training sets containing 25 and 89 views per object were regularly spaced; those with 100 views were distributed uniformly. Testing was done on an independent test set of 100 random views per object. All views were taken from the upper viewing hemisphere. For further discussion, see Sec. 2.2.1.

degree:	1	3	6	9	12	15	20	25
entry level:								
25 grey scale	26.0	17.7	15.4	13.9	13.1	13.0	13.0	14.6
89 grey scale	14.5	3.4	2.4	2.0	1.8	1.8	1.8	2.1
100 grey scale	17.1	5.6	4.2	3.5	3.2	2.8	2.4	2.8
25 silhouettes	27.1	19.6	17.9	16.7	16.2	15.6	15.4	16.3
89 silhouettes	17.2	4.3	3.3	2.7	2.5	2.2	2.2	2.8
100 silhouettes	18.2	6.9	5.4	4.8	4.2	4.0	4.0	4.7
entry level with edge detection:								
25 grey scale	9.0	8.0	6.7	5.8	5.5	5.3	4.9	5.6
89 grey scale	1.9	1.2	0.8	0.7	0.6	0.5	0.4	0.4
100 grey scale	3.5	2.3	1.8	1.5	1.3	1.1	1.1	1.0
25 silhouettes	9.4	8.2	7.6	7.0	6.6	6.5	6.1	6.0
89 silhouettes	2.4	1.7	1.2	0.8	0.6	0.5	0.5	0.4
100 silhouettes	3.8	3.0	2.6	2.5	2.5	2.4	2.3	2.2
animals:								
25 grey scale	31.6	20.4	15.9	14.8	13.8	13.4	13.0	13.8
89 grey scale	21.8	5.6	3.2	2.5	2.0	1.7	1.7	2.0
100 grey scale	24.5	8.8	5.8	5.2	5.0	4.7	4.8	4.4
25 silhouettes	34.4	22.4	18.2	17.0	16.4	15.6	15.8	16.4
89 silhouettes	27.0	7.4	3.8	2.8	2.5	2.5	2.2	2.8
100 silhouettes	29.1	11.0	7.4	6.3	5.8	5.4	5.2	5.7
animals with edge detection:								
25 grey scale	11.8	9.0	7.9	7.2	6.9	6.8	6.4	6.4
89 grey scale	3.2	1.5	1.1	1.0	0.9	0.9	0.8	0.8
100 grey scale	4.7	3.3	2.7	2.2	2.2	2.0	2.0	2.0
25 silhouettes	12.1	9.9	8.8	8.0	7.6	7.5	7.0	7.1
89 silhouettes	3.7	2.0	1.3	1.2	1.1	1.2	1.1	1.1
100 silhouettes	5.4	4.0	3.2	3.1	3.0	2.9	2.7	2.6

TABLE 2.2: Numbers of SVs for the object recognition systems of Table 2.1, on different databases of 25 objects, using polynomial SV classifiers of varying degrees. The training sets containing 25 and 89 views per object were regularly spaced; the ones with 100 views were distributed uniformly. The numbers of SVs are averages over all 25 binary classifiers separating one object from the rest; they should be seen in relation to the size of the training set, which for the above numbers of views per objects was 625, 2225, and 2500, respectively. The given numbers of SVs thus amount to roughly 10% of the database sizes. For the silhouette databases, the numbers (not shown here) are very similar, only slightly bigger.

degree:	1	3	6	9	12	15	20	25
entry level:								
25 grey scale	86	74	71	70	72	74	79	92
89 grey scale	219	148	132	128	128	133	144	165
100 grey scale	206	139	121	117	119	122	135	158
entry level with edge detection:								
25 grey scale	73	74	77	79	84	87	91	99
89 grey scale	126	119	125	130	137	145	151	161
100 grey scale	123	115	120	125	129	133	143	153
animals:								
25 grey scale	108	96	89	90	91	95	100	112
89 grey scale	231	196	180	177	178	183	193	208
100 grey scale	235	196	176	169	169	174	185	199
animals with edge detection:								
25 grey scale	101	92	93	99	103	107	117	128
89 grey scale	183	170	172	180	188	198	212	227
100 grey scale	187	171	172	177	182	191	201	215

Support Vectors. The numbers of SVs (Table 2.2) of the individual recognizers for each object make up about $5\% - 15\%$ of the whole databases. The fraction decreases with increasing database size.

For polynomial machines of degree 1 (i.e. separating hyperplanes in input space), the problem is not separable. In that case, all training errors show up as SVs (cf. (2.18)), causing a fairly large number of SVs. For degrees higher than 1, the number of SVs slightly increases with increasing polynomial degree. However, the increase is rather moderate, compared with the increase of the dimensionality of the feature space that we are implicitely working on (cf. Sec. 1.3). Interestingly, the number of SVs does

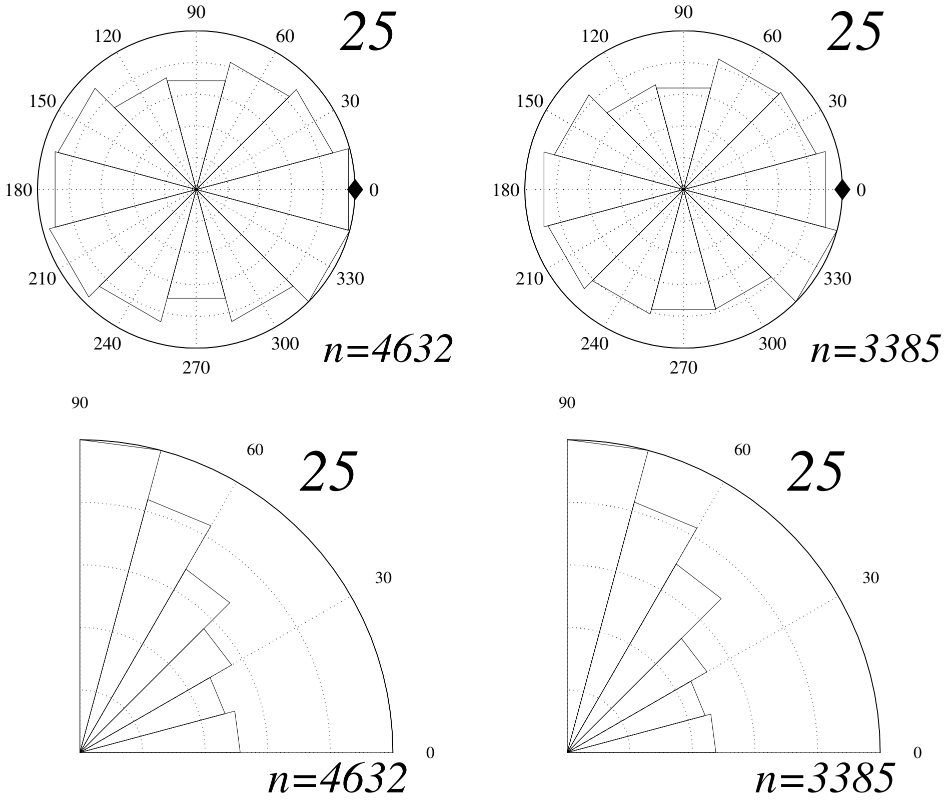

FIGURE 2.8: Angular distribution of the viewing angles of those training views which became SVs, for a polynomial SV machine of degree 20 on the animal *(left)* and entry level *(right)* databases (100 grey level views per object, without edge detection). The plotted distributions for azimuth *(top)* and elevation *(bottom)* have been normalized by the corresponding distributions in the training set (see Fig. A.1). It can be seen that SVs tend to occur more often for top, front and back views. In this and the following plots, views which become SVs for more than one of the 25 binary recognizers are counted according to their frequency of occurence. Consequently, there is no contradiction in the overall number of SVs n exceeding the database size (2500).

not change much if we add edge detection information, even though this increases the input dimensionality by a factor of 5.

As each of the training examples is associated with two viewing angles (θ, ϕ) (cf. Appendix A), we can look at the angular distribution of SVs and errors. It is shown in figures 2.8 – 2.10, and, in more detail, in figures B.2 – B.9 in the appendix (there, we also give an example of a full SV set of one of the binary recognizers, in Fig. B.1). The density of SVs is increased at high polar angles, i.e. for viewing the objects from the top. Also, SVs tend to be found more often for frontal and back views than for views closer to the side. Top, frontal and back views typically are harder to classify than views from more generic points of view (Blanz, 1995). We can thus interpret our finding as an indication that the density of SVs is related to the local complexity of the classification surface, i.e. the local difficulty of the classification task. Indeed, the same qualitative behaviour is found for the distribution of recognition errors (figures 2.9 and 2.10).

There are several factors contributing to the difficulty in classifying top, frontal and back views. First note that since most objects in our databases are bilaterally symmetric, top, frontal and back views contain a large amount of redundancy. In contrast, side views of symmetric objects contain a maximal amount of information. Moreover, many objects in the databases have their main axis of elongation roughly aligned with the direction of the zero view ($\theta = 0, \phi = 0$). Consequently, frontal and back views suffer the drawback of showing a projection of a comparably small area.

As an aside, note that although the SVs live in a high-dimensional space, the particular setup of the presented object recognition experiments made it possible to discuss the relationship between the difficulty of the task, the distribution of SVs, and the distribution of errors. This is due to the low-dimensional parametrization of the examples, arising from the procedure of generating the examples by taking snapshots at well-defined viewing positions.

The hope that Support Vectors are a useful means of analysing recognition tasks will receive further support in Sec. 2.4, where we shall present results which show that different types of SV machines, obtained using different kernel functions, lead to largely the same Support Vectors if trained on the same task.

Comparison with Neural Networks. To evaluate the performance of SV classifiers on this task, benchmark comparisons with other classifiers need to be carried out. We conducted a set of experiments using perceptrons with one hidden layer, endowed with 400 hidden neurons, and hyperbolic tangent activation functions in hidden and output neurons. The networks were trained by back-propagation of mean squared error (Rumelhart, Hinton, and Williams, 1986; LeCun, 1985). We used on-line (stochastic) gradient descent, i.e. the weights were updated after each pattern; training was stopped when the training error dropped below 0.1%, or after 600 learning epochs, whichever occured earlier. Neither this procedure nor the network design was carefully optimized for the task at hand, thus the results reported in the following should be seen as baseline comparisons solely intended to facilitate assessing the reported SV results.[9]

[9]By observing the dependency of the test error on the number of learning epochs, we were able to see that the networks were not overtrained. In addition, experiments with smaller numbers of hidden units gave worse performance (larger networks were not used, for reasons of excessive training times), hence the network capacities did not seem too large.

A full-fledged comparison between SV machines and perceptrons would take into account the following issues in order to obtain optimized network designs: instead of one fully connected hidden layer, more sophisticated *architectures* use several layers with shared weights, extracting features of increasing complexity and invariance, while still limiting the number of free parameters. Other *regularization* techniques useful for improving generalization include weight decay and pruning. Similarly, *early stopping* can be used to deal with issues of overtraining. The training procedure can be optimized by using different *error functions* and *output functions* (e.g. softmax). Finally, for small

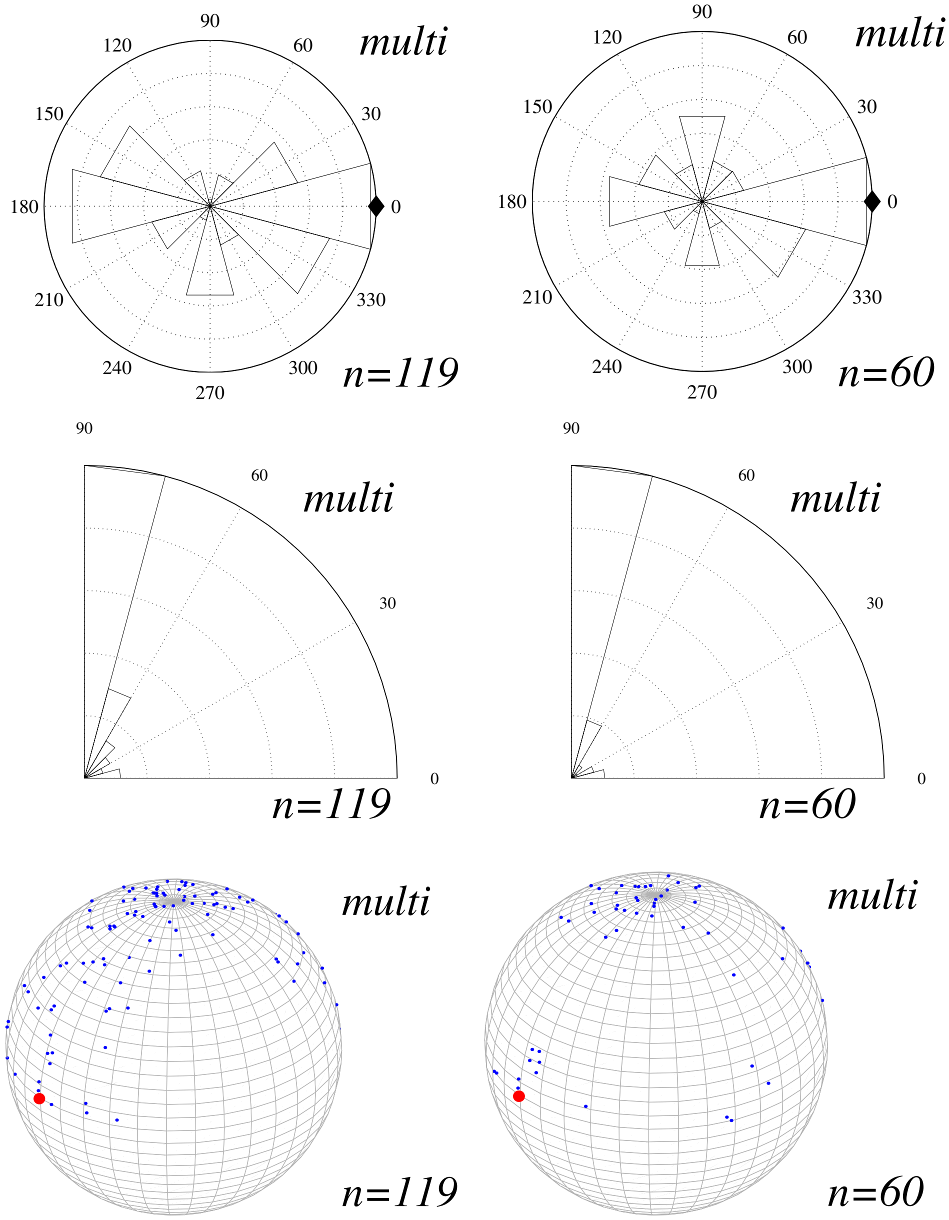

FIGURE 2.9: Angular distribution of the errors made in multi-class classification (25 classes), normalized by the corresponding distribution in the test set, details as in Fig. 2.8 (*left*: animal database, *right*: entry level database). In addition to the separate azimuth and elevation data, the joint distributions *(bottom)* are shown. The zero view $(\theta, \phi) = (0, 0)$ is marked by a bigger dot; θ increases to the top, and ϕ to the right.

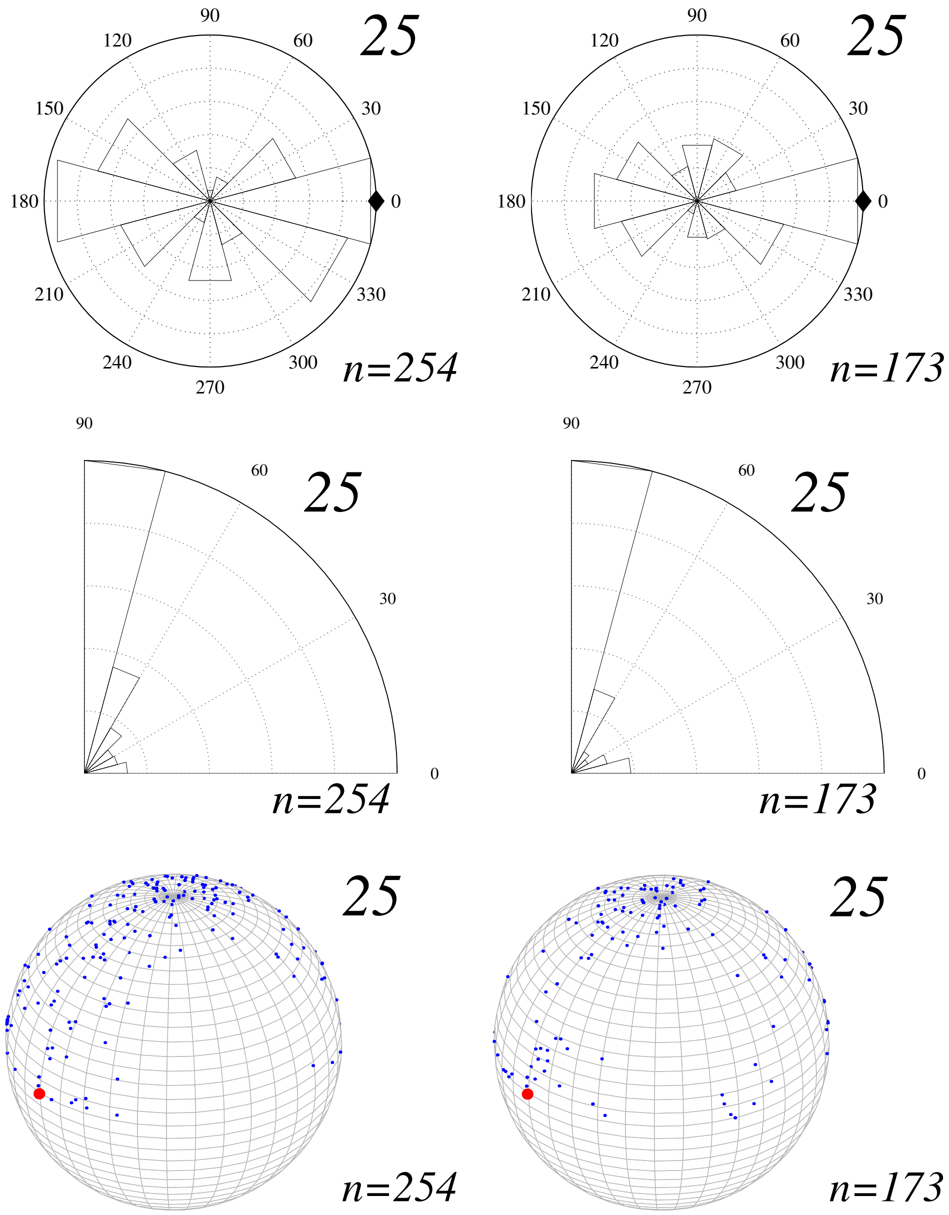

FIGURE 2.10: Angular distribution of the overall errors made by any of the 25 binary recognizers, details as in Fig. 2.8 and Fig. 2.9. The errors visible in the spherical plots *(bottom)* form supersets of the corresponding plots in Fig. 2.9, thus the number of errors n is larger than the actual number of errors in 25-class classification (Fig. 2.9). This is due to the fact that e.g. if for a particular example none of the 25 classifiers answers positively, the arbitration procedure (2.38) will choose the one with the least negative output, potentially leading to correct classification.

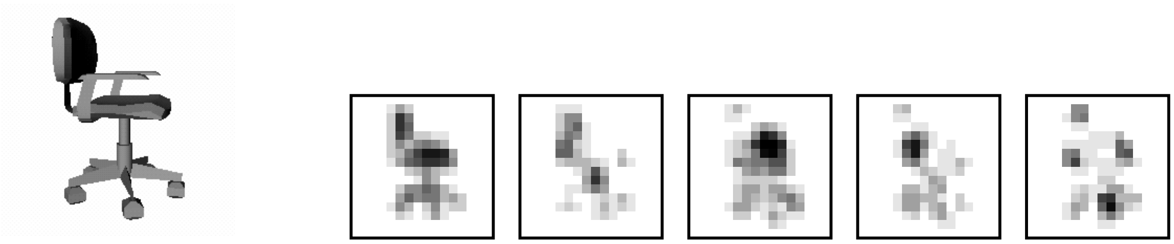

FIGURE 2.11: *Left*: rendered view of a 3-D model from the chair database; *right*: 16×16 downsampled image, and four 16×16 downsampled edge detection patterns.

For the following two reasons, we chose the small training set, with 25 views per object. First, the error rates reported above for the large sets were already very low, and differences in performance are thus more likely to be significant for the smaller training sets. Second, training times of the neural networks were very long (in the cases reported in the following, they were longer than for SV machines by more than an order of magnitude).

On the 25 view-per-object training sets, we obtained error rates of 17.3% and 21.4% on the entry level and animal databases, respectively. Adding edge detection information, the error rates dropped to 6.8%, and 11.2%, respectively. Comparing with the results in (2.1), we note that SV machines in almost all cases performed better. Further performance comparisons between SV machines and other classifiers are reported in the following section.

2.2.2 Chair Recognition Benchmark

In a set of experiments using the MPI *chair database* (Fig. 2.11, Appendix A), different view–based recognition algorithms were compared (Blanz et al., 1996). The SV analysis for this case is less detailed than the one given in Sec. 2.2.1, however, we decided also to report these experiments, since they include further benchmark results obtained with other classifiers. The first one used oriented filters to extract features which are robust with respect to small rigid transformations of the underlying 3-D objects, followed by a decision stage based on comparisons with stored templates (for details, see Blanz, 1995; Vetter, 1994; Blanz et al., 1996). The second one, run as a baseline benchmark, was a perceptron with one hidden layer, trained by error backpropagation to minimize the mean squared error (for further details, see Sec. 2.2.1). The third system was a polynomial Support Vector machine (cf. (2.26)) with degree $d = 15$ and $\gamma = 100$.[10] In addition, we report results of Kressel (1996), who utilized a fully quadratic polynomial classifier (Schürmann, 1996) trained on the first 50

databases with little redundancy it is sometimes advantageous to use batch updates with conjugate gradient descent, or using higher order derivatives of the error function. For details, see LeCun, Boser, Denker, Henderson, Howard, Hubbard, and Jackel (1989); Bishop (1995); Amari, Murata, Müller, Finke, and Yang (1997).

[10] The latter was chosen as in Sec. 2.2.1. Note that these values differ from those used in (Blanz et al., 1996), which in some cases leads to different results.

TABLE 2.3: Recognition test error (in %) for the MPI chair database (Appendix A) on 25×100 random test views from the upper viewing hemisphere, for different training sets (viewing angles either regularly spaced, or uniformly distributed, on the upper viewing hemisphere; views were either just 16×16 images, or images plus edge detection data), and different classifiers: SV: Support Vector machine; MLP: fully connected perceptron with one hidden layer of 400 neurons; OF: oriented filter invariant feature extraction, see text; PC: quadratic polynomial classifier trained on the first 50 principal components (Kressel, 1996). Where marked with '–', results are not available.

training set			classifier			
input	distribution	views per obj.	SV	MLP	OF	PC
images+e.d.	regul. spaced	25	5.0	8.8	5.4	–
images+e.d.	regul. spaced	89	1.0	1.3	4.7	1.7
images+e.d.	random	100	1.4	2.6	–	–
images+e.d.	random	400	0.3	–	–	0.8
images	regul. spaced	25	13.2	25.4	26.0	–
images	regul. spaced	89	2.0	7.2	21.0	–
images	random	100	4.5	7.5	–	–
images	random	400	0.6	–	–	–

principal components of the images.

In all experiments, the Support Vector machine exhibits the highest generalization ability (Table 2.3). Considering that the images of a single object can change drastically with viewpoint (cf. Appendix A), it seems that the Support Vector machine is best in constructing a decision surface sufficiently complex to separate the 25 classes of chairs. This, in turn, can be related to the fact that SV machines use kernel functions to construct hyperplanes in very high-dimensional feature spaces without overfitting.

Note, moreover, that this was achieved with an SV machine which does not utilize prior information about the problem at hand. The oriented filter approach, in constrast, does use prior information about the process by which the images arose from underlying 3-D objects. This knowledge was used to handcraft the robust features used for recognition. The SV machine has to extract all information from the given training data, making it understandable that its advantage over the oriented filter system gets smaller for smaller training set sizes (Table 2.3). In Chapter 4, we try to deal with this shortcoming by proposing methods to incorporate prior knowledge into SV machines.

2.2.3 Discussion

Realistically rendered computer graphics images of objects provide a useful basis for evaluating object recognition algorithms. This setup enabled us to study shape recognition under controlled conditions. Real-world recognition systems, however, face additional problems. For instance, segmentation of objects in cluttered scenes is a problem not addressed in the above experiments. Partly, these additional problems can be outweighed by additional sources of information. Objects with different albedo and color would facilitate segmentation and recognition significantly.

The impact of noise, characteristic of many real-life problems, should not be too big, at least in the case where we trained our systems on the image data only: in that case, all the processing is done in the low spatial frequency domain.

On all three databases, high recognition accuracies were reported. The highest accuracies were obtained using the regularly spaced 89 view per object training sets, and the edge detection data.

As the number of classes was 25 in all cases, we can compare the performance of the SV systems across tasks. It correlates with the intuitive difficulty of the tasks: accuracies are highest for the entry level database, were the objects have the largest differences, followed by the animal database, and by the subordinate level chair database.

2.3 Digit Recognition Using Different Kernels

Handwritten digit recognition has long served as a test bed for evaluating and benchmarking classifiers (e.g. LeCun et al., 1989; Bottou et al., 1994; LeCun et al., 1995). Thus, it is imperative to evaluate the SV method on some widely used digit recognition task. In the present chapter, we use the US Postal Service (USPS) database for this purpose (Appendix C). We put particular emphasis on comparing different types of SV classifiers obtained by choosing different kernels. We report results for polynomial kernels (2.26), radial basis function kernels (2.27), and sigmoid kernels (2.28); all of them were obtained with $\gamma = 10$ (our default choice, used wherever not stated otherwise — cf. (2.19)).

Results for the three different kernels are summarized in Table 2.4. In all three cases, error rates around 4% can be achieved. They should be compared with values achieved on the same database with a five-layer neural net (*LeNet1*, LeCun, Boser, Denker, Henderson, Howard, Hubbard, and Jackel, 1989), 5.0%, a neural net with one hidden layer, 5.9%, and the human performance, 2.5% (Bromley and Säckinger, 1991). Results of classical RBF machines, along with further reference results, are quoted in Sec. 2.5.3.

The results show that the Support Vector algorithm allows the construction of various learning machines, all of which are performing well. The similar performance for the three different functions k suggests that among these cases, the choice of the set of decision functions is less important than capacity control in the chosen type of structure. This phenomenon is well-known for the Parzen density estimator in $\mathbf{R}^N$,

$$p(\mathbf{x}) = \frac{1}{\ell} \sum_{i=1}^{\ell} \frac{1}{\omega^N} k\left(\frac{\mathbf{x} - \mathbf{x}_i}{\omega}\right). \tag{2.39}$$

There, it is of great importance to choose an appropriate value of the bandwidth

TABLE 2.4: Performance on the USPS set, for three different types of classifiers, constructed with the Support Vector algorithm by choosing different functions k in (2.25) and (2.29). Given are raw errors (i.e. no rejections allowed) on the test set. The normalization factor $c = 1.04$ in the sigmoid case is chosen such that $c \cdot \tanh(2) = 1$. For each of the ten-class-classifiers, we also show the average number of Support Vectors of the ten two-class-classifiers. The normalization factors of 256 are tailored to the dimensionality of the data, which is 16×16.

polynomial: $k(\mathbf{x}, \mathbf{y}) = ((\mathbf{x} \cdot \mathbf{y})/256)^d$

d	1	2	3	4	5	6	7
raw error/%	8.9	4.7	4.0	4.2	4.5	4.5	4.7
av. # of SVs	282	237	274	321	374	422	491

RBF: $k(\mathbf{x}, \mathbf{y}) = \exp\left(-\|\mathbf{x} - \mathbf{y}\|^2/(256\ c)\right)$

c	4.0	2.0	1.2	0.8	0.5	0.2	0.1
raw error/%	5.3	5.0	4.9	4.3	4.4	4.4	4.5
av. # of SVs	266	240	233	235	251	366	722

sigmoid: $k(\mathbf{x}, \mathbf{y}) = 1.04 \tanh(2(\mathbf{x} \cdot \mathbf{y})/256 + \Theta)$

$-\Theta$	0.8	0.9	1.0	1.1	1.2	1.3	1.4
raw error/%	6.3	4.8	4.1	4.3	4.3	4.4	4.8
av. # of SVs	206	242	254	267	278	289	296

parameter ω for a given amount of data (e.g. Härdle, 1990; Bishop, 1995). Similar parallels can be drawn to the solution of ill-posed problems (for a complete discussion, see Vapnik, 1995b).

2.4 Universality of the Support Vector Set[11]

In the present section, we report empirical evidence that the SV set provides a novel possibility for extracting a small subset of a database which contains all the information necessary to solve a given classification task: using the Support Vector algorithm to train three different types of handwritten digit classifiers, we observed that these types of classifiers construct their decision surface from strongly overlapping yet small subsets of the database.

Overlap of SV Sets. To study the Support Vector sets for three different types of SV classifiers, we used the optimal parameters on the USPS set according to Table 2.4.

TABLE 2.5: First row: total number of different Support Vectors of three different ten-class-classifiers (i.e. number of elements of the union of the ten two-class-classifier Support Vector sets) obtained by choosing different functions k in (2.25) and (2.29); second row: average number of Support Vectors per two-class-classifier (USPS database size: 7291).

	Polynomial	RBF	Sigmoid
total # of SVs	1677	1498	1611
average # of SVs	274	235	254

TABLE 2.6: Percentage of the Support Vector set of [column] contained in the support set of [row]; for ten-class classifiers *(top)* and binary recognizers for digit class 7 *(bottom)* (USPS set).

	Polynomial	RBF	Sigmoid
Polynomial	100	93	94
RBF	83	100	87
Sigmoid	90	93	100

	Polynomial	RBF	Sigmoid
Polynomial	100	84	93
RBF	89	100	92
Sigmoid	93	86	100

TABLE 2.7: Comparison of all three Support Vector sets at a time (USPS set). For each of the (ten-class) classifiers, "% intersection" gives the fraction of its Support Vector set shared with both the other two classifiers. Out of a total of 1834 different Support Vectors, 1355 are shared by all three classifiers; an additional 242 is common to two of the classifiers.

	Poly	RBF	tanh	intersection	shared by 2	union
no. of SVs	1677	1498	1611	1355	242	1834
% intersection	81	90	84	100	–	–

Table 2.5 shows that all three classifiers use around 250 Support Vectors per two-class-classifier (less than 4% of the training set). The total number of different Support Vectors of the ten-class-classifiers is around 1600. The reason why it is less than 2500 (ten times the above 250) is the following: a particular vector that has been used as a positive SV (i.e. $y_i = +1$ in (2.25)) for digit 7 might at the same time be a negative SV ($y_i = -1$) for digit 1, say.

Tables 2.6 and 2.7 show that the Support Vector sets of the different classifiers have

TABLE 2.8: SV set overlap experiments on the MNIST set (Fig. C.2), using the binary recognizer for digit 0. *Top three tables:* performances (on the 60000 element test set) and numbers of SVs for three different kernels and various parameter choices. The numbers of SVs, which should be compared to the database size, 60000, were used to select the parameters for the SV set comparison: to get a balanced comparison of the different SV sets, we decided to select parameter values such that the respective SV sets have approximately equal size (polynomial degree $d = 4$, radial basis function width $c = 0.6$, and sigmoid threshold $\Theta = -1.5$). *Bottom:* SV set comparison. For each of the binary classifiers, "% intersection" gives the fraction of its Support Vector set shared with both the other two classifiers. The scaling factor 784 in the kernels stems from the dimensionality of the data; it ensures that the values of the kernels lie in similar ranges for different polynomial degrees.

polynomial: $k(\mathbf{x}, \mathbf{y}) = ((\mathbf{x} \cdot \mathbf{y})/784)^d$

d	2	3	4	5	6	7
# of test errors	163	147	135	131	127	127
# of SVs	994	1083	1187	1292	1401	1537

RBF: $k(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2/(784\ c))$

c	1	0.75	0.6	0.5	0.4	0.3
# of test errors	147	145	145	141	137	134
# of SVs	1061	1118	1179	1264	1308	1460

sigmoid: $k(\mathbf{x}, \mathbf{y}) = 1.04 \tanh(2(\mathbf{x} \cdot \mathbf{y})/784 + \Theta)$

$-\Theta$	1.3	1.4	1.5	1.6	1.7	1.8
# of test errors	139	138	138	141	145	144
# of SVs	1137	1162	1194	1211	1223	1217

	Polyn	RBF	tanh	intersection	shared by 2	union
no. of SVs	1187	1179	1194	1054	124	1328
% intersection	89	89	88	100	–	–

about 90% overlap. This surprising result, first published in (Schölkopf, Burges, and Vapnik, 1995), has meanwhile been reproduced on the MNIST character recognition set (Table 2.8), with SV sets which amounted to just 2% of the whole database. Together with K. Sung at MIT, we have reproduced this result also on a face detection task (binary classification, faces vs. non-faces).

As mentioned previously, the Support Vector expansion (2.11) need not be unique. Depending on the way the quadratic programming problem is solved, one can potentially get different expansions and therefore different Support Vector sets. It is possible to conceive of problems where all patterns do lie on the decision boundary, yet only

TABLE 2.9: Percentage of the Support Vector set of [column] contained in the support set of [row]; for the binary recognizers for digit class 7 *(bottom)* (USPS set). The training sets for the classifiers in [row] and [column] were permuted with respect to each other (control experiment for Table 2.6); still, the overlap between the SV sets persists.

	Polynomial	RBF	Sigmoid
Polynomial	92	82	90
RBF	88	92	84
Sigmoid	91	84	93

a few of them are necessary at a time for *expressing* the decision function. In such a case, the actual SV set extracted could strongly depend on the ordering of the training set, especially if the quadratic programming algorithm processes the data in chunks. In our experiments, we did use the same ordering of the training set in all three cases. To exclude the possibility that it is this ordering that causes the reported overlaps, we ran a control experiment where two classifiers with the same kernel were trained twice, on the original training set, and on a permuted version of it, respectively. We found that the two cases produced highly overlapping (to around 90%) SV sets, which means that the training set ordering does hardly have an effect on the SV sets extracted — it only changes around 10% of the SV sets. In addition, repeating the experiments of Table 2.6 on permuted training sets gave results consistent with this finding: Table 2.9 shows that the overlap between SV sets of different classifiers is hardly changed when one of the training sets is permuted. We may also add that the overlap is not due to SVs corresponding to errors on the training set (cf. (2.18), with $\xi_i > 1$): the considered classifiers had very few training errors.

Using a leave-one-out procedure similar to Proposition 2.1.2, Vapnik and Watkins have subsequently put forward a theoretical argument for shared SVs. We state it in the following form: If the SV set of three SV classifiers had no overlap, we could obtain a fourth classifier which has zero test error.

To see why this is the case, note that if a pattern is left out of the training set, it will always be classified correctly by voting between the three SV classifiers trained on the remaining examples: otherwise, it would have been been an SV of at least two of them, if kept in the training set. The expectation of the number of patterns which are SVs of at least two of the three classifiers, divided by the training set size, thus forms an upper bound on the expected test error of the voting system.

Training on SV Sets. As described in Sec. 2.1.2, the Support Vector set contains all the information a given classifier needs for constructing the decision function. Due to the overlap in the Support Vector sets of different classifiers, one can even train classifiers on Support Vector sets of *another* classifier. Table 2.10 shows that this leads to results comparable to those after training on the whole database. In Sec. 4.2.1, we

TABLE 2.10: Training classifiers on the Support Vector sets of other classifiers leads to performances on the test set which are as good as the results for training on the full database (shown are numbers of errors on the 2007-element test set, for two-class classifiers separating digit 7 from the rest). Additionally, the results for training on a random subset of the database of size 200 are displayed.

kernel	trained on: *size:*	poly-SVs *178*	rbf-SVs *189*	tanh-SVs *177*	full db *7291*	rnd. subs. *200*
Poly		13	13	12	13	23
RBF		17	13	17	15	27
tanh		15	13	13	15	25

will use this finding as a motivation for a method to make SV machines transformation invariant.

Discussion. Learning can be viewed as inferring regularities from a set of training examples. Much research has been devoted to the study of various learning algorithms which allow the extraction of these underlying regularities. No matter how different the outward appearance of these algorithms is, they all must rely on intrinsic regularities of the data. If the learning has been successful, these intrinsic regularities are captured in the values of some parameters of a learning machine; for a polynomial classifier, these parameters are the coefficients of a polynomial, for a neural net they are weights and biases, and for a radial basis function classifier they are weights and centers. This variety of different representations of the intrinsic regularities, however, conceals the fact that they all stem from a common root.

The Support Vector algorithm enables us to view these algorithms in a unified theoretical framework. The presented *empirical* results show that different types of SV classifiers construct their decision functions from highly overlapping subsets of the training set, and thus extract a very similar structure from the observations, which can in this sense be viewed as a characteristic of the data: the set of Support Vectors. This finding may lead to methods for compressing databases significantly by disposing of the data which is not important for the solution of a given task (cf. also Guyon, Matić, and Vapnik, 1996).

In the next section, we will take a closer look at one of the types of learning machines implementable by the SV algorithm.

2.5 Comparison to Classical RBF Networks

By using Gaussian kernels (2.27), the SV algorithm can construct learning machines with a Radial Basis Function (RBF) architecture. In contrast to classical approaches for training RBF networks, the SV algorithm automatically determines centers, weights and threshold that minimize an upper bound on the expected test error. The present section is devoted to an experimental comparison of these machines with a classical approach, where the centers are determined by k-means clustering and the weights are computed using error backpropagation. We consider three machines, namely a classical RBF machine, an SV machine with Gaussian kernel, and a hybrid system with the centers determined by the SV method and the weights trained by error backpropagation. Our results show that on the US postal service database of handwritten digits, the SV machine achieves the highest recognition accuracy, followed by the hybrid system.

2.5.1 Different Ways of Training RBF Classifiers

Consider Fig. 2.12. Suppose we want to construct a radial basis function classifier

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell} w_i \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{c_i}\right) + b\right) \tag{2.40}$$

(b and c_i being constants, the latter positive) separating balls from circles, i.e. taking different values on balls and circles. How do we choose the *centers* $\mathbf{x}_i$? Two extreme cases are conceivable:

The first approach consists of choosing the centers for the two classes separately, irrespective of the classification task to be solved. The classical technique of finding the centers by some clustering technique (*before* tackling the classification problem) is such an approach. The weights w_i are then usually found by either error backpropagation (Rumelhart, Hinton, and Williams, 1986) or the pseudo-inverse method (Poggio and Girosi, 1990).

An alternative approach (Fig. 2.13) consists of choosing as centers points which are *critical* for the classification task at hand. The Support Vector algorithm implements the latter idea. By simply choosing a suitable kernel function (2.27), it allows the construction of radial basis function classifiers. The algorithm automatically computes the number and location of the above centers, the weights w_i, and the threshold b. By the kernel function, the patterns are mapped nonlinearly into a high-dimensional space. There, an optimal separating hyperplane is constructed, expressed in terms of those examples which are closest to the decision boundary. These are the Support Vectors which correspond to the centers in input space.

The goal of the present section is to compare real-world results obtained with k-means clustering and classical RBF training to those obtained when the centers, weights and threshold are automatically chosen by the Support Vector algorithm. To this end, we decided to undertake a performance study by combining expertise on the

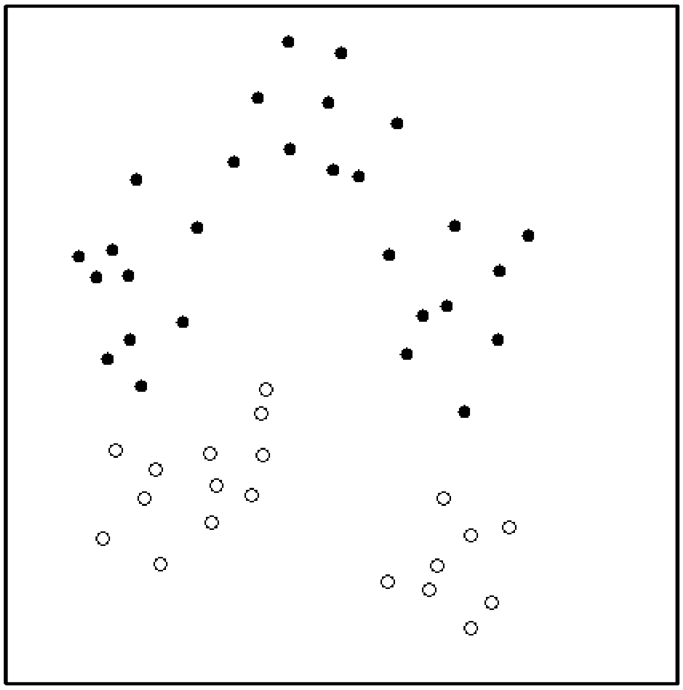

FIGURE 2.12: A simple 2-dimensional classification problem: find a decision function separating balls from circles. The box, as in all following pictures, depicts the region $[-1,1]^2$.

Support Vector algorithm (AT&T Bell Laboratories) and on the classical radial basis function networks (Massachusetts Institute of Technology).

Three different RBF systems took part in the performance comparison:

- **SV system.** A standard SV machine with Gaussian kernel function was constructed (cf. (2.27)).
- **Classical RBF system.** The MIT side of the performance comparison constructed networks of the form

$$\begin{aligned} g(\mathbf{x}) &= \operatorname{sgn}\left(\sum_{i=1}^{K} w_i \mathcal{G}_i(\mathbf{x}) + b\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{K} w_i \frac{1}{(2\pi)^{N/2}\sigma_i^N} \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{2\sigma_i^2}\right) + b\right), \end{aligned}$$

  with the number of centers k identical to the one automatically found by the SV algorithm. The centers $\mathbf{c}_i$ were computed by k-means clustering (e.g. Duda and Hart, 1973), and the weights w_i are trained by on-line mean squared error back propagation.

  The training procedure constructs ten binary recognizers for the digit classes, with RBF hidden units and logistic outputs, trained to produce the target values

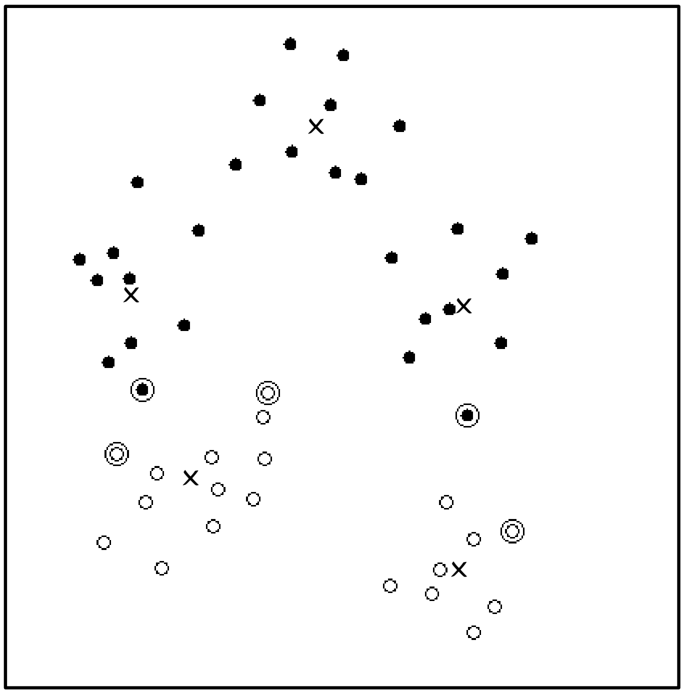

FIGURE 2.13: RBF centers automatically computed by the Support Vector algorithm (indicated by extra circles), using $c_i = 1$ for all i (cf. (2.27), (2.40)). The number of SV centers accidentally coincides with the number of identifiable clusters (indicated by crosses found by k-means clustering with $k = 2$ and $k = 3$ for balls and circles, respectively) but the naive correspondence between clusters and centers is lost; indeed, 3 of the SV centers are circles, and only 2 of them are balls. Note that the SV centers are chosen with respect to the classification task to be solved.

1 and 0 for positive and negative examples, respectively. The networks were trained without weight decay, however, a bootstrap procedure was used to limit their complexity. The final RBF network for each class contains every Gaussian kernel from its target class, but only several kernels from the other 9 classes, selected such that no false positive mistakes are made. For further details, see (Sung, 1996; Moody and Darken, 1989).

- **Hybrid system.** To assess the relative influence of the automatic SV center choice and the SV weight optimization, respectively, another RBF system was built, constructed with centers that are simply the Support Vectors arising from the SV optimization, and with the weights trained separately using mean squared error back propagation.

Computational Complexity. By construction, the resulting classifiers after training will have the same architecture and comparable sizes. Thus the three machines are comparable in classification speed and memory requirements.

Differences were, however, noticeable in training. Regarding training time, the SV

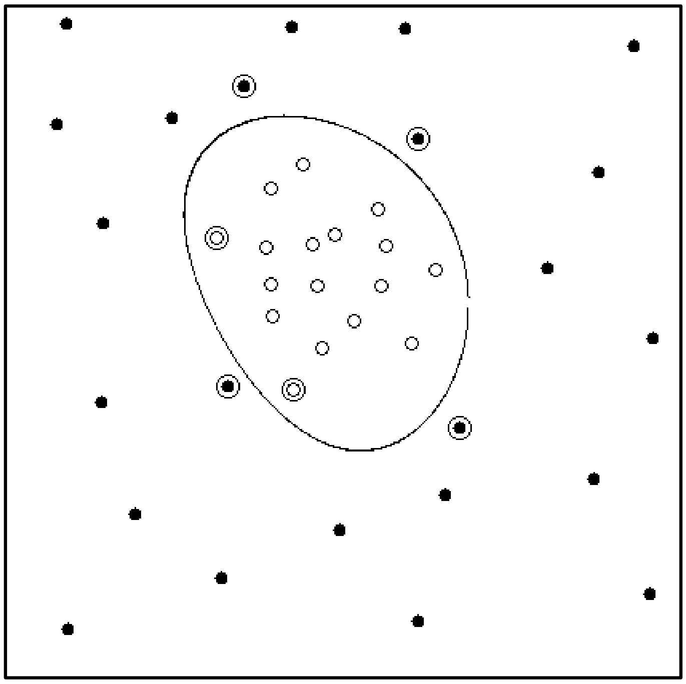

FIGURE 2.14: Two-class classification problem solved by the Support Vector algorithm ($c_i = 1$ for all i; cf. Eq. 2.40).

machine was faster than the RBF system by about an order of magnitude.[12] The optimization, however, requires to work with potentially large matrices. In the implementation that we used, the training data is processed in chunks, and matrix sizes were of the order 500×500. For problems with very large numbers of SVs, a modified training algorithm has recently been proposed by Osuna, Freund, and Girosi (1997).

Error Functions. Due to the constraints (2.18) and the target function (2.19), the SV algorithm puts emphasis on correctly separating the training data. In this respect, it is different from the classical RBF approach of training in the least-squares metric, which is more concerned with the general problem of estimating posterior probabilities than with directly solving a classification task at hand. There exist, however, studies investigating the question how to select RBF centers or exemplars to minimize the number of misclassifications, see for instance (Chang and Lippmann, 1993; Duda and Hart, 1973; Reilly, Cooper, and Elbaum, 1982; Barron, 1984). A classical RBF system could also be made more discriminant by using moving centers (e.g. Poggio and Girosi, 1990), or a different cost function, as the classification figure of merit (Hampshire and Waibel, 1990). In fact, it can be shown that Gaussian RBF regularization networks are equivalent to SV machines if the regularization operator and the cost function are chosen appropriately (Smola and Schölkopf, 1997b).

[12]For noisy regression problems, on the other hand, Support Vector machines can be slower (Müller et al., 1997).

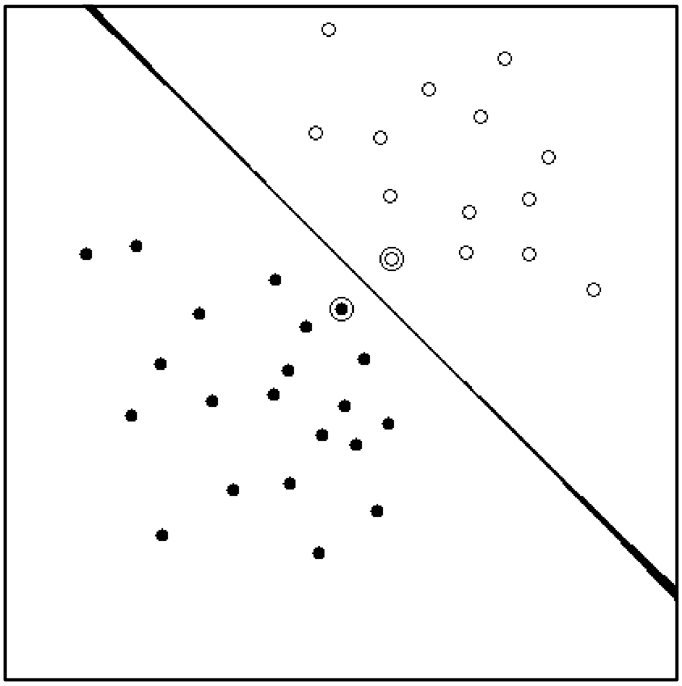

FIGURE 2.15: A simple two-class classification problem as solved by the SV algorithm ($c_i = 1$ for all i; cf. Eq. 2.40). Note that the RBF centers (indicated by extra circles) are closest to the decision boundary. Interestingly, the decision boundary is a straight line, even though a nonlinear Gaussian RBF kernel was used. This is due to the fact that only two SVs are required to solve the problem. The translational and unitary invariance of the RBF kernel then renders the situation completely symmetric.

It is important to stress that the SV machine does not minimize the empirical risk (misclassification error on the training set) alone. Instead it minimizes the sum of an upper bound on the empirical risk and a penalty term that depends on the complexity of the classifier used.

2.5.2 Toy Examples: What are the Support Vectors?

Support Vectors are elements of the data set that are "important" in separating the two classes from each other. In general, the SVs with zero slack variables (2.17) lie on the boundary of the decision surface, as they precisely satisfy the inequality (2.18) in the high-dimensional space. Figures 2.15 and 2.14 illustrate that for the used Gaussian kernel, this is also the case in input space. This raises an interesting question from the point of view of interpreting the structure of trained RBF networks. The traditional view of RBF networks has been one where the centers were regarded as "templates" or stereotypical patterns. It is this point of view that leads to the clustering heuristic for training RBF networks. In contrast, the SV machine posits an alternate point of view, with the centers being those examples which are critical for a given classification task.

TABLE 2.11: Numbers of centers (Support Vectors) automatically extracted by the Support Vector machine. The first row gives the total number for each binary classifier, including both positive and negative examples; in the second row, we only counted the positive SVs. The latter number was used in the initialization of the k-means algorithm, cf. Sec. 2.5.

digit class	0	1	2	3	4	5	6	7	8	9
# of SVs	274	104	377	361	334	388	236	235	342	263
# of pos. SVs	172	77	217	179	211	231	147	133	194	166

TABLE 2.12: Two-class-classification: numbers of test errors (out of 2007 test patterns) for the three systems described in Sec. 2.5.

digit class	0	1	2	3	4	5	6	7	8	9
classical RBF	20	16	43	38	46	31	15	18	37	26
RBF with SV centers	9	12	27	24	32	24	19	16	26	16
full SV machine	16	8	25	19	29	23	14	12	25	16

2.5.3 Handwritten Digit Recognition

We used the USPS database of handwritten digits (Appendix C). The SV machine results reported in the following were obtained with our default choice $\gamma = 10$ (cf. (2.19), Sec. 2.3), and $c = 0.3 \cdot N$ (cf. (2.27)), where $N = 256$ is the dimensionality of input space.[13]

Two-class classification. Table 2.11 shows the numbers of Support Vectors, i.e. RBF centers, extracted by the SV algorithm. Table 2.12 gives the results of binary classifiers separating single digits from the rest, for the systems described in Sec. 2.5.

Ten-class classification. For each test pattern, the arbitration procedure in all three systems simply returns the digit class whose recognizer gives the strongest response (cf. (2.38)). Table 2.13 shows the 10-class digit recognition error rates for our original system and the two RBF-based systems.

The fully automatic SV machine exhibits the highest test accuracy of the three systems.[14] Using the SV algorithm to choose the centers for the RBF network is also better than the baseline procedure of choosing the centers by a clustering heuristic as described above. It can be seen that in contrast to the k-means cluster centers, the centers chosen by the SV algorithm allow zero training error rates.

The considered recognition task is known to be rather hard — the human error rate

[13]The SV machine is rather insensitive to different choices of c. For all values in $0.1, 0.2, \ldots, 1.0$, the performance is about the same (in the area of 4% – 4.5%).

[14]An analysis of the errors showed that about 85% of the errors committed by the SV machine were also made by the other systems. This makes the differences in error rates very reliable.

TABLE 2.13: 10-class digit recognition error rates for three RBF classifiers constructed with different algorithms. The first system is a classical one, choosing its centers by k-means clustering. In the second system, the Support Vectors were used as centers, and in the third one, the entire network was trained using the Support Vector algorithm.

	Classification Error Rate		
USPS DB	classical RBF	RBF with SV centers	full SV machine
Training	1.7%	0.0%	0.0%
Test	6.7%	4.9%	4.2%

is 2.5% (Bromley and Säckinger, 1991), almost matched by a memory-based Tangent-distance classifier (2.6%, Simard, LeCun, and Denker, 1993). Other results on this database include a Euclidean distance nearest neighbour classifier (5.9%, Simard, LeCun, and Denker, 1993), a perceptron with one hidden layer, 5.9%, and a convolutional neural network (5.0%, LeCun et al., 1989). By incorporating translational and rotational invariance using the Virtual SV technique (see below, Sec. 4.2.1), we were able to improve the performance of the considered Gaussian kernel SV machine (same values of γ and c) from 4.2% to 3.2% error.

2.5.4 Summary and Discussion

The Support Vector algorithm provides a principled way of choosing the number and the locations of RBF centers. Our experiments on a real-world pattern recognition problem have shown that in contrast to a corresponding number of centers chosen by k-means, the centers chosen by the Support Vector algorithm allowed a training error of zero, even if the weights were trained by classical RBF methods. The interpretation of this finding is that the Support Vector centers are specifically chosen for the classification task at hand, whereas k-means does not care about picking those centers which will make a problem separable.

In addition, the SV centers yielded lower test error rates than k-means. It is interesting to note that using SV centers, while sticking to the classical procedure for training the weights, improved training and test error rates by approximately the same amount (2%). In view of the guaranteed risk bound (1.5), this can be understood in the following way: the improvement in test error (risk) was solely due to the lower value of the training error (empirical risk); the confidence term (the second term on the right hand side of (1.5)), depending on the VC-dimension and thus on the norm of the weight vector, did not change, as we stuck to the classical weight training procedure. However, when we also trained the weights with the Support Vector algorithm, we minimized the norm of the weight vector (see (2.19), (2.4)) in feature space, and thus the confidence term, while still keeping the training error zero. Thus, consistent with (1.5), the Support Vector machine achieved the highest test accuracy of the three systems.[15]

2.6 Model Selection

2.6.1 Choosing Polynomial Degrees[16]

In the case where the available amount of training data is limited, it is important to have a means for achieving the best possible generalization by controlling characteristics of the learning machine, without having to put aside parts of the training set for validation purposes. One of the strengths of SV machines consists in the automatic capacity tuning, which was related to the fact that the minimization of (2.19) is connected to structural risk minimization, based on the bound (1.5). This capacity tuning takes place within a set of functions specified a priori by the choice of a kernel function. In the following, we go one step further and use the bound (1.5) also to predict the kernel *degree* which yields the best generalization for polynomial classifiers (Schölkopf, Burges, and Vapnik, 1995).

Since for SV machines we have an upper bound on the VC-dimension (Proposition 2.1.1), we can use (1.5) to get an upper bound on the expected error on an independent test set in terms of the training error and the value of $\|\mathbf{w}\|$ (or, equivalently, the margin $2/\|\mathbf{w}\|$). This bound can then be used to try to choose parameters of the learning machines such that the test error gets minimal, without actually looking at the test set.

We consider polynomial classifiers with the kernel (2.26), varying their degree d, and make the assumption that the bound (2.4) gives a reliable indication of the actual VC-dimension, i.e. that the VC-dimension can be estimated by

$$h \approx c_1 h_{est.} = R^2 \|\mathbf{w}\|^2 \tag{2.41}$$

with some $c_1 < 1$ which is independent of the polynomial degree.

For the USPS digit recognition problem, training errors are very small. In that case, the right hand side of the bound (1.5) is dominated by the confidence term, which is minimized when the VC-dimension is minimized. For the latter, we use (2.41), with

[15] Two remarks on the interpretation of our findings are in order. The first result, comparing the error rates of the classical and the hybrid system, does not necessarily rule out the possibility of reducing the training error also for k-means centers by using different cost functions or codings of the output units. It should be considered as a statement comparing two sets of centers, using the same weight training algorithm to build RBF networks from them. Along similar lines, the second result, indicating the superior performance of the full SV RBF system, refers to the systems as described in this study. It does not rule out the possibility of improving classical RBF systems by suitable methods of complexity control. Indeed, the results for the SV RBF system do show that using the same architecture, but different weight training procedures, can significantly improve results.

[16] *Copyright notice:* the material in this section is based on the article "Extracting support data for a given task" by B. Schölkopf, C. Burges and V. Vapnik, which appeared in: Proceedings, First International Conference on Knowledge Discovery & Data Mining, pp. 252 – 257, 1995. AIII Press.

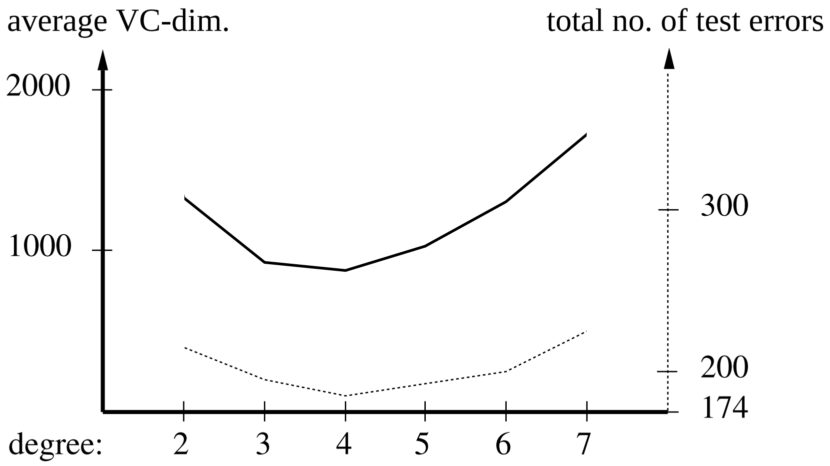

FIGURE 2.16: Average VC-dimension (solid) and total number of test errors of the ten two-class-classifiers (dotted) for polynomial degrees 2 through 7 (for degree 1, R_{emp} is comparably big, so the VC-dimension alone is not sufficient for predicting R, cf. (1.5)). The baseline on the error scale, 174, corresponds to the total number of test errors of the ten best binary classifiers out of the degrees 2 through 7. The graph shows that the VC-dimension allows us to predict that degree 4 yields the best overall performance of the two-class-classifiers on the test set. This is not necessarily the case for the performances of the *ten-class* classifiers, which are built from the two-class-classifier outputs before applying the sgn functions (cf. Sec. 2.1.6).

$||\mathbf{w}||$ determined by the Support Vector algorithm (note that $||\mathbf{w}||$ is computed in feature space, using the kernel). Thus, in order to compute $h_{est.}$, we need to compute R, the radius of the smallest sphere enclosing the training data in feature space. This can be reduced to a quadratic programming problem similar to the one used in constructing the optimal hyperplane:[17]

We formulate the problem as follows:

$$\text{Minimize } \; R^2 \; \text{ subject to } \; ||\mathbf{z}_i - \mathbf{z}^*||^2 \leq R^2, \tag{2.42}$$

where $\mathbf{z}^*$ is the (to be determined) center of the sphere. We use the Lagrangian

$$R^2 - \sum_i \lambda_i (R^2 - (\mathbf{z}_i - \mathbf{z}^*)^2), \tag{2.43}$$

and compute the derivatives by $\mathbf{z}^*$ and R to get

$$\mathbf{z}^* = \sum_i \lambda_i \mathbf{z}_i \tag{2.44}$$

and the Wolfe dual problem: maximize

$$\sum_i \lambda_i \cdot (\mathbf{z}_i \cdot \mathbf{z}_i) - \sum_{i,j} \lambda_i \lambda_j \cdot (\mathbf{z}_i \cdot \mathbf{z}_j) \tag{2.45}$$

[17]The following derivation is due to Chris Burges.

subject to

$$\sum_i \lambda_i = 1, \ \lambda_i \geq 0, \tag{2.46}$$

where the λ_i are Lagrange multipliers. As in the Support Vector algorithm, this problem has the property that the $\mathbf{z}_i$ appear only in dot products, so as before one can compute the dot products in feature space, replacing $(\mathbf{z}_i \cdot \mathbf{z}_j)$ by $k(\mathbf{x}_i, \mathbf{x}_j)$ (where the $\mathbf{x}_i$ live in input space, and the $\mathbf{z}_i$ in feature space).

In this way, we compute the radius of the minimal enclosing sphere for all the USPS training data for polynomial classifiers of degrees 1 through 7. For the same degrees, we then train a binary polynomial classifier for each digit. Using the obtained values for $h_{est.}$, we can predict, for each digit, which degree polynomial will give the best generalization performance. Clearly, this procedure is contingent upon the validity of the assumption that c_1 is approximately the same for all degrees. We can then compare this prediction with the actual polynomial degree which gives the best performance on the test set. The results are shown in Table 2.14; cf. also Fig. 2.16.

TABLE 2.14: Performance of the classifiers with degree predicted by the VC-bound. Each row describes one two-class-classifier separating one digit (stated in the first column) from the rest. The remaining columns contain: deg: the degree of the best polynomial as predicted by the described procedure, param.: the dimensionality of the high dimensional space, which is also the VC-dimension for the set of all separating hyperplanes in that space, $h_{est.}$: the VC-dimension estimate for the actual classifiers, which is much smaller than the number of free parameters of linear classifiers in that space, 1 – 7: the numbers of errors on the test set for polynomial classifiers of degrees 1 through 7. The table shows that the decribed procedure chooses polynomial degrees which are optimal or close to optimal.

	chosen classifier			errors on the test set for degrees 1 – 7						
digit	deg	param.	$h_{est.}$	1	2	3	4	5	6	7
0	3	$2.8 \cdot 10^6$	547	36	14	$\boxed{11}$	11	11	12	17
1	7	$1.5 \cdot 10^{13}$	95	17	15	14	11	10	9	$\boxed{10}$
2	3	$2.8 \cdot 10^6$	832	53	32	$\boxed{28}$	26	28	27	32
3	3	$2.8 \cdot 10^6$	1130	57	25	$\boxed{22}$	22	22	22	23
4	4	$1.8 \cdot 10^8$	977	50	32	32	$\boxed{30}$	30	29	33
5	3	$2.8 \cdot 10^6$	1117	37	20	$\boxed{22}$	24	24	26	28
6	4	$1.8 \cdot 10^8$	615	23	12	12	$\boxed{15}$	17	17	19
7	5	$9.5 \cdot 10^9$	526	25	15	12	10	$\boxed{11}$	13	14
8	4	$1.8 \cdot 10^8$	1466	71	33	28	$\boxed{24}$	28	32	34
9	5	$9.5 \cdot 10^9$	1210	51	18	15	11	$\boxed{11}$	12	15

The above method for predicting the optimal classifier functions gives good results. In four cases, the theory predicted the correct degree; in the other cases, the predicted degree gave performance close to the best possible one.

2.6.2 The Choice of the Regularization Parameter γ

In addition to kernel parameters as the polynomial degree, there is another parameter whose value needs to be set for SV training: the regularization constant γ, determining the trade-off between minimizing training error and controlling complexity (cf. (2.19)). The optimal value of γ should depend both on characteristics of the problem at hand and on the sample size. Although our experience suggests that for problems with little noise, the results are reasonably insensitive with respect to changes of γ, it would still be desirable to have a principled method for choosing γ. The remainder of this section is an attempt at developing such a method.

As in the last section, the starting point is the risk bound (1.5). The idea is to adjust γ such that minimization of the SV objective function (2.19) amounts to minimizing (1.5). As the solution $\mathbf{w}$ depends on the value of γ chosen (in (2.19)), we cannot use (1.5) and (2.4) to determine the value of γ a priori. Instead, we will resort to an iterative strategy.

Following (2.4), we write

$$h = c_1 R^2 \|\mathbf{w}\|^2 \tag{2.47}$$

with some $c_1 < 1$. Substituting this into the bound (1.5), (1.6), we obtain

$$\sqrt{\frac{c_1 R^2 \|\mathbf{w}\|^2 \left(\log \frac{2\ell}{c_1 R^2 \|\mathbf{w}\|^2} + 1\right) - \log(\eta/4)}{\ell}} + R_{emp}(\alpha) \tag{2.48}$$

as an upper bound on the risk that we want to minimize.

Remembering that $\sum_i \xi_i$ is an upper bound on the number of training errors, we additionally write

$$\sum_i \xi_i = c_2 \ell R_{emp}(\mathbf{w}), \tag{2.49}$$

with some $c_2 \geq 1$, where ℓ is the number of training examples. Minimizing the objective function (2.19) then amounts to minimizing

$$\frac{1}{2\gamma c_2 \ell} \|\mathbf{w}\|^2 + R_{emp}(\mathbf{w}). \tag{2.50}$$

Identifying $\mathbf{w}$ with the function index α in (2.48), we now have a formulation where the second terms of (2.50) and (2.48) are identical. The first terms cannot coincide in general: unlike the first term of (2.48), $\|\mathbf{w}\|^2/(2\gamma c_2 \ell)$ is proportional to $\|\mathbf{w}\|^2$. However, a necessary condition to ensure that the minimum of the function that we are minimizing, (2.50), is close to that of the one that we would like to minimize, (2.48), is that the gradients of the first terms with respect to $\mathbf{w}$ coincide at the minimum. Hence, we have to choose γ such that

$$\frac{1}{\gamma c_2 \ell}\mathbf{w} = \frac{c_1 R^2 (\log \frac{2\ell}{c_1 R^2 \|\mathbf{w}\|^2} - 1)}{\sqrt{\ell (c_1 R^2 \|\mathbf{w}\|^2 \log \frac{2\ell}{c_1 R^2 \|\mathbf{w}\|^2} - \log(\eta/4))}}\mathbf{w}. \tag{2.51}$$

For $\mathbf{w} \neq 0$, we thus obtain

$$\gamma = \frac{\sqrt{\ell (c_1 R^2 \|\mathbf{w}\|^2 \log \frac{2\ell}{c_1 R^2 \|\mathbf{w}\|^2} - \log(\eta/4))}}{c_1 c_2 \ell R^2 (\log \frac{2\ell}{c_1 R^2 \|\mathbf{w}\|^2} - 1)}. \tag{2.52}$$

Eq. (2.52) establishes a relationship between γ and $\mathbf{w}$ at the point of the solution. If we start with a non-optimal value of γ, however, we will obtain a non-optimal $\mathbf{w}$ and thus (2.52) will not tell us exactly how to adjust γ. For instance, suppose we start with a γ which is too big. Then too much weight will be put on minimizing empirical risk (cf. (2.19)), and the margin will become too small, i.e. $\mathbf{w}$ will become too big. We will resort to the following method: we use (2.52) to determine a new value γ', and iterate the procedure.

The value of $\|\mathbf{w}\|^2$ is obtained by solving the SV quadratic programming problem (in feature space, we have $\|\mathbf{w}\|^2 = \sum_{ij} y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$); R^2 is computed as in Sec. 2.6.1, and c_2 is obtained from $\sum_i \xi_i$ and the training error using (2.49). The values of c_1 and η must be chosen by the user. The constant c_1 characterizes the tightness of the VC-dimension bound (cf. (2.47)), and $1 - \eta$ is a lower bound on the probability with which the risk bound (2.48) holds. As long as η is not too close to 0, it does hardly affect our procedure. The value of c_1 is more difficult to choose correctly, however, reasonable results can already be obtained with the default choice $c_1 = 1$, as we shall see below.

Statements on the convergence of this procedure are hard to obtain: to compute the mapping from γ to γ', we have to train an SV machine and then evaluate (2.52), thus one cannot compute its derivative in closed form. In an empirical study to be decribed presently, the procedure exhibited well-behaved convergence behaviour. In the experiment, we used the small MNIST database (Appendix C). We found that the iteration converged no matter whether we started with a very large γ or with a tiny γ. In the following, we report results obtained when starting with a huge value of γ, which effectively leads to the construction of an optimal margin classifier (i.e. with zero training error — cf. Sec. 2.1.2: for $\gamma = \infty$, (2.22) reduces to (2.14)).

Table 2.15 shows partly encouraging results. For all 10 binary digit recognizers, the iteration converges very fast (about two steps were required to reach a value very close to the final one). In seven cases, the number of test errors decreased, in only two case did it increase. By combining the resulting binary classifiers, we obtained a 10-class classifier with an error rate (on the 10000 element small MNIST test set) of 3.9%, slightly better than the error rate obtained both with the starting value used in the iteration, $\gamma = 10^{10}$, and with our default choice $\gamma = 10$: in these cases, we obtained 4.0% error (cf. below, in Table 4.6).

Clearly, further experiments are necessary to validate or improve this method. In particular, it would be interesting to study a noisy classification problem, where the choice of γ should potentially have a greater impact on the quality of the solution.

We conclude this section with a note on the relationship of the model selection methods described in sections 2.6.1 and 2.6.2. Both of the proposed methods are based on the bound (1.5). In principle, we could also apply the method of 2.6.1 for choosing γ. In that case, we would *try out* a series of values of γ, and pick the one which minimizes (1.5). The advantage of the present method, however, is that it does not require scanning a whole range of values. Instead, γ is chosen such that, with the help of a few iterations, the SV optimization automatically minimizes (1.5) over γ, in addition to the built-in minimization over the weights of the SV machine (cf. the remarks at the beginning of Sec. 2.6.1).[18]

TABLE 2.15: Iterative choice of the regularization constant γ (cf. Sec. 2.6.2) for all ten digit recognizers on the small MNIST database. Each table shows SV machine parameters and results for the starting point ($\gamma = 10^{10}$), and for five subsequent iteration steps. In all cases, we used $c_1 = 1$, $\eta = 0.2$ (corresponding to a risk bound holding with probability of at least 0.8), and a polynomial classifier of degree 5. The constant c_2 is undefined before the first run of the algorithm. After each run, it is computed using (2.49); if R_{emp} is 0, we set $c_2 = 1$. For $\gamma = 10^{10}$, we are effectively computing an optimal separating hyperplane, with zero training errors. The iteration converges very fast; moreover, in seven of the ten cases, it reduced the number of test errors (in two cases, the opposite happened).

	γ	c_2	train. errors	test errors	# of SVs	$\|\|\mathbf{w}\|\|$
digit 0	10^{10}	-	0	38	177	36.156
	0.723763	1	3	32	187	29.460
	0.052130	10.0	3	32	194	29.288
	0.050580	10.3	3	31	194	29.312
	0.047947	10.8	3	30	192	29.416
	0.051618	10.1	3	31	188	29.316

	γ	c_2	train. errors	test errors	# of SVs	$\|\|\mathbf{w}\|\|$
digit 1	10^{10}	-	0	33	141	48.998
	1.248717	1	3	30	153	34.286
	0.047532	14.0	3	31	160	34.063
	0.045677	14.4	3	30	161	33.988
	0.042151	15.5	3	29	157	34.054
	0.046176	14.2	3	31	154	34.038

[18] We performed another set of experiments to find out whether the leave-one-out generalization bound (Proposition 2.1.2) could be used for selecting γ. On the small MNIST database, the results were negative, leading to values of γ which were too large (in the range of 10^5 to 10^{10}).

	γ	c_2	train. errors	test errors	# of SVs	$\|\mathbf{w}\|$
digit 2	10^{10}	-	0	104	340	58.816
	1.853855	1	4	88	355	49.055
	0.100126	12.5	4	87	354	48.810
	0.094182	13.2	4	87	352	48.757
	0.092732	13.3	4	87	352	48.833
	0.095662	13.0	4	88	351	48.906
	γ	c_2	train. errors	test errors	# of SVs	$\|\mathbf{w}\|$
digit 3	10^{10}	-	0	96	377	70.455
	3.047037	1	5	93	392	57.387
	0.139778	12.5	5	93	397	56.860
	0.122975	13.9	5	94	387	57.143
	0.142765	12.1	5	96	385	56.868
	0.123462	13.9	5	93	383	57.272
	γ	c_2	train. errors	test errors	# of SVs	$\|\mathbf{w}\|$
digit 4	10^{10}	-	0	74	282	66.786
	2.586497	1	5	79	312	52.781
	0.134502	10.8	6	77	313	52.409
	0.150066	9.6	6	77	312	52.374
	0.152267	9.4	6	77	313	52.324
	0.147880	9.7	6	77	311	52.338
	γ	c_2	train. errors	test errors	# of SVs	$\|\mathbf{w}\|$
digit 5	10^{10}	-	0	87	339	65.051
	2.400509	1	4	101	358	53.578
	0.116545	12.9	5	99	353	53.182
	0.126663	11.7	5	99	362	53.263
	0.129597	11.5	5	98	358	53.250
	0.126985	11.7	5	101	363	53.272
	γ	c_2	train. errors	test errors	# of SVs	$\|\mathbf{w}\|$
digit 6	10^{10}	-	0	80	231	46.857
	1.144632	1	2	80	260	37.923
	0.037990	20.6	2	80	264	37.623
	0.037135	20.8	2	78	256	37.773
	0.039888	19.5	2	78	253	37.834
	0.041169	19.0	2	79	258	37.771
	γ	c_2	train. errors	test errors	# of SVs	$\|\mathbf{w}\|$
digit 7	10^{10}	-	0	122	253	69.716
	2.945887	1	9	109	272	48.730
	0.187035	6.6	10	109	271	48.263
	0.196960	6.2	10	108	278	48.104
	0.189135	6.4	10	108	270	48.241
	0.197877	6.1	10	108	272	48.258

	γ	c_2	train. errors	test errors	# of SVs	$\|\mathbf{w}\|$
digit 8	10^{10}	-	0	127	440	77.167
	4.246777	1	3	126	473	63.982
	0.102221	22.4	5	126	463	63.590
	0.156304	14.4	5	122	464	63.703
	0.165320	13.7	5	126	457	63.605
	0.166110	13.6	5	127	462	63.492
	γ	c_2	train. errors	test errors	# of SVs	$\|\mathbf{w}\|$
digit 9	10^{10}	-	0	146	412	94.997
	21.34817	1	2	146	446	74.464
	0.103062	35.8	11	140	437	71.822
	0.417387	7.8	10	137	434	71.893
	0.383747	8.5	11	141	435	71.857
	0.439163	7.4	11	139	440	71.900

2.7 Why Do SV Machines Work Well?

The presented experimental results show that Support Vector machines obtain high accuracies which are competitive with state-of-the-art techniques. This was true for several visual recognition tasks. Care should be exercised, however, when generalizing this statement to other types of pattern recognition tasks. There, empirical studies have yet to be carried out, in particular since the tasks that we considered were all characterized by relatively low overlap of the different classes (for instance, in the USPS task, the human error rate is around 2.5%). In any case, the results obtained here are encouraging, in particular when taking into account that the SV algorithm was developed only recently. Below, we summarize different aspects providing insight in why SV machines generalize well:

Capacity Control. The kernel method allows to reduce a large class of learning machines to separating hyperplanes in some space. For those, an upper bound on the VC-dimension can be given (Proposition 2.1.1). As argued in Sec. 2.1.3, minimizing the SV objective function (2.19) corresponds to trying to separate the data with a classifier of low VC-dimension, thereby approximately performing structural risk minimization. The problem of constructing the decision function requires minimizing a positive quadratic form subject to box constraints and can thus be solved efficiently.

As we saw, low VC-dimension is related to a large separation margin. Thus, analyses of the generalization performance in terms of separation margins and fat shattering dimension also bear relevance to SV machines (e.g. Schapire, Freund, Bartlett, and Lee, 1997; Shawe-Taylor, Bartlett, Williamson, and Anthony, 1996; Bartlett, 1997).

Compression. The leave-one-out bound (Proposition 2.1.2) relates SV generalization ability to the fact that the decision function is expressed in terms of a (possibly small) subset of the data. This can be viewed in the context of Algorithmic Complexity and Minimum Description Length (Vapnik, 1995b, Chapter 5, footnote 6).

Regularization. In (Smola and Schölkopf, 1997b), a regularization framework is proposed which contains the SV algorithm as a special case. For kernel-based function expansions, it is shown that given a regularization operator P (Tikhonov and Arsenin, 1977) mapping the functions of the learning machine into some dot product space $\mathcal{D}$, the problem of minimizing the regularized risk

$$R_{reg}[f] = R_{emp}[f] + \lambda \|Pf\|^2, \tag{2.53}$$

(with a regularization parameter $\lambda \geq 0$) can be written as a constrained optimization problem. For particular choices of the cost function, it further reduces to a SV type quadratic programming problem. The latter thus is not specific to SV machines, but is common to a much wider class of approaches. What gets lost in this case, however, is the fact that the solution can usually be expressed in terms of a small number of SVs. This specific feature of SV machines is due to the fact that the type of regularization and the class of functions which are considered as admissible solutions are intimately related (cf. Poggio and Girosi, 1990; Girosi, Jones, and Poggio, 1993; Smola and Schölkopf, 1997a; Smola, Schölkopf, and Müller, 1997): the SV algorithm is equivalent to minimizing the regularized risk on the set of functions

$$f(\mathbf{x}) = \sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b, \tag{2.54}$$

provided that k and P are interrelated by

$$k(\mathbf{x}_i, \mathbf{x}_j) = ((Pk)(\mathbf{x}_i, .) \cdot (Pk)(\mathbf{x}_j, .)) \,. \tag{2.55}$$

(Here, $(Pk)(\mathbf{x}_i, .)$ denotes the result of applying P to the function obtained by fixing k's first argument to $\mathbf{x}_i$.) To this end, k is chosen as Green's function of P^*P. For instance, an RBF kernel thus corresponds to regularization with a functional containing a specific differential operator (Yuille and Grzywacz, 1988).

Chapter 3

Kernel Principal Component Analysis

In the last chapter, we tried to show that the idea of implicitely mapping the data into a high-dimensional feature space has been a very fruitful one in the context of SV machines. Indeed, it is mainly this feature which distinguishes them from the Generalized Portrait algorithm which has been known for more than 20 years (Vapnik and Chervonenkis, 1974), and which makes them applicable to complex real-world problems which are not linearly separable. Thus, it was natural to ask the question whether the same idea could prove fruitful in other domains of learning.

The present chapter proposes a new method for performing a nonlinear form of Principal Component Analysis. By the use of Mercer kernels, one can efficiently compute principal components in high-dimensional feature spaces, related to input space by some nonlinear map. We give the derivation of the method and present experimental results on polynomial feature extraction for pattern recognition (Schölkopf, Smola, and Müller, 1996b, 1997b).

3.1 Introduction

Principal Component Analysis (PCA) is a powerful technique for extracting structure from possibly high-dimensional data sets. It is readily performed by solving an Eigenvalue problem, or by using iterative algorithms which estimate principal components. For reviews of the existing literature, see Jolliffe (1986) and Diamantaras & Kung (1996); some of the classical papers are due to Pearson (1901); Hotelling (1933); Karhunen (1946). PCA is an orthogonal transformation of the coordinate system in which we describe our data. The new coordinate values by which we represent the data are called *principal components*. It is often the case that a small number of principal components is sufficient to account for most of the structure in the data. These are sometimes called *factors* or *latent variables* of the data.

The present work studies PCA in the case where we are not interested in principal components in input space, but rather in principal components of variables, or *features*, which are nonlinearly related to the input variables. Among these are for instance variables obtained by taking arbitrary higher-order correlations between input variables. In the case of image analysis, this amounts to finding principal components in the space of products of input pixels.

To this end, we are computing dot products in feature space by means of kernel functions in input space (cf. Sec. 1.3). Given *any* algorithm which can be expressed solely in terms of dot products, i.e. without explicit usage of the variables themselves, this kernel method enables us to construct different nonlinear versions of it. Even though this general fact was known (Burges, private communication), the machine learning community has made little use of it, the exception being Vapnik's Support Vector machines (Chapter 2). In this chapter, we give an example of applying this method in the domain of unsupervised learning, to obtain a nonlinear form of PCA.

In the next section, we will first review the standard PCA algorithm. In order to be able to generalize it to the nonlinear case, we formulate it in a way which uses exclusively dot products. Using kernel representations of dot products (Sec. 1.3), Sec. 3.3 presents a kernel-based algorithm for nonlinear PCA and explains some of the differences to previous generalizations of PCA. First experimental results on kernel-based feature extraction for pattern recognition are given in Sec. 3.4. We conclude with a discussion (Sec. 3.5). Some technical material which is not essential for the main thread of the argument has been relegated to Appendix D.2.

3.2 Principal Component Analysis in Feature Spaces

Given a set of centered observations $\mathbf{x}_k \in \mathbf{R}^N$, $k = 1, \ldots, M$, $\sum_{k=1}^M \mathbf{x}_k = 0$, PCA diagonalizes the covariance matrix[1]

$$C = \frac{1}{M} \sum_{j=1}^{M} \mathbf{x}_j \mathbf{x}_j^\top. \tag{3.1}$$

To do this, one has to solve the Eigenvalue equation

$$\lambda \mathbf{v} = C\mathbf{v} \tag{3.2}$$

for Eigenvalues $\lambda \geq 0$ and Eigenvectors $\mathbf{v} \in \mathbf{R}^N \backslash \{0\}$. As

$$\lambda \mathbf{v} = C\mathbf{v} = \frac{1}{M} \sum_{j=1}^{M} (\mathbf{x}_j \cdot \mathbf{v}) \mathbf{x}_j, \tag{3.3}$$

all solutions $\mathbf{v}$ with $\lambda \neq 0$ must lie in the span of $\mathbf{x}_1 \ldots \mathbf{x}_M$, hence (3.2) in that case is equivalent to

$$\lambda(\mathbf{x}_k \cdot \mathbf{v}) = (\mathbf{x}_k \cdot C\mathbf{v}) \text{ for all } k = 1, \ldots, M. \tag{3.4}$$

[1]More precisely, the covariance matrix is defined as the expectation of $\mathbf{x}\mathbf{x}^\top$; for convenience, we shall use the same term to refer to the estimate (3.1) of the covariance matrix from a finite sample.

In the remainder of this section, we describe the same computation in another dot product space F, which is related to the input space by a possibly nonlinear map

$$\Phi : \mathbf{R}^N \to F, \quad \mathbf{x} \mapsto \mathbf{X}. \tag{3.5}$$

Note that the *feature space* F could have an arbitrarily large, possibly infinite, dimensionality. Here and in the following, upper case characters are used for elements of F, while lower case characters denote elements of $\mathbf{R}^N$.

Again, we assume that we are dealing with centered data, $\sum_{k=1}^{M} \Phi(\mathbf{x}_k) = 0$ — we shall return to this point later. In F, the covariance matrix takes the form

$$\bar{C} = \frac{1}{M} \sum_{j=1}^{M} \Phi(\mathbf{x}_j)\Phi(\mathbf{x}_j)^\top. \tag{3.6}$$

Note that if F is infinite-dimensional, we think of $\Phi(\mathbf{x}_j)\Phi(\mathbf{x}_j)^\top$ as a linear operator on F, mapping

$$\mathbf{X} \mapsto \Phi(\mathbf{x}_j)(\Phi(\mathbf{x}_j) \cdot \mathbf{X}). \tag{3.7}$$

We now have to find Eigenvalues $\lambda \geq 0$ and Eigenvectors $\mathbf{V} \in F \backslash \{0\}$ satisfying

$$\lambda \mathbf{V} = \bar{C}\mathbf{V}. \tag{3.8}$$

Again, all solutions $\mathbf{V}$ with $\lambda \neq 0$ lie in the span of $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_M)$. For us, this has two useful consequences: first, we may instead consider the set of equations

$$\lambda(\Phi(\mathbf{x}_k) \cdot \mathbf{V}) = (\Phi(\mathbf{x}_k) \cdot \bar{C}\mathbf{V}) \text{ for all } k = 1, \ldots, M, \tag{3.9}$$

and second, there exist coefficients α_i $(i = 1, \ldots, M)$ such that

$$\mathbf{V} = \sum_{i=1}^{M} \alpha_i \Phi(\mathbf{x}_i). \tag{3.10}$$

Combining (3.9) and (3.10), we get

$$\lambda \sum_{i=1}^{M} \alpha_i (\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}_i)) = \frac{1}{M} \sum_{i=1}^{M} \alpha_i \left(\Phi(\mathbf{x}_k) \cdot \sum_{j=1}^{M} \Phi(\mathbf{x}_j)(\Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}_i)) \right)$$
$$\text{for all } k = 1, \ldots, M. \tag{3.11}$$

Defining an $M \times M$ matrix K by

$$K_{ij} := (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)), \tag{3.12}$$

this reads

$$M\lambda K\boldsymbol{\alpha} = K^2\boldsymbol{\alpha}, \tag{3.13}$$

where $\boldsymbol{\alpha}$ denotes the column vector with entries $\alpha_1, \ldots, \alpha_M$. To find solutions of (3.13), we solve the Eigenvalue problem

$$M\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha} \tag{3.14}$$

for nonzero Eigenvalues. In Appendix D.2.1, we show that this gives us all solutions of (3.13) which are of interest for us.

Let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_M$ denote the Eigenvalues of K (i.e. the solutions $M\lambda$ of (3.14)), and $\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^M$ the corresponding complete set of Eigenvectors, with λ_p being the first nonzero Eigenvalue (assuming that Φ is not identically 0). We normalize $\boldsymbol{\alpha}^p, \ldots, \boldsymbol{\alpha}^M$ by requiring that the corresponding vectors in F be normalized, i.e.

$$(\mathbf{V}^k \cdot \mathbf{V}^k) = 1 \text{ for all } k = p, \ldots, M. \tag{3.15}$$

By virtue of (3.10) and (3.14), this translates into a normalization condition for $\boldsymbol{\alpha}^p, \ldots, \boldsymbol{\alpha}^M$:

$$\begin{aligned} 1 &= \sum_{i,j=1}^{M} \alpha_i^k \alpha_j^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)) = \sum_{i,j=1}^{M} \alpha_i^k \alpha_j^k K_{ij} \\ &= (\boldsymbol{\alpha}^k \cdot K\boldsymbol{\alpha}^k) = \lambda_k(\boldsymbol{\alpha}^k \cdot \boldsymbol{\alpha}^k) \end{aligned} \tag{3.16}$$

For the purpose of principal component extraction, we need to compute projections onto the Eigenvectors $\mathbf{V}^k$ in F ($k = p, \ldots, M$). Let $\mathbf{x}$ be a test point, with an image $\Phi(\mathbf{x})$ in F, then

$$(\mathbf{V}^k \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{M} \alpha_i^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})) \tag{3.17}$$

may be called its nonlinear principal components corresponding to Φ.

In summary, the following steps were necessary to compute the principal components: first, compute the matrix K, second, compute its Eigenvectors and normalize them in F; third, compute projections of a test point onto the Eigenvectors.[2]

For the sake of simplicity, we have above made the assumption that the observations are centered. This is easy to achieve in input space, but more difficult in F, as we cannot explicitely compute the mean of the mapped observations in F. There is, however, a way to do it, and this leads to slightly modified equations for kernel-based PCA (see Appendix D.2.2).

To conclude this section, note that Φ can be an arbitrary nonlinear map into the possibly high-dimensional space F, e.g. the space of all d-th order monomials in the entries of an input vector. In that case, we need to compute dot products of input vectors mapped by Φ, with a possibly prohibitive computational cost. The solution to this problem, however, is to use kernel functions (1.14) — we *exclusively* need

[2]Note that in our derivation we could have used the known result (e.g. Kirby & Sirovich, 1990) that PCA can be carried out on the dot product matrix $(\mathbf{x}_i \cdot \mathbf{x}_j)_{ij}$ instead of (3.1), however, for the sake of clarity and extendability (in Appendix D.2.2, we shall consider the question how to center the data in F), we gave a detailed derivation.

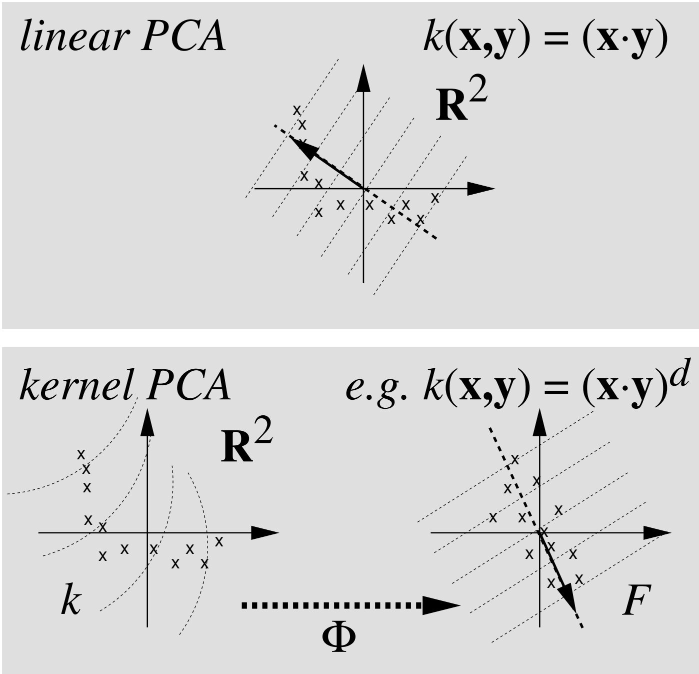

FIGURE 3.1: The basic idea of kernel PCA. In some high-dimensional feature space F (bottom right), we are performing linear PCA, just as a PCA in input space (top). Since F is nonlinearly related to input space (via Φ), the contour lines of constant projections onto the principal Eigenvector (drawn as an arrow) become *nonlinear* in input space. Note that we cannot draw a pre-image of the Eigenvector in input space, as it may not even exist. Crucial to kernel PCA is the fact that there is no need to perform the map into F: all necessary computations are carried out by the use of a kernel function k in input space (here: $\mathbf{R}^2$).

to compute dot products between mapped patterns (in (3.12) and (3.17)); *we never need the mapped patterns explicitely.* Therefore, we can use the kernels described in Sec. 1.3. The particular kernel used then implicitely determines the space F of all possible features. The proposed algorithm, on the other hand, is a mechanism for *selecting* features in F.

3.3 Kernel Principal Component Analysis

The Algorithm. To perform kernel-based PCA (Fig. 3.1), from now on referred to as *kernel PCA*, the following steps have to be carried out: first, we compute the matrix $K_{ij} = (k(\mathbf{x}_i, \mathbf{x}_j))_{ij}$. Next, we solve (3.14) by diagonalizing K, and normalize the Eigenvector expansion coefficients α^n by requiring $\lambda_n(\boldsymbol{\alpha}^n \cdot \boldsymbol{\alpha}^n) = 1$. To extract the principal components (corresponding to the kernel k) of a test point $\mathbf{x}$, we then

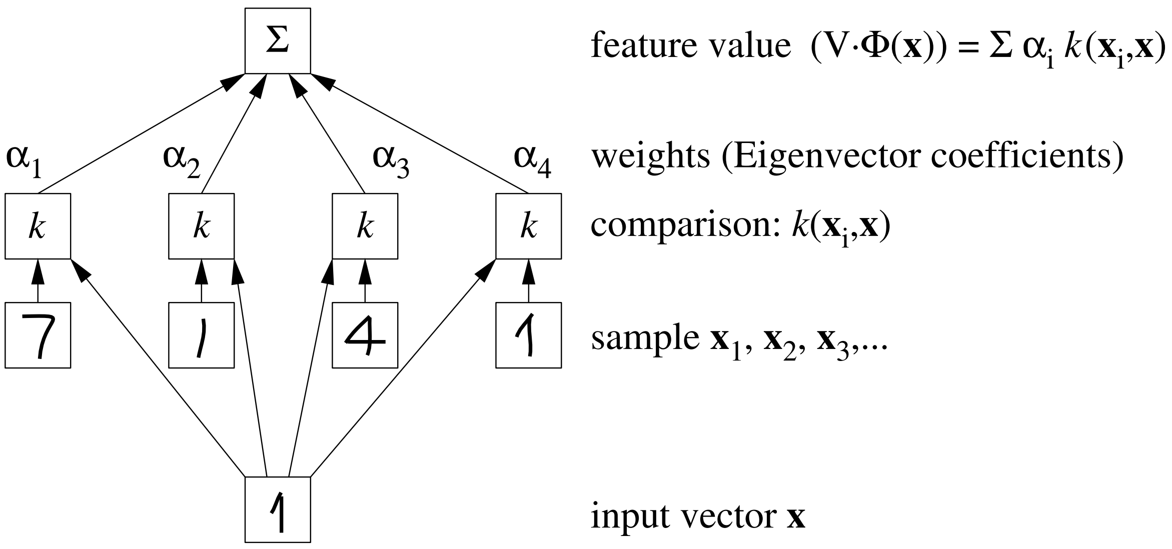

FIGURE 3.2: Feature extractor constructed by kernel PCA (cf. (3.18)). In the first layer, the input vector is compared to the sample via a kernel function, chosen a priori (e.g. polynomial, Gaussian, or sigmoid). The outputs are then linearly combined using weights which are found by solving an Eigenvector problem. As shown in the text, the depicted network's function can be thought of as the projection onto an Eigenvector of a covariance matrix in a high-dimensional feature space. As a function on input space, it is nonlinear.

compute projections onto the Eigenvectors by (cf. Eq. (3.17), Fig. 3.2),

$$(\mathbf{V}^n \cdot \Phi(\mathbf{x})) = \sum_{i=1}^{M} \alpha_i^n k(\mathbf{x}_i, \mathbf{x}). \tag{3.18}$$

If we use a kernel satisfying Mercer's conditions (Proposition 1.3.2), we know that this procedure exactly corresponds to standard PCA in some high-dimensional feature space, except that we do not need to perform expensive computations in that space.

Properties of Kernel-PCA. For Mercer kernels, we know that we are in fact doing a standard PCA in F. Consequently, all mathematical and statistical properties of PCA (see for instance Jolliffe, 1986; Diamantaras & Kung, 1996) carry over to kernel-based PCA, with the modifications that they become statements about a set of points $\Phi(\mathbf{x}_i), i = 1, \ldots, M$, in F rather than in $\mathbf{R}^N$. In F, we can thus assert that PCA is the orthogonal basis transformation with the following properties (assuming that the Eigenvectors are sorted in ascending order of the Eigenvalue size):

- the first q ($q \in \{1, \ldots, M\}$) principal components, i.e. projections on Eigenvectors, carry more variance than any other q orthogonal directions
- the mean-squared approximation error in representing the observations by the first q principal components is minimal[3]

[3]To see this, in the simple case where the data $\mathbf{z}_1, \ldots, \mathbf{z}_\ell$ are centered, we consider an orthogonal

- the principal components are uncorrelated
- the first q principal components have maximal mutual information with respect to the inputs (this holds under Gaussianity assumptions, and thus depends on the particular kernel chosen and on the data)

To translate these properties of PCA in F into statements about the data in input space, they need to be investigated for specific choices of a kernels. We conclude this section by noting one general property of kernel PCA in input space: for kernels which depend only on dot products or distances in input space (as all the examples that we have given so far do), kernel PCA has the property of unitary invariance. This follows directly from the fact that both the Eigenvalue problem and the feature extraction only depend on kernel values. This ensures that the features extracted do not depend on which orthonormal coordinate system we use for representing our input data.

Computational Complexity. As mentioned in Sec. 1.3, a fifth order polynomial kernel on a 256-dimensional input space yields a 10^{10}-dimensional feature space. For two reasons, kernel PCA can deal with this huge dimensionality. First, as pointed out in Sect. 3.2 we do not need to look for Eigenvectors in the full space F, but just in the subspace spanned by the images of our observations $\mathbf{x}_k$ in F. Second, we do not need to compute dot products explicitely between vectors in F (which can be impossible in practice, even if the vectors live in a lower-dimensional subspace), as we are using kernel functions. Kernel PCA thus is computationally comparable to a linear PCA on ℓ observations with an $\ell \times \ell$ dot product matrix. If k is easy to compute, as for polynomial kernels, e.g., the computational complexity is hardly changed by the fact that we need to evaluate kernel functions rather than just dot products. Furthermore, in the case where we need to use a large number ℓ of observations, we may want to work with an algorithm for computing only the largest Eigenvalues, as for instance the power method with deflation (for a discussion, see Diamantaras & Kung, 1996). In addition, we can consider using an estimate of the matrix K, computed from a subset of $M < \ell$ examples, while still extracting principal components from all ℓ examples (this approach was chosen in some of our experiments described below).

The situation can be different for principal component extraction. There, we have to evaluate the kernel function M times for each extracted principal component (3.18),

---

basis transformation W, and use the notation P_q for the projector on the first q canonical basis vectors $\{\mathbf{e}_1, \ldots, \mathbf{e}_q\}$. Then the mean squared reconstruction error using q vectors is

$$\frac{1}{\ell}\sum_i \|\mathbf{z}_i - W^\top P_q W \mathbf{z}_i\|^2 = \frac{1}{\ell}\sum_i \|W\mathbf{z}_i - P_q W \mathbf{z}_i\|^2 = \frac{1}{\ell}\sum_i \sum_{j>q} (W\mathbf{z}_i \cdot \mathbf{e}_j)^2$$

$$= \frac{1}{\ell}\sum_i \sum_{j>q} (\mathbf{z}_i \cdot W^\top \mathbf{e}_j)^2 = \frac{1}{\ell}\sum_i \sum_{j>q} (W^\top \mathbf{e}_j \cdot \mathbf{z}_i)(\mathbf{z}_i \cdot W^\top \mathbf{e}_j) = \sum_{j>q} (W^\top \mathbf{e}_j \cdot C W^\top \mathbf{e}_j).$$

It can easily be seen that the values of this quadratic form (which gives the variances in the directions $W^\top \mathbf{e}_j$) are minimal if the $W^\top \mathbf{e}_j$ are chosen as its (orthogonal) Eigenvectors with smallest Eigenvalues.

rather than just evaluating one dot product as for a linear PCA. Of course, if the dimensionality of F is 10^{10}, this is still vastly faster than linear principal component extraction in F. Still, in some cases, e.g. if we were to extract principal components as a preprocessing step for classification, we might want to speed up things. This can be carried out by the reduced set technique of Burges (1996) (cf. Appendix D.1.1), proposed in the context of Support Vector machines. In the present setting, we approximate each Eigenvector

$$\mathbf{V} = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i) \tag{3.19}$$

(Eq. (3.10)) by another vector

$$\tilde{\mathbf{V}} = \sum_{j=1}^{m} \beta_j \Phi(\mathbf{z}_j), \tag{3.20}$$

where $m < \ell$ is chosen a priori according to the desired speed-up, and $\mathbf{z}_j \in \mathbf{R}^N, j = 1, \ldots, m$. This is done by minimizing the squared difference

$$\rho = \|\mathbf{V} - \tilde{\mathbf{V}}\|^2. \tag{3.21}$$

This can be carried out without explicitely dealing with the possibly high-dimensional space F. Since

$$\rho = \|\mathbf{V}\| + \sum_{i,j=1}^{m} \beta_i \beta_j k(\mathbf{z}_i, \mathbf{z}_j) - 2 \sum_{i=1}^{\ell} \sum_{j=1}^{m} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{z}_j), \tag{3.22}$$

the gradient of ρ with respect to the β_j and the $\mathbf{z}_j$ is readily expressed in terms of the kernel function, thus ρ can be minimized by standard gradient methods. For the task of handwritten character recognition, this technique led to a speed-up by a factor of 50 at almost no loss in accuracy (Burges & Schölkopf, 1996; cf. Sec. 4.4.1).

Finally, we add that although kernel principal component extraction is computationally more expensive than its linear counterpart, this additional investment can pay back afterwards. In experiments on classification based on the extracted principal components, we found when we trained on nonlinear features, it was sufficient to use a linear Support Vector machine to construct the decision boundary. Linear Support Vector machines, however, are much faster in classification speed than nonlinear ones. This is due to the fact that for $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})$, the Support Vector decision function (2.25) can be expressed with a single weight vector $\mathbf{w} = \sum_{i=1}^{\ell} y_i \alpha_i \mathbf{x}_i$ as $f(\mathbf{x}) = \mathrm{sgn}((\mathbf{x} \cdot \mathbf{w}) + b)$. Thus the final stage of classification can be done extremely fast; the speed of the principal component extraction phase, on the other hand, and thus the accuracy-speed trade-off of the whole classifier, can be controlled by the number of components which we extract, or by the number m (cf. Eq. (3.20)).

Interpretability and Variable Selection. In PCA, it is sometimes desirable to be able to select specific axes which span the subspace into which one projects in doing principal component extraction. This way, it may for instance be possible to choose variables which are more accessible to interpretation. In the nonlinear case, there is an additional problem: some elements of F do not have pre-images in input space. To make this plausible, note that the linear span of the training examples mapped into feature space can have dimensionality up to M (the number of examples). If this exceeds the dimensionality of input space, it is rather unlikely that each vector of the form (3.10) has a pre-image (cf. Appendix D.1.2). To get interpretability, we thus need to find directions in input space (i.e. input variables) whose images under Φ span the PCA subspace in F. This can be done with an approach akin to the one described above: we could parametrize our set of desired input variables and run the minimization of (3.22) only over those parameters. The parameters can be e.g. group parameters which determine the amount of translation, say, starting from a set of images.

Dimensionality Reduction, Feature Extraction, and Reconstruction. Unlike linear PCA, the proposed method allows the extraction of a number of principal components which *can* exceed the input dimensionality. Suppose that the number of observations M exceeds the input dimensionality N. Linear PCA, even when it is based on the $M \times M$ dot product matrix, can find at most N nonzero Eigenvalues — they are identical to the nonzero Eigenvalues of the $N \times N$ covariance matrix. In contrast, kernel PCA can find up to M nonzero Eigenvalues — a fact that illustrates that it is impossible to perform kernel PCA directly on an $N \times N$ covariance matrix. Even more features could be extracted by using several kernels.

Being just a basis transformation, standard PCA allows the reconstruction of the original patterns $\mathbf{x}_i, i = 1, \ldots, \ell$, from a complete set of extracted principal components $(\mathbf{x}_i \cdot \mathbf{v}_j), j = 1, \ldots, \ell$, by expansion in the Eigenvector basis. Even from an incomplete set of components, good reconstruction is often possible. In kernel PCA, this is more difficult: we can reconstruct the image of a pattern in F from its nonlinear components; however, if we only have an approximate reconstruction, there is no guarantee that we can find an exact pre-image of the reconstruction in input space. In that case, we would have to resort to an approximation method (cf. (3.22)). Alternatively, we could use a suitable regression method for estimating the reconstruction mapping from the kernel-based principal components to the inputs.

Comparison to Other Methods for Nonlinear PCA. Starting from some of the properties characterizing PCA (see above), it is possible to develop a number of possible generalizations of linear PCA to the nonlinear case. Alternatively, one may choose an iterative algorithm which adaptively estimates principal components, and make some of its parts nonlinear to extract nonlinear features. Rather than giving a full review of this field here, we briefly describe just three approaches, and refer the reader to Diamantaras & Kung (1996) for more details.

Hebbian Networks. Initiated by the pioneering work of Oja (1982), a number of unsupervised neural-network type algorithms computing principal components have been proposed (e.g. Sanger, 1989). Compared to the standard approach of diagonalizing the covariance matrix, they have advantages for instance in cases where the data are nonstationary. Nonlinear variants of these algorithms are obtained by adding nonlinear activation functions. The algorithms then extract features that the authors have referred to as nonlinear principal components. These approaches, however, do not have the geometrical interpretation of kernel PCA as a standard PCA in a feature space nonlinearly related to input space, and it is thus more difficult to understand what exactly they are extracting. For a discussion of some approaches, see (Karhunen and Joutsensalo, 1995).

Autoassociative Multi-Layer Perceptrons. Consider a linear perceptron with one hidden layer, which is smaller than the input. If we train it to reproduce the input values as outputs (i.e. use it in autoassociative mode), then the hidden unit activations form a lower-dimensional representation of the data, closely related to PCA (see for instance Diamantaras & Kung, 1996). To generalize to a nonlinear setting, one uses nonlinear activation functions and additional layers.[4] While this of course can be considered a form of nonlinear PCA, it should be stressed that the resulting network training consists in solving a hard nonlinear optimization problem, with the possibility to get trapped in local minima, and thus with a dependence of the outcome on the starting point of the training. Moreover, in neural network implementations there is often a risk of getting overfitting. Another drawback of neural approaches to nonlinear PCA is that the number of components to be extracted has to be specified in advance. As an aside, note that hyperbolic tangent kernels can be used to extract neural network type nonlinear features using kernel PCA (Fig. 3.6). The principal components of a test point $\mathbf{x}$ in that case take the form (Fig. 3.2) $\sum_i \alpha_i^n \tanh(\kappa(\mathbf{x}_i, \mathbf{x}) + \Theta)$.

Principal Curves. An approach with a clear geometric interpretation in input space is the method of principal curves (Hastie & Stuetzle, 1989), which iteratively estimates a curve (or surface) capturing the structure of the data. The data are projected onto (i.e. mapped to the closest point on) a curve, and the algorithm tries to find a curve with the property that each point on the curve is the average of all data points projecting onto it. It can be shown that the only straight lines satisfying the latter are principal components, so principal curves are indeed a generalization of the latter. To compute principal curves, a nonlinear optimization problem has to be solved. The dimensionality of the surface, and thus the number of features to extract, is specified in advance. Some authors (e.g. Ritter, Martinetz, and Schulten, 1990) have discussed parallels between the Principal Curve algorithm and self-organizing feature maps (Kohonen, 1982) for dimensionality reduction.

[4]Simply using nonlinear activation functions in the hidden layer would not suffice: already the linear activation functions lead to the best approximation of the data (given the number of hidden nodes), so for the nonlinearities to have an effect on the components, the architecture needs to be changed (see e.g. Diamantaras & Kung, 1996).

Kernel PCA. Kernel PCA is a nonlinear generalization of PCA in the sense that (a) it is performing PCA in feature spaces of arbitrarily large (possibly infinite) dimensionality, and (b) if we use the kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})$, we recover original PCA. Compared to the above approaches, kernel PCA has the main advantage that no nonlinear optimization is involved — it is essentially linear algebra, as simple as standard PCA. In addition, we need not specify the number of components that we want to extract in advance. Compared to neural approaches, kernel PCA could be disadvantageous if we need to process a very large number of observations, as this results in a large matrix K. Compared to principal curves, kernel PCA is so far harder to interpret in input space; however, at least for polynomial kernels, it has a very clear interpretation in terms of higher-order features.

3.4 Feature Extraction Experiments

In this section, we present a set of experiments where we used kernel PCA (in the form given in Appendix D.2.2) to extract principal components. First, we shall take a look at a simple toy example; following that, we describe real-world experiments where we assess the utility of the extracted principal components by classification tasks.

Toy Examples. To provide some intuition on how PCA in F behaves in input space, we show a set of experiments with an artificial 2-D data set, using polynomial kernels (cf. Eq.(2.26)) of degree 1 through 4 (see Fig. 3.3). Linear PCA (on the left) only leads to 2 nonzero Eigenvalues, as the input dimensionality is 2. In contrast, nonlinear PCA allows the extraction of further components. In the figure, note that nonlinear PCA produces contour lines of constant feature value which reflect the structure in the data better than in linear PCA. In all cases, the first principal component varies monotonically along the parabola which underlies the data. In the nonlinear cases, also the second and the third components show behaviour which is similar for different polynomial degrees. The third component, which comes with small Eigenvalues (rescaled to sum to 1), seems to pick up the variance caused by the noise, as can be nicely seen in the case of degree 2. Dropping this component would thus amount to noise reduction.

In Fig. 3.3, it can be observed that for larger polynomial degrees, the principal component extraction functions become increasingly flat around the origin. Thus, different data points not too far from the origin would only differ slightly in the value of their principal components. To understand this, consider the following example: suppose we have two data points

$$\mathbf{x}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \mathbf{x}_2 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \tag{3.23}$$

and a kernel $k(\mathbf{x}, \mathbf{y}) := (\mathbf{x} \cdot \mathbf{y})^2$. Then the differences between the entries of $\mathbf{x}_1$ and

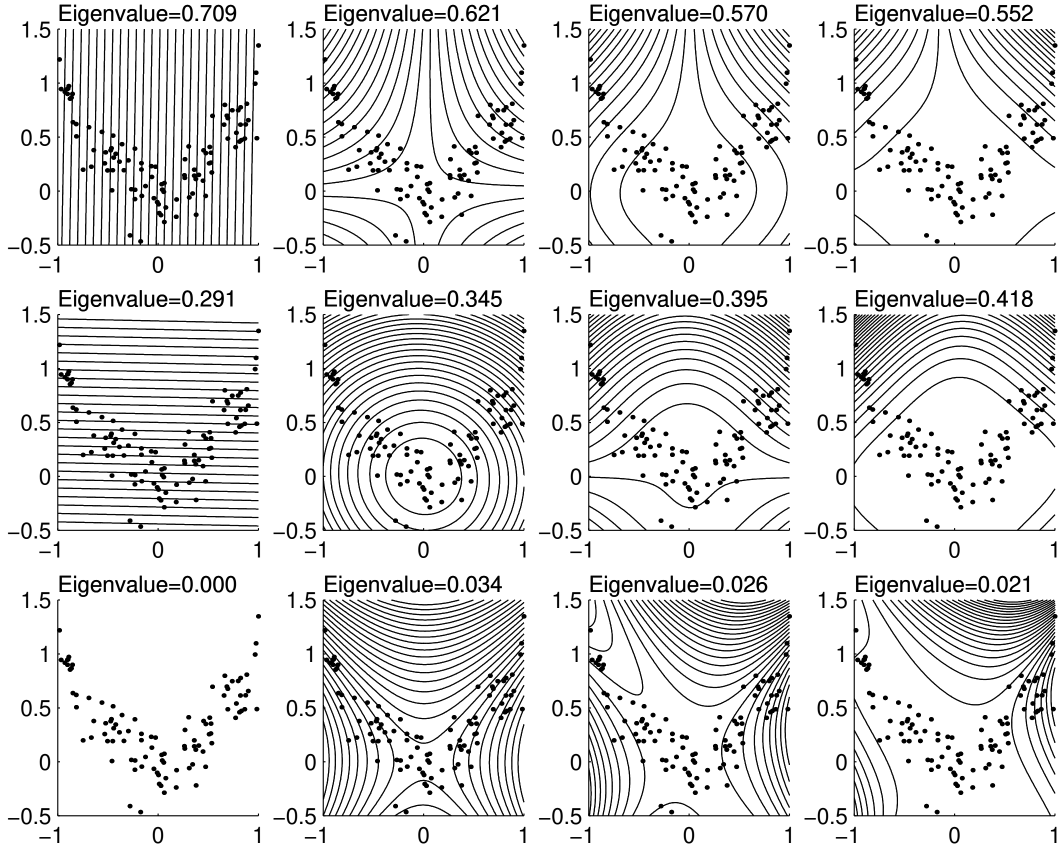

FIGURE 3.3: Two-dimensional toy example, with data generated in the following way: x-values have uniform distribution in $[-1, 1]$, y-values are generated from $y_i = x_i^2 + \xi$, were ξ is normal noise with standard deviation 0.2. From left to right, the polynomial degree in the kernel (2.26) increases from 1 to 4; from top to bottom, the first 3 Eigenvectors are shown, in order of decreasing Eigenvalue size. The figures contain lines of constant principal component value (contour lines); in the linear case, these are orthogonal to the Eigenvectors. We did not draw the Eigenvectors, as in the general case, they live in a higher-dimensional feature space.

$\mathbf{x}_2$ get scaled up by the kernel, namely $k(\mathbf{x}_1, \mathbf{x}_1) = 1$, but $k(\mathbf{x}_2, \mathbf{x}_2) = 16$. We can compensate for this by rescaling the individual entries of each vector $\mathbf{x}_i$ by

$$(\mathbf{x}_i)_k \mapsto \operatorname{sign}((\mathbf{x}_i)_k) \cdot |(\mathbf{x}_i)_k|^{\frac{1}{2}}. \tag{3.24}$$

Indeed, Fig. 3.4 shows that when the data are preprocessed according to (3.24) (where higher degrees are treated correspondingly), the first principal component extractors do hardly depend on the degree anymore, as long as it is larger than 1. If necessary,

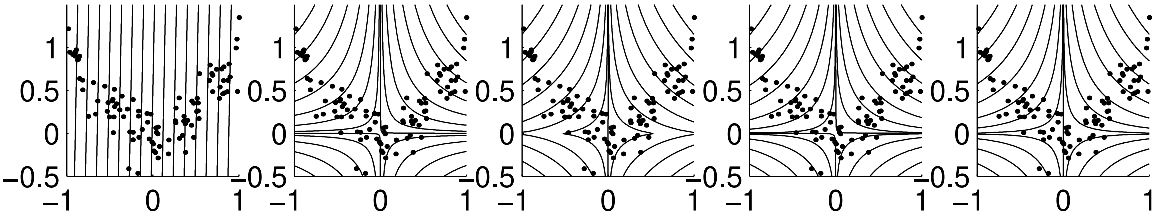

FIGURE 3.4: PCA with kernel (2.26), degrees $d = 1, \ldots, 5$. 100 points $((x_i)_1, (x_i)_2)$ were generated from $(x_i)_2 = (x_i)_1^2+$ noise (Gaussian, with standard deviation 0.2); all $(x_i)_j$ were rescaled according to $(x_i)_j \mapsto \text{sgn}((x_i)_j) \cdot |(x_i)_j|^{1/d}$. Displayed are contour lines of constant value of the first principal component. Nonlinear kernels ($d > 1$) extract features which nicely increase along the direction of main variance in the data; linear PCA ($d = 1$) does its best in that respect, too, but it is limited to straight directions.

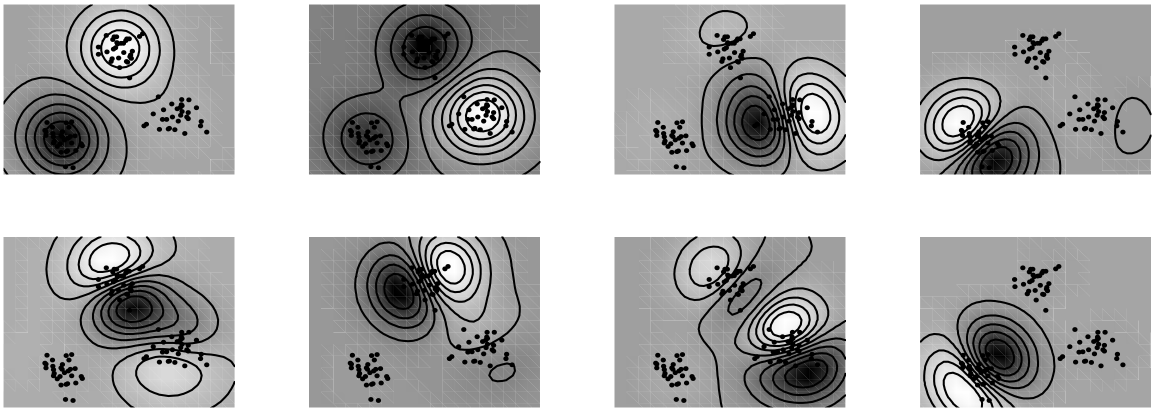

FIGURE 3.5: Two-dimensional toy example with three data clusters (Gaussians with standard deviation 0.1, depicted region: $[-1, 1] \times [-0.5, 1]$): first 8 nonlinear principal components extracted with $k(\mathbf{x}, \mathbf{y}) = \exp\left(-\|\mathbf{x} - \mathbf{y}\|^2/0.1\right)$. Note that the first 2 principal component (top left) nicely separate the three clusters. The components 3 – 5 split up the clusters into halves. Similarly, the components 6 – 8 split them again, in a way orthogonal to the above splits. Thus, the first 8 components divide the data into 12 regions.

we can thus use (3.24) to preprocess our data. Note, however, that the above scaling problem is irrelevant for the character and object databases to be considered below: there, most entries of the patterns are ± 1.

Further toy examples, using radial basis function kernels (1.28) and neural network type sigmoid kernels (1.29), are shown in figures 3.5 – 3.8.

Object Recognition. In this set of experiments, we used the MPI chair database with 89 training views per object (Appendix A). We computed the matrix K from all

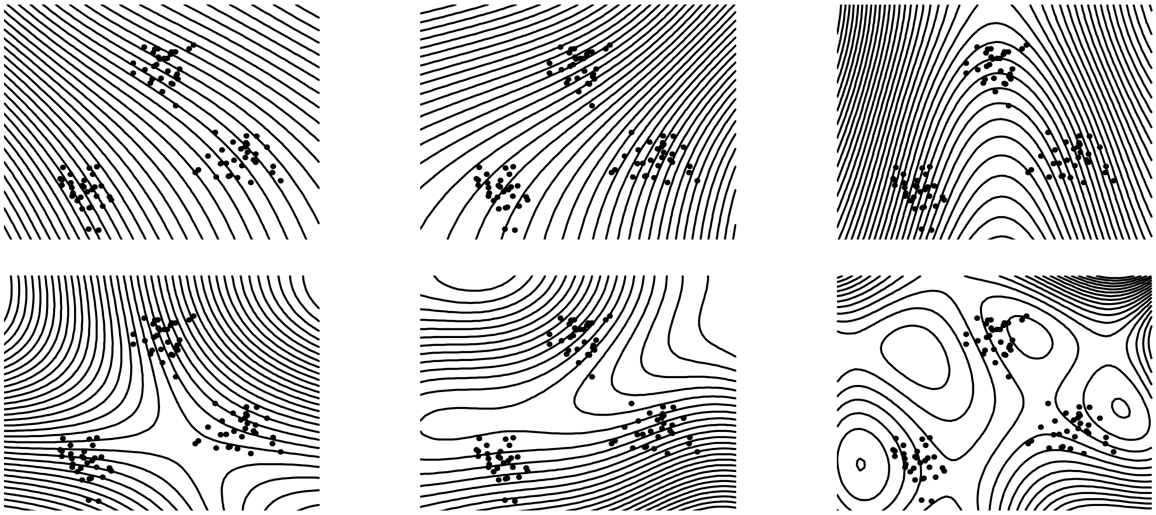

FIGURE 3.6: Two-dimensional toy example with three data clusters (Gaussians with standard deviation 0.1, depicted region: $[-1, 1] \times [-0.5, 1]$): first 6 nonlinear principal components extracted with $k(\mathbf{x}, \mathbf{y}) = \tanh(2(\mathbf{x} \cdot \mathbf{y}) - 1)$ (the gain and threshold values were chosen according to the values used in SV machines, cf. Table 2.4). Note that the first 2 principal components are sufficient to separate the three clusters, and the third and fourth component simultaneously split all clusters into halves.

2225 training examples, and used polynomial kernel PCA to extract nonlinear principal components from the training and test set. To assess the utility of the components, we trained a soft margin hyperplane classifier (Sec. 2.1.3) on the classification task. This is a special case of Support Vector machines, using the standard dot product as a kernel function. Table 3.1 summarizes our findings: in all cases, nonlinear components as extracted by polynomial kernels (Eq. (2.26) with $d > 1$) led to classification accuracies superior to standard PCA. Specifically, the nonlinear components afforded top test performances between 2% and 4% error, whereas in the linear case we obtained 17%.

Character Recognition. To validate the above results on a widely used pattern recognition benchmark database, we repeated the same experiments on the US postal service database of handwritten digits (Appendix C). This database contains 9298 examples of dimensionality 256; 2007 of them make up the test set. For computational reasons, we decided to use a subset of 3000 training examples for the matrix K. Table 3.2 illustrates two advantages of using nonlinear kernels: first, performance of a linear classifier trained on nonlinear principal components is better than for the same number of linear components; second, the performance for nonlinear components can be further improved by using more components than possible in the linear case. The latter is related to the fact that of course there are many more higher-order features than there are pixels in an image. Regarding the first point, note that extracting a certain number of features in a 10^{10}-dimensional space constitutes a much higher reduc-

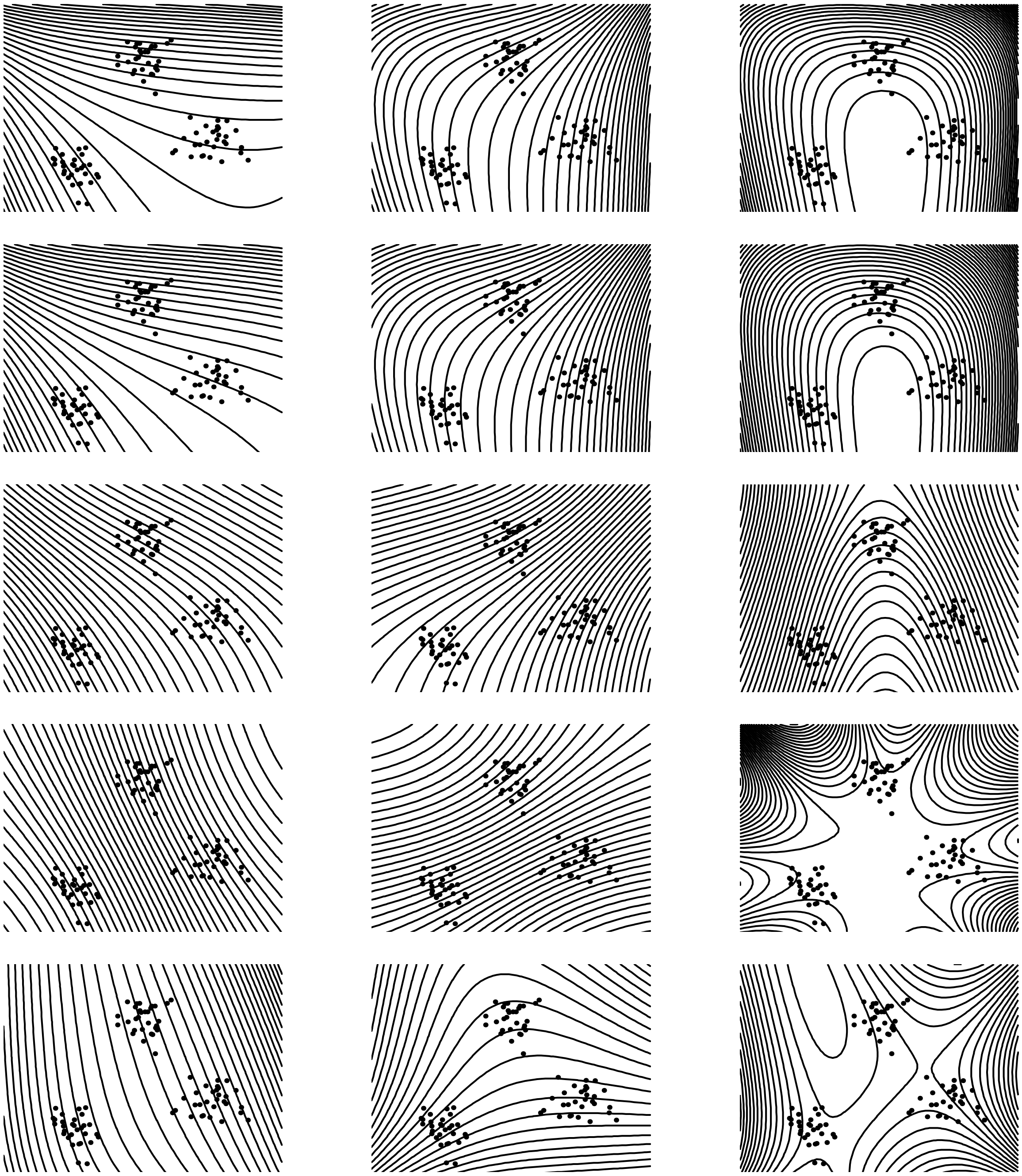

FIGURE 3.7: For different threshold values Θ (from top to bottom: $\Theta = -4, -2, -1, 0, 2$), kernel PCA with hyperbolic tangent kernels $k(\mathbf{x}, \mathbf{y}) = \tanh(2(\mathbf{x} \cdot \mathbf{y}) + \Theta)$ exhibits qualitatively similar behaviour (data as in the previous figures). In all cases, the first two components capture the main structure of the data, whereas the third components split the clusters.

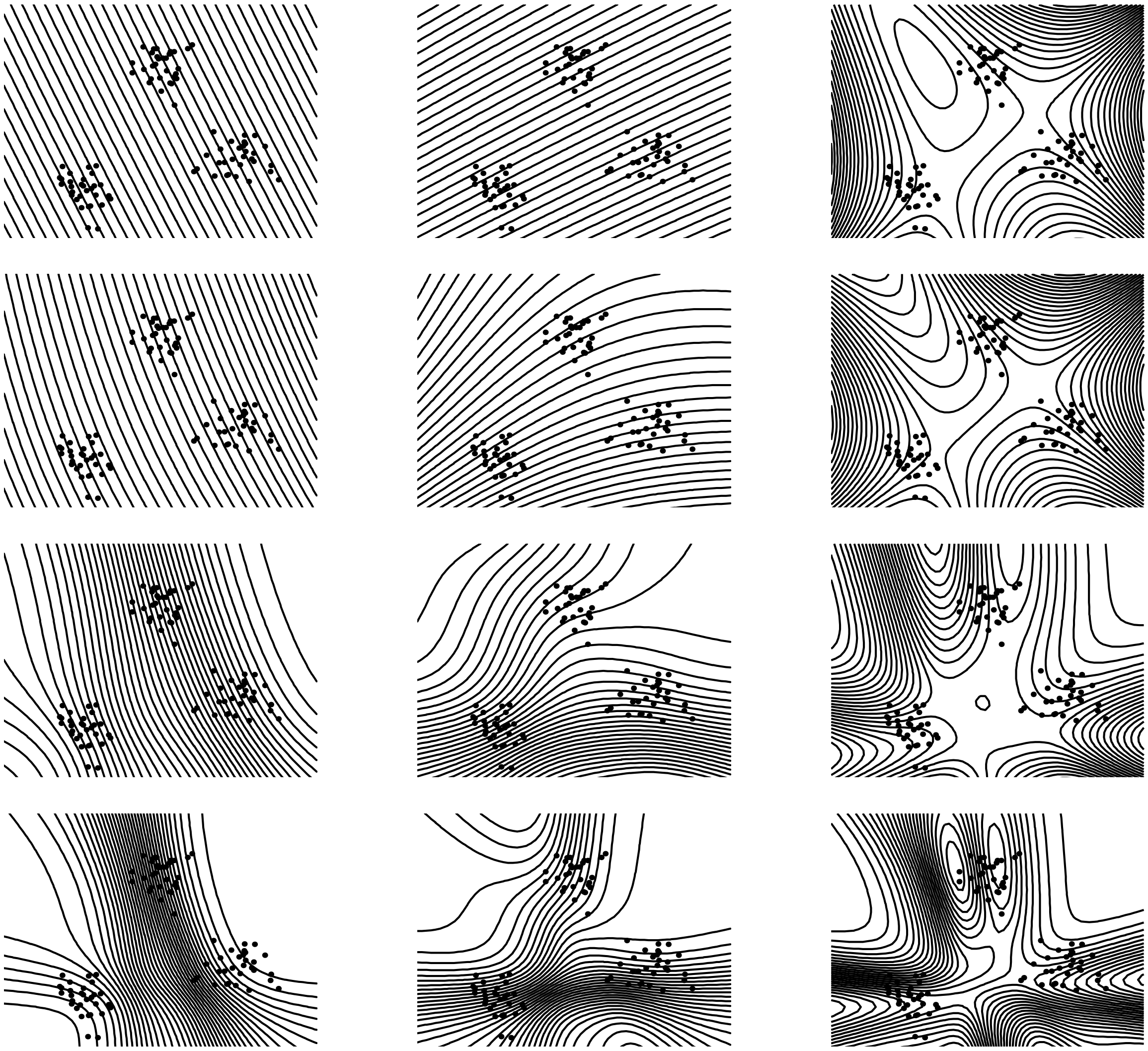

FIGURE 3.8: A smooth transition from linear PCA to nonlinear PCA is obtained by using hyperbolic tangent kernels $k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + 1)$ with varying gain κ: from top to bottom, $\kappa = 0.1, 1, 5, 10$ (data as in the previous figures). For $\kappa = 0.1$, the first two features look like linear PCA features. For large κ, the nonlinear region of the tanh function becomes effective. In that case, kernel PCA can exploit this nonlinearity to allocate the highest feature gradients to regions where there are data points, as can be seen nicely in the case $\kappa = 10$.

# of components	Test Error Rate for degree						
	1	2	3	4	5	6	7
64	23.0	21.0	17.6	16.8	16.5	16.7	16.6
128	17.6	9.9	7.9	7.1	6.2	6.0	5.8
256	16.8	6.0	4.4	3.8	3.4	3.2	3.3
512	n.a.	4.4	3.6	3.9	2.8	2.8	2.6
1024	n.a.	4.1	3.0	2.8	2.6	2.6	2.4
2048	n.a.	4.1	2.9	2.6	2.5	2.4	2.2

TABLE 3.1: Test error rates on the MPI chair database for linear Support Vector machines trained on nonlinear principal components extracted by PCA with polynomial kernel (2.26), for degrees 1 through 7. In the case of degree 1, we are doing standard PCA, with the number of nonzero Eigenvalues being at most the dimensionality of the space, 256; thus, we can extract at most 256 principal components. The performance for the nonlinear cases (degree > 1) is significantly better than for the linear case, illustrating the utility of the extracted nonlinear components for classification.

# of components	Test Error Rate for degree						
	1	2	3	4	5	6	7
32	9.6	8.8	8.1	8.5	9.1	9.3	10.8
64	8.8	7.3	6.8	6.7	6.7	7.2	7.5
128	8.6	5.8	5.9	6.1	5.8	6.0	6.8
256	8.7	5.5	5.3	5.2	5.2	5.4	5.4
512	n.a.	4.9	4.6	4.4	5.1	4.6	4.9
1024	n.a.	4.9	4.3	4.4	4.6	4.8	4.6
2048	n.a.	4.9	4.2	4.1	4.0	4.3	4.4

TABLE 3.2: Test error rates on the USPS handwritten digit database for linear Support Vector machines trained on nonlinear principal components extracted by PCA with kernel (2.26), for degrees 1 through 7. In the case of degree 1, we are doing standard PCA, with the number of nonzero Eigenvalues being at most the dimensionality of the space, 256. Clearly, nonlinear principal components afford test error rates which are superior to the linear case (degree 1).

tion of dimensionality than extracting the same number of features in 256-dimensional input space.

For all numbers of features, the optimal degree of kernels to use is around 4, which is compatible with Support Vector machine results on the same data set (cf. Sec. 2.3 and Fig. 2.16). Moreover, with only one exception, the nonlinear features are superior to their linear counterparts. The resulting error rate for the best of our classifiers (4.0%) is competitive with convolutional 5-layer neural networks (5.0% were reported by LeCun et al., 1989) and nonlinear Support Vector classifiers (4.0%, Table 2.4); it is much better than linear classifiers operating directly on the image data (a linear Support Vector machine achieves 8.9%; Table 2.4). Our results were obtained without using any prior knowledge about symmetries of the problem at hand, explaining why the performance is inferior to Virtual Support Vector classifiers (3.2%, Table 4.1), and Tangent Distance Nearest Neighbour classifiers (2.6%, Simard, LeCun, & Denker, 1993). We believe that adding e.g. local translation invariance, be it by generating "virtual" translated examples (cf. Sec. 4.2.1) or by choosing a suitable kernel (e.g. as the ones that we shall describe in Sec. 4.3), could further improve the results.

3.5 Discussion

Feature Extraction for Classification. This chapter was devoted to the presentation of a new technique for nonlinear PCA. To develop this technique, we made use of a kernel method so far only used in supervised learning (Vapnik, 1995; Sec. 1.3). Kernel PCA constitutes a mere first step towards exploiting this technique for a large class of algorithms.

In experiments comparing the utility of kernel PCA features for pattern recognition using a linear classifier, we found two advantages of nonlinear kernels: first, nonlinear principal components afforded better recognition rates than corresponding numbers of linear principal components; and second, the performance for nonlinear components can be further improved by using more components than possible in the linear case. We have not yet compared kernel PCA to other techniques for nonlinear feature extraction and dimensionality reduction. We can, however, compare results to other feature extraction methods which have been used in the past by researchers working on the USPS classification problem (cf. Sec. 3.4). Our system of kernel PCA feature extraction plus linear support vector machine for instance performed better than LeNet1 (LeCun et al., 1989). Even though the latter result has been obtained a number of years ago, it should be stressed that LeNet1 provides an architecture which contains a great deal of prior information about the handwritten character classification problem. It uses shared weights to improve transformation invariance, and a hierarchy of feature detectors resembling parts of the human visual system. These feature detectors were for instance used by Bottou and Vapnik (1992) as a preprocessing stage in their experiments in local learning. Note that, in addition, our features were extracted without taking into account that we want to do classification. Clearly, in supervised learning, where we are given a set of labelled observations $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)$, it would seem advisable to make use of the labels not only during the training of the final classifier, but already in the stage of feature extraction.

To conclude this paragraph on feature extraction for classification, we note that a similar approach could be taken in the case of regression estimation.

Feature Space and the Curse of Dimensionality. We are doing PCA in 10^{10}-dimensional feature spaces, yet getting results in finite time which are comparable to state-of-the-art techniques. In fact, of course, we are not working in the full feature space, but just in a comparably small linear subspace of it, whose dimension equals at most the number of observations. The method automatically chooses this subspace and provides a means of taking advantage of the lower dimensionality — an approach which consisted in explicitely mapping into feature space and then performing PCA would have severe difficulties at this point: even if PCA was done based on an $M \times M$ dot product matrix (M being the sample size) whose diagonalization is tractable, it would still be necessary to evaluate dot products in a 10^{10}-dimensional feature space to compute the *entries* of the matrix in the first place. Kernel-based methods avoid this problem — they do not explicitely compute all dimensions of F (loosely speaking, all possible features), but only work in a relevant subspace of F.

Note, moreover, that we did not get overfitting problems when training the linear SV classifier on the extracted features. The basic idea behind this two-step approach is very similar in spirit to nonlinear SV machines: one maps into a very complex space to be able to approximate a large class of possible decision functions, and then uses a low VC-dimension classifier in that space to control generalization.

Conclusion. Compared to other techniques for nonlinear feature extraction, kernel PCA has the advantages that it does not require nonlinear optimization, but only the solution of an Eigenvalue problem, and by the possibility to use different kernels, it comprises a fairly general class of nonlinearities that can be used. Clearly, the last point has yet to be evaluated in practice, however, for the support vector machine, the utility of different kernels has already been established. Different kernels (polynomial, sigmoid, Gaussian) led to fine classification performances (Table 2.4). The general question of how to select the ideal kernel for a given task (i.e. the appropriate feature space), however, is an open problem.

We conclude this chapter with a twofold outlook. The scene has been set for using the kernel method to construct a wide variety of rather general and still feasible nonlinear variants of classical algorithms. It is beyond the scope of the present work to explore all the possibilities, including many distance-based algorithms, in detail. Some of them are currently being investigated, for instance nonlinear forms of k-means clustering and kernel-based independent component analysis (Schölkopf, Smola, & Müller, 1996). Other domains where researchers have recently started to investigate the use of Mercer kernels include Gaussian Processes (Williams, 1997).

Linear PCA is being used in numerous technical and scientific applications, including noise reduction, density estimation, image indexing and retrieval systems, and the analysis of natural image statistics. Kernel PCA can be applied to all domains where traditional PCA has so far been used for feature extraction, and where a nonlinear extension would make sense.

Chapter 4

Prior Knowledge in Support Vector Machines

In 1995, LeCun et al. published a pattern recognition performance comparison noting the following:

> *"The optimal margin classifier [i.e. SV machine, the author] has excellent accuracy, which is most remarkable, because unlike the other high performance classifiers, it does not include* ***a priori*** *knowledge about the problem. In fact, this classifier would do just as well if the image pixels were permuted by a fixed mapping. [...] However, improvements are expected as the technique is relatively new."*

One of the key points in developing SV technology is thus the incorporation of prior knowledge about given tasks. Moreover, it is also a key point if we want to learn anything general about the processing of visual information in animals from SV machines: having been exposed to the world for all their life, animals extensively exploit any available knowledge on regularities and invariances of the world.

Two years after the above statement was published, we are now in the position to be able to devote the present chapter to three techniques for incorporating task-specific prior knowledge in SV machines (Schölkopf, Burges, and Vapnik, 1996a; Schölkopf, Simard, Smola, and Vapnik, 1997a).

4.1 Introduction

When we are trying to extract regularities from data, we often have additional knowledge about functions that we estimate (for a review, see Abu-Mostafa, 1995). For instance, in image classification tasks, there exist transformations which leave class membership *invariant* (e.g. translations); moreover, it is usually the case that images have a *local* structure in the sense that not all correlations between image regions carry equal amounts of information. We presently investigate the question how to make use of these two sources of knowledge.

We first present the Virtual SV method of incorporating prior knowledge about transformation invariances by applying transformations to Support Vectors, the training examples most critical for determining the classification boundary (Sec. 4.2.1).

In Sec. 4.2.2, we design kernel functions which lead to invariant classification hyperplanes. This method is applicable to invariances under the action of differentiable local 1-parameter groups of local transformations, e.g. translational invariance in pattern recognition; the Virtual SV method is applicable to any type of invariance. In the third method proposed in this chapter, we also modify the kernel functions; however, this time not to incorporate transformation invariance, but to take into account image locality by using localized receptive fields (Sec. 4.3). Following that, Sec. 4.4 and Sec. 4.5 give experimental results and a discussion, respectively.

4.2 Incorporating Transformation Invariances

In many applications of learning procedures, certain transformations of the input are known to leave function values unchanged. At least three different ways of exploiting this knowledge have been used (illustrated in Fig. 4.1):

In the first case, the knowledge is used to generate artificial training examples ("virtual examples", Poggio and Vetter, 1992; Baird, 1990) by transforming the training examples accordingly. It is then hoped that given sufficient time, the learning machine will extract the invariances from the artificially enlarged training data.

In the second case, the learning algorithm itself is modified. This is typically done by using a modified error function which forces a learning machine to construct a function with the desired invariances (Simard et al., 1992).

Finally, in the third case, the invariance is achieved by changing the representation of the data by first mapping them into a more suitable space; an approach pursued for instance by Segman, Rubinstein, and Zeevi (1992), or Vetter and Poggio (1997). The data representation can also be changed by using a modified distance metric, rather than actually changing the patterns (e.g. Simard, LeCun, and Denker, 1993).

Simard et al. (1992) compare the first two techniques and find that for the considered problem — learning a function with three plateaus where function values are locally invariant — training on the artificially enlarged data set is significantly slower, due to both correlations in the artificial data and the increase in training set size. Moving to real-world applications, the latter factor becomes even more important. If the size of a training set is multiplied by a number of desired invariances (by generating a corresponding number of artificial examples for each training pattern), the resulting training sets can get rather large (as the ones used by Drucker, Schapire, and Simard, 1993). However, the method of generating virtual examples has the advantage of being readily implemented for all kinds of learning machines and symmetries. If instead of Lie groups of symmetry transformations one is dealing with discrete symmetries, as the bilateral symmetries of Vetter, Poggio, and Bülthoff (1994); Vetter and Poggio (1994), derivative-based methods such as the ones of Simard et al. (1992) are not applicable. It would thus be desirable to have an intermediate method which has the advantages of the virtual examples approach without its computational cost.

The two methods described in the following try to combine merits of all the approaches mentioned above. The Virtual SV method (Sec. 4.2.1) retains the flexibility

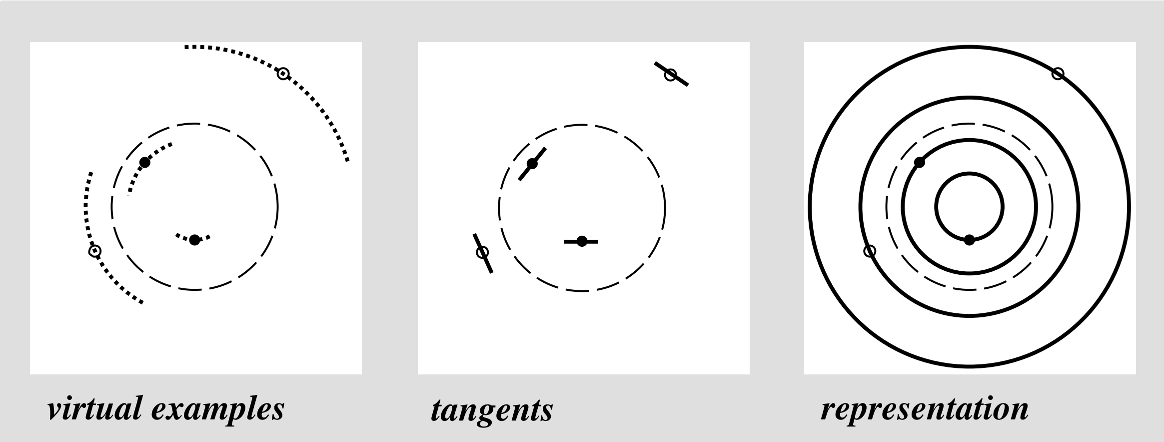

FIGURE 4.1: Different ways of incorporating invariances in a decision function. The dashed line marks the "true" boundary, disks and circle are the training examples. We assume that prior information tells us that the classification function only depends on the norm of the input vector (the origin being in the center of each picture). Lines crossing an example indicate the type of information conveyed by the different methods of incorporating prior information. *Left:* generating virtual examples in a localized region around each training example; *middle:* incorporating a regularizer to learn tangent values (cf. Simard, Victorri, LeCun, and Denker, 1992); *right:* changing the representation of the data by first mapping each example to its norm. If feasible, the latter method yields the most information. However, if the necessary nonlinear transformation cannot be found, or if the desired invariances are of localized nature, one has to resort to one of the former techniques. Finally, the reader may note that examples close to the boundary allow us to exploit prior knowledge very effectively: given a method to get a first approximation of the true boundary, the examples closest to it would allow good estimation of the true boundary. A similar two-step approach is pursued in Sec. 4.2.1. (From Schölkopf, Burges, and Vapnik (1996a).)

and simplicity of virtual examples approaches, while cutting down on their computational cost significantly. The Invariant Hyperplane method (Sec. 4.2.2), on the other hand, is comparable to the method of Simard et al. (1992) in that it is applicable for all differentiable local 1-parameter groups of local symmetry transformations, comprising a fairly general class of invariances. In addition, it has an equivalent interpretation as a preprocessing operation applied to the data before learning. In this sense, it can also be viewed as changing the representation of the data to a more invariant one, in a task-dependent way.

4.2.1 The Virtual SV Method

In Sec. 2.4, it has been argued that the SV set contains all information necessary to solve a given classification task. It particular, it was possible to train any one of three different types of SV machines solely on the Support Vector set extracted by another machine, with a test performance not worse than after training on the full database. Using this finding as a starting point, we now investigate the question whether it might be sufficient to generate virtual examples from the Support Vectors only. After all, one might hope that it does not add much information to generate virtual examples of patterns which are not close to the boundary. In high-dimensional cases, however, care has to be exercised regarding the validity of this intuitive picture. Thus, an experimental test on a high-dimensional real-world problem is imperative. In our experiments, we proceeded as follows (cf. Fig. 4.2):

1. Train a Support Vector machine to extract the Support Vector set.
2. Generate artificial examples by applying the desired invariance transformations to the Support Vectors. In the following, we will refer to these examples as *Virtual Support Vectors (VSVs).*
3. Train another Support Vector machine on the generated examples.[1]

If the desired invariances are incorporated, the curves obtained by applying Lie symmetry transformations to points on the decision surface should have tangents parallel to the latter (cf. Simard et al., 1992). If we use small small Lie group transformations to generate the virtual examples, this implies that the Virtual Support Vectors should be approximately as close to the decision surface as the original Support Vectors. Hence, they are fairly likely to become Support Vectors after the second training run. Vice versa, if a substantial fraction of the Virtual Support Vectors turn out to become support vectors in the second run, we have reason to expect that the decision surface does have the desired shape.

4.2.2 Constructing Invariance Kernels

Invariance by a Self-Consistency Argument. We face the following problem: to express the condition of invariance of the decision function, we already need to know its coefficients which are found only during the optimization, which in turn should already take into account the desired invariances. As a way out of this circle, we use the following ansatz: consider decision functions $f = \mathrm{sgn} \circ g$, where g is defined as

$$g(\mathbf{x}_j) := \sum_{i=1}^{\ell} \alpha_i y_i (B\mathbf{x}_j \cdot B\mathbf{x}_i) + b, \tag{4.1}$$

with a matrix B to be determined below. This follows Vapnik (1995b), who suggested to incorporate invariances by modifying the dot product used: any nonsingular B defines a dot product, which can equivalently be written in the form $(\mathbf{x}_j \cdot A\mathbf{x}_i)$, with a positive definite matrix $A = B^\top B$.

[1]Clearly, the scheme can be iterated; however, care has to exercised, since the iteration of local invariances would lead to global ones which are not always desirable — cf. the example of a '6' rotating into a '9' (Simard, LeCun, and Denker, 1993).

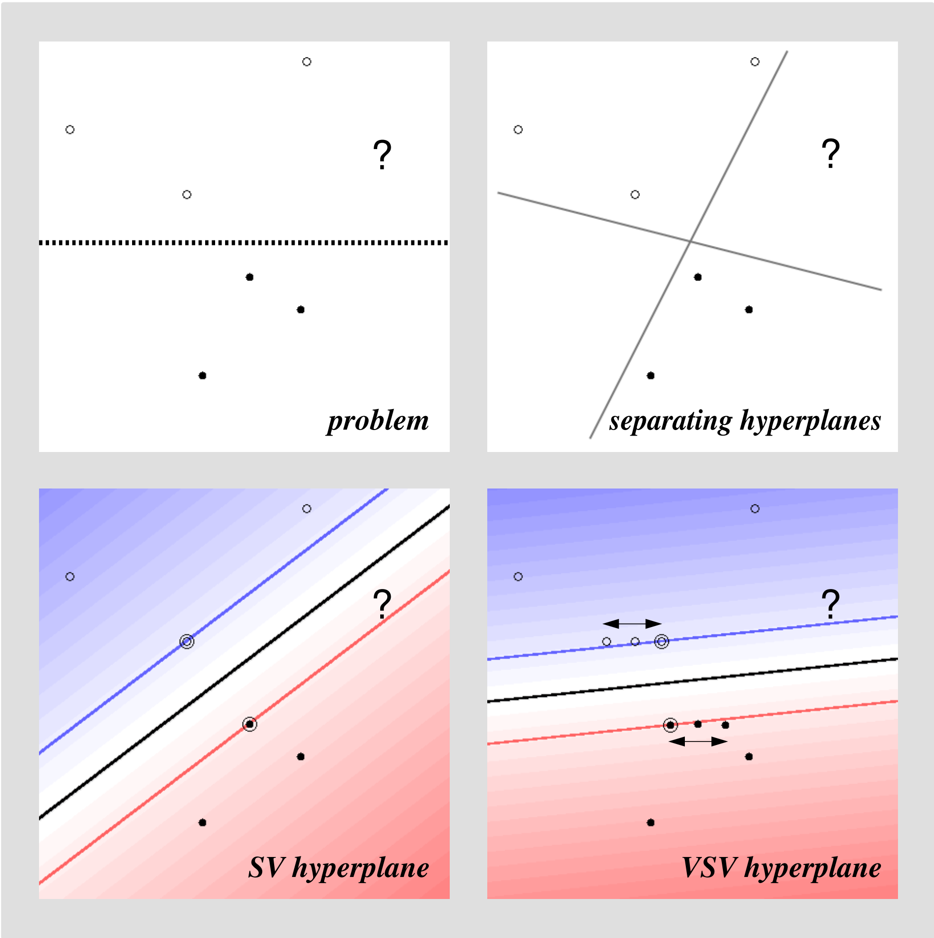

FIGURE 4.2: Suppose we have prior knowledge indicating that the decision function should be invariant with respect to horizontal translations. The true decision boundary is drawn as a dotted line (*top left*); however, as we are just given a limited training sample, different separating hyperplanes are conveivable (*top right*). The SV algorithm finds the unique separating hyperplane with maximal margin (*bottom left*, which in this case is quite different from the true boundary. For instance, it would lead to wrong classification of the ambiguous point indicated by the question mark. Making use of the prior knowledge by generating Virtual Support Vectors from the Support Vectors found in a first tranining run, and retraining on these, yields a more accurate decision boundary (*bottom right*). Note, moreover, that for the considered example, it is sufficient to train the SV machine *only* on virtual examples generated from the Support Vectors.

Clearly, invariance of g under local transformations of all $\mathbf{x}_j$ is a *sufficient* condition for the same invariance to hold for $f = \text{sgn} \circ g$, which is what we are aiming for. Strictly speaking, however, invariance of g is not *necessary* at points which are not Support Vectors, since these lie in a region where $(\text{sgn} \circ g)$ is constant.

The above notion of invariance refers to invariance when evaluating the decision function. A different notion could ask the question whether the separating hyperplane, including its margin, would change if the training examples were transformed. It turns out that when discussing the invariance of g rather than f, these two concepts are closely related. In the following argument, we restrict ourselves to the optimal margin case ($\xi_i = 0$ for all $i = 1, \ldots, \ell$), where the margin is well-defined. As the separating hyperplane and its margin are expressed in terms of Support Vectors, *locally* transforming a Support Vector $\mathbf{x}_i$ will change the hyperplane or the margin if $g(\mathbf{x}_i)$ changes: if $|g|$ gets smaller than 1, the transformed pattern will lie in the margin, and the recomputed margin will be smaller; if $|g|$ gets larger than 1, the margin might become bigger, depending on whether the pattern can be expressed in terms of the other SVs (cf. the remark in point 2 of the enumeration preceding Proposition 2.1.2). In terms of the mechanical analogy of Sec. 2.1.2: moving Support Vectors changes the mechanical equilibrium for the sheet separating the classes. Conversely, a *local* transformation of a non-Support Vector will never change f, even if the value of g changes, as the solution of the programming problem is expressed in terms of the Support Vectors only.

In this sense, invariance of f under local transformations of the given data corresponds to invariance of (4.1) for the Support Vectors. Note, however, that this criterion is not readily applicable: before finding the Support Vectors in the optimization, we already need to know how to enforce invariance. Thus the above observation cannot be used directly, however it could serve as a starting point for constructing heuristics or iterative solutions. In the Virtual SV method (Sec. 4.2.1), a first run of the standard SV algorithm is carried out to obtain an initial SV set; similar heuristics could be applied in the present case.

Local invariance of g for each pattern $\mathbf{x}_j$ under transformations of a differentiable local 1-parameter group of local transformations $\mathcal{L}_t$,

$$\frac{\partial}{\partial t}\Big|_{t=0} g(\mathcal{L}_t \mathbf{x}_j) = 0, \tag{4.2}$$

can be approximately enforced by minimizing the regularizer

$$\frac{1}{\ell} \sum_{j=1}^{\ell} \left(\frac{\partial}{\partial t}\Big|_{t=0} g(\mathcal{L}_t \mathbf{x}_j) \right)^2 . \tag{4.3}$$

Note that the sum may run over labelled as well as unlabelled data, so in principle one could also require the decision function to be invariant with respect to transformations of elements of a *test* set. Moreover, we could use different transformations for different patterns.

For (4.1), the local invariance term (4.2) becomes

$$\begin{aligned} \frac{\partial}{\partial t}\Big|_{t=0} g(\mathcal{L}_t \mathbf{x}_j) &= \frac{\partial}{\partial t}\Big|_{t=0} \left(\sum_{i=1}^{\ell} \alpha_i y_i (B\mathcal{L}_t \mathbf{x}_j \cdot B\mathbf{x}_i) + b \right) \\ &= \sum_{i=1}^{\ell} \alpha_i y_i \frac{\partial}{\partial t}\Big|_{t=0} (B\mathcal{L}_t \mathbf{x}_j \cdot B\mathbf{x}_i) \end{aligned}$$

$$= \sum_{i=1}^{\ell} \alpha_i y_i \partial_1 (B\mathcal{L}_0 \mathbf{x}_j \cdot B\mathbf{x}_i) \cdot B \frac{\partial}{\partial t}\Big|_{t=0} \mathcal{L}_t \mathbf{x}_j, \tag{4.4}$$

using the chain rule. Here, $\partial_1(B\mathcal{L}_0\mathbf{x}_j \cdot B\mathbf{x}_i)$ denotes the gradient of $(\mathbf{x} \cdot \mathbf{y})$ with respect to $\mathbf{x}$, evaluated at the point $(\mathbf{x} \cdot \mathbf{y}) = (B\mathcal{L}_0\mathbf{x}_j \cdot B\mathbf{x}_i)$.

As a side remark, note that a sufficient, albeit rather strict condition for invariance is thus that $\frac{\partial}{\partial t}\big|_{t=0}(B\mathcal{L}_t\mathbf{x}_j \cdot B\mathbf{x}_i)$ vanish for all i, j; however, we will proceed in our derivation, with the goal to impose weaker conditions, which apply for one specific decision function rather than simultaneously for all decision functions expressible by different choices of the coefficients $\alpha_i y_i$.

Substituting (4.4) into (4.3), using the facts that $\mathcal{L}_0 = I$ and $\partial_1(\mathbf{x}, \mathbf{y}) = \mathbf{y}^\top$, yields the regularizer

$$\begin{aligned}
& \frac{1}{\ell} \sum_{j=1}^{\ell} \left(\sum_{i=1}^{\ell} \alpha_i y_i (B\mathbf{x}_i)^\top B \frac{\partial}{\partial t}\Big|_{t=0} \mathcal{L}_t \mathbf{x}_j \right)^2 \\
= \; & \frac{1}{\ell} \sum_{j=1}^{\ell} \left(\sum_{i=1}^{\ell} \alpha_i y_i (B\mathbf{x}_i)^\top B \frac{\partial}{\partial t}\Big|_{t=0} \mathcal{L}_t \mathbf{x}_j \right) \left(\sum_{k=1}^{\ell} \alpha_k y_k (B \frac{\partial}{\partial t}\Big|_{t=0} \mathcal{L}_t \mathbf{x}_j)^\top (B\mathbf{x}_k) \right) \\
= \; & \sum_{i,k=1}^{\ell} \alpha_i y_i \alpha_k y_k (B\mathbf{x}_i \cdot BCB^\top B\mathbf{x}_k)
\end{aligned} \tag{4.5}$$

where

$$C := \frac{1}{\ell} \sum_{j=1}^{\ell} \left(\frac{\partial}{\partial t}\Big|_{t=0} \mathcal{L}_t \mathbf{x}_j \right) \left(\frac{\partial}{\partial t}\Big|_{t=0} \mathcal{L}_t \mathbf{x}_j \right)^\top. \tag{4.6}$$

We now choose B such that (4.5) reduces to the standard SV target function (2.7) in the form obtained by substituting (2.11) into it (cf. the quadratic term of (2.13)), utilizing the dot product chosen in (4.1), i.e. such that

$$(B\mathbf{x}_i \cdot BCB^\top B\mathbf{x}_k) = (B\mathbf{x}_i \cdot B\mathbf{x}_k). \tag{4.7}$$

Assuming that the $\mathbf{x}_i$ span the whole space, this condition becomes

$$B^\top BCB^\top B = B^\top B, \tag{4.8}$$

or, by requiring B to be nonsingular, i.e. that no information get lost during the preprocessing, $BCB^\top = I$. This can be satisfied by a preprocessing matrix

$$B = C^{-\frac{1}{2}}, \tag{4.9}$$

the nonnegative square root of the inverse of the nonnegative matrix C defined in (4.6). In practice, we use a matrix

$$C_\lambda := (1 - \lambda)C + \lambda I, \tag{4.10}$$

with $0 < \lambda \leq 1$, instead of C. As C is nonnegative, C_λ is invertible. For $\lambda = 1$, we recover the standard SV optimal hyperplane algorithm, other values of λ determine the trade-off between invariance and model complexity control. It can be shown that using C_λ corresponds to using an objective function $\tau(\mathbf{w}) = (1-\lambda)\sum_i(\mathbf{w} \cdot \frac{\partial}{\partial t}|_{t=0}\mathcal{L}_t\mathbf{x}_i)^2 + \lambda\|\mathbf{w}\|^2$ (see Appendix D.3).

By choosing the preprocessing matrix B according to (4.9), we have obtained a formulation of the problem where the standard SV quadratic optimization technique does in effect minimize the tangent regularizer (4.3): the maximum of (2.13) subject to (2.14) and (2.15), using the modified dot product as in (4.1), coincides with the minimum of (4.3) subject to the separation conditions $y_i \cdot g(\mathbf{x}_i) \geq 1$, where g is defined as in (4.1).

Note that preprocessing with B does not affect classification speed: since $(B\mathbf{x}_j \cdot B\mathbf{x}_i) = (\mathbf{x}_j \cdot B^\top B\mathbf{x}_i)$, we can precompute $B^\top B\mathbf{x}_i$ for all SVs $\mathbf{x}_i$ and thus obtain a machine (with modified SVs) which is as fast as a standard SV machine (cf. (4.1)).

In the nonlinear case, where kernel functions $k(\mathbf{x}, \mathbf{y})$ are substituted for every occurence of a dot product, the above analysis of transformation invariance leads to the regularizer

$$\frac{1}{\ell}\sum_{j=1}^{\ell}\left(\sum_{i=1}^{\ell}\alpha_i y_i \partial_1 k(B\mathbf{x}_j, B\mathbf{x}_i) \cdot B\frac{\partial}{\partial t}\Big|_{t=0}\mathcal{L}_t\mathbf{x}_j\right)^2. \tag{4.11}$$

The derivative of k must be evaluated for specific kernels, e.g. for $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$, $\partial_1 k(\mathbf{x}, \mathbf{y}) = d \cdot (\mathbf{x} \cdot \mathbf{y})^{d-1} \cdot \mathbf{y}^\top$. To obtain a kernel-specific constraint on the matrix B, one would need to equate the result with the quadratic term in the nonlinear objective function,

$$\sum_{i,k}\alpha_i y_i \alpha_k y_k k(B\mathbf{x}_i, B\mathbf{x}_k). \tag{4.12}$$

Relationship to Principal Component Analysis. Let us now provide some interpretation of (4.9) and (4.6). The tangent vectors $\pm\frac{\partial}{\partial t}|_{t=0}\mathcal{L}_t\mathbf{x}_j$ have zero mean, thus C is a sample estimate of the covariance matrix of the random vector $s \cdot \frac{\partial}{\partial t}|_{t=0}\mathcal{L}_t\mathbf{x}$, $s \in \{\pm 1\}$ being a random sign. Based on this observation, we call C (4.6) the *Tangent Covariance Matrix* of the data set $\{\mathbf{x}_i : i = 1, \ldots, \ell\}$ with respect to the transformations $\mathcal{L}_t$.

Being positive definite,[2] C can be diagonalized, $C = SDS^\top$, with an orthogonal matrix S consisting of C's Eigenvectors and a diagonal matrix D containing the corresponding positive Eigenvalues. Then we can compute

$$B = C^{-\frac{1}{2}} = SD^{-\frac{1}{2}}S^\top, \tag{4.13}$$

where $D^{-\frac{1}{2}}$ is the diagonal matrix obtained from D by taking the inverse square

[2] it is understood that we use C_λ if C is not definite (cf. (4.10))

roots of the diagonal elements. Since the dot product is invariant under orthogonal transformations, we may drop the leading S and (4.1) becomes

$$g(\mathbf{x}_j) = \sum_{i=1}^{\ell} \alpha_i y_i (D^{-\frac{1}{2}} S^\top \mathbf{x}_j \cdot D^{-\frac{1}{2}} S^\top \mathbf{x}_i) + b. \tag{4.14}$$

A given pattern $\mathbf{x}$ is thus first transformed by projecting it onto the Eigenvectors of the tangent covariance matrix C, which are the rows of $S^\top$. The resulting feature vector is then rescaled by dividing by the square roots of C's Eigenvalues.[3] In other words, the directions of main variance of the random vector $\frac{\partial}{\partial t}|_{t=0}\mathcal{L}_t\mathbf{x}$ are scaled back, thus more emphasis is put on features which are less variant under $\mathcal{L}_t$. For example, in image analysis, if the $\mathcal{L}_t$ represent translations, more emphasis is put on the relative proportions of ink in the image rather than the positions of lines. The PCA interpretation of our preprocessing matrix suggests the possibility to regularize and reduce dimensionality by discarding part of the features, as it is common usage when doing PCA. As an aside, note that the resulting matrix will still satisfy (4.8).[4]

Combining the PCA interpretation with the considerations following (4.1) leads to an interesting observation: by computing the tangent covariance matrix from the SVs only, rather than from the full data set, it can be rendered a task-dependent covariance matrix. Although the summation in (4.6) does not take into account class labels y_i, it then implicitely depends on the task to be solved via the SV set, which is computed for given the task. Thus, it allows the extraction of features which are invariant in a task-dependent way: it does not matter whether features for "easy" patterns change with transformations, it is more important that the "hard" patterns, close to the decision boundary, lead to invariant features.

The Nonlinear Tangent Covariance Matrix. We are now in a position to describe a feasible way how to generalize to the nonlinear case. To this end, we use kernel principal component analysis (Chapter 3). This technique allows us to compute principal components in a space F nonlinearly related to input space. The kernel function k plays the role of the dot product in F, i.e. $k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$. To generalize (4.14) to the nonlinear case, we compute the tangent covariance matrix C (Eq. 4.6) in feature space F, and its projection onto the subspace of F which is given by the linear span of the tangent vectors in F. There, the considerations of the linear case

[3] As an aside, note that our goal to build invariant SV machines has thus serendipitously provided us with an approach for an open problem in SV learning, namely the one of scaling: in SV machines, there has so far been no way of automatically assigning different weight to different directions in input space — in a trained SV machine, the weights of the first layer (the SVs) form a subset of the training set. Choosing these Support Vectors from the training set only gives rather limited possibilities for appropriately dealing with different scales in different directions of input space.

[4] To see this, first note that if B solves $B^\top BCB^\top B = B^\top B$, and B's polar decomposition is $B = UB_s$, with $UU^\top = 1$ and $B_s = B_s^\top$, then B_s also solve it. Thus, we may restrict ourselves to symmetrical solutions. For our choice $B = C^{-\frac{1}{2}}$, B commutes with C, hence they can be diagonalized simultaneously. In this case, $B^2CB^2 = B^2$ clearly can also be satisfied by any matrix which is obtained from B by setting an arbitrary selection of Eigenvalues to 0 (in the diagonal representation).

apply. The whole procedure reduces to computing dot products in F, which can be done using k, without explicitly mapping into F:

In rewriting (4.6) for the nonlinear case, we substitute finite differences, with $t > 0$, for derivatives:

$$C := \frac{1}{\ell t^2} \sum_{j=1}^{\ell} \left(\Phi(\mathcal{L}_t \mathbf{x}_j) - \Phi(\mathbf{x}_j)\right) \left(\Phi(\mathcal{L}_t \mathbf{x}_j) - \Phi(\mathbf{x}_j)\right)^\top . \tag{4.15}$$

For the sake of brevity, we have omitted the summands corresponding to derivatives in the opposite direction, which ensure that the data set is centered. For the final tangent covariance matrix C, they do not make a difference, as the two negative signs cancel out.

In high-dimensional feature spaces, C cannot be computed explicitely. In complete analogy to Chapter 3, we compute another matrix whose Eigenvalues and Eigenvectors will allow us to extract features corresponding to Eigenvectors and Eigenvalues of C. This is done by taking dot products from both sides with $\Phi(\mathcal{L}_t \mathbf{x}_i) - \Phi(\mathbf{x}_i)$ (the Eigenvectors in F can be expanded in terms of the latter, by the same argument as the one leading to (3.10)). Defining

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j), \tag{4.16}$$

$$K^t_{ij} = k(\mathbf{x}_i, \mathcal{L}_t \mathbf{x}_j) + k(\mathcal{L}_t \mathbf{x}_i, \mathbf{x}_j), \tag{4.17}$$

and

$$K^{tt}_{ij} = k(\mathcal{L}_t \mathbf{x}_i, \mathcal{L}_t \mathbf{x}_j), \tag{4.18}$$

we get

$$\begin{aligned} & (\Phi(\mathcal{L}_t \mathbf{x}_i) - \Phi(\mathbf{x}_i))^\top C (\Phi(\mathcal{L}_t \mathbf{x}_k) - \Phi(\mathbf{x}_k)) \\ = \; & \frac{1}{\ell t^2} \sum_{j=1}^{\ell} (K^{tt}_{ij} - K^t_{ij} + K_{ij})(K^{tt}_{jk} - K^t_{jk} + K_{jk}) \\ = \; & \left(\frac{1}{\ell t^2} (K^{tt} - K^t + K)^2 \right)_{ik} . \end{aligned} \tag{4.19}$$

Using (4.19), and Eigenvector expansions

$$\mathbf{V} = \sum_{k=1}^{\ell} \alpha_k (\Phi(\mathcal{L}_t \mathbf{x}_k) - \Phi(\mathbf{x}_k)), \tag{4.20}$$

the Eigenvalue problem that we need to solve (cf. (3.9)),

$$\begin{aligned} & \lambda (\Phi(\mathcal{L}_t \mathbf{x}_i) - \Phi(\mathbf{x}_i))^\top \sum_{k=1}^{\ell} \alpha_k (\Phi(\mathcal{L}_t \mathbf{x}_k) - \Phi(\mathbf{x}_k)) \\ = \; & (\Phi(\mathcal{L}_t \mathbf{x}_i) - \Phi(\mathbf{x}_i))^\top C \sum_{k=1}^{\ell} \alpha_k (\Phi(\mathcal{L}_t \mathbf{x}_k) - \Phi(\mathbf{x}_k)), \end{aligned} \tag{4.21}$$

then takes the form

$$\lambda(K^{tt} - K^t + K)\boldsymbol{\alpha} = \frac{1}{\ell t^2}(K^{tt} - K^t + K)^2\boldsymbol{\alpha}. \tag{4.22}$$

To find solutions of (4.22), we solve the Eigenvalue problem (cf. (3.14))[5]

$$\lambda\boldsymbol{\alpha} = \frac{1}{\ell t^2}(K^{tt} - K^t + K)\boldsymbol{\alpha}. \tag{4.23}$$

Normalization of each Eigenvector (4.20) is carried out by requiring $(V \cdot V) = 1$, which, as in (3.16), translates into

$$\lambda(\boldsymbol{\alpha} \cdot \boldsymbol{\alpha}) = 1, \tag{4.24}$$

using the corresponding Eigenvalue λ.

Feature extraction for a test point $\mathbf{x}$ is done by computing the projection of $\Phi(\mathbf{x})$ onto Eigenvectors $\mathbf{V}$,

$$\begin{aligned} \mathbf{V}^\top\Phi(\mathbf{x}) &= \sum_{k=1}^{\ell} \alpha_k (\Phi(\mathcal{L}_t\mathbf{x}_k) - \Phi(\mathbf{x}_k))^\top \Phi(\mathbf{x}) \\ &= \sum_{k=1}^{\ell} \alpha_k (k(\mathcal{L}_t\mathbf{x}_k, \mathbf{x}) - k(\mathbf{x}_k, \mathbf{x})). \end{aligned} \tag{4.25}$$

In Appendix D.3, we give an alternative justification of this procedure, which naturally arises from requiring invariance in feature space, without the need for a PCA interpretation.

4.3 Image Locality and Local Feature Extractors

By using a kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$, one implicitly constructs a decision boundary in the space of all possible products of d pixels. This may not be desirable, since in natural images, correlations over short distances are much more reliable as features than long-range correlations are. To take this into account, we define a kernel $k_p^{d_1, d_2}$ as follows (cf. Fig. 4.3):

1. compute a third image $\mathbf{z}$, defined as the pixel-wise product of $\mathbf{x}$ and $\mathbf{y}$
2. sample $\mathbf{z}$ with pyramidal receptive fields of diameter p, centered at all locations (i, j), to obtain the values $\mathbf{z}_{ij}$
3. raise each $\mathbf{z}_{ij}$ to the power d_1, to take into account local correlations within the range of the pyramid

[5]If we expand V in a different set of vectors, we instead arrive at a problem of simultaneous diagonalization of two matrices.

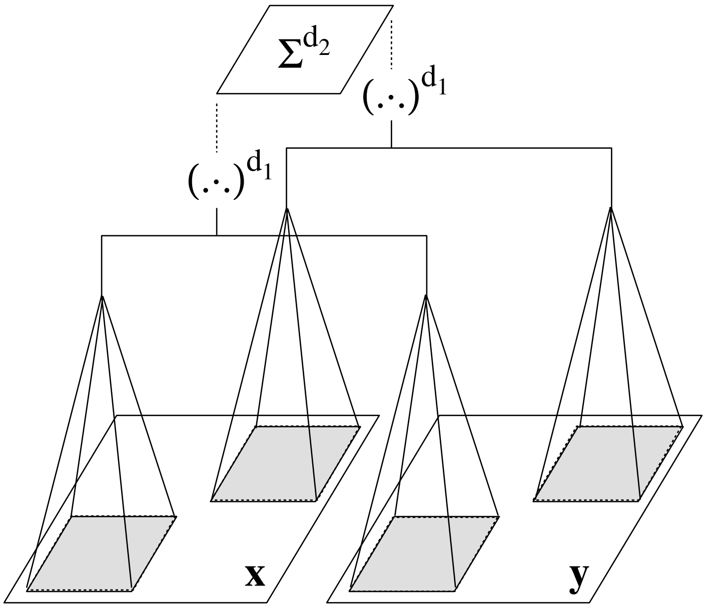

FIGURE 4.3: Kernel utilizing local correlations in images. To compute $k(\mathbf{x}, \mathbf{y})$ for two images $\mathbf{x}$ and $\mathbf{y}$, we sum over products between corresponding pixels of the two images in localized regions (in the figure, this is indicated by dot products $(. \cdot .)$), weighed by pyramidal receptive fields. To the outputs, a first nonlinearity in form of an exponent d_1 is applied. The resulting numbers for all patches (only two are displayed) are summed, and the d_2-th power of the result is taken as the value $k(\mathbf{x}, \mathbf{y})$. This kernel corresponds to a dot product in a polynomial space which is spanned mainly by localized correlations between pixels (see Sec. 4.3).

4. sum $\mathbf{z}_{ij}^{d_1}$ over the whole image, and raise the result to the power d_2 to allow for longe-range correlations of order d_2

The resulting kernel will be of order $d_1 \cdot d_2$, however, it will *not* contain *all* possible correlations of $d_1 \cdot d_2$ pixels.

4.4 Experimental Results

4.4.1 Virtual Support Vectors

USPS Digit Recognition. The first set of experiments was conducted on the USPS database of handwritten digits (Appendix C). This database has been used extensively in the literature, with a LeNet1 Convolutional Network achieving a test error rate of 5.0% (LeCun et al., 1989). As in Sec. 2.3, we used $\gamma = 10$.

Virtual Support Vectors were generated for the set of all different Support Vectors of the ten classifiers. Alternatively, one can carry out the procedure separately for the ten binary classifiers, thus dealing with smaller training sets during the training of the second machine. Table 4.1 shows that incorporating only translational invariance

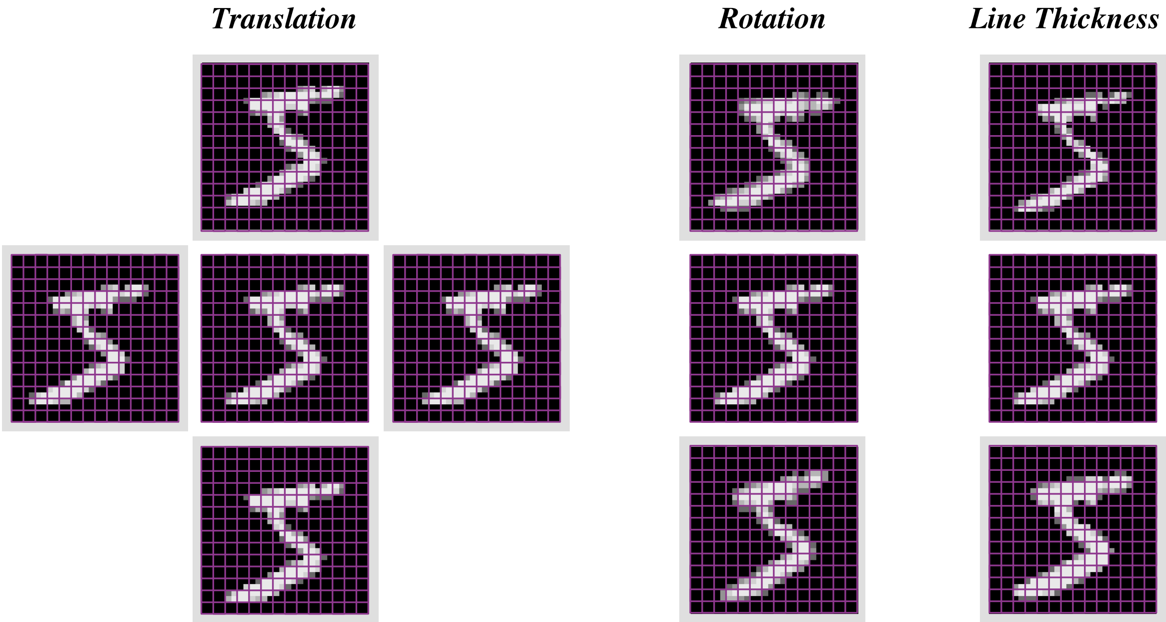

FIGURE 4.4: Different invariance transformations in the case of handwritten digit recognition (MNIST database). In all three cases, the central pattern is the original which is transformed into "virtual examples" (marked by grey frames) with the same class membership by applying small transformations.

TABLE 4.1: Comparison of Support Vector sets and performance for training on the original database and training on the generated Virtual Support Vectors. In both training runs, we used a polynomial classifier of degree 3. Virtual Support Vectors were generated by simply shifting the images by one pixel in the four principal directions. Adding the unchanged Support Vectors, this leads to a training set of the second classifier which has five times the size of the first classifier's overall Support Vector set (i.e. the union of the 10 Support Vector sets of the binary classifiers, of size 1677 — note that due to some overlap, this is smaller than the sum of the ten support set sizes). Note that training on virtual patterns generated from *all* training examples does not lead to better results than in the Virtual SV case; moreover, although the training set in this case is much larger, it hardly leads to more SVs.

classifier trained on	size	av. no. of SVs	test error
full training set	7291	274	4.0%
overall SV set	1677	268	4.1%
Virtual SV set	8385	686	3.2%
Virtual patterns from full DB	36455	719	3.4%

already improves performance significantly, from 4.0% to 3.2% error rate. For other types of invariances (Fig. 4.4), we also found improvements, albeit smaller ones: generating Virtual Support Vectors by rotation or by the line thickness transformation of Drucker, Schapire, and Simard (1993), we constructed polynomial classifiers with 3.7% error rate (in both cases).

Note, moreover, that generating Virtual examples from the full database rather than just from the SV sets did not improve the accuracy, nor did it enlarge the SV set of the final classifier substantially. This finding was reproduced for the Virtual SV system mentioned in Sec. 2.5.3: in that case, similar to Table 4.1, generating Virtual examples from the full database led to identical performance, and only slightly increased SV set size (861 instead of 806). From this, we conclude that for the considered recognition task, it is sufficient to generate Virtual examples only from the SVs — Virtual examples generated from the other patterns do not add much useful information.

MNIST Digit Recognition. The larger a database, the more information about invariances of the decision function is already contained in the differences between patterns of the same class. To show that it is nevertheless possible to improve classifi-

TABLE 4.2: Application of the Virtual SV method to the MNIST database. Virtual SVs were generated by translating the original SVs in all four principal directions (by 1 pixel). Results are given for the original SV machine, and two VSV systems utilizing different kernel degrees; in all cases, we used $\gamma = 10$ (cf. (2.19)). SV: degree 5 polynomial SV classifier; *VSV1*: VSV machine with degree 5 polynomial kernel; *VSV2*: same with degree 9 kernel. The *first table* gives the performance: for the ten binary recognizers, as numbers of errors; for multi class-classification (T1), in terms of error rates (in %), both on the 60000 element test set. The second multi-class error rate (T2) was computed by testing only on a 10000 element subset of the full 60000 element test set. These results are given for reference purpose, they are the ones usually reported in MNIST performance studies. The *second table* gives numbers of SVs for all ten binary digit recognizers.

Errors

	binary recognizers										10-class	
system	0	1	2	3	4	5	6	7	8	9	T1	T2
SV	131	97	243	240	212	241	195	259	343	409	1.6	1.4
VSV1	95	84	186	176	173	171	127	217	233	289	1.1	1.0
VSV2	81	66	164	146	141	147	119	179	196	254	1.0	0.8

Support Vectors

system	0	1	2	3	4	5	6	7	8	9
SV	1206	757	2183	2506	1784	2255	1347	1712	3053	2720
VSV1	2938	1887	5015	4764	3983	5235	3328	3968	6978	6348
VSV2	3941	2136	6598	7380	5127	6466	4128	5014	8701	7720

cation accuracies with our technique, we applied the method to the MNIST database (Appendix C) of 60000 handwritten digits. This database has become the standard for performance comparisons at AT&T Bell Labs; the error rate record of 0.7% is held by a boosted LeNet4 (Bottou et al., 1994; LeCun et al., 1995), i.e. by an ensemble of learning machines. The best single machine in the performance comparisons so far was a LeNet5 convolutional neural network (0.9%); other high performance systems include Tangent Distance nearest neighbour classifiers (1.1%), and LeNet4 with a last layer using methods of local learning (1.1%, cf. Bottou and Vapnik, 1992).

Using Virtual Support Vectors generated by 1-pixel translations, we improved a degree 5 polynomial SV classifier from 1.4% to 1.0% error rate on the 10000 element test set (Table 4.2). In this case, we applied our technique separately for all ten Support Vector sets of the binary classifiers (rather than for their union) in order to avoid having to deal with large training sets in the retraining stage. Note, moreover, that for the MNIST database, we did not compare results of the VSV technique to those for generating Virtual examples from the whole database: the latter is computationally exceedingly expensive, as it entails training on a very large training set. We did, however, make a comparison for the *small* MNIST database (Appendix C). There, a degree 5 polynomial classifier was improved from 3.8% to 2.5% error by the Virtual SV method, with an increase of the average SV set sizes from 324 to 823. By generating Virtual examples from the full training set, and retraining on these, we obtained a system which had slightly more SVs (939), but an unchanged error rate.

After retraining, the number of SVs more than doubled (Table 4.2). Thus, although the training sets for the second set of binary classifiers were substantially smaller than the original database (for four Virtual SVs per SV, four times the size of the original SV sets, in our case amounting to around 10^4), we concluded that the amount of data in the region of interest, close to the decision boundary, had more than doubled. Therefore, we reasoned that it should be possible to use a more complex decision function in the second stage (note that the risk bound (1.5) depends on the *ratio* of VC-dimension and training set size). Indeed, using a degree 9 polynomial led to an error rate of 0.8%, very close to the record performance of 0.7%.

Another interesting performance measure is the rejection error rate, defined as the percentage of patterns that would have to be rejected to attain a specified error rate (in the benchmark studies of Bottou et al. (1994) and LeCun et al. (1995), 0.5%). Note that this percentage is computed on the *test* set. In our case, using the confidence measure of Sec. 2.1.6, it was measured to be 0.9%, realizing a large improvement compared to the original SV system (2.4%). In the above benchmark studies, only the boosted LeNet4 ensemble performed better (0.5%).

Further improvements can possibly be achieved by combining different types of invariances. Another intriguing extension of the scheme would be to use techniques based on image correspondence (e.g. Vetter and Poggio, 1997) to *extract* invariance transformations from the training set. Those transformations can then be used to generate Virtual Support Vectors.[6]

[6]Together with Thomas Vetter, we have recently started working on this approach.

FIGURE 4.5: Virtual SVs in gender classification. A: 2-D image of a 3-D head model (from the MPI head database (Troje and Bülthoff, 1996; Vetter and Troje, 1997)); B: 2D image of the rotated 3D head; C: artificial image, generated from A using the assumption that it belongs to a cylinder-shaped 3D object (rotation by the same angle as B).

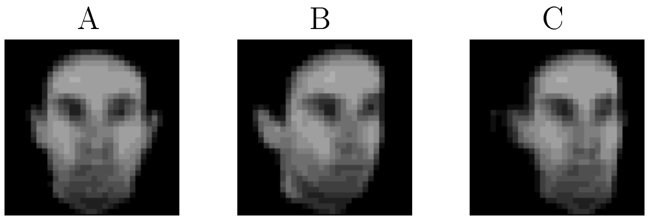

TABLE 4.3: Numbers of test errors for gender classification in novel pose, using Virtual SVs (qualitatively similar to Fig. 4.5). The training set contained 100 views of male and female heads (divided 49:51), taken at an azimuth of $24°$, downsampled to 32×32. The test set contained 100 frontal views of the same heads. We used polynomial SV classifiers of different degrees, generating one virtual SV per original SV. Clearly, training and test views are differently distributed, however, the amount of rotation ($24°$) was known to the classifier in the sense that it was used for generating the Virtual SVs (Fig. 4.5): first, a simplified head model was inferred by averaging over in-depth revolutions of all the 2D views. VSVs were generated by projecting the original SVs onto the head model, then rotating the head to the frontal view, and computing the new 2-D view.

	degree				
prior knowledge	1	2	3	4	5
no virtual SVs	25	24	23	21	19
virtual SVs from 3D model	11	10	10	9	10

Face Classification. Certain types of transformations, as the above used translations and rotations, apply equally well to object recognition as they do to character recognition. There are, however, types of transformations which are specific to the class of images considered (cf. Sec. 1.1). For instance, line thickness transformations (Fig. 4.4) are specific to character recognition. To provide an example of Virtual SVs which are specific to object recognition, we generated virtual SVs corresponding to object rotations in depth, by making assumptions about the 3D shape of objects. Clearly, such an approach would have a hard time if applied to complex objects as chairs (Appendix A). For human heads, however, it is possible to formulate 2-D image transformations which can be applied to generate *approximate* novel views of heads (Fig. 4.5). Using these views improved accuracies in a small gender classification experiment. Table 4.3 gives details and results of the experiment.

TABLE 4.4: Test error rates for two object recognition databases, for views of resolution 16×16, using different types of approximate invariance transformations to generate Virtual SVs, and polynomial kernels of degree 20 (cf. Table 2.1). The second training run in the Virtual SV systems was done on the original SVs and the generated Virtual SVs. The training sets with 25 and 89 views per object are regularly spaced; for them, mirroring does not provide additional information. The interesting case is the one where we trained on the 100-view-per-object sets. Here, a combination of virtual SVs from mirroring and rotation substantially improves accuracies on both databases.

database:	animal			entry level		
	training set: views per object					
Virtual SVs	25	89	100	25	89	100
none (orig. system)	13.0	1.7	$\boxed{4.8}$	13.0	1.8	$\boxed{2.4}$
mirroring	13.6	1.8	4.8	14.2	2.8	3.2
translations	16.4	1.6	4.3	17.1	11.1	4.8
rotations	9.0	0.7	3.0	10.3	1.8	2.5
rotations & mirroring	9.0	0.7	$\boxed{1.7}$	9.6	0.9	$\boxed{1.7}$

Discrete Symmetries in Object Recognition. As mentioned above, rigid transformations of 3-D objects, however, do not in general correspond to simple transformations of the produced 2-D images (cf. Sec. 4.4.1). For the MPI object databases (Appendix A), however, there exists a type of invariance transformation which can easily be computed from the images: as all the objects used are (approximately) bilaterally symmetric, we can easily produce another valid view of the same object, with a different viewing angle, by performing a mirror operation with respect to a vertical axis in the center of the images, say (Vetter, Poggio, and Bülthoff, 1994). If the objects were exactly symmetric, we would not expect any additional information to be gained in the case of the regularly spaced object sets (25 and 89 views per object), as in these the snapshots are already sampled symmetrically around the zero view direction, which in most cases coincided with the symmetry plane. The slight *decrease* in performance in that case (Table 4.4) indicates that for some objects, the symmetry only holds approximately (for snapshots, see Appendix A).

To get more robust, we tried combining this type of invariance transformation with other types. As in the case of character recognition, we simply used translations (by 1 pixel in all four directions) and image-plane rotations (by 10 degrees in both directions). Even though these transformation are but very crude approximations of transformations which occur when a 3-D object is rotated in space, they did in some cases yield significant performance improvements.[7]

[7] The following may serve as a partial explanation why rotations were more useful than translations. First, different snapshots at large elevations can be transformed into each other by an approximate image plane rotation, and second, image plane rotations retain the centering which was applied to

To examine the effect of the mirror symmetry Virtual SVs, we need to focus on the non-regularly spaced training set with 100 views per object. There, by far the best performance for both the entry level and the animal database was obtained by using both mirroring and rotations (Table 4.4).

TABLE 4.5: Speed improvement using the Reduced Set method. The second through fourth columns give numbers of errors on the 10000 element MNIST test set for the original system, the Virtual Support Vector system, and the reduced set system (for the 10-class classifiers, the error is given in %). The last three columns give, for each system, the number of vectors whose dot product must be computed in the test phase.

Digit	*SV* err	*VSV1* err	*RS* err	*SV* #	*VSV1* #	*RS* #
0	17	15	18	1206	2938	59
1	15	13	12	757	1887	38
2	34	23	30	2183	5015	100
3	32	21	27	2506	4764	95
4	30	30	35	1784	3983	80
5	29	23	27	2255	5235	105
6	30	18	24	1347	3328	67
7	43	39	57	1712	3968	79
8	47	35	40	3053	6978	140
9	56	40	40	2720	6348	127
10-class	1.4%	1.0%	1.1%			

Virtual SV Combined with Reduced Set. Apart from the increase in overall training time (by a factor of two, in our experiments), the VSV technique has the *computational* disadvantage that many of the Virtual Support Vectors become Support Vectors for the second machine, increasing the cost of evaluating the decision function (2.25). However, the latter problem can be solved with the Reduced Set (RS) method (Burges, 1996, see Appendix D.1.1), which reduces the complexity of the decision function representation by approximating it in terms of fewer vectors. In a study combining the VSV and RS methods, we achieved a factor of fifty speedup in test phase over the Virtual Support Vector machine, with only a small decrease in performance (Burges and Schölkopf, 1997). We next briefly report the results of this study. The RS results reported were obtained by Chris Burges.

As a starting point for the RS computation, we used the *VSV1* machine (Table 4.2), which achieved 1.0% error rate on the 10000 element MNIST test set.[8] The

the original images. Both points suggest that virtual examples generated by rotations should be more "realistic" than those generated by translations.

[8] At the time when the described study was carried out, *VSV1* was our best system; *VSV2* was not available yet.

improvement in accuracy compared to the *SV* machine (Table 4.2) comes at a cost in classification speed of approximately a factor of 2. Furthermore, the speed of *SV* was comparatively slow to start with (cf. LeCun et al., 1995), requiring approximately 14 million multiply adds for one classification. In order to become competitive with systems with comparable accuracy (LeCun et al., 1995), we need approximately a factor of fifty improvement in speed. We therefore approximated *VSV1* with a reduced set system *RS* with a factor of fifty fewer vectors than the number of Support Vectors in *VSV1*.

Table 4.5 compares results on the 10000 element test set for the three systems. Overall, the *SV* machine performance of 1.4% error is improved to 1.1%, with a machine requiring a factor of 22 fewer multiply adds (*RS*). For details on the computation of the RS solution, see (Burges and Schölkopf, 1997).

4.4.2 Invariant Hyperplane Method

In the experiments exploring the invariant hyperplane method (Sec. 4.2.2), we used the small MNIST database (Appendix C). We start by giving some baseline classification results.

Using a standard linear SV machine (i.e. a separating hyperplane, Sec. 2.1.3), we obtain a test error rate of 9.8%; by using a polynomial kernel of degree 4, this drops to 4.0%. In all of the following experiments, we use degree 4 kernels of various types. The number 4 was chosen as it can be written as a product of two integers, thus we could compare results to a kernel $k_p^{d_1,d_2}$ with $d_1 = d_2 = 2$ (cf. sections 4.3 and 4.4.3). For the considered classification task, results for higher polynomial degrees are very similar.

In a series of experiments with a homogeneous polynomial kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^4$, using preprocessing with Gaussian smoothing kernels of standard deviation in the range $0.1, 0.2, \ldots, 1.0$, we obtained error rates which gradually increased from 4.0% to 4.3%. We concluded that no improvement of the original 4.0% performance was possible by a simple smoothing operation.

Invariant Hyperplanes Results. Table 4.6 reports results obtained by preprocessing all patterns with B (cf. (4.9)), choosing different values of λ (cf. Eq. (4.10)). In the

TABLE 4.6: Classification error rates for modifying the kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^4$ with the invariant hyperplane preprocessing matrix $B_\lambda = C_\lambda^{-\frac{1}{2}}$; cf. Eqs. (4.9) – (4.10). Enforcing invariance with $\lambda = 0.2, 0.3, \ldots, 0.9$ leads to improvements over the original performance ($\lambda = 1$).

λ	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
error rate in %	4.2	3.8	3.6	3.6	3.7	3.8	3.8	3.9	3.9	4.0

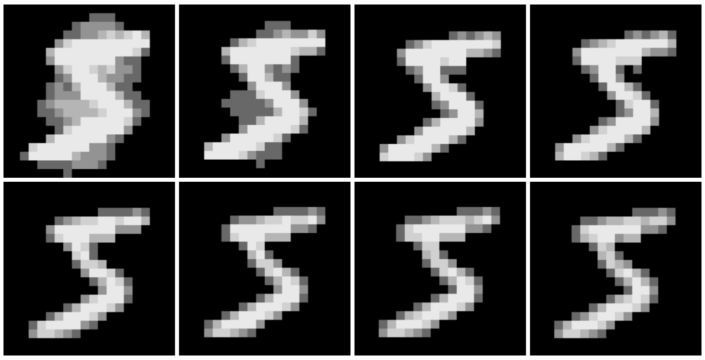

FIGURE 4.6: The first pattern in the small MNIST database, preprocessed with $B_\lambda = C_\lambda^{-\frac{1}{2}}$ (cf. equations (4.9) – (4.10)), enforcing various amounts of invariance. *Top row*: $\lambda = 0.1, 0.2, 0.3, 0.4$; *bottom row*: $\lambda = 0.5, 0.6, 0.7, 0.8$. For some values of γ, the preprocessing resembles a smoothing operation, however, it leads to higher classification accuracies (see Sec. 4.4.2) than the latter.

experiments, the patterns were first rescaled to have entries in $[0, 1]$, then B was computed, using horizontal and vertical translations, and preprocessing was carried out; finally, the resulting patterns were scaled back again (for snapshots of the resulting patterns, see Fig. 4.6). The scaling was done to ensure that patterns and derivatives lie in comparable regions of $\mathbf{R}^N$ (note that if the pattern background level is a constant -1, then its derivative is 0). The results show that even though (4.6) was derived for the linear case, it leads to improvements in the nonlinear case (here, for a degree 4 polynomial).

Dimensionality Reduction. The above $[0, 1]$ scaling operation is affine rather than linear, hence the argument leading to (4.14) does not hold for this case. We thus only report results on dimensionality reduction for the case where the data is kept in $[0, 1]$ scaling during the whole procedure. Dropping principal components which are less important leads to substantial improvements (Table 4.7); cf. the explanation following (4.14)).

The results in Table 4.7 are somewhat distorted by the fact that the polynomial kernel is not translation invariant, and performs poorly when none of the principal

TABLE 4.7: Dropping directions corresponding to small Eigenvalues of C, i.e. dropping less important principal components (cf. (4.14)), leads to substantial improvements. All results given are for the case $\lambda = 0.4$ (cf. Table 4.6); degree 4 homogeneous polynomial kernel.

PCs discarded	0	50	100	150	200	250	300	350
error rate in %	8.7	5.4	4.9	4.4	4.2	3.9	3.7	3.9

components are discarded. Hence this result should not be compared to the performance of the polynomial kernel on the data in $[-1, 1]$ scaling. (Recall that we obtained 3.6% in that case, for $\lambda = 0.4$.) In practice, of course, we may choose the scaling of the data as we like, in which case it would seem pointless to use a method which is only applicable for a rather disadvantageous representation of the data. However, nothing prevents us from using a translation invariant kernel. We opted for a radial basis function kernel (2.27) with $c = 0.5$. On the $[-1, 1]$ data, for $\lambda = 0.4$, this leads to the same performance as the degree 4 polynomial, 3.6% (without invariance preprocessing, i.e. for $\lambda = 1$, the performance is 3.9%). To get the identical system on $[0, 1]$ data, the RBF width was rescaled accordingly, to $c = 0.125$. Table 4.8 shows that discarding principal components can further improve performance, up to 3.3%.

TABLE 4.8: Dropping directions corresponding to small Eigenvalues of C, i.e. dropping less important principal components (cf. (4.14)), for the translation invariant RBF kernel (see text). All results given are for the case $\lambda = 0.4$ (cf. Table 4.6).

PCs discarded	0	50	100	150	200	250	300	350
error rate in %	3.6	3.6	3.6	3.5	3.5	3.4	3.3	3.6

4.4.3 Kernels Using Local Correlations

Character Recognition. As in Sec. 4.4.2, the present results were obtained on the small MNIST database (Appendix C). As a reference result, we use the degree 4 polynomial SV machine, performing at 4.0% error (Sec. 4.4.2). To exploit locality in images, we used pyramidal receptive field kernel $k_p^{d_1,d_2}$ with diameter $p = 9$ (cf. Sec. 4.3) and $d_1 \cdot d_2 = 4$, i.e. degree 4 polynomials kernels which do not use *all* products of 4 pixels (Sec. 4.2.2). For $d_1 = d_2 = 2$, we obtained an improved error rate of 3.1%, another degree 4 kernel with *only* local correlations ($d_1 = 4, d_2 = 1$) led to 3.4% (Table 4.9).

Albeit better than the 4.0% for the degree 4 homogeneous polynomial, this is still worse than the Virtual SV result: generating Virtual SVs by image translations, the latter led to 2.8%. As the two methods, however, exploit different types of prior knowledge, it could be expected that combining them leads to still better performance; and indeed, this yielded the best performance of all (2.0%), halving the error rate of the original system.

For the purpose of benchmarking, we also ran our system on the USPS database. In that case, we obtained the following test error rates: SV with degree 4 polynomial kernel 4.2% (Table 2.4), Virtual SV (same kernel) 3.5%, SV with $k_7^{2,2}$ 3.6% (for the smaller USPS images, we used a k_7 kernel rather than k_9), Virtual SV with $k_7^{2,2}$ 3.0%. The latter compares favourably to almost all known results on that database, and is second only to a memory-based tangent-distance nearest neighbour classifier at 2.6% (Simard, LeCun, and Denker, 1993).

TABLE 4.9: Summary: error rates for various methods of incorporating prior knowledge, on the small MNIST database (Appendix C). In all cases, degree 4 polynomial kernels were used, either of the local type (Sec. 4.3), or (by default) of the complete polynomial type (2.26). In all cases, we used $\gamma = 10$ (cf. (2.19)).

Classifier	Test Error / %
SV	4.0
Virtual SV (Sec. 4.2.1), with translations	2.8
Invariant hyperplane (Sec. 4.2.2), $\lambda = 0.4$	3.6
same, on first 100 principal components (Table 4.7)	3.7
semi-local kernel $k_9^{2,2}$ (Sec. 4.4.3)	3.1
purely local kernel $k_9^{4,1}$ (Sec. 4.4.3)	3.4
Virtual SV with $k_9^{2,2}$	2.0

Object Recognition. The above results have been confirmed on the two object recognition databases used in Sec. 2.2.1 (cf. Appendix A). As in the case of the small MNIST database, we used $k_9^{d_1,d_2}$. In the present case, we chose $d_1 = d_2 = 3$, which yields a degree 9 ($= 3 \cdot 3$) polynomial classifier which differs from a standard polynomial (2.26) in that it does not utilize all products of 9 pixels, but mainly local ones. Comparing the results to those obtained with standard polynomials of equal degree shows that this pre-selection of useful features significantly improves recognition results (Table 4.10).

As in the case of digit recognition, we combined this method with the Virtual SV method (Sec. 4.2.1). Based on the fact that prior knowledge about image locality is different from prior knowledge on invariances, we expected the possibility to get further improvements. We used the same types of Virtual SVs as in Sec. 4.4.1. The results (Table 4.11) further improve upon Table 4.10, confirming the digit recognition findings reported above. In 4 of 6 cases, the resulting classifiers are better than those of Table 4.4.[9]

4.5 Discussion

For Support Vector learning machines, invariances can readily be incorporated by generating virtual examples from the Support Vectors, rather than from the whole training set. The method yields a significant gain in classification accuracy at a moderate cost in time: it requires two training runs (rather than one), and it constructs classification rules utilizing more Support Vectors, thus slowing down classification speed (cf. (2.25)) — in our case, both points amounted to a factor of about 2. Given that Support Vector

[9] Note that in Table 4.4, the VSV method was used for degree 20 kernels, which on the object recognition tasks does far better than degree 9, cf. Table 2.1.

TABLE 4.10: Test error rates for two object recognition tasks, comparing kernels local in the image to complete polynomial kernels. Local kernels of degree 9 outperform complete polynomial kernels of corresponding degree. Moreover, they performed at least as well as the best polynomial classifier out of all degrees in $\{1, 3, 6, 9, 12, 15, 20, 25\}$ (cf. Table 2.1).

kernel:	degree 9 polyn.	best polynomial	$k_9^{3,3}$ (cf. Sec. 4.3)
entry level:			
25 grey scale	13.9	13.0	12.0
89 grey scale	2.0	1.8	1.8
100 grey scale	3.5	2.4	2.0
25 silhouettes	16.7	15.4	15.0
89 silhouettes	2.7	2.2	2.1
100 silhouettes	4.8	4.0	3.9
animals:			
25 grey scale	14.8	13.0	12.0
89 grey scale	2.5	1.7	1.6
100 grey scale	5.2	4.4	4.0
25 silhouettes	17.0	15.6	15.2
89 silhouettes	2.8	2.2	2.0
100 silhouettes	6.3	5.2	4.9

TABLE 4.11: Test error rates for two object recognition databases, using different types of approximate invariance transformations to generate Virtual SVs (as in Table 4.4), and local polynomial kernels $k_9^{3,3}$ of degree 9 (cf. Sec. 4.3, Table 4.10, Table 4.4, and Table 2.1). The second training run in the Virtual SV systems was done on the original SVs and the generated Virtual SVs. The training sets with 25 and 89 views per object are regularly spaced; for them, mirroring does not provide additional information. For the non-regularly spaced 100-view-per-object sets, a combination of virtual SVs from mirroring and rotation substantially improves accuracies on both databases.

database:	animal			entry level		
	training set: views per object					
Virtual SVs	25	89	100	25	89	100
none (orig. system)	12.0	1.6	4.0	12.0	1.8	2.0
mirroring	12.5	1.7	4.6	13.1	2.9	3.3
rotations & mirroring	8.8	1.0	1.4	8.5	1.2	1.6

machines are known to allow for short training times (Bottou et al., 1994), the first point is usually not critical. Certainly, training on virtual examples generated from the whole database would be significantly slower. To compensate for the second point, we used the reduced set method of Burges (1996) for increasing speed. This way, we obtained a system which was both fast and accurate.

As an alternative approach, we have built in known invariances directly into the SVM objective function via the choice of a kernel. With its rather general class of admissible kernel functions, the SV algorithm provides ample possibilities for constructing task-specific kernels. We have considered two forms of domain knowledge: first, pattern classes were required to be locally translationally invariant, and second, local correlations in the images were assumed to be more reliable than long-range correlations. The second requirement can be seen as a more general form of prior knowledge — it can be thought of as arising partially from the fact that patterns possess a whole variety of transformations; in object recognition, for instance, we have object rotations and deformations. Mostly, these transformations are continuous, which implies that local relationships in an image are fairly stable, whereas global relationships are less reliable.

Both types of domain knowledge led to improvements on the considered pattern recognition tasks.

The method for constructing kernels for *transformation invariant* SV machines (invariant hyperplanes), put forward to deal with the first type of domain knowledge, so far has only been applied in the linear case, which probably explains why it only led to moderate improvements, especially when compared with the large gains achieved by the Virtual SV method. It is applicable for differentiable transformations — other types, e.g. for mirror symmetry, have to be dealt with using other techniques, as the Virtual Support Vector method. Its main advantages compared to the latter technique is that it does not slow down testing speed, and that using more invariances leaves training time almost unchanged. In addition, it is more attractive from a theoretical point of view, establishing a surprising connection to invariant feature extraction, preprocessing, and principal component analysis.

The proposed kernels respecting *locality* in images, on the other hand, led to large improvements; they are applicable not only in image classification but to all cases where the relative importance of subsets of products features can be specified appropriately. They do, however, slow down both training and testing by a constant factor which depends on the cost of evaluating the specific kernel used.

Clearly, SV machines have not yet been developed to their full potential, which could explain the fact that our highest accuracies are still slightly worse that the record on the MNIST database. However, SVMs present clear opportunities for further improvement. More invariances (for example, for the pattern recognition case, small rotations, or varying ink thickness) could be incorporated, possibly combined with techniques for dealing with optimization problems involving very large numbers of SVs (Osuna, Freund, and Girosi, 1997). Further, one might use only those Virtual Support Vectors which provide new information about the decision boundary, or use a measure of such information to keep only the most important vectors. Finally, if local kernels (Sec. 4.3) will prove to be as useful on the full MNIST database as they were on the small version of it, accuracies could be substantially increased — at a cost in classification speed, though.

We conclude this chapter by noting that all three described techniques should be directly applicable to other kernel-based methods as SV regression (Vapnik, 1995b) and kernel PCA (Chapter 3). Future work will include the nonlinear Tangent Covariance Matrix (cf. our considerations in Sec. 4.2.2), the incorporation of invariances other than translation, and the construction of kernels incorporating local feature extractors (e.g. edge detectors) different from the pyramids described in Sec. 4.3.

Chapter 5

Conclusion

We believe that Support Vector machines and Kernel Principal Component Analysis are only the first examples of a series of potential applications of Mercer-kernel-based methods in learning theory. *Any* algorithm which can be formulated solely in terms of dot products can be made nonlinear by carrying it out in feature spaces induced by Mercer kernels. However, already the above two fields are large enough to render an exhaustive discussion in this thesis infeasible. Thus, we have tried to focus on some aspects of SV learning and Kernel PCA, hoping that we have succeeded in illustrating how nonlinear feature spaces can beneficially be used in complex learning tasks.

On the **Support Vector** side, we presented two chapters. Apart from a tutorial introduction to the theory of SV learning, the first one focused on empirical results related to the accuracy and the Support vector sets of different SV classifiers. Considering three well-known classifier types which are included in the SV approach as special cases, we showed that they lead to similarly high accuracies and construct their decision surface from almost the same Support Vectors. Our first question raised in the Preface was *which of the observations should be used to construct the decision boundary?* Against the backdrop of our empirical findings, we can now take the position that the Support Vectors, if constructed in an appropriate nonlinear feature space, constitute such a subset of observations. The second SV chapter focused on algorithms and empirical results for the incorporation of prior knowledge in SV machines. We showed that this can be done both by modifying kernels and by generating Virtual examples from the set of Support Vectors. In view of the high performances obtained, we can reinforce and generalize the above answer, to include also Virtual Support Vectors, and specialize it, saying that the appropriate feature space should be constructed using prior knowledge of the task at hand. Our best performing systems used both methods simultaneously, Virtual Support Vectors and kernels incorporating prior knowledge about the local structure of images.

On **Kernel Principal Component Analysis**, we presented one chapter, describing the algorithm and giving first experimental results on feature extraction for pattern recognition. We saw that features extracted in nonlinear feature spaces led to recognition performances much higher than those extracted in input space (i.e. with traditional PCA). This lends itself to an answer of the second question raised in the

Preface, *which features should be extracted from each observation?* From our present point of view, these should be nonlinear Kernel PCA features. As Kernel PCA operates in the same types of feature spaces as Support Vector machines, the choice of the kernel, and the design of kernels to incorporate prior knowledge, should also be of importance here. As the Kernel PCA method is very recent, however, these questions have not been thoroughly investigated yet. We hope that given a few years time, we will be in a position to specialize our answer to the second question exactly as it was done for the first one.

We conclude with an outlook, revisiting the question of visual processing in biological systems. If the Support Vector set should prove to be a characteristic of the data largely independent of the type of learning machine used (which we have shown for three types of learning machines), one would hope that it could also be of relevance in biological learning. If a subset of observations characterizes a *task* rather than a particular algorithm's favourite examples, there is reason to hope that every system trying to solve this task — in particular animals — should make use of this subset in one way or another. Regarding Kernel PCA, it would be interesting to study the types of feature extractors that Kernel PCA constructs when performed on collections of images resembling those that animals are naturally exposed to. Comparing those with the ones found in neurophysiological studies could potentially assist us in trying to understand natural visual systems. If applied on the same data, and similar tasks, optimal machine learning algorithms could be as fruitful to biological thinking as biological solutions can be to engineering.

Support Vector Learning!

Appendix A

Object Databases

In this section, we briefly describe three object recognition databases (chairs, entry level objects, and animals) generated at the Max-Planck-Institut für biologische Kybernetik (Liter et al., 1997). We start by describing the procedure for creating the databases, and then show some images of the resulting patterns.

The training and test data was generated according to the following procedure (Blanz et al., 1996; Liter et al., 1997):

Database Generation

Snapshot Sampling. 25 different object models with uniform grey surface were rendered in perspective projection in front of a white background on a Silicon Graphics workstation using Inventor software. The initial images had a resolution of 256×256 pixels. In all viewing directions, the image plane orientation was such that the vertical axis of the object was projected in an upright orientation. Thus, each view of an object is fully characterised by two camera position angles, the elevation θ ($0°$ at the horizon, and $90°$ from the top) and the azimuth $\phi \in [0°, 360°)$ (increasing clockwise when viewed from the top). Only views on the upper half of the viewing sphere were used, i.e. $\theta \in [0°, 90°]$. The directions of lighting and camera were chosen to coincide.

For each database, we generated different training sets: two of them consisted of 25 and 89 equally spaced views of each object, respectively; the other one contained 100 random views per object (cf. Fig. A.1).[1] Thus, we obtained training sets of sizes 625, 2225 and 2500, respectively. The test set of size 2500 comprised 100 random views of each object, independent from the above sets.

Centering. The resulting grey level pictures were centered with respect to the center of mass of the binarized image. As the objects were shown on a white background, the binarized image separates figure from ground.

Edge Detection. Four one-dimensional differential operators (vertical, horizontal, and two diagonal ones) were applied to the images, followed by taking the modulus.

[1] In one case, we also generated a set with 400 random views per object.

Downsampling. In all five resulting images, the resolution was reduced to 16×16, leading to five images $r_0 \ldots r_4$. In this representation, each view requires $5 \cdot 16 \cdot 16 = 1280$ pixels.

Containing edge detection data, the parts $r_1 \ldots r_4$ already provide useful features for recognition algorithms. To study the ability of an algorithm to extract features by itself, one can alternatively use only the actual image part r_0 of the data, and thus train on the 256-dimensional downsampled images rather than on the 1280-dimensional inputs. In our experiments, we used both variants of the databases.

Standardization. On the chair database, the standard deviation of the 16×16 images with pixel values in $[0, 1]$ was around 30 (measured on the training sets). We rescaled all databases, separately for each part $r_0 \ldots r_4$, such that each part separately gives rise to training sets with standard deviation 30. This hardly affects the r_0 part, however, it does change the edge detection parts $r_1, \ldots, r_4$. In the resulting 5×256-dimensional representation, the different parts arising from edge detection, or just downsampling, have comparable scaling.

Pixel Rescaling. Before we ran the algorithms on the databases, each pixel value x was rescaled according to $x \mapsto 2x - 1$. Thus, the background level was -1, and maximal intensities were about 1.

Databases

Using the above procedure, three object recognition databases were generated.

MPI Chair Database. The first object recognition database contains 25 different chairs (figures A.2, A.3, A.4). For benchmarking purposes, the downsampled views are available via ftp://ftp.mpik-tueb.mpg.de/pub/chair_dataset/. As all 25 objects belong to the same object category, recognition of chairs in the database is a subordinate level task.

MPI Entry Level Database. The entry level databases contains 25 objects (figures A.5, A.6, A.7), for which psychophysical evidence suggests that they belong to different entry levels in object recognition (cf. Sec. 2.2.1).

MPI Animal Database. The animal database contains 25 different animals (figures A.8, A.9, A.10). Note that some of these animals are also contained in the entry level database (Fig. A.5).

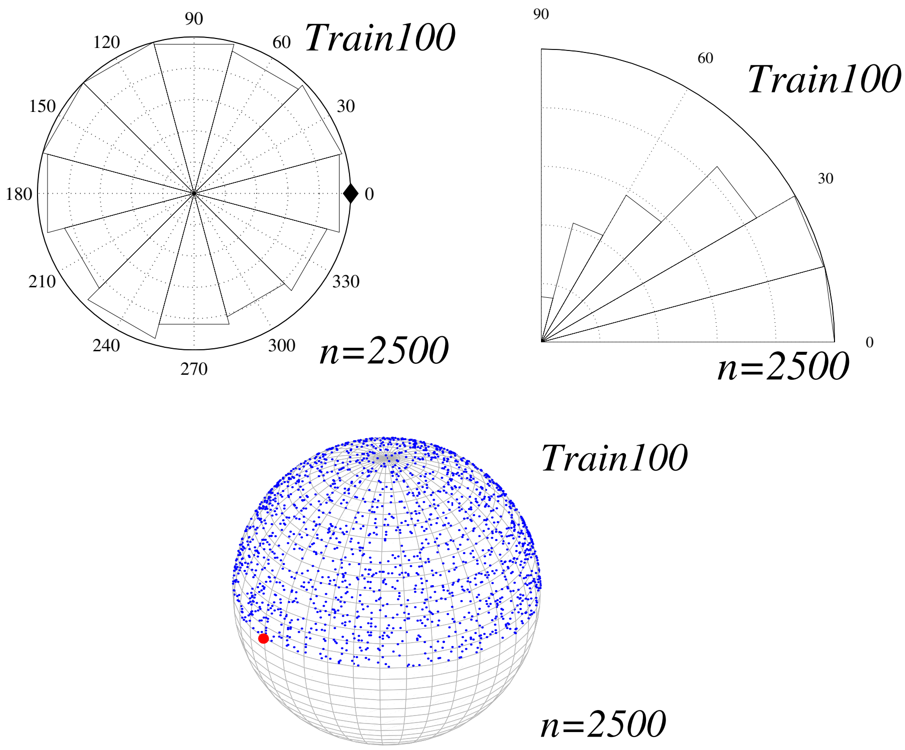

FIGURE A.1: Distribution of views in the 100 view-per-object training sets (*top left*: azimuth ϕ, *top right*: elevation θ, *bottom*: joint distribution; cf. Appendix A).

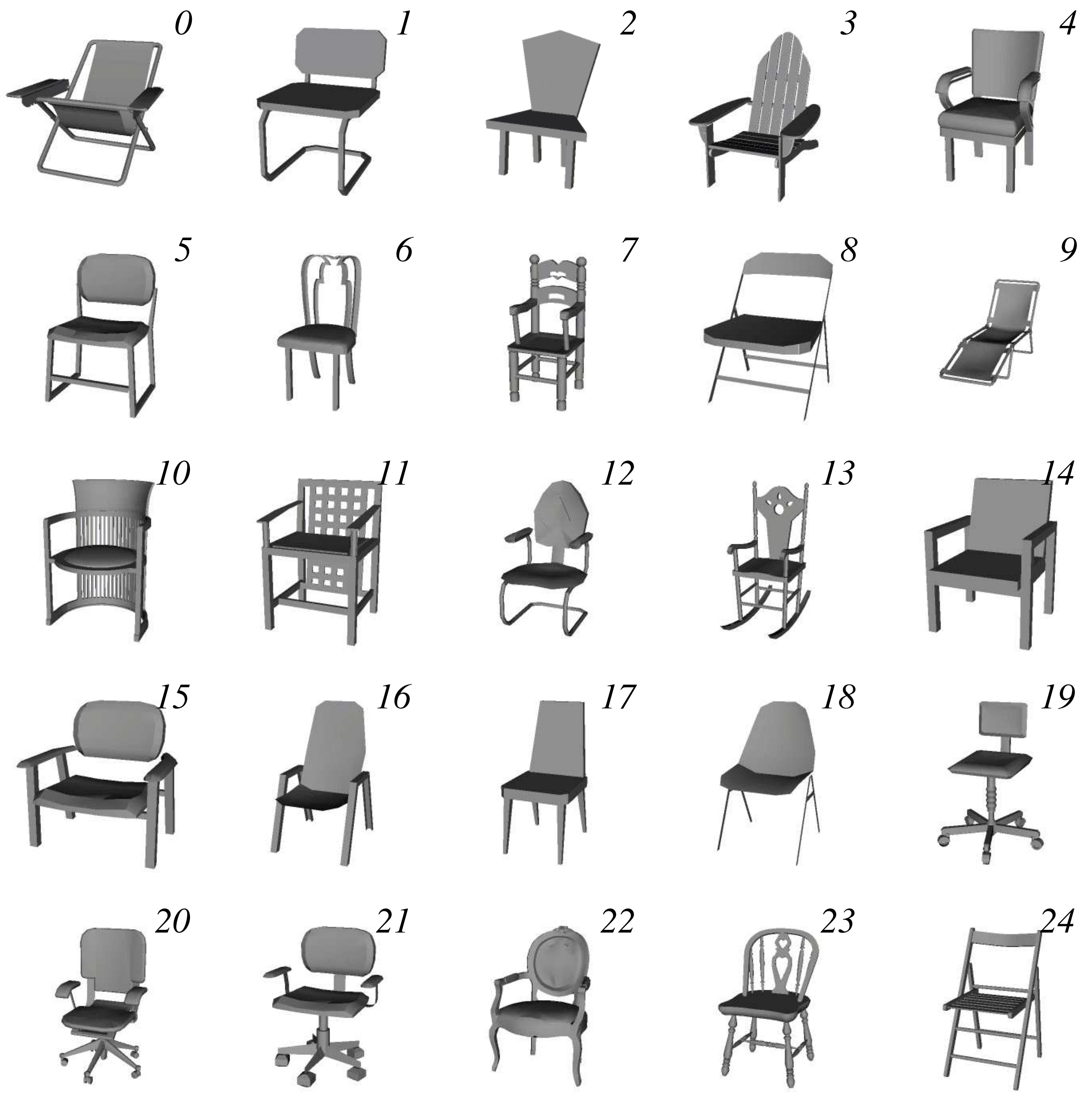

FIGURE A.2: 25 rendered 3-D models of chairs (images taken from Blanz et al. (1996)).

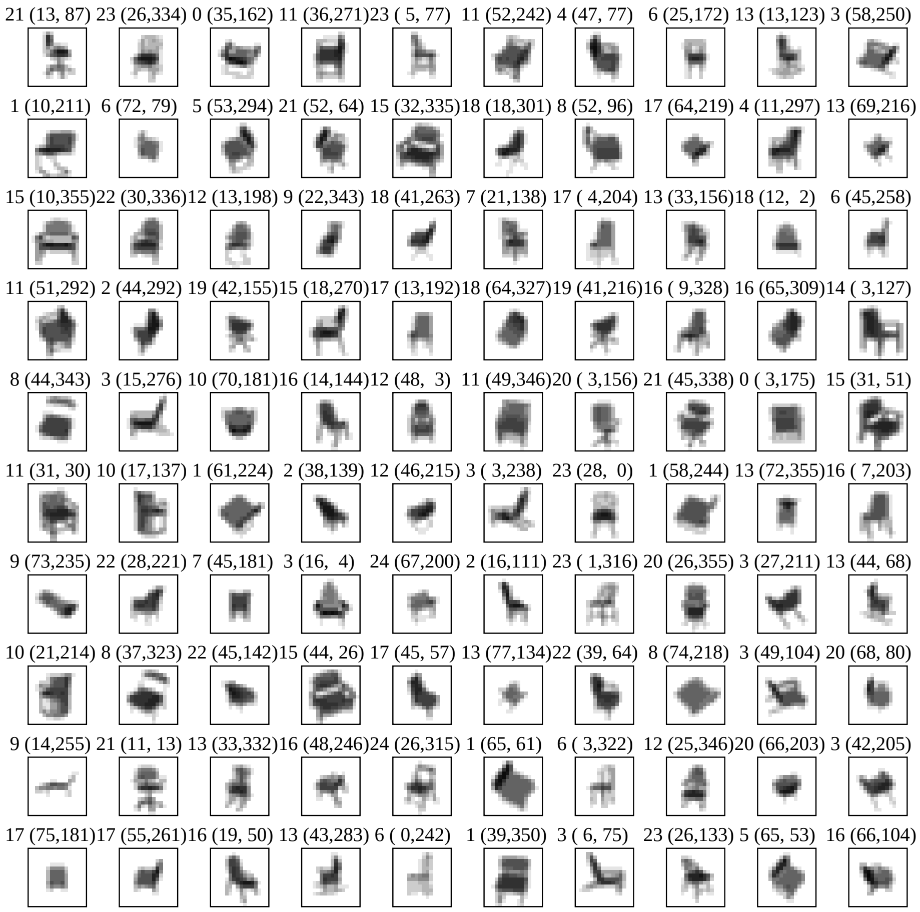

FIGURE A.3: 100 random views of the chairs depicted in Fig. A.2, taken from the upper viewing hemisphere, downsampled to 16×16 (from the chair training set). The numbers above the images are the class labels and the two polar angles (θ, ϕ), measured in degrees, of the viewing direction in which the snapshots were taken.

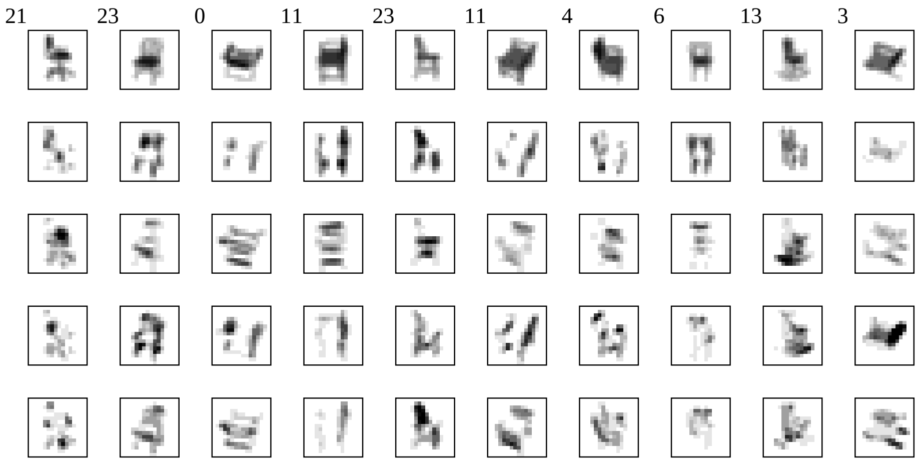

FIGURE A.4: Image (top row) and edge detection data computed with vertical, horizontal, and two diagonal filters (four bottom rows), for the first 10 images in Fig. A.3.

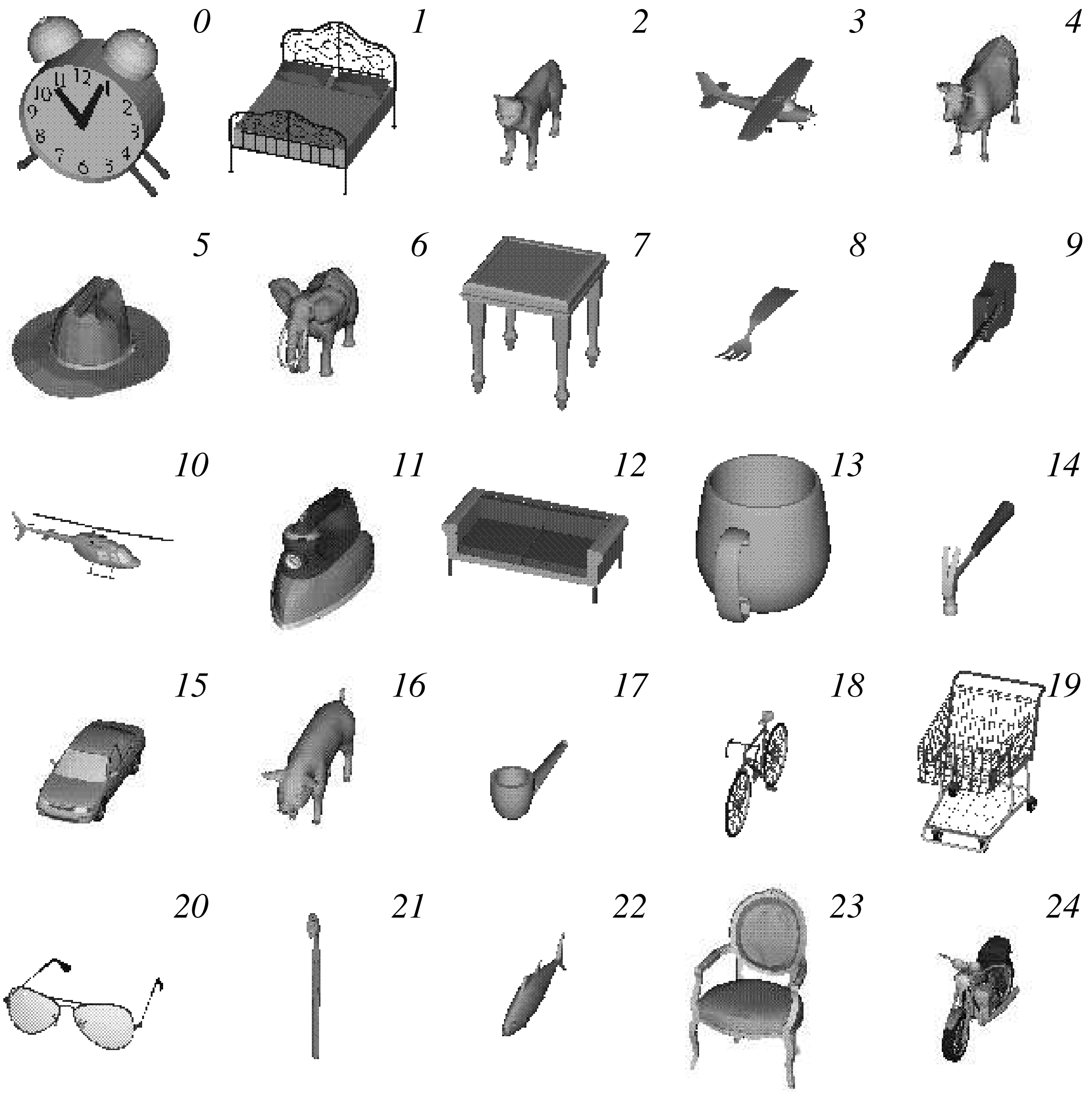

FIGURE A.5: Snapshots of the 3-D models in the MPI entry level database (taken from Liter et al. (1997)). The views shown were taken at elevation $\theta = 30°$ and azimuth $\phi = -32°$.

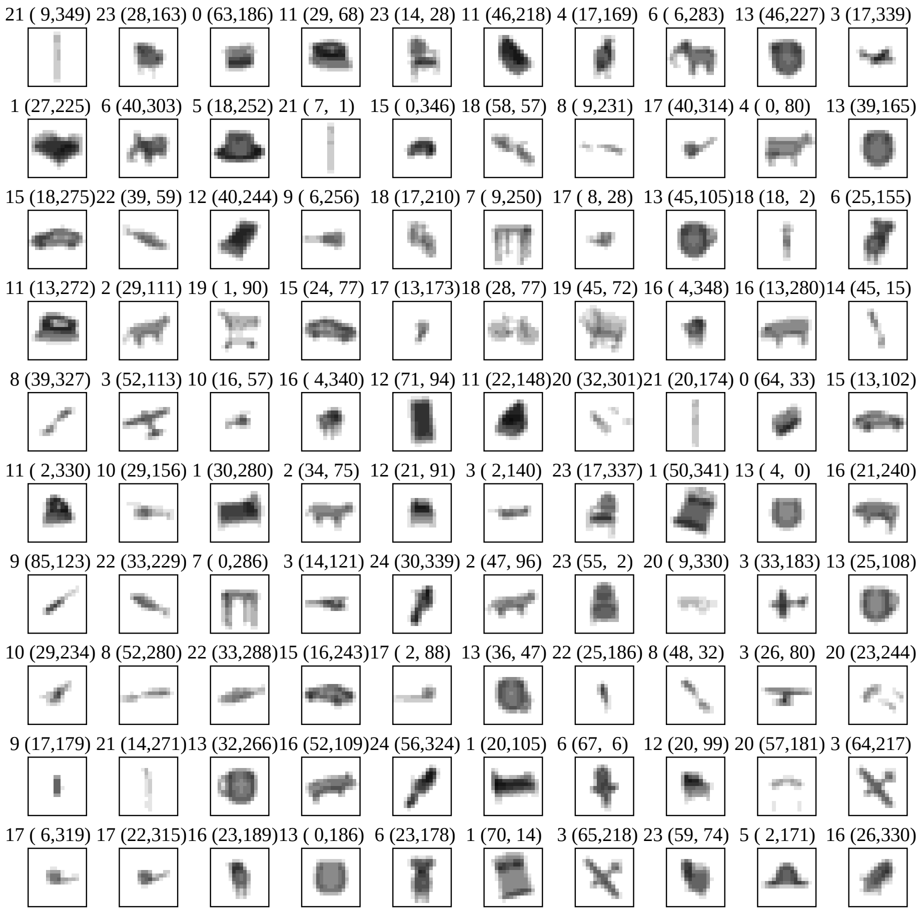

FIGURE A.6: A random subset of 100 training examples, taken from the entry level training set containing 100 views per object. The numbers above the images are the class labels and the two polar angles (θ, ϕ), measured in degrees, of the viewing direction in which the snapshots were taken. For object identities, see Fig. A.5.

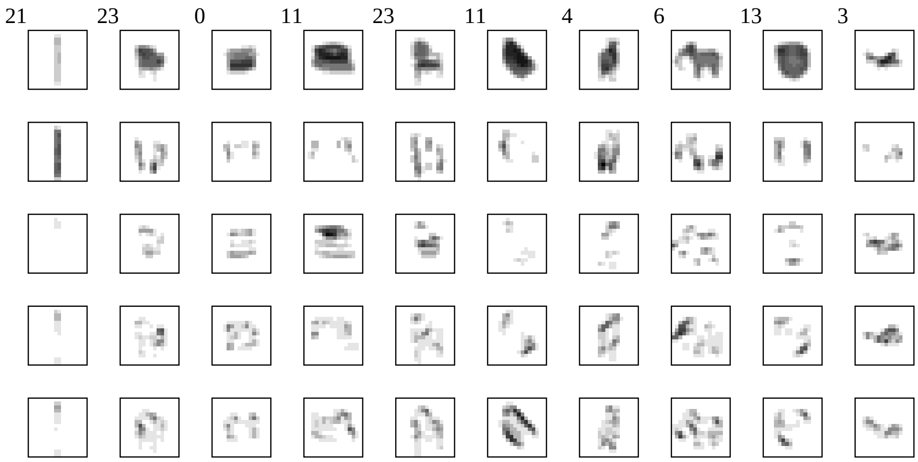

FIGURE A.7: Image (top row) and edge detection data computed with vertical, horizontal, and two diagonal filters (four bottom rows), for the first 10 images in Fig. A.6.

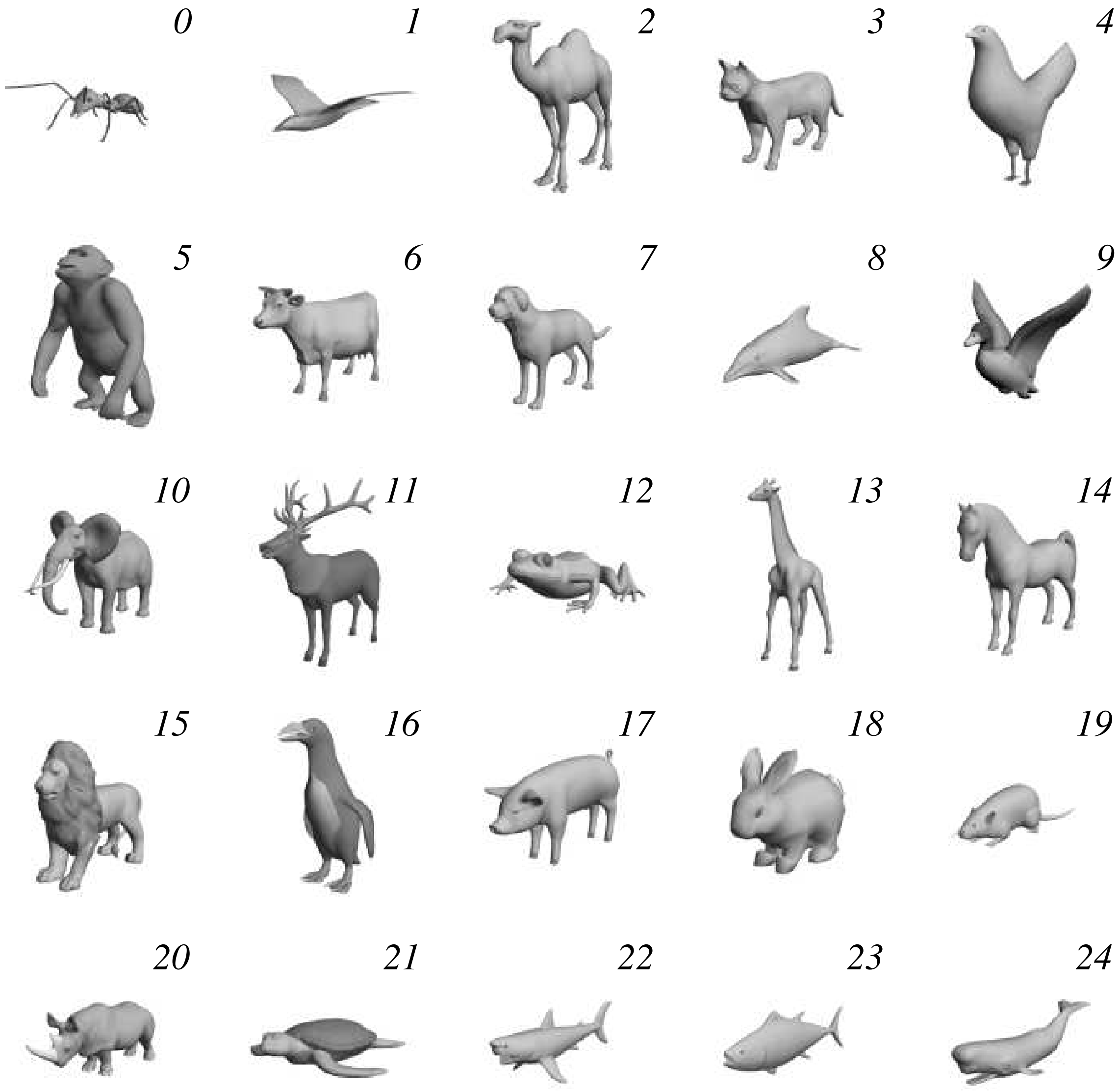

FIGURE A.8: Snapshots of the 3-D models in the MPI animal data base. The views shown are lit from the side; they have been used in a naming study (Liter et al., 1997). They were taken at elevation $\theta = 12°$ and azimuth $\phi = -40°$.

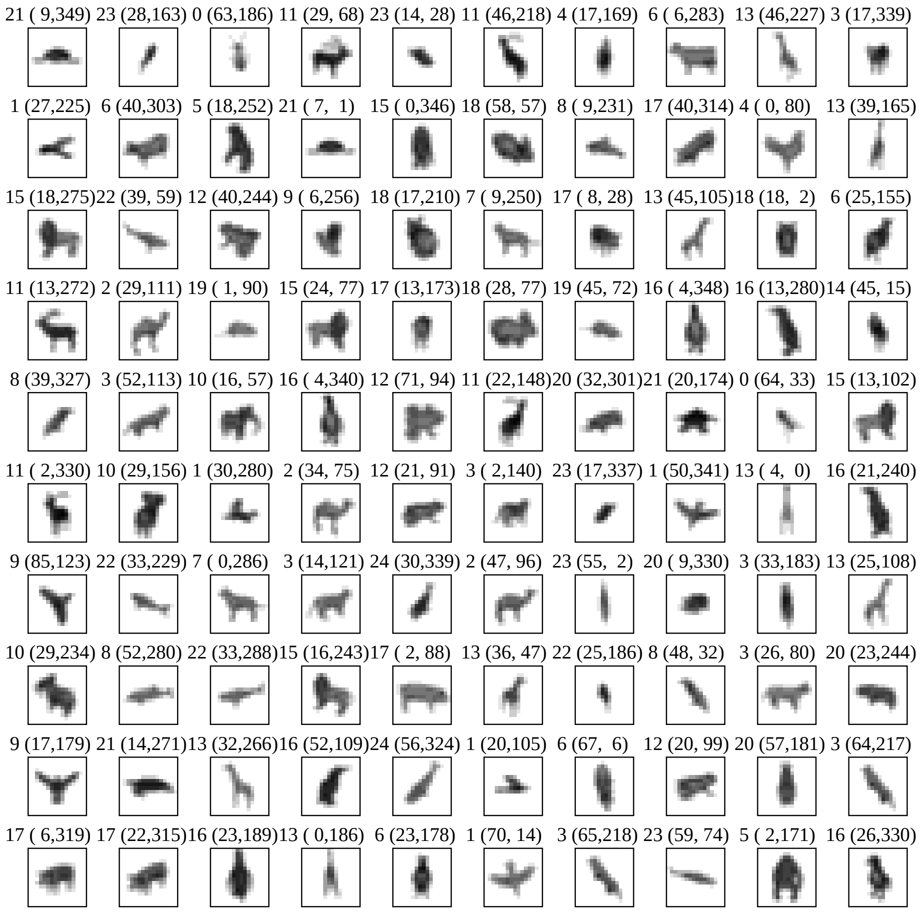

FIGURE A.9: A random subset of 100 training examples, taken from the animal training set containing 100 views per object. The numbers above the images are the class labels and the two polar angles (θ, ϕ), measured in degrees, of the viewing direction in which the snapshots were taken. For object identities, cf. Fig. A.8.

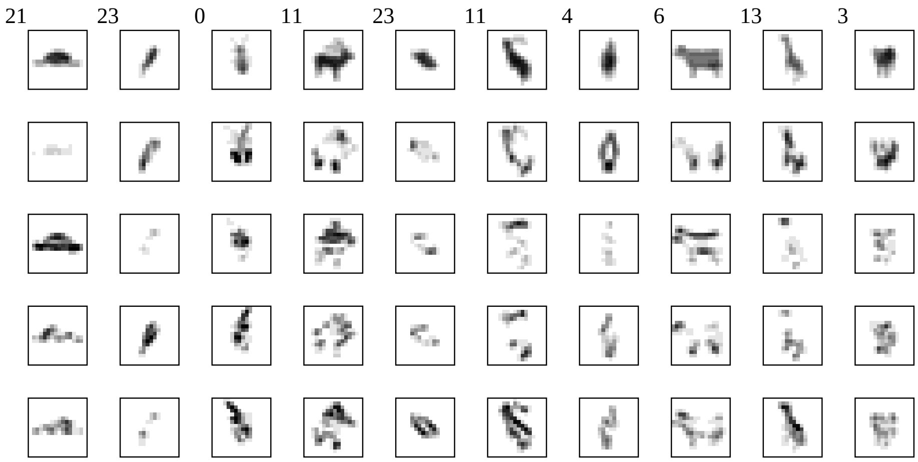

FIGURE A.10: Image (top row) and edge detection data computed with vertical, horizontal, and two diagonal filters (four bottom rows), for the first 10 images in Fig. A.9.

Appendix B

Object Recognition Results

In this part of the appendix, we present more detailed results of the experiments reported in Sec. 2.2.1. Without extensive discussion, we show plots of the type given in Figures 2.8 – 2.10, albeit separately for each object. Unlike in Fig. 2.9 and Fig. 2.10, the plots shown in the appendix do not include the distributions of errors. This is due to the fact that the error numbers of the individual binary object recognizers are mostly very small. Consequently, they do not admit reliable conclusions to be drawn. Note however, that the anisotropy effect, when summed over *all* binary classifiers, was even larger for the errors than it was for the SVs (Figures 2.8 – 2.10).

For lack of space, we cannot give detailed distribution plots for all results reported in Sec. 2.2.1. Instead, we focus on just one case, namely the polynomial classifier of degree 20, trained on the 100-view-per-object image databases (without edge detection data), and show plots for both entry level and animal data.

Although the detailed angular distributions of SVs shown in Figures B.2 – B.9 are provided mainly for reference purpose, we want to give a few examples of how to interpret them. For instance, Fig. B.6 shows that for most animals, the largest number of SVs is needed for viewing from the top. However, there are some exceptions, most notably the graphs 5, 12, and 21. If we look up object identities in Fig. A.8, this becomes plausible: objects 12 and 21 are a frog and a turtle, respectively. They share the trait of having a rather distinctive and easily recognizable top view, whereas their side views are harder than for most other animals. Object 5, the monkey, is very hard to recognize from the back, shown in graph 5 in Fig. B.7. Figures B.6 and B.7 show the SVs separated according to the binary classifier they belong to — remember that for instance the recognizer for frogs will have frogs as well as other animals as SVs, using them as positive and negative examples, respectively. In contrast, figures B.8 and B.9 separate the SVs into 25 groups according to their actual class membership. Thus, the fact that e.g. the distribution of frog SV polar angles (graph 12 in Fig. B.8) is less skewed towards the horizontal direction than the one in Fig. B.6 (the frog recognizer SVs) amounts to saying that many of the *horizontal* SVs of the frog classifier were not frogs, but many of the SVs which are views from *above* are frogs, i.e. positive examples.

Another instructive point can be made by examining objects which are shared by both databases. For instance, *consider the cat*, which is object number 2 in the entry level database (Fig. A.5) and object number 3 in the animal database (Fig. A.8). As an element of the animal database, SVs are fairly evenly distributed over all viewing azimuths (Figures B.7 and B.9). In the basic level database, on the other hand, where there are fewer similar objects that could be mistaken for a cat (as for instance the dog), the side view is more distinctive and accordingly requires a smaller fraction of the SVs. Moreover, the overall number of cat SVs drops. The SVs extracted thus strongly depend on the task that is being solved. In a world without dogs, cats are simpler to recognize.

For one of the 25 binary recognizers (number 21) constructed to build SV classifiers for the animal database, we show images of all Support Vectors. It can be seen that these comprise both positive examples (turtles, in this case) and negative examples. Some of the negative examples clearly could be mistaken for turtles, for other ones, this intuitive interpretation does not apply (Fig. B.1).

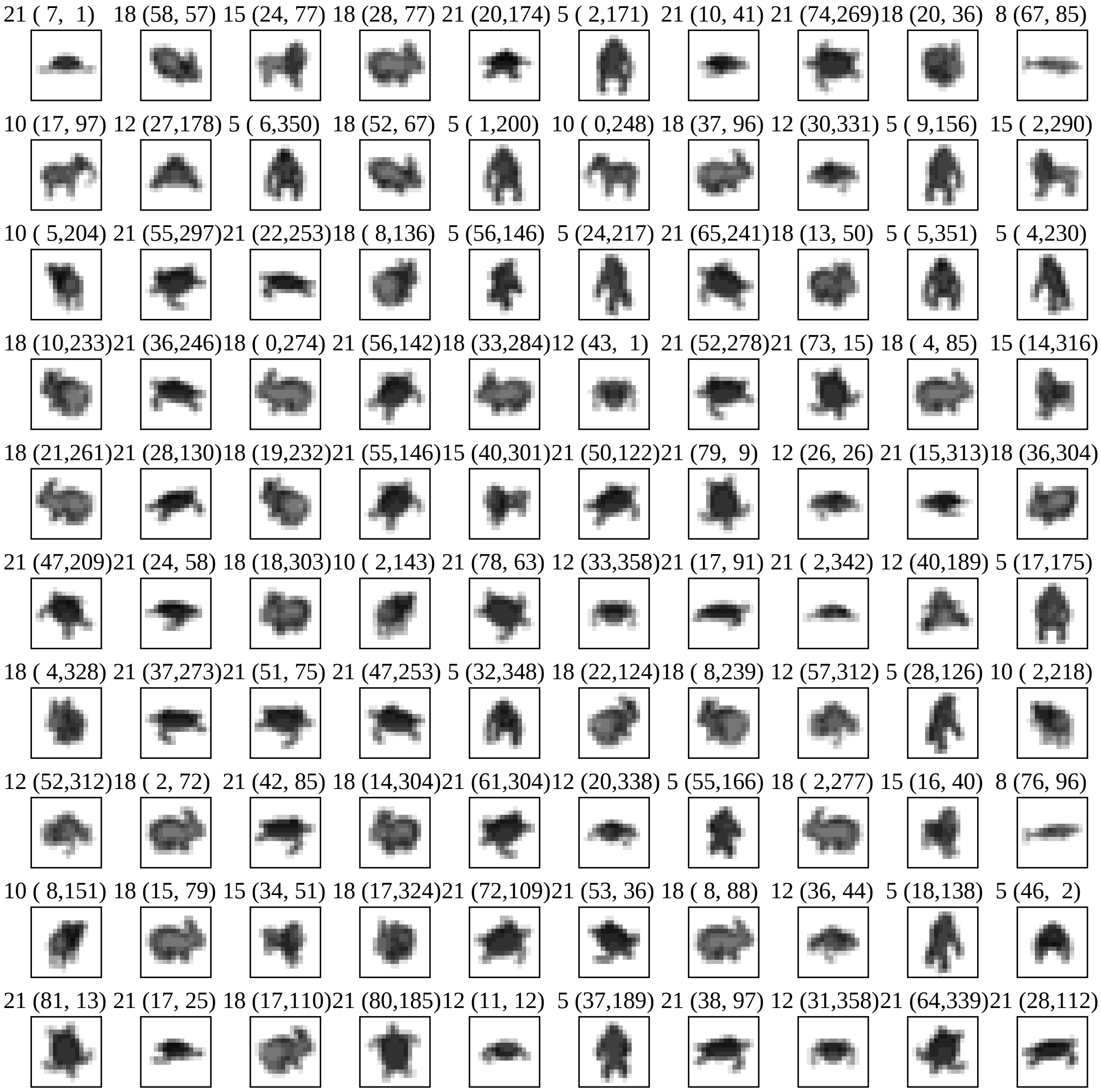

FIGURE B.1: Example of a SV set of a binary recognizer. In the depicted case (which was chosen simply because the size of the SV set happened to be 100), the recognizer's job is to recognize turtles (class 21). For object identities of other classes, cf. Fig. A.8. Out of the shown SVs, 34 are turtles, which is less than 50%, but more than the relative frequency of turtle views in the database (4%).

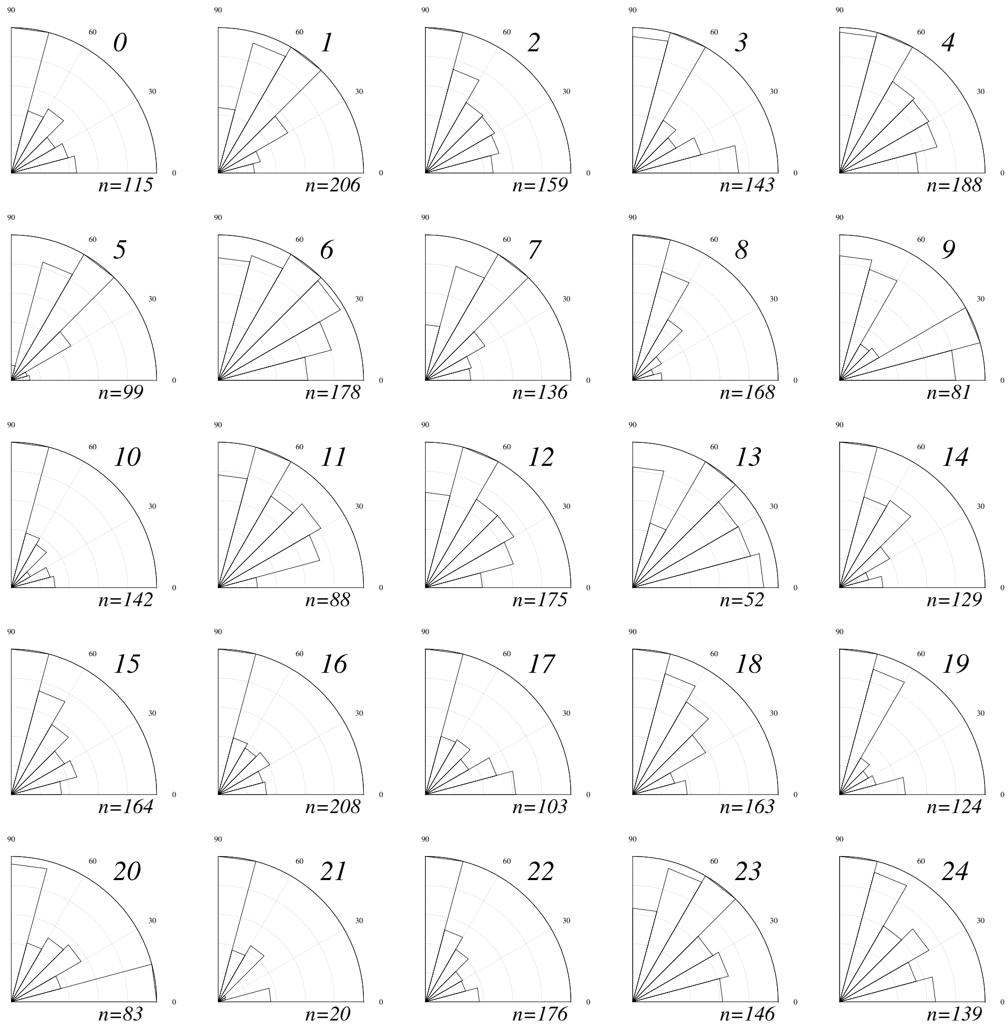

FIGURE B.2: SVs of entry level binary object recognizers, elevation θ (0° at the horizon, and 90° from the top, cf. Appendix A).

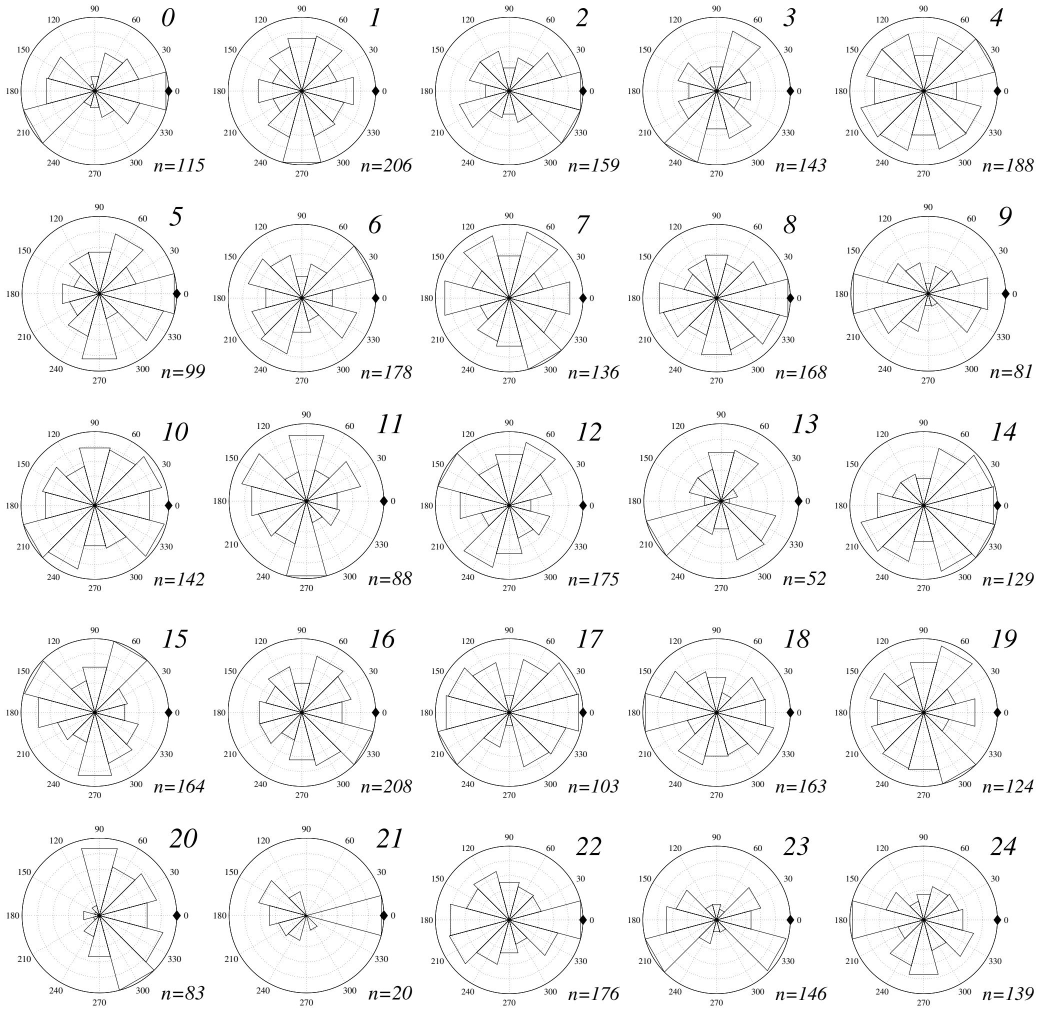

FIGURE B.3: SVs of entry level binary object recognizers, azimuth ϕ (cf. Appendix A)

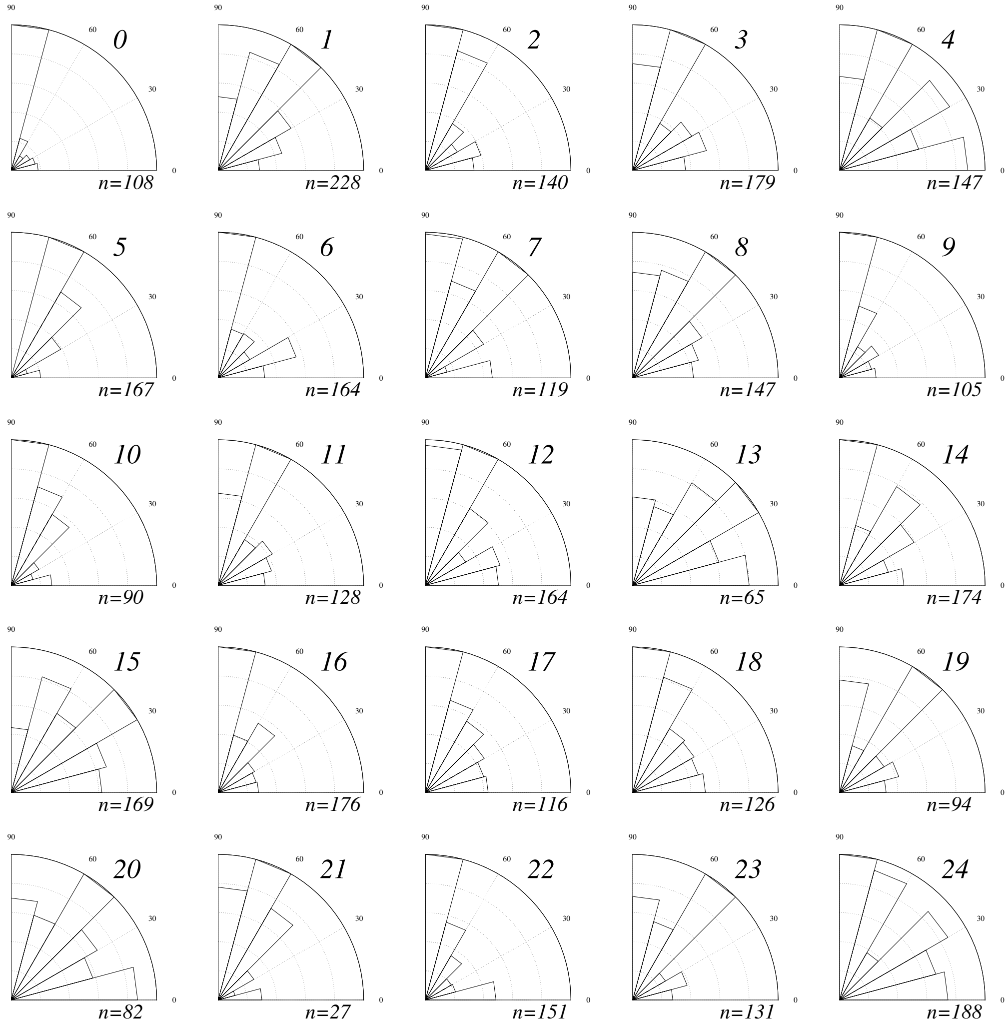

FIGURE B.4: SVs of each entry level object class, elevation θ

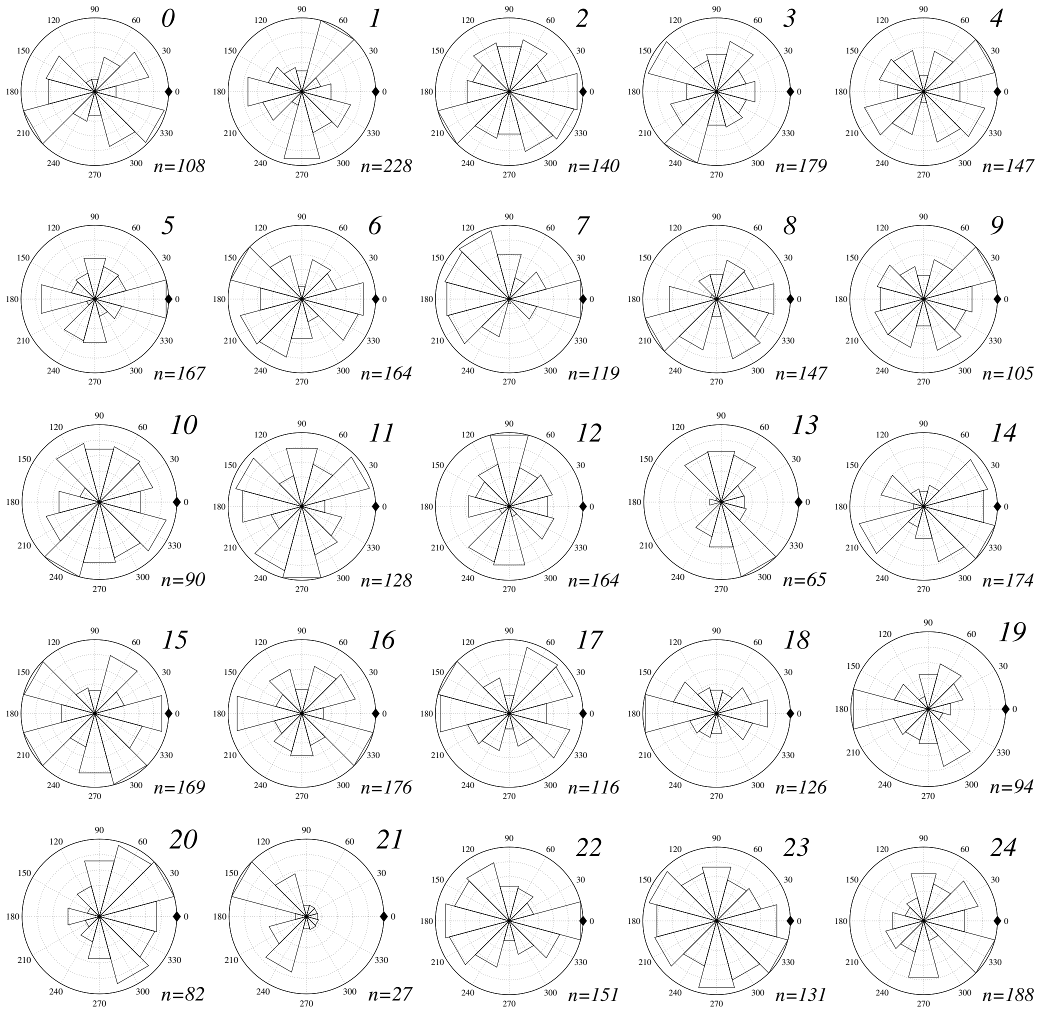

FIGURE B.5: SVs of each entry level object class, azimuth ϕ

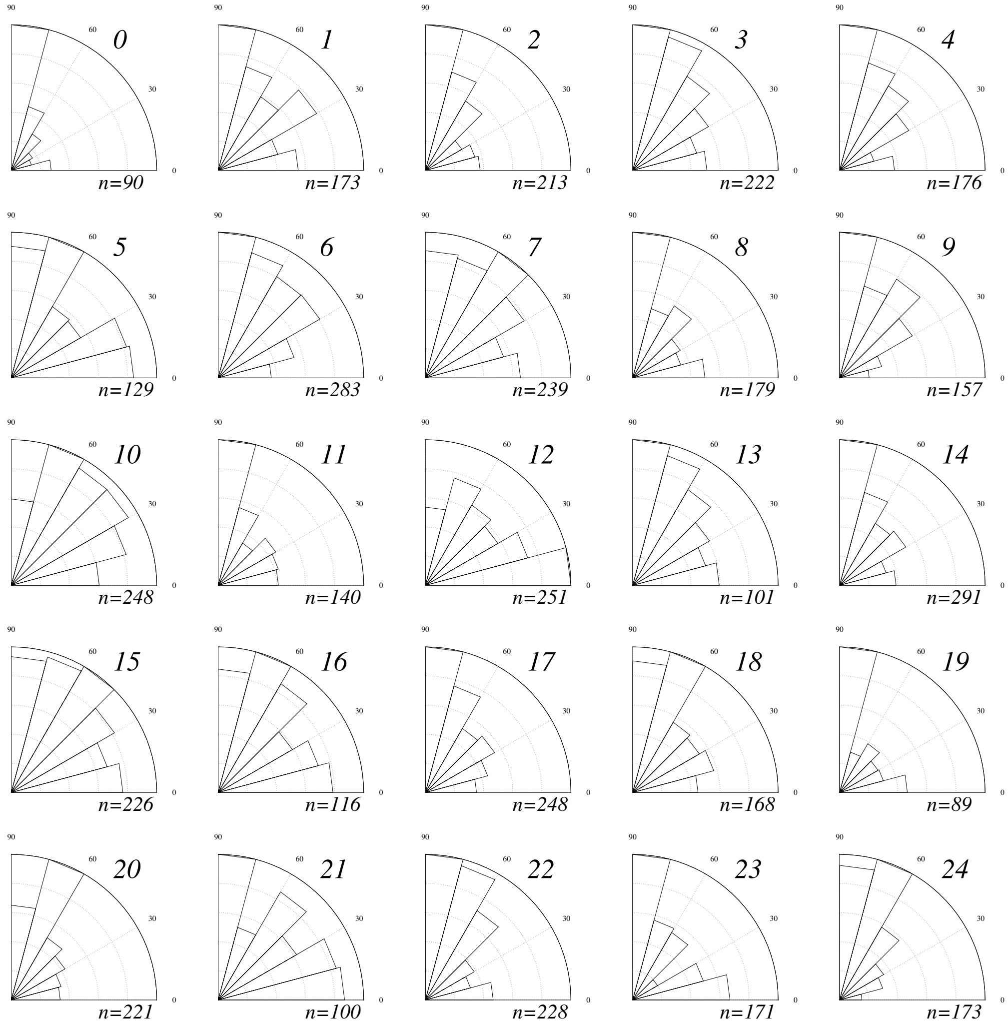

FIGURE B.6: SVs of binary animal recognizers, elevation θ

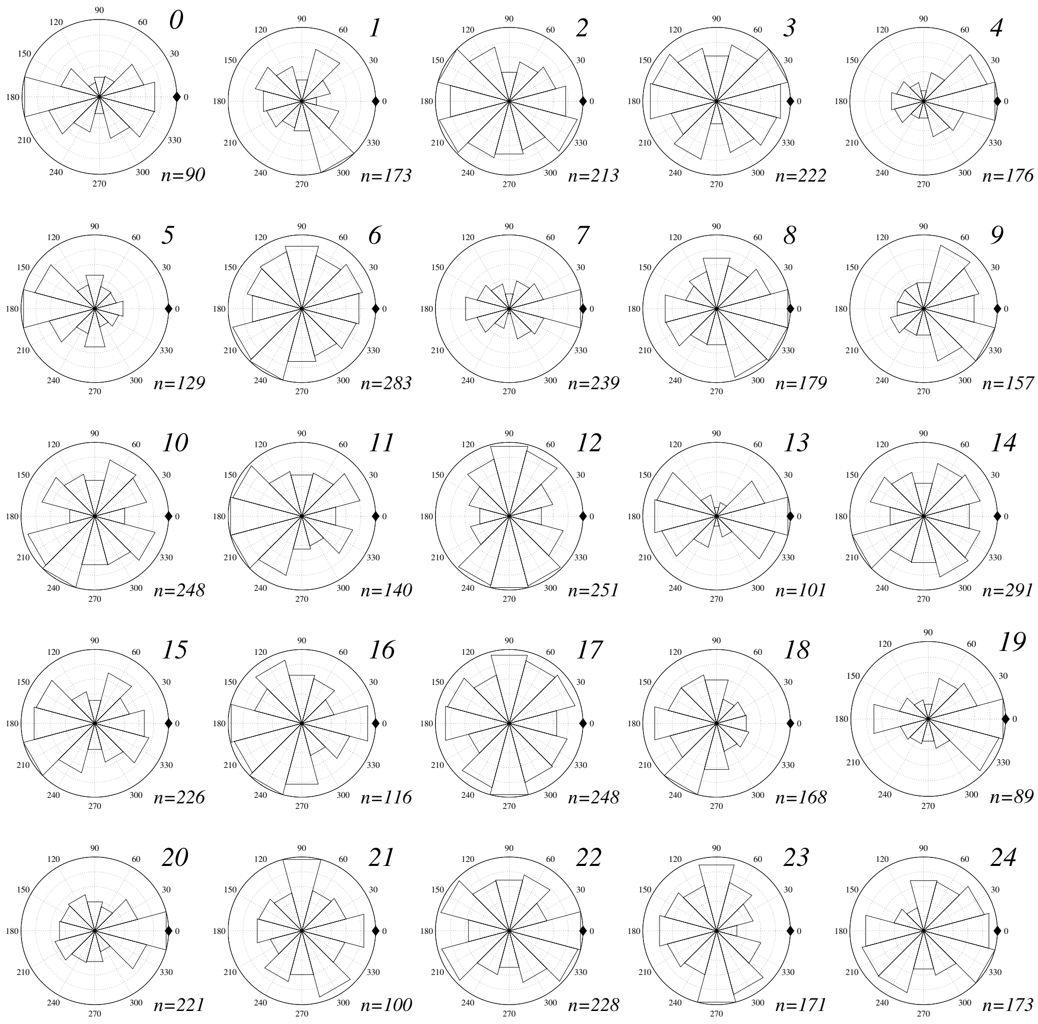

FIGURE B.7: SVs of binary animal recognizers, azimuth ϕ

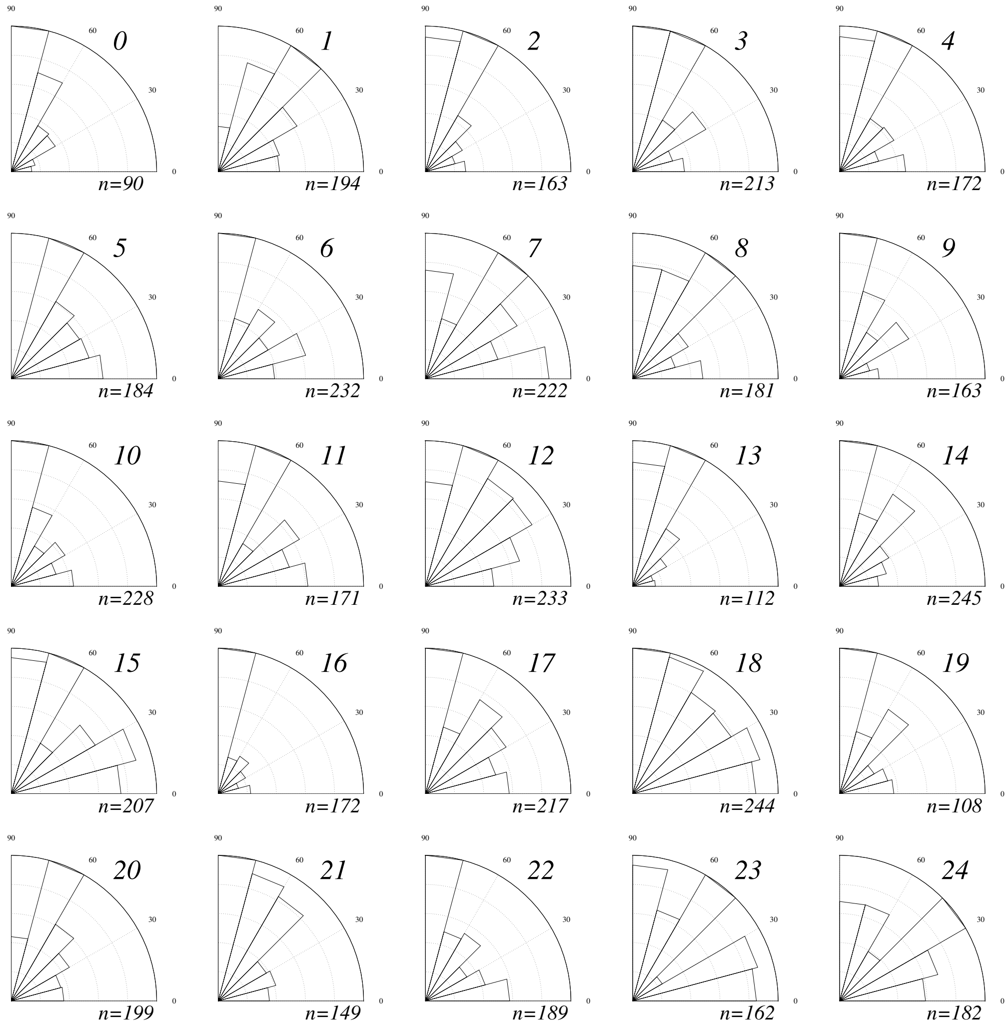

FIGURE B.8: SVs of each animal class, elevation θ

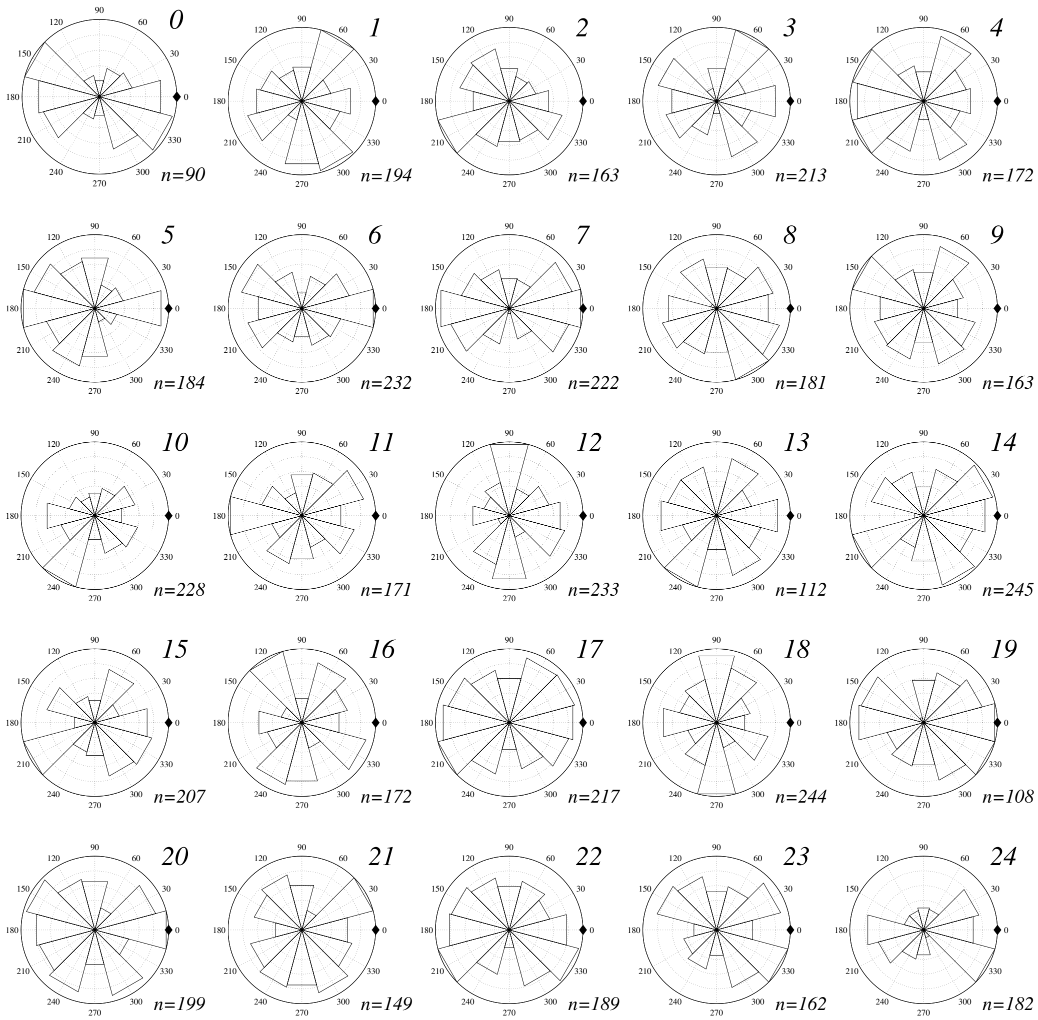

FIGURE B.9: SVs of each animal class, azimuth ϕ

Appendix C

Handwritten Character Databases

US Postal Service Database. The *US Postal Service (USPS)* database (see Fig. C.1) contains 9298 handwritten digits (7291 for training, 2007 for testing), collected from mail envelopes in Buffalo (cf. LeCun et al., 1989). Each digit is a 16×16 image, represented as a 256-dimensional vector with entries between -1 and 1. Preprocessing consisted of smoothing with a Gaussian kernel of width $\sigma = 0.75$.

It is known that the USPS test set is rather difficult — the human error rate is 2.5% (Bromley and Säckinger, 1991). For a discussion, see (Simard, LeCun, and Denker, 1993). Note, moreover, that some of the results reported in the literature for the USPS set have been obtained with an enhanced training set: for instance, Drucker, Schapire, and Simard (1993) used an enlarged training set of size 9709, containing some additional machine-printed digits, and note that this improves accuracies. In our experiments, only 7291 training examples were used.

MNIST Database. The *MNIST* database (Fig. C.2) contains 120000 handwritten digits, equally divided into training and test set. The database is a modified version of *NIST Special Database 3* and *NIST Test Data 1*. Training and test set consist of patterns generated by different writers. The images were first size normalized to fit into a 20×20 pixel box, and then centered in a 28×28 image (Bottou et al., 1994).

Test results on the MNIST database which are given in the literature (e.g. Bottou et al., 1994; LeCun et al., 1995) for some reason do not use the full MNIST test set of 60000 characters. Instead, a subset of 10000 characters is used, consisting of the test set patterns from 24476 to 34475. To obtain results which can be compared to the literature, we also use this test set, although the larger one is preferable from the point of view of obtaining more reliable test error estimates.

Small MNIST Database. The USPS database has been criticised (Burges, LeCun, private communication; Bottou et al. (1994)) as not providing the most adequate classifier benchmark. First, it only comes with a small test set, and second, the test set contains a number of corrupted patterns which not even humans can classify correctly. The MNIST database, which is the currently used classifier benchmark in the AT&T and Bell Labs learning research groups, does not have these drawbacks; moreover, its

FIGURE C.1: The first 100 USPS training images, with class labels.

training set is much larger. In some cases, however, it is useful to be able to study an algorithm on a small database. First, this can save computation time, and second, this allows the study of learning from smaller sample sizes. We thus decided to generate a smaller version of the MNIST database which we used in some of the more recent experiments. It is smaller in two respects: first, the patterns are used in a resolution of 20×20, obtained from the above 28×28 patterns by downsampling (combined with a Gaussian smoothing of standard deviation 0.75 pixels, to avoid aliasing effects). Second, it only comprises a subset of the training set, namely the first 5000 patterns. As test set, we use the same 10000 test examples as above. We refer to this database as the *small MNIST* database (Fig. C.3).

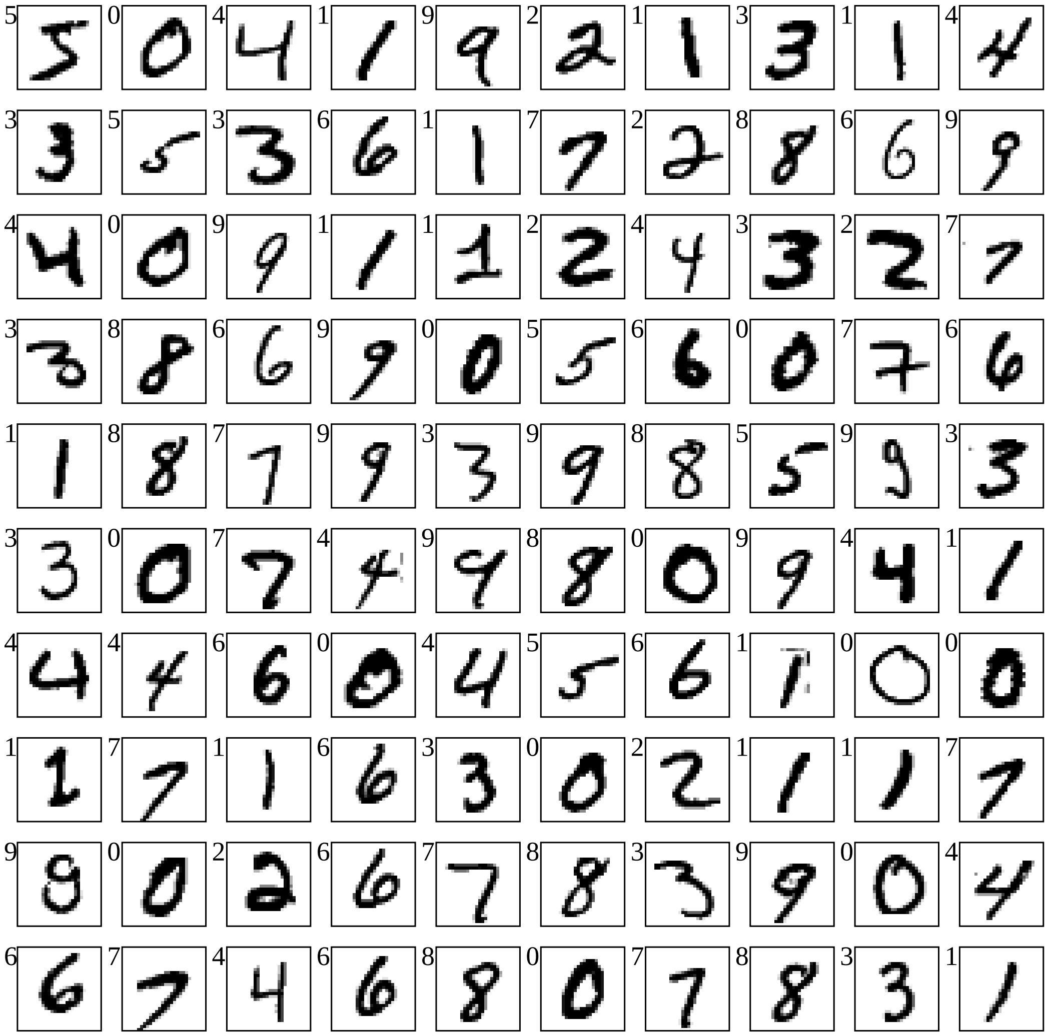

FIGURE C.2: The first 100 MNIST training images, with class labels.

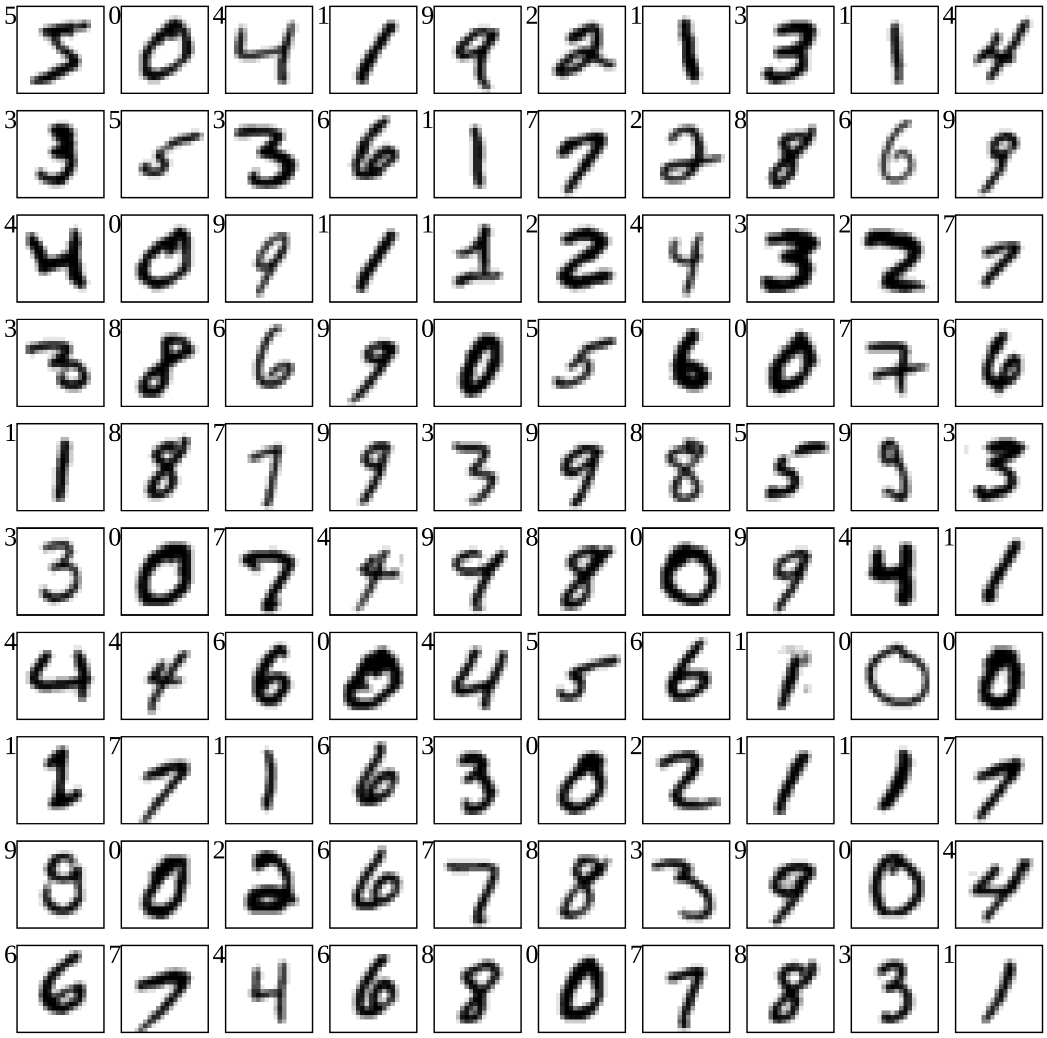

FIGURE C.3: The first 100 small-MNIST training images, with class labels.

Appendix D

Technical Addenda

D.1 Feature Space and Kernels

In this section, we collect some material related to Mercer kernels and the correspondig feature spaces. If not stated otherwise, we assume that k is a Mercer kernel (cf. Proposition 1.3.2), and Φ is the corresponding map into a feature space F such that $k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$.

D.1.1 The Reduced Set Method

Given a vector $\Psi \in F$, written in terms of images of input patterns,

$$\Psi = \sum_{i=1}^{\ell} \alpha_i \Phi(\mathbf{x}_i), \tag{D.1}$$

with $\alpha_i \in \mathbf{R}$, one can try to approximate it by

$$\Psi' = \sum_{i=1}^{N_z} \beta_i \Phi(\mathbf{z}_i), \tag{D.2}$$

with $N_z << \ell$, $\beta_i \in \mathbf{R}$. To this end, we have to minimize

$$\rho = \|\Psi - \Psi'\|^2. \tag{D.3}$$

The crucial point is that even if Φ is not given explicitely, ρ can be computed (and minimized) in terms of kernels, using $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})) = k(\mathbf{x}, \mathbf{y})$ (Burges, 1996).

In Sec. 4.4.1, this method is used to approximate Support Vector decision boundaries in order to speed up classification.

D.1.2 Inverting the Map Φ

If Φ is nonlinear, the dimension of the linear span of the Φ-images of a set of input vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$ can exceed the dimension of their span in input space. Thus, we need not expect that there is a pre-image under Φ for each vector that can be expressed as a linear combination of the vectors $\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_\ell)$. Nevertheless, it might be desirable to have a means of constructing the pre-image in the case where it does exist.

To this end, suppose we have a vector in F given in terms of an expansion of images of input data, with an unknown pre-image $\mathbf{x}_0$ under Φ in input space $\mathbf{R}^N$, i.e.

$$\Phi(\mathbf{x}_0) = \sum_{j=1}^{\ell} \alpha_j \Phi(\mathbf{x}_j). \tag{D.4}$$

Then, for any $\mathbf{x} \in \mathbf{R}^N$,

$$k(\mathbf{x}_0, \mathbf{x}) = \sum_{j=1}^{\ell} \alpha_j k(\mathbf{x}_j, \mathbf{x}). \tag{D.5}$$

Assume moreover that the kernel $k(\mathbf{x}, \mathbf{y})$ is an invertible function f_k of $(\mathbf{x} \cdot \mathbf{y})$,

$$k(\mathbf{x}, \mathbf{y}) = f_k((\mathbf{x} \cdot \mathbf{y})), \tag{D.6}$$

e.g. $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$ with odd d, or $k(\mathbf{x}, \mathbf{y}) = \sigma((\mathbf{x} \cdot \mathbf{y}) + \Theta)$ with a strictly monotonic sigmoid function σ and a threshold Θ. Given any a priori chosen basis of input space $\{\mathbf{e}_1, \ldots, \mathbf{e}_N\}$, we can then expand $\mathbf{x}_0$ as

$$\begin{aligned} \mathbf{x}_0 &= \sum_{i=1}^{N} (\mathbf{x}_0 \cdot \mathbf{e}_i) \mathbf{e}_i \\ &= \sum_{i=1}^{N} f_k^{-1}(k(\mathbf{x}_0, \mathbf{e}_i)) \mathbf{e}_i \\ &= \sum_{i=1}^{N} f_k^{-1} \left(\sum_{j=1}^{\ell} \alpha_j k(\mathbf{x}_j, \mathbf{e}_i) \right) \mathbf{e}_i. \end{aligned} \tag{D.7}$$

By using (D.5), we thus reconstructed $\mathbf{x}_0$ from the values of dot products between images (in F) of training examples and basis elements.

Clearly, a crucial assumption in this construction was the existence of the pre-image $\mathbf{x}_0$. If this does not hold, then the discrepancy

$$\sum_{j=1}^{\ell} \alpha_j \Phi(\mathbf{x}_j) - \Phi(\mathbf{x}_0) \tag{D.8}$$

will be nonzero. There is a number of things that we could do to make the discrepancy small:

(a) We can try to find a suitable basis in which we expand the pre-images.

(b) We can repeat the scheme, by trying to find a pre-image for the discrepancy vector. This problem has precisely the same structure as the original one (D.4), with one more term in the summation on the right hand side. Iterating this method gives an expansion of the vector in F in terms of reconstructed approximate pre-images.

(c) We have the freedom to choose the scaling of the vector in F. To see this, note that for any nonzero α, we have, similar to (D.7),

$$\mathbf{x}_0 = \sum_{i=1}^{N} \alpha \cdot (\mathbf{x}_0 \cdot \mathbf{e}_i/\alpha)\mathbf{e}_i = \sum_{i=1}^{N} \alpha f_k^{-1} \left(\sum_{j=1}^{\ell} \alpha_j k(\mathbf{x}_j, \mathbf{e}_i/\alpha) \right) \mathbf{e}_i. \tag{D.9}$$

(d) Related to this scaling issue, we could also have started with

$$\beta\Phi(\mathbf{x}_0) = \sum_{j=1}^{\ell} \alpha_j \Phi(\mathbf{x}_j), \tag{D.10}$$

obtaining a reconstruction (cf. (D.7))

$$\mathbf{x}_0 = \sum_{i=1}^{N} f_k^{-1} \left(\sum_{j=1}^{\ell} \frac{\alpha_j}{\beta} \cdot k(\mathbf{x}_j, \mathbf{e}_i) \right) \mathbf{e}_i \tag{D.11}$$

with the property that (D.10) holds *if* such an $\mathbf{x}_0$ exists.

The success of using different values of β or α could be monitored by computing the squared norm of the discrepancy,

$$\left\| \sum_{j=1}^{\ell} \alpha_j \Phi(\mathbf{x}_j) - \beta\Phi(\mathbf{x}_0) \right\|^2, \tag{D.12}$$

which can be evaluated in terms of the kernel function.

Finally, we note that same approach can also be applied for more general kernel functions which cannot be written as an invertible function of $(\mathbf{x} \cdot \mathbf{y})$. All we need is a kernel which allows the reconstruction of $(\mathbf{x} \cdot \mathbf{y})$ — and nothing prevents us from requiring the evaluation of the kernel on several pairs of points for this purpose. Consider the following example: assume that

$$k(\mathbf{x}, \mathbf{y}) = f_k \left(\|\mathbf{x} - \mathbf{y}\|^2 \right) \tag{D.13}$$

with an invertible f_k (e.g., if k is a Gaussian RBF function, cf. (1.28)). Then, by the polarization identity, we have

$$(x_0 \cdot \mathbf{e}_i) = \frac{1}{4} \left(\|\mathbf{x}_0 + \mathbf{e}_i\|^2 - \|\mathbf{x}_0 - \mathbf{e}_i\|^2 \right) = \frac{1}{4} \left(f_k^{-1}(k(\mathbf{x}_0, -\mathbf{e}_i)) - f_k^{-1}(k(\mathbf{x}_0, \mathbf{e}_i)) \right). \tag{D.14}$$

The same also works if $k(\mathbf{x}, \mathbf{y}) = f_k(\|\mathbf{x} - \mathbf{y}\|)$, e.g.: we just have to raise the results of f_k^{-1} to the power of 2.

Similar methods can be applied to deal with other kernels.

D.1.3 Mercer Kernels

In this section, we give some further material related to Sec. 1.3.

First, we mention that if a finite number of Eigenvalues is negative, the expansion (1.25) is still valid. In that case, k corresponds to a Lorentzian symmetric bilinear form in a space with indefinite signature. For the SV algorithm, this would entail problems, as the optimization problem would become indefinite. The diagonalization required for kernel PCA, however, can still be performed, and (3.16) can be modified such that it allows for negative Eigenvalues. The main difference is that we can no longer interpret the method as PCA in some feature space. Nevertheless, it could still be viewed as a type of nonlinear factor analysis.

Next, we note that the polynomial kernels given in (1.17) satisfy Mercer's conditions of Proposition 1.3.2. As compositions of continuous functions, they are continuous, thus we only need to show positivity, which follows immediately if we consider their dot product representation

$$(\mathbf{x} \cdot \mathbf{y})^d = \sum_{i=1}^{N_F} \left(\Phi_d(\mathbf{x})\right)_i \left(\Phi_d(\mathbf{y})\right)_i . \tag{D.15}$$

Namely, more generally, if an integral operator kernel k admits a uniformly convergent dot product representation on some compact set $C \times C$,

$$\sum_{i=1}^{\infty} \phi_i(\mathbf{x})\phi_i(\mathbf{y}), \tag{D.16}$$

it is necessarily positive: for $f \in L^2(C)$, we have

$$\begin{aligned}
&\int_{C\times C} \left(\sum_{i=1}^{\infty} \phi_i(\mathbf{x})\phi_i(\mathbf{y})\right) f(\mathbf{x}) f(\mathbf{y})\, d\mathbf{x}\, d\mathbf{y} \\
= &\sum_{i=1}^{\infty} \int_{C\times C} \phi_i(\mathbf{x}) f(\mathbf{x}) \phi_i(\mathbf{y}) f(\mathbf{y})\, d\mathbf{x}\, d\mathbf{y} \qquad \text{(D.17)} \\
= &\sum_{i=1}^{\infty} \left(\int_C \phi_i(\mathbf{x}) f(\mathbf{x})\, d\mathbf{x}\right)^2 \geq 0, \qquad \text{(D.18)}
\end{aligned}$$

establishing the converse of Proposition 1.3.2.

We conclude this section with some considerations on Proposition 1.3.3. Is it possible to give a more general class of kernels, such that the expansion (1.25) is no longer valid, but the mapping of Proposition 1.3.3 can still be constructed? One would expect that if k does not correspond to a compact operator (as it did in the case of Mercer kernels, cf. Dunford and Schwartz (1963); in fact, in the Mercer case, we even have trace class operators, cf. Nashed and Wahba (1974)), with a discrete spectrum, then the mapping (1.26) should no longer map into an l^2 space, but into some separable Hilbert space of functions on a non-discrete measure space.

To this end, let ϕ be a map from input space into some Hilbert space H,

$$\phi : \mathbf{x} \mapsto f_{\mathbf{x}}, \tag{D.19}$$

and $T \geq 0$ be a positive bounded operator on H. Moreover, define a kernel

$$k_T(\mathbf{x}, \mathbf{y}) := (f_{\mathbf{x}} \cdot T f_{\mathbf{y}}). \tag{D.20}$$

Then

$$\Phi : \mathbf{x} \mapsto \sqrt{T}\, f_{\mathbf{x}} \tag{D.21}$$

clearly is a map such that

$$k_T(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})). \tag{D.22}$$

As an aside, to see the connection to Mercer's theorem, we may formally set $f_{\mathbf{x}}$ to be $\delta_{\mathbf{x}}$, and assume that T is an integral operator with kernel k. In this case, the right hand side of (D.20) would equal $k(\mathbf{x}, \mathbf{y})$.

The connection to (1.26) becomes clearer if we use the spectral representation of T, and construct a Φ different from the one in (D.21): T can be written as

$$T = U^* M_v U, \tag{D.23}$$

where v is a continuous function with corresponding multiplication operator M_v, U is a unitary operator

$$U : H \to L^2(\mathbf{R}, \mu), \tag{D.24}$$

and μ is a probability measure (the spectral measure of T) (e.g. Reed and Simon, 1980). Since $T \geq 0$, we have $M_v \geq 0$ and $v \geq 0$. Then, for all $\mathbf{x}$ and $\mathbf{y}$,

$$\begin{aligned}
k_T(\mathbf{x}, \mathbf{y}) &= (f_{\mathbf{x}} \cdot U^* M_v U f_{\mathbf{y}}) && \text{(D.25)}\\
&= (U f_{\mathbf{x}} \cdot M_v U f_{\mathbf{y}}) && \text{(D.26)}\\
&= (\sqrt{M_v} U f_{\mathbf{x}} \cdot \sqrt{M_v} U f_{\mathbf{y}}) && \text{(D.27)}\\
&= (M_{\sqrt{v}} U f_{\mathbf{x}} \cdot M_{\sqrt{v}} U f_{\mathbf{y}}) && \text{(D.28)}\\
&= (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})), && \text{(D.29)}
\end{aligned}$$

defining

$$\begin{aligned}
\Phi : \mathbf{R}^N &\to L^2(\mathbf{R}, \mu) && \text{(D.30)}\\
\Phi(\mathbf{x}) &= M_{\sqrt{v}} U f_{\mathbf{x}}. && \text{(D.31)}
\end{aligned}$$

To see the relationship to (1.26), it should be noted that the spectrum of T coincides with the essential range of v.

For simplicity, we have above made the assumption that T is bounded. The same argument, however, also works in the case of unbounded T (e.g. Reed and Simon, 1980).

For the purpose of practical applications, we are interested in maps ϕ and operators $T \geq 0$ such that the kernel k defined by (D.20) can be computed analytically.

Without going into detail, we briefly mention an example of a map ϕ. Define

$$\phi : \mathbf{x} \mapsto k(\mathbf{x}, .), \tag{D.32}$$

where k is some a priori specified kernel, and $T = P^*P$, with a regularization operator P (Tikhonov and Arsenin, 1977). Then

$$k_T(\mathbf{x}, \mathbf{y}) = ((Pk)(\mathbf{x}, .) \cdot (Pk)(\mathbf{y}, .)) \tag{D.33}$$

coincides with a dot product matrix arising in a kernel-based regularization framework for learning problems (Smola and Schölkopf, 1997b). If k is chosen as Green's function of P^*P, then k_T and k can be shown to coincide, and the regularization approach is equivalent to the SV approach (Smola, Schölkopf, and Müller, 1997).

D.1.4 Polynomial Kernels and Higher Order Correlations

Consider the mappings corresponding to kernels of the form (1.20): suppose the monomials $x_{i_1} x_{i_2} \dots x_{i_d}$ are written such that $i_1 \leq i_2 \leq \dots \leq i_d$. Then the coefficients (as the $\sqrt{2}$ in (1.21)), arising from the fact that different combinations of indices occur with different frequencies, are largest for $i_1 < i_2 < \dots < i_d$ (let us assume here that the input dimensionality is not smaller than the polynomial degree d): in that case, we have a coefficient of $\sqrt{d!}$. If $i_1 = i_2$, say, the coefficient will be $\sqrt{(d-1)!}$. In general, if n of the x_i are equal, and the remaining ones are different, then the coefficient in the corresponding component of Φ is $\sqrt{(d-n+1)!}$. Thus, the terms belonging to the d-th order correlations will be weighted with an extra factor $\sqrt{d!}$ compared to the terms x_i^d, and compared to the terms where only $d-1$ different components occur, they are still weighted stronger by $\sqrt{d}$. Consequently, kernel PCA with polynomial kernels will tend to pick up variance in the d-th order correlations mainly.

D.2 Kernel Principal Component Analysis

D.2.1 The Eigenvalue Problem in the Space of Expansion Coefficients

We presently give a justification for solving (3.14) rather than (3.13) in computing the Eigensystem of the covariance matrix in F (cf. Sec. 3.2).

Being symmetric, K has an orthonormal basis of Eigenvectors $(\boldsymbol{\beta}^i)_i$ with corresponding Eigenvalues μ_i, thus for all i, we have $K\boldsymbol{\beta}^i = \mu_i\boldsymbol{\beta}^i$ $(i = 1,\dots,M)$. To understand the relation between (3.13) and (3.14), we proceed as follows: first suppose $\lambda, \boldsymbol{\alpha}$ satisfy (3.13). We may expand $\boldsymbol{\alpha}$ in K's Eigenvector basis as

$$\boldsymbol{\alpha} = \sum_{i=1}^{M} a_i\boldsymbol{\beta}^i. \tag{D.34}$$

Equation (3.13) then reads

$$M\lambda \sum_i a_i\mu_i\boldsymbol{\beta}^i = \sum_i a_i\mu_i^2\boldsymbol{\beta}^i, \tag{D.35}$$

or, equivalently, for all $i = 1,\dots,M$,

$$M\lambda a_i\mu_i = a_i\mu_i^2. \tag{D.36}$$

This in turn means that for all $i = 1,\dots,M$,

$$M\lambda = \mu_i \quad \mathit{or} \quad a_i = 0 \quad \mathit{or} \quad \mu_i = 0. \tag{D.37}$$

Note that the above are not exclusive *or*-s. We next assume that $\lambda, \boldsymbol{\alpha}$ satisfy (3.14). In that case, we find that (3.14) is equivalent to

$$M\lambda \sum_i a_i\boldsymbol{\beta}^i = \sum_i a_i\mu_i\boldsymbol{\beta}^i, \tag{D.38}$$

i.e. for all $i = 1,\dots,M$,

$$M\lambda = \mu_i \quad \mathit{or} \quad a_i = 0. \tag{D.39}$$

Comparing (D.37) and (D.39), we see that all solutions of the latter satisfy the former. However, they do not give its full set of solutions: given a solution of (3.14), we may always add multiples of Eigenvectors of K with Eigenvalue $\mu_i = 0$ and still satisfy (3.13), with the same Eigenvalue.[1] Note that this means that there exist solutions of (3.13) which belong to different Eigenvalues yet are not orthogonal in the space of the $\boldsymbol{\alpha}^k$ (for instance, take any two Eigenvectors with different Eigenvalues, and add a multiple of the same Eigenvector with Eigenvalue 0 to both of them). This, however, does not mean that the Eigenvectors of $\bar{C}$ in F are not orthogonal. Indeed, note that if $\boldsymbol{\alpha}$ is an Eigenvector of K with Eigenvalue 0, then the corresponding vector $\sum_i \alpha_i\Phi(\mathbf{x}_i)$ is orthogonal to *all* vectors in the span of the $\Phi(\mathbf{x}_j)$ in F, since

$$\left(\Phi(\mathbf{x}_j) \cdot \sum_i \alpha_i\Phi(\mathbf{x}_i)\right) = (K\boldsymbol{\alpha})_j = 0 \text{ for all } j, \tag{D.40}$$

which means that $\sum_i \alpha_i\Phi(\mathbf{x}_i) = 0$. Thus, the above difference between the solutions of (3.13) and (3.14) is not relevant, since we are interested in vectors in F rather than vectors in the space of the expansion coefficients of (3.10). We therefore only need to diagonalize K in order to find all relevant solutions of (3.13).

Note, finally, that the rank of K determines the dimensionality of the span of the $\Phi(\mathbf{x}_j)$ in F, i.e. of the subspace that we are working in.

[1]This observation could be used to change the vectors $\boldsymbol{\alpha}$ of the solution, e.g. to make them maximally sparse, without changing the solution.

D.2.2 Centering in Feature Space

In Sec. 3.2, we made the assumption that our mapped data is centered in F, i.e.

$$\sum_{n=1}^{M} \Phi(\mathbf{x}_n) = 0. \tag{D.41}$$

We shall now drop this assumption. First note that given any Φ and any set of observations $\mathbf{x}_1, \ldots, \mathbf{x}_M$, the points

$$\tilde{\Phi}(\mathbf{x}_i) := \Phi(\mathbf{x}_i) - \frac{1}{M}\sum_{i=1}^{M} \Phi(\mathbf{x}_i) \tag{D.42}$$

are centered. Thus, the assumptions of Sec. 3.2 now hold, and we go on to define covariance matrix and $\tilde{K}_{ij} = (\tilde{\Phi}(\mathbf{x}_i) \cdot \tilde{\Phi}(\mathbf{x}_j))$ in F. We arrive at our already familiar Eigenvalue problem

$$\tilde{\lambda}\tilde{\boldsymbol{\alpha}} = \tilde{K}\tilde{\boldsymbol{\alpha}}, \tag{D.43}$$

with $\tilde{\boldsymbol{\alpha}}$ being the expansion coefficients of an Eigenvector (in F) in terms of the points (D.42),

$$\tilde{\mathbf{V}} = \sum_{i=1}^{M} \tilde{\alpha}_i \tilde{\Phi}(\mathbf{x}_i). \tag{D.44}$$

We cannot compute $\tilde{K}$ directly; however, we can express it in terms of its non-centered counterpart K. In the following, we shall use $K_{ij} = (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$, in addition, we shall make use of the notation $1_{ij} = 1$ for all i, j.

$$\begin{aligned}
\tilde{K}_{ij} &= (\tilde{\Phi}(\mathbf{x}_i) \cdot \tilde{\Phi}(\mathbf{x}_j)) &&\text{(D.45)}\\
&= \left((\Phi(\mathbf{x}_i) - \frac{1}{M}\sum_{m=1}^{M} \Phi(\mathbf{x}_m)) \cdot (\Phi(\mathbf{x}_j) - \frac{1}{M}\sum_{n=1}^{M} \Phi(\mathbf{x}_n)) \right) \\
&= (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)) - \frac{1}{M}\sum_{m=1}^{M} (\Phi(\mathbf{x}_m) \cdot \Phi(\mathbf{x}_j)) \\
&\quad -\frac{1}{M}\sum_{n=1}^{M} (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_n)) + \frac{1}{M^2}\sum_{m,n=1}^{M} (\Phi(\mathbf{x}_m) \cdot \Phi(\mathbf{x}_n)) \\
&= K_{ij} - \frac{1}{M}\sum_{m=1}^{M} 1_{im}K_{mj} - \frac{1}{M}\sum_{n=1}^{M} K_{in}1_{nj} + \frac{1}{M^2}\sum_{m,n=1}^{M} 1_{im}K_{mn}1_{nj}
\end{aligned}$$

Using the matrix $(1_M)_{ij} := 1/M$, we get the more compact expression

$$\tilde{K}_{ij} = K - 1_M K - K 1_M + 1_M K 1_M. \tag{D.46}$$

We thus can compute $\tilde{K}$ from K, and then solve the Eigenvalue problem (D.43). As in (3.16), the solutions $\tilde{\boldsymbol{\alpha}}^k$ are normalized by normalizing the corresponding vectors $\tilde{\mathbf{V}}^k$ in F, which translates into

$$\tilde{\lambda}_k(\tilde{\boldsymbol{\alpha}}^k \cdot \tilde{\boldsymbol{\alpha}}^k) = 1. \tag{D.47}$$

For feature extraction, we compute projections of centered Φ-images of test patterns $\mathbf{t}$ onto the Eigenvectors of the covariance matrix of the centered points,

$$(\tilde{\mathbf{V}}^k \cdot \Phi(\mathbf{t})) = \sum_{i=1}^{M} \tilde{\alpha}_i^k (\tilde{\Phi}(\mathbf{x}_k) \cdot \tilde{\Phi}(\mathbf{t})). \tag{D.48}$$

Consider a set of test points $\mathbf{t}_1, \ldots, \mathbf{t}_L$, and define two $L \times M$ matrices by

$$K_{ij}^{test} = (\Phi(\mathbf{t}_i) \cdot \Phi(\mathbf{x}_j)) \tag{D.49}$$

and

$$\tilde{K}_{ij}^{test} = \left(((\Phi(\mathbf{t}_i) - \frac{1}{M} \sum_{m=1}^{M} \Phi(\mathbf{x}_m)) \cdot (\Phi(\mathbf{x}_j) - \frac{1}{M} \sum_{n=1}^{M} \Phi(\mathbf{x}_n))) \right). \tag{D.50}$$

Similar to (D.45), we can express $\tilde{K}^{test}$ in terms of K^{test}, and arrive at

$$\tilde{K}^{test} = K^{test} - 1'_M K - K^{test} 1_M + 1'_M K 1_M, \tag{D.51}$$

where $1'_M$ is the $L \times M$ matrix with all entries equal to $1/M$. As the test points can be chosen arbitrarily, we have thus in effect computed a centered version not only of the dot product matrix, but also of the kernel itself.

D.3 On the Tangent Covariance Matrix

In this section, we give an alternative derivation of (4.10), obtained by modifying the analysis of Sec. 2.1.2 (Vapnik, 1998). There, we had to maximize (2.7) subject to (2.6). When we want to construct invariant hyperplanes, the situation is slightly different. We do not only want to separate the training data, but we want to separate it in a way such that submitting a pattern to a transformation of an a priori specified Lie group will not alter its class assignment. This can be achieved by enforcing that the classification boundary be such that group actions move patterns parallel to the decision boundary, rather than across it. A local statement of this property is the requirement that the Lie derivatives should be orthogonal to the normal $\mathbf{w}$ which determines the separating hyperplane. Thus we modify (2.7) by adding a second term enforcing invariance:

$$\tau(\mathbf{w}) = \frac{1}{2} \left((1-\lambda) \frac{1}{\ell} \sum_{i=1}^{\ell} \left(\mathbf{w} \cdot \frac{\partial}{\partial t}\Big|_{t=0} \mathcal{L}_t \mathbf{z}_i \right)^2 + \lambda \|\mathbf{w}\|^2 \right) \tag{D.52}$$

For $\lambda = 1$, we recover the original objective function; for values $1 > \lambda \geq 0$, different amounts of importance are assigned to invariance with respect to the Lie group of transformations $\mathcal{L}_t$.

The above sum can be rewritten as

$$\begin{aligned} \frac{1}{\ell} \sum_{i=1}^{\ell} \left(\mathbf{w} \cdot \frac{\partial}{\partial t}\Big|_{t=0} \mathcal{L}_t \mathbf{z}_i \right)^2 &= \frac{1}{\ell} \sum_{i=1}^{\ell} \left(\mathbf{w} \cdot \frac{\partial}{\partial t}\Big|_{t=0} \mathcal{L}_t \mathbf{z}_i \right) \left(\frac{\partial}{\partial t}\Big|_{t=0} \mathcal{L}_t \mathbf{z}_i \cdot \mathbf{w} \right) \\ &= (\mathbf{w} \cdot C\mathbf{w}), \end{aligned} \tag{D.53}$$

where the matrix C is defined as in (4.6),

$$C := \frac{1}{\ell}\sum_{i=1}^{\ell}\left(\frac{\partial}{\partial t}\Big|_{t=0}\mathcal{L}_t\mathbf{z}_i\right)\left(\frac{\partial}{\partial t}\Big|_{t=0}\mathcal{L}_t\mathbf{z}_i\right)^{\top} \tag{D.54}$$

(if we want to use more than one derivative operator, we also sum over these; in that case, we may want to orthonormalize the derivatives for each observation $\mathbf{z}_i$). To solve the optimization problem, one introduces a Lagrangian

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\left((1-\lambda)(\mathbf{w}\cdot C\mathbf{w}) + \lambda\|\mathbf{w}\|^2\right) - \sum_{i=1}^{\ell}\alpha_i\left(y_i((\mathbf{z}_i\cdot\mathbf{w}) + b) - 1\right) \tag{D.55}$$

with Lagrange multipliers α_i. At the point of the solution, the gradient of L with respect to $\mathbf{w}$ must vanish:

$$(1-\lambda)C\mathbf{w} + \lambda\mathbf{w} - \sum_{i=1}^{\ell}\alpha_i y_i\mathbf{z}_i = 0 \tag{D.56}$$

As the left hand side of (D.53) is non-negative for any $\mathbf{w}$, C is a positive (not necessarily definite) matrix. It follows that for

$$C_\lambda := (1-\lambda)C + \lambda I \tag{D.57}$$

to be invertible (I denoting the identity), $\lambda > 0$ is a sufficient condition. In that case, we get the following expansion for the solution vector:

$$\mathbf{w} = \sum_{i=1}^{\ell}\alpha_i y_i C_\lambda^{-1}\mathbf{z}_i \tag{D.58}$$

Together with (2.3), (D.58) yields the decision function

$$f(\mathbf{z}) = \operatorname{sgn}\left(\sum_{i=1}^{\ell}\alpha_i y_i(\mathbf{z}\cdot C_\lambda^{-1}\mathbf{z}_i) + b\right). \tag{D.59}$$

Substituting (D.58), and the fact that at the point of the solution, the partial derivative of L with respect to b must vanish ($\sum_{i=1}^{\ell}\alpha_i y_i = 0$), into the Lagrangian (D.55), we get

$$\begin{aligned} W(\boldsymbol{\alpha}) = {} & \frac{1}{2}\sum_{i=1}^{\ell}\alpha_i y_i\mathbf{z}_i^{\top}\left(C_\lambda^{-1}\right)^{\top}\left(C_\lambda C_\lambda^{-1}\right)\sum_{j=1}^{\ell}\alpha_j y_j\mathbf{z}_j \\ & - \sum_{i=1}^{\ell}\alpha_i y_i\mathbf{z}_i C_\lambda^{-1}\sum_{j=1}^{\ell}\alpha_j y_j\mathbf{z}_j + \sum_{i=1}^{\ell}\alpha_i. \end{aligned} \tag{D.60}$$

By virtue of the fact that C_λ and thus also C_λ^{-1} is symmetric, the dual form of the optimization problem takes the following form: maximize

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j (\mathbf{z}_i \cdot C_\lambda^{-1} \mathbf{z}_j) \tag{D.61}$$

subject to (2.14) and (2.15).

The same derivation can be carried out for the nonseparable case, leading to the corresponding result with modified constraints (2.22) and (2.23) (cf. Sec. 2.1.3).

We conclude by generalizing to the nonlinear case. As in Sec. 2.1.4, we now think of the patterns $\mathbf{z}_i$ no longer as living in input space, but as patterns in some feature space F related to input space by a nonlinear map

$$\Phi : \mathbf{R}^N \rightarrow F \tag{D.62}$$

$$\mathbf{x}_i \mapsto \mathbf{z}_i = \Phi(\mathbf{x}_i). \tag{D.63}$$

Unfortunately, (D.59) and (D.61) are not simply written in terms of dot products between images of input patterns under Φ. Hence, substituting kernel functions for dot products will not do. Note, moreover, that C_λ now is an operator in a possibly infinite-dimensional space, with C being defined as in (4.15). We cannot compute it explicitely, but we can nevertheless compute (D.59) and (D.61), which is all we need. First note that for all $\mathbf{x}, \mathbf{y} \in \mathbf{R}^N$,

$$(\Phi(\mathbf{x}) \cdot C_\lambda^{-1} \Phi(\mathbf{y})) = (C_\lambda^{-\frac{1}{2}} \Phi(\mathbf{x}) \cdot C_\lambda^{-\frac{1}{2}} \Phi(\mathbf{y})), \tag{D.64}$$

with $C_\lambda^{-\frac{1}{2}}$ being the positive square root of C_λ^{-1}. At this point, methods similar to kernel PCA come to our rescue. As C_λ is symmetric, we may diagonalize it as

$$C_\lambda = SDS^\top, \tag{D.65}$$

hence

$$C_\lambda^{-\frac{1}{2}} = SD^{-\frac{1}{2}} S^\top. \tag{D.66}$$

Substituting (D.66) into (D.64), and using the fact that S is unitary, we obtain

$$(\Phi(\mathbf{x}) \cdot C_\lambda^{-1} \Phi(\mathbf{y})) = (SD^{-\frac{1}{2}} S^\top \Phi(\mathbf{x}) \cdot SD^{-\frac{1}{2}} S^\top \Phi(\mathbf{y})) \tag{D.67}$$

$$= (D^{-\frac{1}{2}} S^\top \Phi(\mathbf{x}) \cdot D^{-\frac{1}{2}} S^\top \Phi(\mathbf{y})). \tag{D.68}$$

This, however, is simply a dot product between kernel PCA feature vectors: $S^\top \Phi(\mathbf{x})$ computes projections onto Eigenvectors of C_λ (i.e. features), and $D^{-\frac{1}{2}}$ rescales them. Note that we have thus again arrived at the nonlinear tangent covariance matrix of Sec. 4.2.2; this time, however, the approach was motivated solely by constructing invariant hyperplanes in feature space, and the nonlinear feature extraction by the tangent covariance matrix is a mere by-product.

To carry out kernel PCA on C_λ, we essentially have to go through the analysis of kernel PCA using C_λ instead of the covariance matrix of the mapped data in F. The modifications arising from the fact that we are dealing with tangent vectors were already described in Sec. 4.2.2, hence, we shall presently only sketch the additional modifications for $\lambda > 0$: here, we are looking for solutions of the Eigenvalue equation $\mu\mathbf{V} = C_\lambda\mathbf{V}$ with $\mu > \lambda$ (let us assume that $\lambda < 1$, otherwise all Eigenvalues are identical to λ, the minimal Eigenvalue).[2] These lie in the span of the tangent vectors. In complete analogy to (3.14), we then arrive at

$$\ell\mu\boldsymbol{\alpha} = ((1-\lambda)K + \lambda I)\boldsymbol{\alpha}, \tag{D.69}$$

and the normalization condition for the coefficients $\boldsymbol{\alpha}^k$ of the k-th Eigenvector reads

$$1 = \frac{\mu_k - \lambda}{1-\lambda}(\boldsymbol{\alpha}\cdot\boldsymbol{\alpha}), \tag{D.70}$$

where the $\mu_k > \lambda$ are the Eigenvalues of $(1-\lambda)K + \lambda I$. Feature extraction is carried out as in (4.25).

We conclude by noting an essential difference to the approach of (4.11), which we believe is an advantage of the present method: in (4.11), the pattern preprocessing was assumed to be linear. In the present method, the goal to get invariant hyperplanes in feature space naturally led to a nonlinear preprocessing operation.

[2]If we want λI also to have an effect outside of the span of the tangent vectors, we have to modify the set in which we expand our solutions.

Bibliography

Y. S. Abu-Mostafa. Hints. *Neural Computation*, 7(4):639–671, 1995.

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.

S. Amari, N. Murata, K.-R. Müller, M. Finke, and H. Yang. Aymptotic statistical theory of overtraining and cross-validation. *IEEE Trans. on Neural Networks*, 8(5), 1997.

J. K. Anlauf and M. Biehl. The adatron: an adaptive perceptron algorithm. *Europhys. Letters*, 10:687 – 692, 1989.

H. Baird. Document image defect models. In *Proceddings, IAPR Workshop on Syntactic and Structural Pattern Recognition*, pages 38 – 46, Murray Hill, NJ, 1990.

H. Barlow. The neuron doctrine in perception. In M. Gazzaniga, editor, *The Cognitive Neurosciences*, pages 415 – 435. MIT Press, Cambridge, MA, 1995.

A. Barron. Predicted squared error: a criterion for automatic model selection. In S. Farlow, editor, *Self-organizing Methods in Modeling*. Marcel Dekker, New York, 1984.

Peter L. Bartlett. For valid generalization the size of the weights is more important than the size of the network. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 134, Cambridge, MA, 1997. MIT Press.

D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 1995.

D. Beymer and T. Poggio. Image representations for visual learning. *Science*, 272 (5270):1905–1909, 1996.

R. Bhatia. *Matrix Analysis.* Springer Verlag, New York, 1997.

C. M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, 1995.

V. Blanz. Bildbasierte Objekterkennung und die Bestimmung optimaler Ansichten. Diplomarbeit in Physik, Universität Tübingen, 1995.

V. Blanz, B. Schölkopf, H. Bülthoff, C. Burges, V. Vapnik, and T. Vetter. Comparison of view-based object recognition algorithms using realistic 3D models. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks — ICANN'96*, pages 251 – 256, Berlin, 1996. Springer Lecture Notes in Computer Science, Vol. 1112.

B. E. Boser, I .M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Müller, E. Säckinger, P. Simard, and V. Vapnik. Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th International Conference on Pattern Recognition and Neural Networks, Jerusalem*, pages 77 – 87. IEEE Computer Society Press, 1994.

L. Bottou and V. N. Vapnik. Local learning algorithms. *Neural Computation*, 4(6): 888–900, 1992.

J. Bromley and E. Säckinger. Neural-network and k-nearest-neighbor classifiers. Technical Report 11359–910819–16TM, AT&T, 1991.

H. H. Bülthoff and S. Edelman. Psychophysical support for a 2-D view interpolation theory of object recognition. *Proceedings of the National Academy of Science*, 89: 60 – 64, 1992.

C. J. C. Burges. Simplified support vector decision rules. In L. Saitta, editor, *Proceedings, 13th Intl. Conf. on Machine Learning*, pages 71–77, San Mateo, CA, 1996. Morgan Kaufmann.

C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector learning machines. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 375–381, Cambridge, MA, 1997. MIT Press.

E. I. Chang and R. L. Lippmann. A boundary hunting radial basis function classifier which allocates centers constructively. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann.

C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273 – 297, 1995.

R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume 1. Interscience Publishers, Inc, New York, 1953.

K. I. Diamantaras and S. Y. Kung. *Principal Component Neural Networks*. Wiley, New York, 1996.

H. Drucker, R. Schapire, and P. Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7:705 – 719, 1993.

R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

C.J. Duffy and R.H. Wurtz. Sensitivity of MST neurons to optic flow field stimuli. I. A continuum of response selectivity to large-field stimuli. *Journal of Neurophysiology*, 65:1329–1345, 1991.

N. Dunford and J. T. Schwartz. *Linear Operators Part II: Spectral Theory, Self Adjoint Operators in Hilbert Space*. Number VII in Pure and Applied Mathematics. John Wiley & Sons, New York, 1963.

S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

F. Girosi, M. Jones, and T. Poggio. Priors, stabilizers and basis functions: From regularization to radial, tensor and additive splines. A.I. Memo No. 1430, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1993.

I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large VC-dimension classifiers. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 147–155. Morgan Kaufmann, San Mateo, CA, 1993.

I. Guyon, N. Matić, and V. Vapnik. Discovering informative patterns and data cleaning. In U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smythand R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 181 – 203. MIT Press, Cambridge, MA, 1996.

J. B. Hampshire and A. Waibel. A novel objective function for improved phoneme recognition using time-delay neural networks. *IEEE Trans. Neural Networks*, 1: 216 – 228, 1990.

W. Härdle. *Applied Nonparametric Regression*. Cambridge University Press, Cambridge, 1990.

T. Hastie and W. Stuetzle. Principal curves. *JASA*, 84:502 – 516, 1989.

S. Haykin. *Neural Networks : A Comprehensive Foundation*. Macmillan, New York, 1994.

T. Hofmann and J. M. Buhmann. Pairwise data clustering by deterministic annealing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(1):1–25, 1997.

H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441 and 498–520, 1933.

P. Jolicoeur, M.A. Gluck, and S.M. Kosslyn. From pictures to words: Making the connection. *Cognitive Psychology*, 16:243–275, 1984.

I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

M. Kac and S. M. Ulam. *Mathematics and Logic*. Praeger, Britannica perspective, New York, 1968.

J. Karhunen and J. Joutsensalo. Generalizations of principal component analysis, optimization problems, and neural networks. *Neural Networks*, 8(4):549–562, 1995.

K. Karhunen. Zur Spektraltheorie stochastischer Prozesse. *Ann. Acad. Sci. Fenn.*, 34, 1946.

M. Kirby and L. Sirovich. Application of the Karhunen-Loève procedure for the characterization of human faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(1):103–108, 1990.

F. Klein. *Vergleichende Betrachtungen über neuere geometrische Forschungen*. Verlag von Andreas Deichert, Erlangen, 1872.

T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59 – 69, 1982.

A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1 – 7, 1965.

H. G. Krapp and R. Hengstenberg. Estimation of self-motion by optic flow processing in single visual interneurons. *Nature*, 384:463 – 466, 1996.

U. Kressel. Private communication. The quoted results are summarized on ftp://ftp.mpik-tueb.mpg.de/pub/chair_dataset/README, 1996.

Y. LeCun. Une procédure d'apprentissage pour Réseau à seuil assymmétrique. In *Cognitiva: A la Frontière de l'Intelligence Artificielle des Sciences de la Connaissance des Neurosciences*, pages 599–604, Paris, France, 1985. CESTA.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. J. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541 – 551, 1989.

Y. LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Müller, E. Säckinger, P. Simard, and V. Vapnik. Comparison of learning algorithms for handwritten digit recognition. In F. Fogelman-Soulié and P. Gallinari, editors, *Proceedings ICANN'95 — International Conference on Artificial Neural Networks*, volume II, pages 53 – 60, Nanterre, France, 1995. EC2.

J. Liter et al. Psychophysical and computational experiments on the MPI object databases. Technical report, Max-Planck-Institut für biologische Kybernetik, 1997.

N. K. Logothetis, J. Pauls, and T. Poggio. Shape representation in the inferior temporal cortex of monkeys. *Current Biology*, 5:552 – 563, 1995.

D. G. Luenberger. *Introduction to Linear and Nonlinear Programming.* Addison-Wesley, Reading, MA, 1973.

M. Mishkin, L.G. Ungerleider, and K.A. Macko. Object vision and spatial vision: two cortical pathways. *Trends in Neurosciences*, 6:414–417, 1983.

J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In *ICANN'97*, page 999. Springer Lecture Notes in Computer Science, 1997.

M. Z. Nashed and G. Wahba. Generalized inverses in reproducing kernel spaces: An approach to regularization of linear operator equations. *SIAM J. Math. Anal.*, 5: 974–987, 1974.

E. Oja. A simplified neuron model as a principal component analyzer. *J. Math. Biology*, 15:267 – 273, 1982.

E. Osuna, R. Freund, and F. Girosi. Improved training algorithm for support vector machines. In *NNSP'97*, 1997. in press.

K. Pearson. On lines and planes of closest fit to points in space. *Philosophical Magazine*, 2 (sixth series):559–572, 1901.

T. Poggio. On optimal nonlinear associative recall. *Biological Cybernetics*, 19:201–209, 1975.

T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9), 1990.

T. Poggio and T. Vetter. Recognition and structure from one 2D model view: observations on prototypes, object classes, and symmetries. A.I. Memo No. 1347, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1992.

H. Primas. *Chemistry, Quantum Mechanics and Reductionism.* Springer-Verlag, Berlin, 1983. second edition.

R. P. N. Rao and D. H. Ballard. Localized receptive fields may mediate transformation-invariant recognition in the visual cortex. Technical Report 97.2, National Resource Laboratory for the Study of Brain and Behavior, Computer Science Department, University of Rochester, 1997.

M. Reed and B. Simon. *Method of modern mathematical physics. Vol. 1: Functional Analysis.* Academic Press, San Diego, 1980.

D. Reilly, L. N. Cooper, and C. Elbaum. A neural model for category learning. *Biol. Cybern.*, 45:35 – 41., 1982.

B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, 1996.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

H. J. Ritter, T. M. Martinetz, and K. J. Schulten. *Neuronale Netze: Eine Einführung in die Neuroinformatik selbstorganisierender Abbildungen.* Addison-Wesley, Munich, Germany, 1990.

E. Rosch, C.B. Mervis, W. Gray, D. Johnson, and P. Boyes-Braem. Basic objects in natural categories. *Cognitive Psychology*, 8:382–439, 1976.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(9):533–536, 1986.

T. D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward network. *Neural Networks*, 2:459–473, 1989.

R. E. Schapire, Y. Freund, P. Bartlett, and W.S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Machine Learning: Proceedings of the Fourteenth International Conference*, 1997.

B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings, First International Conference on Knowledge Discovery & Data Mining.* AAAI Press, Menlo Park, CA, 1995.

B. Schölkopf. Künstliches Lernen. In S. Bornholdt and P. H. Feindt, editors, *Komplexe adaptive Systeme (Forum für Interdisziplinäre Forschung Bd. 15)*, pages 93 – 117. Röll, Dettelbach, 1996.

B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks — ICANN'96*, pages 47 – 52, Berlin, 1996a. Springer Lecture Notes in Computer Science, Vol. 1112.

B. Schölkopf, P. Simard, A. Smola, and V. Vapnik. Prior knowledge in support vector kernels. Accepted for *NIPS'97* (Proceedings to be published by MIT Press), 1997a.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. Technical Report 44, Max-Planck-Institut für biologische Kybernetik, 1996b. in press *(Neural Computation)*.

B. Schölkopf, A. Smola, and K.-R. Müller. Kernel principal component analysis. In *ICANN'97*, page 583. Springer Lecture Notes in Computer Science, 1997b.

B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. A.I. Memo No. 1599, Massachusetts Institute of Techology, 1996c.

J. Schürmann. *Pattern Classification: a unified view of statistical and neural approaches*. Wiley, New York, 1996.

J. Segman, J. Rubinstein, and Y. Y. Zeevi. The canonical coordinates method for pattern deformation: theoretical and computational considerations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:1171 – 1183, 1992.

J. Shawe-Taylor, P. Bartlett, R. Williamson, and M. Anthony. A framework for structural risk minimization. In *COLT*, 1996.

P. Simard, Y. LeCun, and J. Denker. Efficient pattern recognition using a new transformation distance. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 50–58, San Mateo, CA, 1993. Morgan Kaufmann.

P. Simard, B. Victorri, Y. LeCun, and J. Denker. Tangent prop — a formalism for specifying selected invariances in an adaptive network. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 895–903, San Mateo, CA, 1992. Morgan Kaufmann.

A. Smola and B. Schölkopf. From regularization operators to support vector kernels. Accepted for *NIPS'97*, 1997a.

A. Smola and B. Schölkopf. On a kernel-based method for pattern recognition, regression, approximation and operator inversion. *Algorithmica*, 1997b. in press.

A. Smola, B. Schölkopf, and K.-R. Müller. The connection between regularization operators and support vector kernels. *submitted to Neural Networks*, 1997.

A. J. Smola. Regression estimation with support vector learning machines. Diplomarbeit, Technische Universität München, 1996.

K. Sung. *Learning and Example Selection for Object and Pattern Detection.* PhD thesis, Massachusetts Institute of Technology, 1996.

A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-posed Problems.* W. H. Winston, Washington, D.C., 1977.

N. Troje and H. Bülthoff. How is bilateral symmetry of human faces used for recognition of novel views? Technical Report 38, Max-Planck-Institut für biologische Kybernetik, 1996. to appear in *Vision Research.*

S. Ullman. Aligning pictorial descriptions: An approach to object recognition. *Cognition*, 32:193–254, 1989.

S. Ullman. *High-Level Vision.* MIT Press, Cambridge, MA, 1996.

V. Vapnik. *Estimation of Dependences Based on Empirical Data [in Russian].* Nauka, Moscow, 1979. (English translation: Springer Verlag, New York, 1982).

V. Vapnik. Inductive principles of statistics and learning theory. In P. Smolensky, M. C. Mozer, and D. E. Rumelhart, editors, *Mathematical Perspectives on Neural Networks.* Lawrence Erlbaum, Mahwah, NJ, 1995a.

V. Vapnik. *The Nature of Statistical Learning Theory.* Springer Verlag, New York, 1995b.

V. Vapnik. *Statistical Learning Theory.* Wiley, New York, 1998. forthcoming.

V. Vapnik and A. Chervonenkis. Uniform convergence of frequencies of occurence of events to their probabilities. *Dokl. Akad. Nauk SSSR*, 181:915 – 918, 1968.

V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition [in Russian].* Nauka, Moscow, 1974. (German Translation: W. Wapnik & A. Tscherwonenkis, *Theorie der Zeichenerkennung*, Akademie-Verlag, Berlin, 1979).

V. Vapnik, S. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 281–287, Cambridge, MA, 1997. MIT Press.

T. Vetter. An early vision model for 3D object recognition. In *Fachtagung der Gesellschaft für Kognitionswissenschaften*, 1994.

T. Vetter and T. Poggio. Symmetric 3D objects are an easy case for 2D object recognition. *Spatial Vision*, 8(4):443–453, 1994.

T. Vetter and T. Poggio. Linear objectclasses and image synthesis from a single example image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1997. in press.

T. Vetter, T. Poggio, and H. Bülthoff. The importance of symmetry and virtual views in three-dimensional object recognition. *Current Biology*, 4:18 – 23, 1994.

T. Vetter and N. Troje. Separation of texture and shape in images of faces for image coding and synthesis. *Journal of the Optical Society of America*, in press, 1997.

G. Wahba. Convergence rates of certain approximate solutions to Fredholm integral equations of the first kind. *Journal of Approximation Theory*, 7:167 – 185, 1973.

C. K. I. Williams. Prediction with Gaussian processes. Preprint, 1997.

A. Yuille and N. Grzywacz. The motion coherence theory. In *Proceedings of the International Conference on Computer Vision*, pages 344–354, Washington, D.C., 1988. IEEE Computer Society Press.