Foundations and Trends® in
sample
Vol. xx, No xx (xxxx) 1–100
© xxxx xxxxxxxxx
DOI: xxxxxx

Kernel Methods in Computer Vision

Christoph H. Lampert[1]

[1] *Max Planck Institute for Biological Cybernetics, 72076 Tübingen, Germany,* `christoph.lampert@tuebingen.mpg.de`

Abstract

Over the last years, *kernel methods* have established themselves as powerful tools for computer vision researchers as well as for practitioners. In this tutorial, we give an introduction to kernel methods in computer vision from a geometric perspective, introducing not only the ubiquitous support vector machines, but also less known techniques for regression, dimensionality reduction, outlier detection and clustering. Additionally, we give an outlook on very recent, non-classical techniques for the prediction of structure data, for the estimation of statistical dependency and for learning the kernel function itself. All methods are illustrated with examplesss of successful application from the recent computer vision research literature.

Contents

1 Overview

Computer vision is a broad field, spanning all method for building automatic systems for the extraction of information from images, It covers a diverse range of applications, from the organization of visual information, over control and monitoring tasks, to interactive and real-time systems for human-computer interaction. Despite this variabilty, some principled algorithms have emerged over the last years and decades that are useful in many different scenarios and thereby transcend the boundaries of specific applications. One recently very successful class of such algorithms are *kernel methods*. Based on the fundamental concept of defining *similarities* between objects they allow, *e.g.* the prediction of properties of new objects based on the properties of known ones (*classification, regression*), or the identification of common subspaces or subgroups in otherwise unstructured data collections (*dimensionality reduction, clustering*).

1.1 The Goals of this Tutorial

This goal of tutorial is to give an introduction to kernel methods and their use in computer vision. In the *"Introduction to Kernel Methods"*, the problem of binary classification with *support vector machines* is used as introductory example in order to explain and motivate the fundamental concepts underlying all kernel methods. Subsequently, *"Kernels for Computer Vision"* gives an overview of the kernel functions that have been used in the area of computer vision and introduces the most important concepts one needs to know for the design of new kernel functions. Although support vector machine (SVM) are the most popular examples of kernel methods, they are by far not the only useful one. The rest of this text covers a variety of kernel methods that go beyond binary classification, namely algorithms for *"Multiclass Classification"*, *"Outlier Detection"*, *"Regression"*, *"Dimensionality Reduction"*, and *"Clustering"*. Also included are some very recent non-standard techniques, namely *"Structured Prediction"*, *"Dependency Estimation"* and techniques for *"Learning the Kernel"* from data. In each case, after introducing the underlying idea and mathematical concepts, we give examples from the computer vision research literature where the methods have been successfully applied. It is our hope that this double-tracked approach will give pointers into both directions, theory and application, for the common benefit of researchers as well as practitioners.

1.2 What this Tutorial is not

This work is not meant to replace a generic introduction into *machine learning* or generic *kernel kethods*. There are excellent textbooks for this purpose, *e.g.* [Schölkopf and Smola, 2002; Shawe-Taylor and Cristianini, 2004]. In contrast to a formal introduction, we will sometimes take shortcuts and appeal to the reader's geometric intuition. This is not out of disrespect for the underlying theoretical concepts, which are in fact one the main reasons why kernel methods have become so successful, but because otherwise we would not be able to achieve our main goal: to give an concise overview of the plethora of kernel methods and to show how to use them for tackling many interesting computer vision problems.

The limited space that is available in a text like this has another unfortunate consequence: we have to leave out a lot of what most kernel method textbooks spends most of their pages on. In particular, we will not cover: *probabilistic foundations*, *e.g.* the generating process of the data set we work with; *statistical learning theory*, including the highly elegant PAC generalization bounds; *optimization theory*, *e.g.* dualization and convexity; *numerics*, *e.g.* the many methods developed to solve the SVM's and related training problem. All good textbooks on support vector machines and kernel methods cover at least some of these topics, and it is our hope that after reading this introduction into *kernel method for computer vision*, your interest will be raised to do further background reading.

2

Introduction to Kernel Methods

To introduce the concept of *kernels*, let us study a toy problem: the classification of fish by their color. For simplicity, we assume that there are only two possible species, *salmon* and *bass*, and that for each exemplar of a fish we have a way of measuring its color. The task is to come up with a classification rule that can decide for any new fish whether it is a salmon or a bass, based on its color. Because we ourselves know nothing about fish, we will not be able to give the system any useful hints. We can, however, make use of a set of examples, called *training set*, that an expert in fish classification has annotated for us with the correct species label.

2.1 Notation

To describe the task in mathematical terms, we introduce the following notation:

- *fish* are denoted by (color) vectors $x \in \mathbb{R}^d$,
- *class membership* is denoted by $y \in \{-1, +1\}$, with $y = +1$ indicating *salmon* and $y = -1$ indicating *bass*,
- the *training set* of examples for which we have labels is called

$X = \{x_1, \dots, x_n\}$,
- their *class labels* are $Y = \{y_1, \dots, y_n\}$.

From this data, we would like to determine, or *learn*, a classification function

$$F : \mathbb{R}^d \to \{-1, +1\}, \tag{2.1}$$

i.e. a function that decides for any new (previously unseen) fish x which species it belongs to.

Clearly, we have infinitely many choices how to parameterize and determine F. In *kernel methods*, we choose F as the *sign* of a linear (or rather affine) function[1], *i.e.*

$$F(x) = \operatorname{sign} f(x) \tag{2.2}$$

with a *decision function* $f : \mathbb{R}^d \to \mathbb{R}$:

$$f(x) = a^0 + a^1 x^1 + a^2 x^2 + \dots a^d x^d. \tag{2.3}$$

In this $x^1, \dots, x^d$ are the components of x, and $a^0, \dots, a^n$ are the coefficients of f. By writing $w = (a^0, \dots, a^n) \in \mathbb{R}^{d+1}$ and augmenting x with a constant 1 in first position, $\tilde{x} = (1, x^1, \dots, x^d) \in \mathbb{R}^{d+1}$, we can write f more compactly as

$$f(x) = \langle w, \tilde{x} \rangle \tag{2.4}$$

where $\langle \cdot\,, \cdot \rangle$ denotes the inner product of vectors in $\mathbb{R}^{d+1}$. To simplify the notation, we will in the following assume that the augmentation is already part of the original feature vector. We will therefore write only x instead of $\tilde{x}$ and work with vectors in $\mathbb{R}^d$ instead of $\mathbb{R}^{d+1}$.

2.2 Linear Classification

Since the classification function F is completely determined by f, the latter will predominantly be the object of our studies, often also calling it the classifier itself. In particular we are interested in the question, how one can *learn* such functions from a set of training examples.

First, we characterize the properties that we would expect from a *good* classification function:

[1] The rare case $f(x) = 0$ can be resolved in an arbitrary way. We just define: $\operatorname{sign} 0 = 1$.

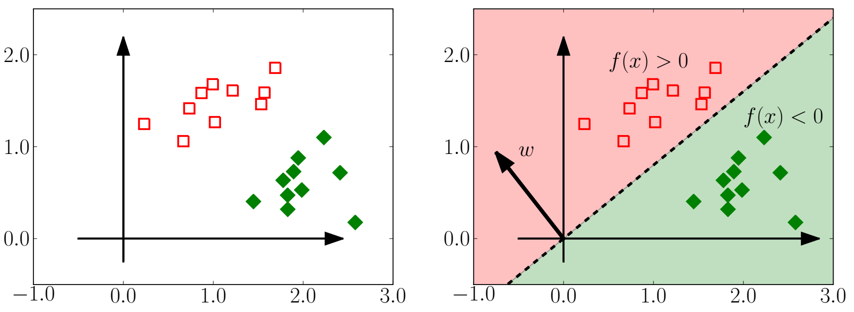


Figure 2.1 Schematic visualization of the *fish* classification task. Left: the training set contains samples of the two classes *salmon* (red) and *bass* (green). Right: a weight w induces a partitioning of the data space by means of a linear decision function $f(x) = \langle w, x\rangle$.

Definition 2.1 (Correctness). A classifier f is called *correct*, if for all samples in the training set, the labels predicted by f coincide with the training labels, *i.e.*

$$\operatorname{sign} f(x_i) = y_i \qquad \text{for } i = 1, \ldots, n. \tag{2.5}$$

Clearly, *correctness* is a desirable property, because it means that the classifier respects the information given to us in form of the training set. However, there are typically many choices for the weight vector w that lead to *correct* classifiers and not all of them are equally good. This can be seen from Figure 2.2: all three classifiers depicted there correctly classify the training points, but nevertheless, most people would agree that the classifier illustrated in the middle should be preferred over the other two, because it appears more *robust* for future decision.

Definition 2.2 (Robustness). The *robustness* of a classifier is the largest amount ρ by which we can perturb the sample vectors in the training set such that the prediction of f does not change:

$$\operatorname{sign} f(x_i + \epsilon) = \operatorname{sign} f(x_i) \qquad \text{for } i = 1, \ldots, n, \tag{2.6}$$

for all perturbations $\epsilon \in \mathbb{R}^d$ with $\|\varepsilon\| < \rho$.

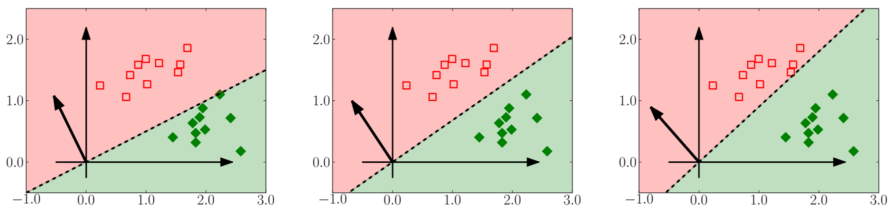


Figure 2.2 *Correctness* (2.5) does not uniquely determine the classifier function: all three classifiers classify predict the correct labels for all training samples.

In contrast to *correctness*, the *robustness* of a classifier is a numerical value. As it turns out, we can explicitly calculate its value for any linear classifier with weight vector w for a given training set $\{x_1, \dots, x_n\}$.

Theorem 2.1. The *robustness* of a classifier function $f(x) = \langle w, x\rangle$ is

$$\rho = \min_{i=1,\dots,n} \left| \langle \frac{w}{\|w\|}, x_i \rangle \right| . \tag{2.7}$$

Proof: From

$$f(x_i + \epsilon) = \langle w, x_i\epsilon\rangle = \langle w, x_i\rangle + \langle w, \epsilon\rangle \tag{2.8}$$

it follows that

$$f(x_i) - \|w\|\|\varepsilon\| \quad \leq \quad f(x_i + \epsilon) \quad \leq \quad f(x_i) + \|w\|\|\varepsilon\|, \tag{2.9}$$

and by checking the cases $\varepsilon = \pm\frac{\|\varepsilon\|}{\|w\|}w$ one sees that these inequalities are sharp. In order to ensure that Equation (2.6) is fulfilled for all training samples, we must have $\min_{i=1,\dots,n} |f(x_i)| \geq \|w\|\|\varepsilon\|$, from which the statement of the theorem follows. □

Geometrically, $\frac{w}{\|w\|}$ is the unit normal vector of the decision hyperplane $H_f = \{f(x) = 0\}$. Therefore $\left|\langle \frac{w}{\|w\|}, x_i\rangle\right|$ measures the distance of a sample x_i to H_f, as illustrated in Figure 2.3. The *robustness* of f is therefore identical to the width of the largest region around decision hyperplane that does not contain any training samples, commonly called the *(geometric) margin*.

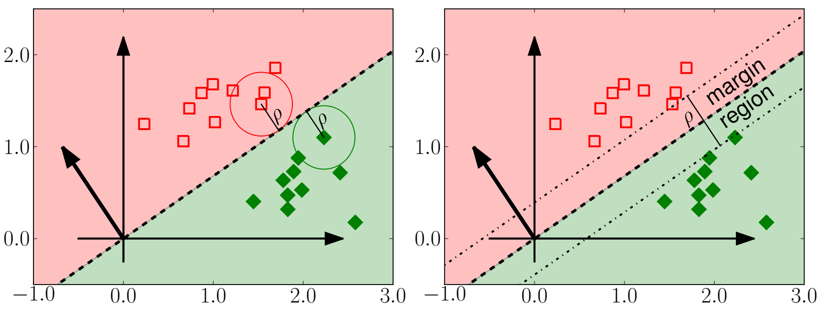


Figure 2.3 The *margin* concept. The *robustness* of a linear classifier, *i.e.* the amount by which training samples can be distorted without them moving across the decision boundary, is identical to the width of the margin, *i.e.* the large region around the decision boundary not containing any training examples.

2.2.1 The Maximum-Margin Classifier

The classifier that combines the properties of *correctness* with maximal *robustness* (*i.e. margin*), is called the *maximum margin classifier*:

Theorem 2.2. Let $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ be a training set with training labels $Y = \{y_1, \dots, y_n\} \subset \{-1, +1\}$. Then the weight vector $w \in \mathbb{R}^d$ of the *maximum margin classifier* is given by the solution to the optimization problem

$$\min_{w \in \mathbb{R}^d} \|w\|^2 \tag{2.10}$$

subject to

$$y_i \langle w, x_i \rangle \geq 1, \qquad \text{for } i = 1, \dots, n. \tag{2.11}$$

Proof: To see that the formulation of Theorem 2.2 indeed describes the *correct* classifier with maximal *robustness*, we first note that the intuitive characterization of a *maximum margin classifier* would be:

$$\max_{w \in \mathbb{R}^d} \rho \tag{2.12}$$

subject to

$$\rho = \min_{i=1,\ldots,n} \left| \langle \frac{w}{\|w\|}, x_i \rangle \right| \tag{2.13}$$

and

$$\operatorname{sign}\langle w, x_i \rangle = y_i, \qquad \text{for } i = 1, \ldots, n. \tag{2.14}$$

We can get rid of the minimization, by constrain ρ just to be smaller than all $\left|\langle \frac{w}{\|w\|}, x_i \rangle\right|$, for $i = 1, \ldots, n$, but at the same time maximizing over it. The previous optimization problem is then equivalent to solving

$$\max_{w \in \mathbb{R}^d, \rho \in \mathbb{R}^+} \quad \rho \tag{2.15}$$

subject to

$$\left| \langle \frac{w}{\|w\|}, x_i \rangle \right| \geq \rho \qquad \text{for } i = 1, \ldots, n, \tag{2.16}$$

and

$$\operatorname{sign}\langle w, x_i \rangle = y_i, \qquad \text{for } i = 1, \ldots, n, \tag{2.17}$$

where $\mathbb{R}^+$ stands for the set of positive real numbers. Inserting the equality constraints into the inequalities, and dividing them by ρ, we obtain the also equivalent

$$\max_{w \in \mathbb{R}^d, \rho \in \mathbb{R}^+} \quad \rho \tag{2.18}$$

subject to

$$y_i \langle \frac{w}{\rho\|w\|}, x_i \rangle \geq 1, \qquad \text{for } i = 1, \ldots, n. \tag{2.19}$$

We now substitute $\tilde{w} = \frac{w}{\rho\|w\|}$. Note that $\tilde{w}$ runs over all of $\mathbb{R}^d$ when $w \in \mathbb{R}^d$ and $\rho \in \mathbb{R}^+$. Using that $\|\tilde{w}\| = \frac{1}{\rho}$, we obtain the equivalent problem

$$\max_{\tilde{w} \in \mathbb{R}^d} \quad \frac{1}{\|\tilde{w}\|} \tag{2.20}$$

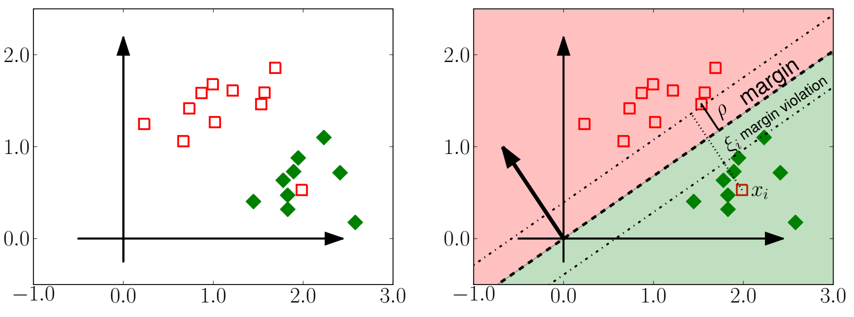


Figure 2.4 Need for soft margin classifiers. Left: because of a red outlier within the green class, there is no *correct* linear classifier for the training set. Right: We can overcome this problem by also allowing linear classifier that do not classify all training samples correctly, *i.e.* that make some mistakes. The outlier point x_i lies on the incorrect side of the margin region, called a *margin violation*.

subject to

$$y_i\langle \tilde{w}, x_i\rangle \geq 1, \qquad \text{for } i = 1, \dots, n, \tag{2.21}$$

which is equivalent to the optimization in Theorem 2.2, because maximizing over $\frac{1}{\|\tilde{w}\|}$ has same effect as minimizing over $\|\tilde{w}\|$, or equivalently over $\|\tilde{w}\|^2$.

From the view of optimization theory, numerically solving the problem in Theorem 2.2 is well understood and rather easy: the function we have to minimize is quadratic, *i.e.* in particular differentiable and *convex*, and all constraints are linear. We can find the *globally optimal* solution vector w in $O(n^2d + n^3)$ operations [Boyd and Vandenberghe, 2004], and there are also many algorithms that find a close to optimal solution much in much shorter time [Platt, 1999; Joachims, 2006; Chapelle, 2007; Shalev-Shwartz et al., 2007].

2.2.2 Soft-Margin Classifiers

So far, we have silently assumed that the maximum margin classifier exists. However, depending on the distribution of training samples, this might not be the case. Figure 2.4 (left) illustrates this case: because of an □ outlier between the ◆ samples, no *correct* linear classifier exists that would separate the classes.

Mathematically, this means that no weight vector w fulfills all constraints (2.11) at the same time, thereby making the optimization (2.10) impossible to solve. However, we can overcome this problem by making the hard constraints (2.11) into *soft constraints*, *i.e.* we allow them to be violated, but every violation is penalized by a certain cost term. In terms of an optimization problem, we obtain

---

Definition 2.3 (Maximum Soft-Margin Classifier). Let $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ be a training set with training labels $Y = \{y_1, \dots, y_n\} \subset \{-1, +1\}$. Then the weight vector $w \in \mathbb{R}^d$ of the *maximum soft-margin classifier* for any *regularization constant* $C > 0$ is given by the solution to the optimization problem

$$\min_{\substack{w \in \mathbb{R}^d \\ \xi_1, \dots, \xi_n \in \mathbb{R}^+}} \quad \|w\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi_i \tag{2.22}$$

subject to

$$y_i \langle w, x_i \rangle \geq 1 - \xi_i, \qquad \text{for } i = 1, \dots, n. \tag{2.23}$$

The new variables ξ_i are called *slack variables.* In contrast to the hard-margin case, the *maximum soft-margin classifier* always exists, because we can always fulfill the constraints (2.23) by just making the ξ_i large enough. However, $\xi_i > 0$ means that the sample x_i lies on the wrong side of the decision hyperplane (for $\xi_i > 1$), or at least inside of the margin region (for $0 < \xi_i \leq 1$). Therefore, we would like the ξ_i to be as small as possible. This is achieved by including the new term $\frac{C}{n} \sum_i \xi_i$ into the minimization of (2.22).

The constant C in Definition 2.3, allows us to trade off between *correctness* and *robustness* of the classifier. This is often a desirable property, as illustrated in Figure 2.5: even in situations where a *correct* classifier exists, *i.e.* a solution to the problem (2.10), we might want to allow for some outliers and search for a *soft-margin solution* instead. This, one would hope, will make the resulting classifier *more robust* for the correctly classified samples and thus make fewer mistakes on future data samples. The constant C allows us to control this behavior:

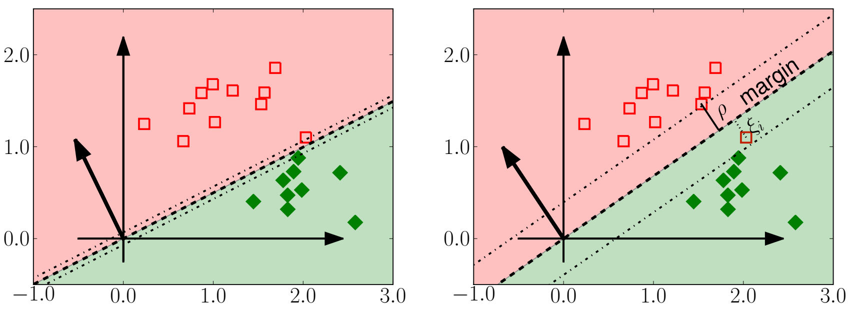


Figure 2.5 *Correctness* vs. *robustness* tradeoff. Even if a *correct* classifier exists (left), we might prefer one that makes some mistakes on the training set (right), if that is much more *robust*.

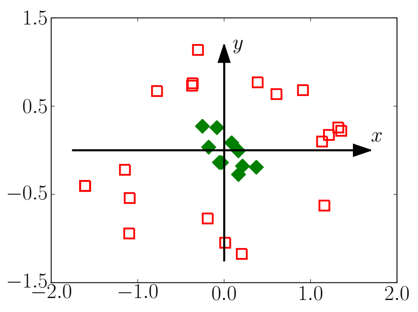
if C is very small, margin violations hardly matter at all, and the optimization concentrates on minimizing $\|w\|^2$, *i.e.* achieving a large ρ. Alternatively, if C is very large, every violation of a margin constraint will be penalized heavily, and, if possible, the soft-margin classifier will be very close or identical to the hard-margin classifier.

2.3 Nonlinear Classification

In some situations, there is no hope for any good linear classifier, even if we allow for margin violations. An example is depicted in Figure 2.6 (left): no linear classifier is going to achieve satisfactory performance.

Consequently, many ideas for non-linear classification have been developed, often based on the concept of combining the decisions of many linear classifiers into one non-linear one, *e.g. decision trees* [Quinlan, 1986; Breiman, 1998], *multi-layer perceptrons* [Bishop, 1995; Hinton et al., 2006] or *boosting* [Schapire and Freund, 1997]. In *kernel methods* we follow a different paradigm: first, we apply a non-linearly transformation to the data, and afterwards, we again learn a single linear classifier to the resulting transformed dataset. As illustrated in Figure 2.6 (right), by choosing a suitable coordinate transformation, a difficult non-linear problem can be become an easy linear one.

This intuition leads to a natural extension of linear classifiers to *generalized linear classifiers*:



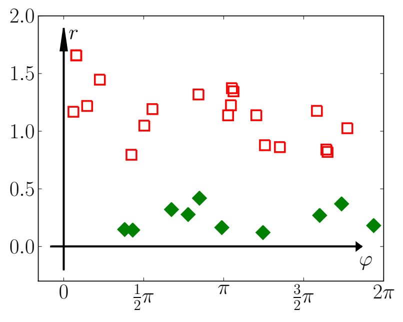


Figure 2.6 Nonlinear preprocessing. Left: for some distribution of training samples, no linear classifier will yield satisfactory results. Right: linear classification can be possible again if we first transform the samples into new a coordinate system, here from Cartesian to polar coordinates: $(x, y) \mapsto (\theta, r)$.

Definition 2.4 (Generalized Linear Max-Margin Classifier). Let $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ be a training set with training labels $Y = \{y_1, \dots, y_n\} \subset \{-1, +1\}$, and let $\varphi : \mathbb{R}^d \to \mathbb{R}^m$ be a (possibly non-linear) map. Then the *generalized linear max-margin classifier* for any *regularization constant* $C > 0$ is given by $F(x) = \operatorname{sign} f(x)$ for

$$f(x) = \langle w, \varphi(x) \rangle, \tag{2.24}$$

where $w \in \mathbb{R}^k$ is the solution to the optimization problem

$$\min_{\substack{w \in \mathbb{R}^k \\ \xi_1, \dots, \xi_n \in \mathbb{R}^+}} \|w\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi_i \tag{2.25}$$

subject to

$$y_i \langle w, \varphi(x_i) \rangle \geq 1 - \xi_i, \qquad \text{for } i = 1, \dots, n. \tag{2.26}$$

In other words, we find the best linear classifier for the transformed dataset $\{\varphi(x_1), \dots, \varphi(x_n)\} \subset \mathbb{R}^m$. This makes the *generalized linear max-margin classifier* linear in $\varphi(x)$, but, depending on φ, possibly non-linear in the original variable x.

2.3.1 Example Feature Mappings

Definition 2.4 does not pose are no restrictions on the feature map $\varphi : \mathbb{R}^d \to \mathbb{R}^m$. However, the performance of the resulting classifier will, of course, depend on this choice. Possible feature maps that have proved to be use for vectorial data are:

- Polar coordinates (for $(x, y) \in \mathbb{R}^2$):

$$\varphi : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} \sqrt{x^2 + y^2} \\ \arctan \frac{y}{x} \end{pmatrix}$$

- m-th degree polynomials (for $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$)

$$\varphi : x \mapsto \Big(1, x_1, \ldots, x_d, x_1^2, \ldots, x_d^2, \ldots, x_1^m, \ldots, x_d^m\Big)$$

- Distance map (for $x \in \mathbb{R}^d$):

$$\varphi : x \mapsto \Big(\|x - p_i\|, \ldots, \|x - p_N\|\Big)$$

  where $\{p_1, \ldots, p_N\} \subset \mathbb{R}^d$ is a set of N prototype vectors.

As one can see from the polynomial and distance map, the dimensionality of the target space can be much larger than that of the original space. This gives a generalized linear classifier with weight vector in $\mathbb{R}^m$ more flexibility than the original linear one in $\mathbb{R}^d$, and one can hope for better classification accuracy. However, by mapping the data into a very high-dimensional space one would expect that the memory usage and the runtime required to solve the optimization problem will increase. It is one of the most interesting properties of kernel method that these worries are unnecessary, as we will see in the following.

2.4 The Representer Theorem

The weight vector w of a generalized linear classifier with preprocessing map $\varphi : \mathbb{R}^d \to \mathbb{R}^m$ has m components that need to be determined. However, it follows from the following *Representer Theorem* that w cannot be arbitrary, but it lies in a subspace of $\mathbb{R}^m$ of dimension n or less, where n is the number of training samples.

Theorem 2.3 (Representer Theorem). A minimizing solution w to the optimization problem (2.25) can always be written as

$$w = \sum_{j=1}^{n} \alpha_j \varphi(x_j) \qquad \text{for coefficients } \alpha_1, \dots, \alpha_n \in \mathbb{R}. \tag{2.27}$$

The proof can be found in most textbooks on kernel methods, *e.g.* [Schölkopf and Smola, 2002].

The most important consequence of the Representer Theorem is the fact that we can rewrite the problem of learning a generalized linear classifier from an m-dimensional optimization problem into an n-dimensional one. For this, we insert the representation $w = \sum_{j=1}^{n} \alpha_j \varphi(x_j)$ into the problem (2.25) and optimize over $(\alpha_1, \dots, \alpha_n) \in \mathbb{R}^n$ instead of $w \in \mathbb{R}^m$. Using the linearity of the inner product, we obtain

$$\min_{\substack{\alpha_1,\dots,\alpha_n \in \mathbb{R} \\ \xi_1,\dots,\xi_n \in \mathbb{R}^+}} \quad \sum_{i,j=1}^{n} \alpha_i \alpha_j \langle\, \varphi(x_i), \varphi(x_j)\,\rangle + \frac{C}{n} \sum_{i=1}^{n} \xi_i \tag{2.28}$$

subject to

$$y_i \sum_{j=1}^{n} \alpha_j \langle\, \varphi(x_j), \varphi(x_i)\,\rangle \geq 1 - \xi_i, \qquad \text{for } i = 1, \dots, n, \tag{2.29}$$

and the generalized linear decision function (2.24) becomes

$$f(x) = \sum_{j=1}^{n} \alpha_j \langle\, \varphi(x_j), \varphi(x)\,\rangle. \tag{2.30}$$

A special property of this problem is, that the coefficient vector $(\alpha_j)_{j=1,\dots,n}$ will be *sparse*, *i.e.* most of the coefficients α_j will be 0 [Schölkopf and Smola, 2002]. This simplifies and speeds up the calculation of the decision function (2.30). The training examples x_j with non-zero coefficients α_j are often called *support vectors*, because they support the decision hyperplane.

2.5 Kernelization

The representer theorem allows us to reduce[2] the size of the optimization problem for finding the generalized max-margin classifier, but calculating and storing $\{\varphi(x_1), \dots, \varphi(x_n)\}$ would still require a lot of memory. We can avoid this bottleneck as well, by observing that the optimization problem (2.28) contains the transformed features $\varphi(x_i)$ only pairwise in inner products, *i.e.* as part of expression $\langle\, \varphi(x_i), \varphi(x_j)\,\rangle$. If we precompute and store these n^2 values ($\frac{n(n+1}{2}$ unique ones, because of symmetry), we do not need to store the $\varphi(x_i)$ vectors themselves anymore.

More generally, we can define the *kernel function* $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ of φ by the identity

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle. \tag{2.31}$$

Rewriting the problem (2.28) by means of this notation, we obtain the optimization problem for the generalized linear max-margin classifier in *kernelized* form, often called **support vector machine (SVM)**.

Theorem 2.4 (Support Vector Machine). Let $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$ be a training set with labels $Y = \{y_1, \dots, y_n\} \subset \{-1, +1\}$, let $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a kernel function and let $C > 0$. Then the decision function of the *kernelized* max-margin classifier is given by

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) \tag{2.32}$$

for coefficients $\alpha_1, \dots, \alpha_n$ obtained by solving

$$\min_{\substack{\alpha_1,\dots,\alpha_n \in \mathbb{R} \\ \xi_1,\dots,\xi_n \in \mathbb{R}^+}} \quad \sum_{i,j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) + \frac{C}{n} \sum_{i=1}^{n} \xi_i, \tag{2.33}$$

[2] If the dimension m of the feature space is larger than the number of training examples n, which is the case for most of the commonly used non-linear feature maps.

subject to

$$y_i \sum_{j=1}^{n} \alpha_j k(x_j, x_i) \geq 1 - \xi_i, \qquad \text{for } i = 1, \dots n. \tag{2.34}$$

2.5.1 Why kernelize?

At first sight, introducing $k(x, x')$ has not improved our situation. Instead of calculating $\langle \varphi(x_i), \varphi(x_j) \rangle$ for $i, j = 1, \dots, n$, we have to calculate $k(x_i, x_j)$, which has exactly the same values. However, there are two potential advantages of the kernelized setup:

Reason 1: Speed. We might find an expression for $k(x_i, x_j)$ that is faster to calculate than forming $\varphi(x_i)$ and then $\langle \varphi(x_i), \varphi(x_j) \rangle$.

As an example, we look at a $2nd$-order polynomial feature map (for simplicity only for $x \in \mathbb{R}^1$). Calculating the feature map

$$\varphi : x \mapsto (1, \sqrt{2}x, x^2) \in \mathbb{R}^3 \tag{2.35}$$

requires 2 multiplication per sample, which makes $2n$ for the whole training set. Subsequently, each inner product requires 5 operations (3 multiplication and 2 additions), which sums up to $\frac{5}{2}n(n+1)$ for the $n(n+1)/2$ unique elements of $\langle \varphi(x_i), \varphi(x_j) \rangle$ for $i, j = 1, \dots, n$. In total, we therefore require $\frac{5}{2}n^2 + \frac{9}{2}n$ operations before we can start the optimization.

The use of a kernel function allows us to simplify the expressions before starting the evaluation:

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle = \langle (1, \sqrt{2}x, x^2)^t, (1, \sqrt{x}x', x'^2)^t \rangle \tag{2.36}$$

$$= 1 + 2xx' + x^2 x'^2 \tag{2.37}$$

$$= (1 + xx')^2 \tag{2.38}$$

Therefore, each evaluation of $k(x_i, x_j)$ requires only 3 operations (2 multiplications and 1 addition) for a total of $\frac{3}{2}n^2 + \frac{3}{2}n$ operations for all unique entries $k(x_i, x_j)$ for $i, j = 1, \dots, n$. Thus, we were able to save more than 40% of operations by the kernelization.

Reason 2: Flexibility. We can construct functions $k(x, x')$, for which we *know* that they corresponds to inner products after some feature mapping φ, but we don't know for what φ is or how to compute it.

This slightly non-intuitive statement is a consequence of the second important theorem in kernel methods:

Theorem 2.5 (Mercer's Condition). Let $\mathcal{X}$ be non-empty set. For any positive definite kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists a Hilbert space $\mathcal{H}$ and a feature map $\varphi : \mathcal{X} \to \mathcal{H}$ such that

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}} \tag{2.39}$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ denotes the inner product in $\mathcal{H}$.

For the proof, see *e.g.* [Schölkopf and Smola, 2002]. Because of this theorem, positive definite kernel functions are also called *Mercer kernels*.

Before discussing the consequences of the theorem, we give the missing definitions of *positive definite kernel* and *Hilbert space*.

Definition 2.5 (Positive Definite Kernel Function). Let $\mathcal{X}$ be a non-empty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called **positive definite kernel function**, if

- k is symmetric, *i.e.* $k(x, x') = k(x', x)$ for all $x, x' \in \mathcal{X}$.
- For any finite set of points $x_1, \dots, x_n \in \mathcal{X}$, the *kernel matrix*

$$K_{ij} = (k(x_i, x_j))_{i,j} \tag{2.40}$$

is positive semidefinite, *i.e.* for all vectors $t \in \mathbb{R}^n$:

$$\sum_{i,j=1}^{n} t_i K_{ij} t_j \geq 0. \tag{2.41}$$

Because all kernels that we study in the following are in fact *positive definite*, we will often just say *kernel* as shorthand for *positive definite kernel function*.

Definition 2.6 (Hilbert Space). A vector space $\mathcal{H}$ is called *Hilbert space*, if

- $\mathcal{H}$ is equipped with an inner product
$$\langle \cdot\, , \cdot \rangle_{\mathcal{H}} : \mathcal{H} \times \mathcal{H} \to \mathbb{R} \tag{2.42}$$
- $\mathcal{H}$ is complete under the induced norm $\|v\|_{\mathcal{H}} = \sqrt{\langle v\, , v\rangle_{\mathcal{H}}}$, *i.e.* all Cauchy sequences of elements in $\mathcal{H}$ converge to a limit that lies in $\mathcal{H}$.

For our purposes, the *completeness* property is of less importance, but the existence of an inner product is crucial for using the kernel function k instead of an explicit evaluation of $\langle \cdot\, , \cdot \rangle_{\mathcal{H}}$.

2.5.2 Classification for non-vectorial data

Theorem 2.5 has some fundamental consequences. In particular, it states that we can use *any* function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and use it in the context of the support vector machine classifier (2.33), as long as k is a positive definite kernel function. The resulting optimization will always correspond to a generalized linear max-margin classifier based on a feature map φ from $\mathbb{R}^d$ into a suitable feature space $\mathcal{H}$. Note that both φ and $\mathcal{H}$ defined only *implicitly*, *i.e.* we know that they exist, but we cannot calculate them. In particular, $\mathcal{H}$ is typically very high or even infinite dimensional, but this does not have to disturb us, because our only contact with $\mathcal{H}$ is through its inner product $\langle \cdot\, , \cdot \rangle_{\mathcal{H}}$, and that we can calculate with help of k.

Furthermore, because the conditions of Theorem 2.5 are not restricted to input domains $\mathbb{R}^d$, we can extend our notion of generalized linear classifiers to *arbitrary input sets* $\mathcal{X}$, as long as we can define a kernel for their elements. We will later see examples for such kernels.

2.6 Constructing Kernel Functions

Although Definition 2.5 of *positive definite kernel functions* is relatively simple, it is in practice difficult to check the criteria for a given function

$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, because the condition (2.41) is a statement about the values of k on arbitrarily large finite subsets of $\mathcal{X}$.

However, it is relatively easy to *construct* functions k that are positive definite kernels, by making use of the following principles:

1) We can *construct kernels from scratch*:

- For any $\varphi : \mathcal{X} \to \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle$ is a kernel.
- If $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *distance function*, *i.e.*
  - $d(x, x') \geq 0$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') = 0$ only for $x = x'$,
  - $d(x, x') = d(x', x)$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') \leq d(x, x'') + d(x'', x')$ for all $x, x', x'' \in \mathcal{X}$,

  then $k(x, x') = \exp(-\rho d(x, x'))$ is a kernel for any $\rho \in \mathbb{R}^+$.

2) We can *construct kernels from other kernels*:

- If k is a kernel and $\alpha \in \mathbb{R}^+$, then $k + \alpha$ and αk are kernels.
- if k_1, k_2 are kernels, then $k_1 + k_2$ and $k_1 \cdot k_2$ are kernels.

2.6.1 Kernel Examples

Using the rule for kernel combinations, we can check that the the following functions are positive definite kernels for vectors in $\mathbb{R}^d$:

- any linear combination $\sum_j \alpha_j k_j$ with k_j kernels and $\alpha_j \geq 0$,
- any polynomial kernel $k(x, x') = (1 + \langle x, x' \rangle)^m$, $m \in \mathbb{N}$,
- the Gaussian kernel $k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$ with $\sigma > 0$.

The following are examples for kernels for other domains $\mathcal{X}$:

- $k(h, h') = \sum_{i=1}^{n} \min(h_i, h'_i)$ for n-bin histograms h, h' [Grauman and Darrell, 2005][3],

[3] The *histogram intersection kernel* is another example of a kernel for which we can derive an explicit feature map φ, but which is much faster to compute in functional form than as an inner product: assume histograms $h = (h_1, \ldots, h_n)$ with positive integer entries (*e.g.*

- $k(p,p') = \exp(-JS(p,p'))$ for probability densities p, p', when
$$JS(p,p') = \frac{1}{2}\int_{\mathbb{R}} p(x)\log\frac{p(x)}{\frac{1}{2}(p(x)+p'(x))} + p'(x)\log\frac{p'(x)}{\frac{1}{2}(p(x)+p'(x))}$$
is the *Jenson-Shannon divergence* [Hein and Bousquet, 2005],
- $k(s,s') = \exp(-D(s,s'))$ for *strings* s, s', when $D(s,s')$ is the *edit distance* [Gusfield, 1997].

However, one can show that the function

- $\tanh(\kappa\langle x, x'\rangle + \theta)$,

which is a popular transfer function in artificial neural networks, is *not a positive definite kernel*[4].

2.7 Choosing the Kernel Function and its Parameters

From the previous section we saw many possibilities how to construct a kernel. Most of these kernels are in fact parameterized families of kernels, *e.g.* each choice of the order m for a polynomial kernels, or of the bandwidth $\sigma > 0$ for a Gaussian kernel gives rise to a legitimate kernel function. However, simply from the existence of an induced Hilbert space and feature map, we cannot infer that the resulting generalized linear classifier will achieve high accuracy.

The two most important questions for practitioners in kernel methods are therefore: *What kernel function should one use?* and *How should one set the kernel parameters?* Unfortunately, there is no uni-

raw counts) that sum at most to N. Then the feature map
$$\psi(h_i) = (\underbrace{1,\dots,1}_{h_i \text{ entries}}, \underbrace{0,\dots,0}_{N-h_i \text{ entries}}), \tag{2.43}$$
i.e. a fixed length unary coding of the histogram, turns each summand into an inner product $\min(h_i, h_i') = \langle\psi(h_i), \psi(h_i')\rangle$. Concatenating these, we obtain a feature map $\varphi(h) = (\psi(h_1), \dots, \psi(h_n))$ such that $k(h,h') = \langle\varphi(h), \varphi(h')\rangle$. However, calculating $k(h,h')$ directly requires $O(n)$ operations whereas by means of $\varphi(h)$ it is at least $O(nN)$.

[4] This did not stop people from successfully using it in a generalized linear classifier setup of the form (2.33). As it turns out, even kernel functions that are not positive definite can yield good classifiers. However, one loses the geometric interpretation that this is because they are maximum-margin classifiers in a latent feature space.

versally correct answer, and typically, the best solution is to try different possibilities and evaluate the resulting classifiers *e.g.* by *cross-validation* [Efron and Gong, 1983; Kohavi, 1995]. If that is not possible, one can rely on the following heuristics to help in the decision:

- A good kernel should have *high values* when applied to two similar objects, and it should have *low values* when applied to two dissimilar objects.[5]

There reasoning behind this statement can be seen from looking at a binary classification scenario and imagining a hypothetical *perfect feature function*, $\varphi : \mathcal{X} \to \mathbb{R}$ that fulfills $\varphi(x) = +1$ for every sample of one class and $\varphi(x) = -1$ for samples of the other class. The resulting kernel function $k(x, x') = \varphi(x)\varphi(x')$ will have the value $+1$ whenever x and x' belong to the same class, and -1 otherwise.

2.7.1 Heuristics

For kernels based on the exponential function

$$k(x, x') = \exp(-\gamma X(x, x')),$$

where X is a distance function, a *rule of thumb* for setting γ is

$$\frac{1}{\gamma} \approx \operatorname*{median}_{i,j=1,\dots,n} X(x_i, x_j). \tag{2.44}$$

This rule is based on the fact that the Gaussian curve has its point of strongest decline where the exponent is -1. Rescaling the distances in a way that brings the median of X to this point ensures that there will be roughly the same number of *similar* as *dissimilar* samples pairs [6].

Figure 2.7 illustrates another heuristic: a kernel function chosen is good for a dataset $\{x_1, \dots, x_n\}$ if the kernel matrix

$$K = k(x_i, x_j)_{i,j=1,\dots,n}$$

[5] Because of this fact, kernels are also often seen just as a special class of *similarity measures*.
[6] More involved variants of this rule of thumb scale the median distance by a factor between 1 and 10, to reflect the fact even within a class not all samples must be similar to each other. Alternatively, the median *between-class* distance can be used.

has a *visible block structure* with respect to the classes in the dataset. Again, this is based on the intuition that the kernel values within a class should be large and between classes should be small. A less heuristic treatment of the question how to choose a kernel function and its parameters can be found in the Chapter *Learning the Kernel*.

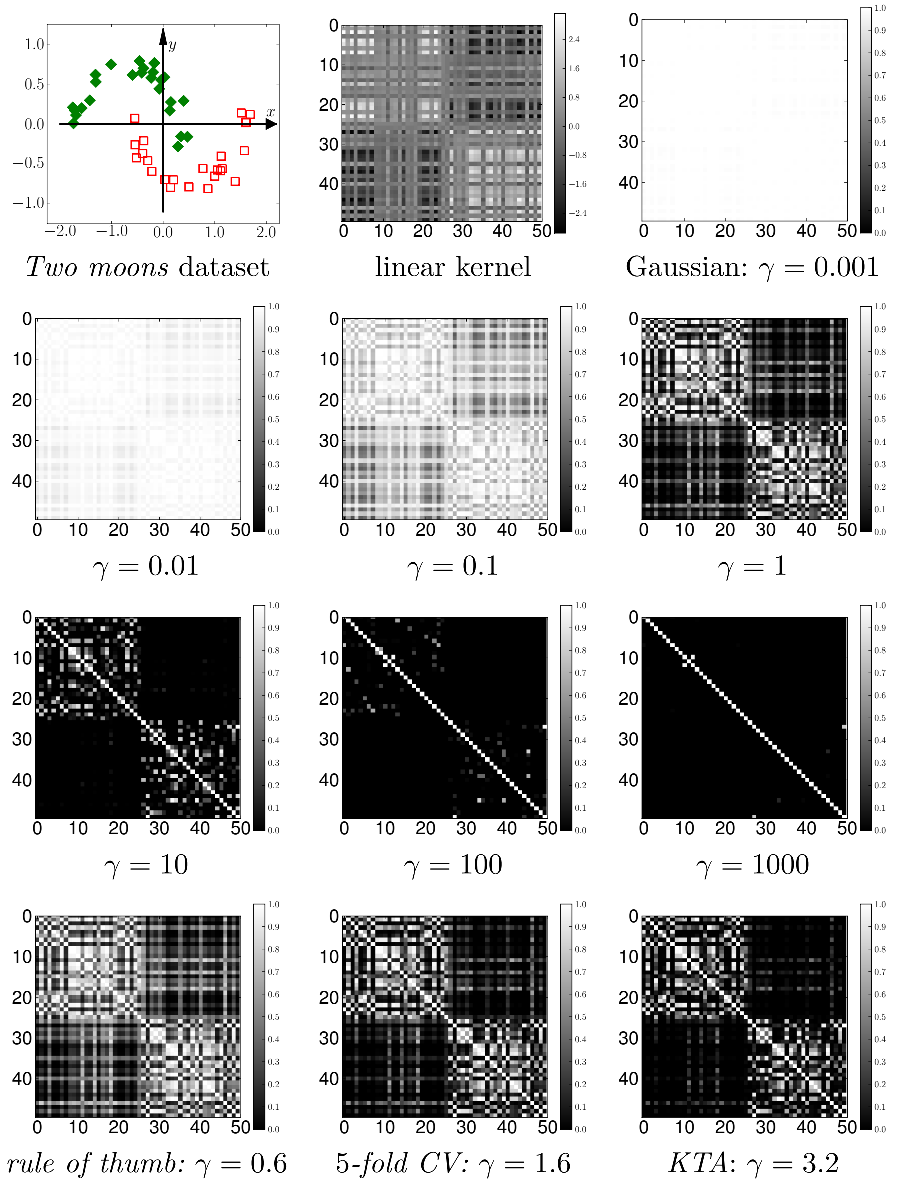


Figure 2.7 Kernel and parameter selection for a *Two Moons* dataset (top row, left). Kernel matrices (in reading order): with a *linear kernel* some samples do not show a clear class membership. *Gaussian* kernels with too small values of γ ($\gamma \in \{0.001, 0.01, 0.1\}$) cause almost all samples to be very similar to each other, even if they are from different classes. Between $\gamma = 1$ and $\gamma = 10$, the classes form visible blocks. Larger values ($\gamma \in \{100, 1000\}$) cause most samples to be only similar to themselves. The *rule of thumb* Eq. (2.44) results in a kernel with reasonable block structure, but *cross-validation (CV)* and *kernel target alignment (KTA)* (that will be introduced in the Chapter "Learning the Kernel") yield slightly more pronounced ones.

3

Kernels for Computer Vision

Images and videos are a data source with a very special characteristic: because each pixel represents a measurement, images are typically very high dimensional. At the same time, images have properties that we don't observe in arbitrary vector representations, *e.g.* strong correlations between neighboring pixels. Therefore, *Computer Vision* researchers have directed special attention on finding good data representations and algorithms to tackle problems, such as

- **Optical character recognition**: classify images of handwritten or printed letters or digits,
- **Object classification**: classify natural images according to which objects they contain,
- **Action recognition**: classify video sequences by what events occur in them,
- **Image segmentation**: partition an image into the exact subregions that correspond to different image aspects, *e.g.* objects or scenery,
- **Content Based Image retrieval**: find images in an image collection or database that are most similar to a query image.

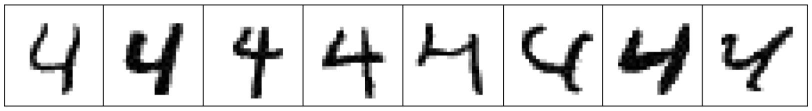


Figure 3.1 Image variability: Humans easily identify all these images as the handwritten digit 4 despite the differences in visual appearance.

Kernel methods have proved successful in all of these areas, mainly because of their interpretability and flexibility: in constructing a kernel function one can integrate knowledge that humans have about the current problem. Typically, this leads to improved performance compared to pure black-box methods that do not allow the integration of prior knowledge. Once a promising kernel function has been designed, it can be re-used in any kernel-based algorithm, not just in the context it was originally designed. This gives researchers as well as practitioners a large pool of established kernel functions to choose from, thereby increasing the chances of finding a well-performing one.

In the following, we introduce some of the existing kernels, the assumptions they were based on, and their applicability to practical computer vision tasks.

3.1 Designing Image Kernels

The easiest way to treat images is as vectors of pixel intensities. Images of size $N \times M$ pixels with values in $[0, 255]$ or $[0, 1]$ become vectors $x \in \mathbb{R}^{NM}$, and we can apply any kernel for vectorial data, *e.g.* linear or Gaussian. In the case of binary images $x \in \{0,1\}^{NM}$, polynomial kernels have also proven successful. Clearly, because these kernels can only compare vectors of identical lengths, all images to be used for learning need to be of the same size.

However, treating images directly as vectors also has other non-intuitive consequences. In particular, a pixel-wise comparison is typically *too specific*: images that humans would consider very similar can be treated as very different when measured by a pixel-wise distance measure like the *Euclidean* norm, see Figure 3.1. The reason for this is that the pixel representation lacks many *invariances* that are implicitly used in the human judgement of image similarity. For example, translations, (small) rotations, (small) changes in size, blur, brightness and contrast are factors that humans typically consider irrelevant when judging if two images show the same object. Other sources of invariance are domain dependent, *e.g.* the stroke width for character images or the illuminant color for natural scenes.

Ultimately, many problems in the design of useful image kernels can be reduced to the decision, which invariances to incorporate into the kernel construction and how to do so efficiently.

3.2 Integrating of Invariance

The need for incorporating image invariances into kernel methods can be understood from looking at the SVM decision function $f(x) = \sum_i \alpha_i k(x_i, x)$ for a test sample x. Only if some values of $k(x_i, x)$ are large, the decision score is significantly different from 0. Otherwise, the sample will lie very close to the decision hyperplane and the classifier becomes unreliable.

For the commonly used Gaussian kernel, $k(x, x_i)$ is significantly larger than 0 only for training samples x_i that are similar enough to x in the Euclidean norm. As discussed above, even images that are visually similar to x might have a large Euclidean distance, if they are geometrically or otherwise distorted. Thus they will not contribute to the decision function. Several methods to overcome this problem have been developed, and they can be categorized into four main approaches:

(1) extending the training set,
(2) image normalization,
(3) invariant kernel functions,
(4) invariant representations.

3.2.1 Extending the Training Set

We saw above that, in order to achieve good classification performance with a Gaussian or similar kernel, we need to have training examples that are similar to the test samples in Euclidean norm. The easiest way to increase the chances for this is by using more and more training

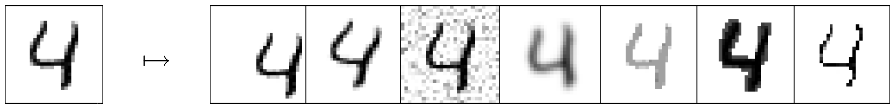


Figure 3.2 Invariance through *virtual samples*: for each training image (left), additional *synthetic* training images (right) are generated and to the training set, *e.g.* through *translation*, *rotation*, *noise*, *blur*, *contrast change*, *stroke width increase* and *thinning*.

samples until the whole data space is covered. In practice, the amount of training samples is usually limited, but we can achieve a similar effect by introducing *artificial* or *virtual* training samples [Decoste and Schölkopf, 2002][1]: for each original training sample we create multiple copies that are distorted with the transformations against which we would like the system to be invariant, *e.g.* translations, rotations and noise, see Figure 3.2. Each new sample is assigned the same training label as the original it was created from and added to the training set.

Advantages: The learning method becomes more invariant to image distortions, because more prototypes are available, in particular also distorted versions.

Disadvantages: We need a *generating procedure* that can create distorted versions the training images. The larger the sample set, the more time and storage is typically needed to train and apply the algorithm.

Conclusion: Use virtual samples if the training set is small and if invariance against sources of distortion is required that can be simulated computationally.

3.2.2 Image Normalization

A lot of image variability can be avoided by *normalizing* training and test images before applying the kernel. A typical procedure for this is the *resizing* of all images to a fixed size. *Brightness* and *contrast*

[1] This trick is not an invention of kernel methods. Similar ideas have been used previously under the name of *jittering* to train *e.g.* neural networks [Linden and Kindermann, 1989; Sietsma and Dow, 1991].

normalization are per-pixel operations that can be applied to reduce unwanted variations in the data range between different images. If we set

$$\hat{x} = \frac{x - x^{min}}{x^{max} - x^{min}}, \tag{3.1}$$

where x^{min} and x^{max} are the minimal and maximal grayvalue of the image $x \in \mathbb{R}^{NM}$, then the gray values of the resulting image $\hat{x}$ will always cover the full range $[0, 1]$, see Figure 3.3(a). For color images, contrast normalization can be applied to each color channel separately.

To make small images or image areas invariant against geometric transformations like *translation* and *rotation*, normalization to a uniform frame can be applied, *e.g.* by moment matching. We calculate the *first* and *second order moment* for an image x with pixels $x[i, j]$:

$$m_x = \frac{1}{m}\sum_{i,j} i\,x[i,j], \qquad m_y = \frac{1}{m}\sum_{i,j} j\,x[i,j], \tag{3.2}$$

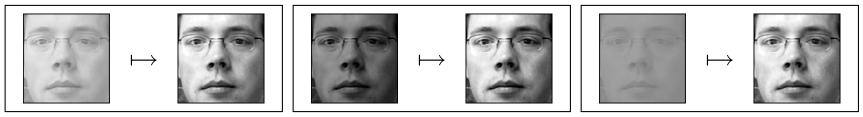
$$m_{xx} = \frac{1}{m}\sum_{i,j}(i - m_x)^2\,x[i,j], \quad m_{yy} = \frac{1}{m}\sum_{i,j}(j - m_y)^2\,x[i,j], \tag{3.3}$$

$$m_{xy} = \frac{1}{m}\sum_{i,j}(i - m_x)(j - m_y)\,x[i,j], \tag{3.4}$$

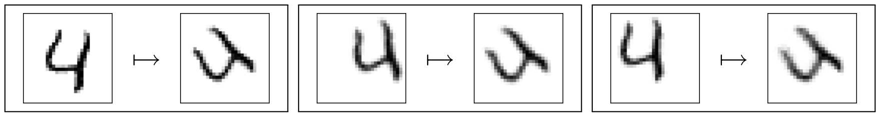
with $m = \sum_{i,j} x[i, j]$. A translation of the image will influence m_x and m_y, where as rotation around the center (m_x, m_y) will influence m_{xx}, m_{xy}, m_{yy}. Shifting the image by (m_x, m_y) and rotating it by $\alpha = \arctan(v_y/v_x)$ where v is the first eigenvector of the 2×2 matrix $M = [m_{xx}, m_{xy}; m_{xy}, m_{yy}]$ will create an image in normalized form with $m_x = m_y = m_{xx} = m_{xy} = m_{yy} = 0$, see Figure 3.3(b).

Advantages: Normalized images show fewer unwanted variations. Visually similar images become more similar also when treated as vectors. Thus, ordinary kernels for vector representations can be applied.

Disadvantages: An *inverse map* is required that counteracts the possible image distortions. Estimating the parameters of the normalization procedure for each image can be unreliable.



(a) *Illumination invariance* by brightness and contrast normalization. The right images are derived from the left by linearly transforming the grayvalues such that their minimum is 0 and their maximum is 1.



(b) *Geometric invariance* by moment normalization. The right images is derived from the left ones by translation and rotation such that $m_x = m_y = m_{xx} = m_{xy} = m_{yy} = 0$.

Figure 3.3 Invariance through *image normalization*: transforming the image to a *standard representation* can remove the effect of image variability.

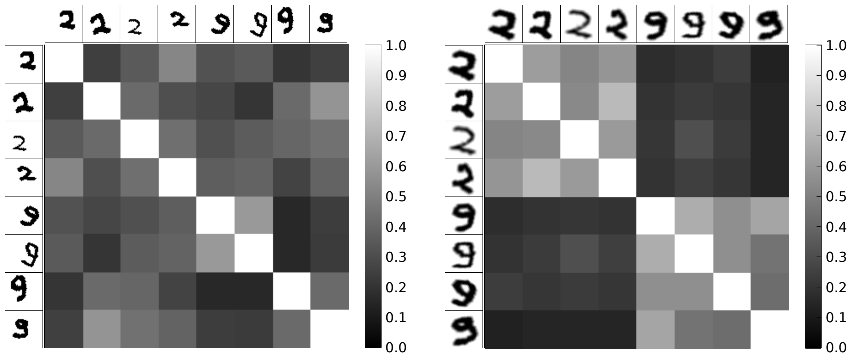


Figure 3.4 Effect of normalization on Gaussian kernel matrix: raw input samples (left) can have large Euclidean distance even if they belong to the same class. This leads to a non-informative kernel. Normalizing each sample to uniform size, translation and rotation (right) makes images of the same class more similar to each other, causing a more discriminative kernel matrix.

Conclusion: Use *image normalization* if you know how to define a *reference* configuration and how to reliably transform the images into it.

3.2.3 Invariant Kernel Functions

Instead of estimating normalizing parameters for each image and calculating the kernel values afterwards, one can make the normalization a part of the kernel function itself. This has the advantage that no globally best reference frame is required, but only transformations suitable to compute the distance between pairs of samples have to be estimated at a time. For example, instead of calculating the Gaussian kernel based on the Euclidean distance $\|.\|$, we define a new (dis)similarity measure $s(x, x')$ based on finding the match between the sample x and all distorted versions of x':

$$d_{dist}(x, x') = \min_{J \in \mathcal{J}} \|x - J(x')\|^2 \tag{3.5}$$

where J runs over all distortion operators from a set $\mathcal{J}$, *e.g.* all rotations or translations. $\mathcal{J}$ is typically multidimensional and a full non-linear optimization over all elements is often too time consuming. Therefore, linear approximations have been developed, *e.g.* the *tangent distance* [Simard et al., 1999] d_{TD}:

$$d_{TD}(x, x') = \min_{\delta_1, \dots, \delta_k \in \mathbb{R}} \|x - (x' + \delta_k t_k)\|^2 \tag{3.6}$$

where t_k are the difference vectors between x' and a slightly distorted versions of x', *e.g.* the image shifted by 1 pixel horizontally or vertically, or rotated by a few degrees. From the (symmetrized) distance function, one forms an invariant kernel function analogously to the Gaussian:

$$k_{inv}(x, x') = e^{-\gamma [d(x,x') + d(x',x)]}. \tag{3.7}$$

An alternatively way of making the kernel values invariant to known image distortions is by averaging over all possible transformations. We define

$$k_{int}(x, x') = \int_{J' \in \mathcal{J}} \int_{J \in \mathcal{J}} k(\, J(x), J'(x')\,) \tag{3.8}$$

for any image kernel k. If the set of transformation $\mathcal{J}$ is suitable, *e.g.* if it is a compact group [Haasdonk and Burkhardt, 2007], the resulting kernel function k_{int} will be exactly invariant to any of the transformations contained, even if k was not.

Advantages: By pairwise normalization, we can achieve invariance even if no global reference frame is available. By integrating over all of transformations, complete invariance can be achieved.

Disadvantages: Each kernel evaluation becomes slower to calculate. Distances functions that include optimization procedures can lead to kernel functions that are not positive definite.

Conclusion: Use an *invariant kernel* if image normalization as a preprocessing step is not possible, and if the transformation space is small enough such that efficient search or integration over it is possible.

3.2.4 Invariant Representations

As we have seen in the *Introduction*, using a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is equivalent to choosing an implicit feature map $\varphi : \mathcal{X} \to \mathcal{H}$, where $\mathcal{X}$ is the set of all images and $\mathcal{H}$ is the Hilbert space induced by k. For invariant kernels, the induced feature map is insensitive to image variations, whereas in image normalization, we used an explicit preprocessing step $\mathcal{N} : \mathcal{X} \to \mathcal{X}$ that removed the influence of the unwanted variations from all images before applying φ. We can generalize both concepts by extracting *invariant features*, followed by an arbitrary kernel for the resulting representation[2]. Mathematically, each of these operations corresponds to an explicit feature mapping $\psi : \mathcal{X} \to \tilde{\mathcal{X}}$, which is followed by the implicit mapping $\varphi : \tilde{\mathcal{X}} \to \mathcal{H}$ induced by a kernel function $k : \tilde{\mathcal{X}} \times \tilde{\mathcal{X}} \to \mathbb{R}$.

Because invariant feature maps form the basis for most relevant Computer Vision kernels, in the rest of this chapter, we will study some typical examples in more detail.

Advantages: *Invariant feature mapping* allow the efficient removal of many sources of unwanted image variations, even if no normalization procedure is known.

Disadvantages: By choosing a too uninformative intermediate representation, *important* sources of variance are easily removed as well. Because the feature extraction is always applied first, there is no way to recover the lost information.

[2] In a manner of speaking, we have already have made use of this construction in order to treat *images* as *vectors*, forgetting on the way, *e.g.*, in which format the image was stored.

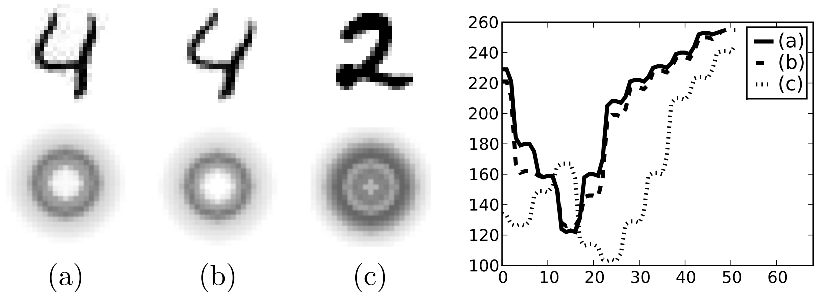


Figure 3.5 Extraction of rotation invariant features by integration. Left (a),(b): The two digits 4 differ by a rotation (top), making them dissimilar them in a Euclidean sense. Averaging over all their rotated versions yields very similar images (bottom). Left (c): The average over all rotated versions of a digit 2 image yields a feature image clearly different from (a) and (b). Right: Representation of the three rotated images by their radial profile. The difference between (a) and (b) is much smaller than that to (c).

Conclusion: *Invariant feature mappings* form the basis for most relevant Computer Vision kernels. However, one has to take care in their design to remove only the right amount of variations.

3.2.4.1 Integration-Based Invariant Features

Already linear or almost-linear transformations, *e.g.* integration against a weight function or the taking of absolute values, can make images more *invariant*. A special role is taken by the Fourier transform $\mathcal{F}$: a common way to achieve translation invariance in signals is by calculating $|\mathcal{F}x|$.

A similar effect as computing the invariant kernel (3.8) can be achieved by extracting features from Haar-integrals, *e.g.* the radial profile after integrating over all rotated version of an image $\tilde{x} = \int_{\theta\in[0,2\pi]} R_\theta(x)\, d\theta$, where $R_\theta x$ denotes the result of rotating the image x by an angle of θ. See Figure 3.2.4 for an illustration.

3.2.4.2 Edge-Based Invariant Features

In natural images, the location and orientation of edges is often more important than absolute intensity values. This has led to a variety of

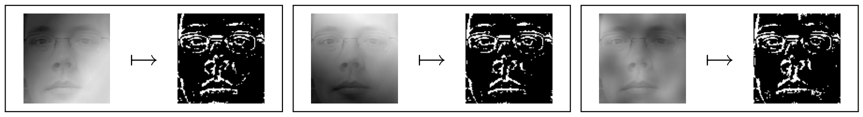


Figure 3.6 Edge detection as invariant preprocessing step: image differentiation followed by thresholding removes global intensity variations while keeping the location of edges.

edge-based feature maps, typically obtained by applying a differentiation step to the image, *e.g.* the *Sobel* or *Laplace* operator, and thresholding the response. Figure 3.6 illustrates this operation. Because differentiation is specifically susceptible to noise, it is often helpful to first convolve the input images with a Gaussian filter in order to *denoise* them. The result is a *Gaussian derivative filter*, which in a similar form is known to also exist in the human visual system [Young, 1987].

3.2.4.3 Histograms-Based Invariant Features

A lot of unwanted variance in natural images is due to changes in the viewpoint, as is illustrated in Figure 3.7. Compared to the in-plane rotations of Figure 3.2.4, it is much more difficult to parameterize and remove such variations, because in perspective projections information is lost that cannot be recovered by a mere postprocessing operation. Therefore, we cannot hope to define a normalizing transformations that exactly removes the effects of perspective from images. However, we can define feature maps that are invariant to perspective effects (or other geometric image properties) using *feature counts* that are aggregated into *histograms*: for a selection of localized base feature one counts how frequently each possible value occurs. In order to allow the efficient handling of the resulting histograms, low-dimensional quantities are preferable, *e.g.* the *grayscale value* or *color* of a pixel, *gradient strength* or *gradient direction*. By quantizing the value range into a fixed number of bins, the resulting histograms have fixed length and can be treated like vectors.

Histogram representations discard all geometric information from an image, even potentially relevant one. A way to control up to what scale the geometric information is removed is provided by *localized* histograms [Smith and Chang, 1996], as illustrated in Figure 3.8. Instead

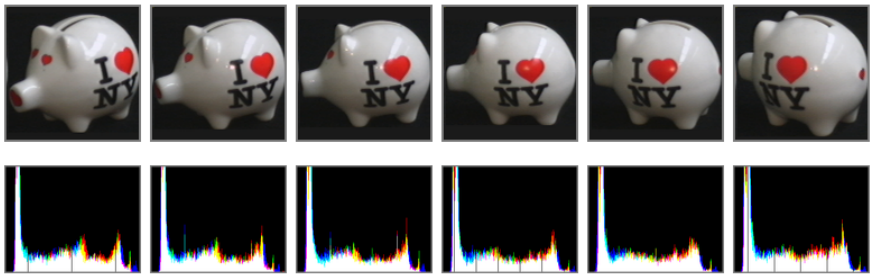


Figure 3.7 Histograms as perspective invariant features: the same object results in different images when viewed from different viewpoints. Nearly the same parts occur, but in different locations. The images differ when treated as vectors, but their color histograms are almost identical. Images: Colorado Object Image Library (COIL-100) [Nene et al., 1996]

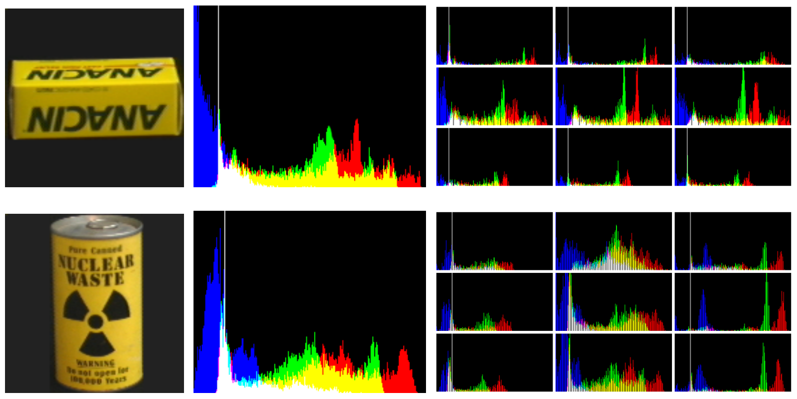


Figure 3.8 Histograms ignore all geometric relations, therefore very different objects (left) can have similar histogram representations (middle). By forming histograms over smaller image regions, *e.g.* a 3×3 grid (right), the feature map can be made more specific. On a global level it retains geometry but locally it is still invariant to changes *e.g.* due to perspective. Images: Colorado Object Image Library (COIL-100) [Nene et al., 1996]

of a single histogram per image, several histograms over pre-defined regions are formed, typically over one or several rectangular grids. By concatenating the resulting histograms, one again obtains a single feature vector. Depending on the size and number of regions chosen, it will be more or less invariant to geometric variations of the image.

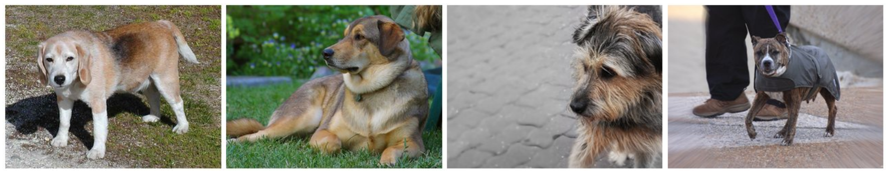


Figure 3.9 Object of the same semantic category, here *dog*, can differ greatly in appearance because of intra-class variance, change of pose, truncation, or occlusion. However, typical characteristic parts are often common for all object instances, *e.g.* eyes, ears or legs, thereby distinguishing them from other categories. *Images courtesy of photos8.com, pdphoto.org.*

3.3 Part-Based Image Representations

Advanced computer vision tasks require generalization not only between different views of the same objects, but between many different objects that only share a semantic aspect, *e.g.* animals of the same species. The visual variations within such a class can be very large, see *e.g.* Figure 3.9. An additional complication arises from the fact that the objects of interest often cover only a small parts of the image and that they can be truncated or occluded. *Part-based* representations of natural images have been developed to overcome all of these problems. They are based on the idea of treating the image as collections of many local parts instead of as single object with global properties.

3.3.1 Interest Points and Descriptors

To identify relevant parts of the image, one first applies an operator for the detection of interest points. Typically, these are low-level differential filters based on *differences of Gaussian* [Harris and Stephens, 1988; Lowe, 2004; Mikolajczyk and Schmid, 2004] or Wavelet coefficients [Loupias et al., 2000; Bay et al., 2006] that at the same time allow the extraction of a characteristic scale of the region. Alternatively, it has been shown that interest points on a regular grid [Zhang et al., 2007] or random locations and scales [Nowak et al., 2006] can also work well in practice.

Each region of interest defines a small image from which one calculates an *invariant representation*, often called a *descriptor*. The popular SIFT descriptor [Lowe, 2004] does this by combining several ideas of the previous sections: it normalizes the image region with respect to size and a canonical orientation, it smoothes the image to remove noise, it calculates image gradients for increased illumination invariance, and it aggregates the gradient directions into localized histograms for a certain degree of geometric invariance. Many other descriptors have been developed that follow similar design goals [Tuytelaars and Mikolajczyk, 2008].

After this first preprocessing step, the image is represented as a set of descriptor vectors, one per region of interest in the image. All descriptors vectors are of the same length, typically between 20 and 500 dimensions. The number of regions and descriptors varies depending on the image contents. Depending on the method for interest point detection, between 100 to 50,000 are common.

3.3.2 Set Kernels for Parts-Based Representations

To define kernels between images in part-based representations, one needs to overcome the problem that different images are represented by regions of interest of different size and location. The simplest solution for this is to ignore the geometric information about the regions of interest, and to keep only the vector valued descriptors.

Analogously to Equations (3.5) and (3.8), two canonical possibilities to compare two images $x = \{d_1, \dots, d_m\}$ and $x' = \{d'_1, \dots, d'_{m'}\}$ are obtained by *summing* or *matching* the values of descriptor-level base kernels:

$$k_{sum}(x, x') := \sum_{i=1}^{m} \sum_{j=1}^{m'} k(d_i, d'_j) \tag{3.9}$$

and

$$k_{match}(x, x') := \frac{1}{m} \sum_{i=1}^{m} \max_{j=1,\dots,m'} k(d_i, d'_j) + \frac{1}{m'} \sum_{j=1}^{m'} \max_{i=1,\dots,m} k(d_i, d'_j), \tag{3.10}$$

where k is any base kernel between vectors.

Unfortunately, such *set kernels* are often computationally inefficient, because the computational effort for each kernel evaluation is quadratic in the number of descriptors per image and at least linear in the dimension of the descriptors used.

3.3.3 Pyramid Match Kernel

The complexity of comparing two images in part-based representation can be made linear instead of quadratic by quantizing the space of possible descriptor values. The *pyramid match kernel (PMK)* [Grauman and Darrell, 2005] does so by subdividing the d-dimensional space of image descriptors into a hierarchy of axis parallel cells in a data dependent way. In the finest layer, each descriptor lies in a cell of its own. Coarser layers are built by merging neighboring cells in any dimension. Consequently, every new cell consists of 2^d cells of the previous layer and this construction is repeated until the coarsest layer has only one cell containing all descriptors.

The base kernel $k(d_i, d'_j)$ of the matching kernel (3.10) is approximated by a count in how many cells both descriptors lie together. This yields a final kernel definition of

$$k_{PMK}(x, x') = \sum_{l=1}^{L} 2^l \sum_{k=1}^{2^{l-1}} \min\left(h^{l,k}(x), h^{l,k}(x')\right), \tag{3.11}$$

where $h^{l,k}(x)$ are histograms of how many features of the image x falls into the k-th cell of the l-th pyramid level. By pre-computing these and storing only the non-empty bins, each evaluation of k_{PMK} requires $O(\max(m, m')dL)$ operations, where L is the number of pyramid levels. Since the number of descriptors per image is typically large, whereas L is small, this is much faster than the $O(mm'd)$ complexity of the naive matching kernel.

3.3.4 Feature Codebooks

Natural images have inherent regularities that cause the extracted descriptors vectors to form *clusters* in the descriptors space. For example, edges and corners are typically much more frequent than, *e.g.*, checker board-like patterns. A quantization of the descriptor space by a regular grid, as used by the pyramid match kernel, does not reflect this clustering: on the one hand, a large number of grid cells will stay empty,

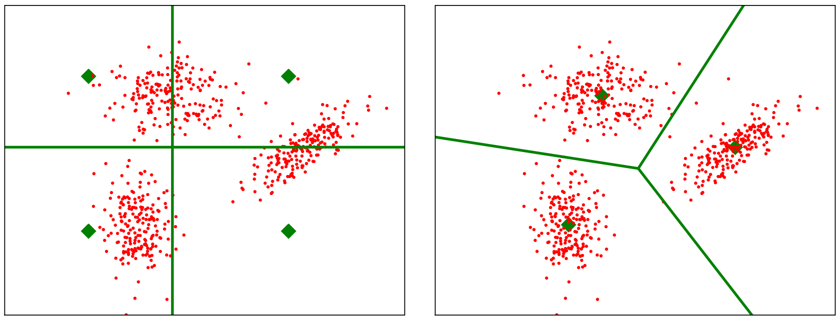


Figure 3.10 Descriptor Codebooks: The descriptors that occur in natural images do not lie uniform in the space of all possible descriptors, but they form clusters. Axis-parallel subdivisions (left) do not respect the cluster structure and can cut through regions of high density. Clustering the descriptors followed by vector quantization (right) divides the descriptor space into Voronoy cells that respect the cluster structure.

and on the other hand, existing clusters might be split apart, see Figure 3.10. A way to overcome this problem is by identifying a relatively small number of cluster centers in the descriptor space and quantizing the feature space by the induced Voronoy tessellation. Cluster centers are usually determined by a K-means clustering of all descriptors in the training data, or a subset thereof. Subsequently, for each descriptor set d_i in an image x, the nearest cluster center c_k (also called *codebook vector* or *visual word*) is identified. As simplest representation, we count for each cluster center how often it occurs as a nearest neighbor of a descriptor in x and form the resulting K-bin histogram. This construction is often called *bag of visual words*, since it is similar to the *bag of words* concept in *natural language processing.* The term *bag* indicates that all (quantized) descriptors are put into a single, unordered, representation, irrespectively of their original local or global arrangement in the image.

3.3.5 Kernels for Bag of Visual Words Representations

The representation of images as feature count histograms leaves us with many possibilities which kernel function to apply to them. A direct analogue of the pyramid match kernel (3.11) is the *histogram intersection* kernel:

$$k_{HI}(x, x') = \sum_{k=1}^{K} \min(h^k, h'^k), \tag{3.12}$$

where we write $h = (h^1, \dots, h^K)$ for the K-bin histogram representation of x and analogously h' for the histogram of x'.[3]

For fixed length histograms we can apply all kernels defined for vectors, *e.g. linear*, *polynomial* or *Gaussian*. If the number of feature points differs between images, it often makes sense to first normalize the histograms, *e.g.* by dividing each histogram bin by the total number of feature points[4]. This allows the use of kernels for empirical probability distributions, *e.g.* the *Bhattacharyya kernel*

$$k_{bhattacharyya}(x, x') = \sum_{k=1}^{K} \sqrt{h^k h'^k} \tag{3.13}$$

or the *symmetrized KL-divergence kernel*

$$k_{symKL}(x, x') = \exp\left(-\frac{1}{2}(KL(h|h') + KL(h'|h))\right), \tag{3.14}$$

where $KL(h|h') = \sum_k h^k \log \frac{h^k}{h'^k}$. A particularly popular choice in Computer Vision is the χ^2-*kernel*:

$$k_{\chi^2}(x, x') = \exp\left(-\gamma \sum_{k=1}^{K} \frac{(h^k - h'^k)^2}{h^k + h'^k}\right), \tag{3.15}$$

which has shown very good performance, *e.g.* in object classification tasks.

[3] This measure of similarity between histogram representations of images has been made popular earlier by [Swain and Ballard, 1991].

[4] Which normalization procedure is the *best* for bag of visual word histograms is as disputed as question which kernel function to use. Apart from normalization the histograms by their L^1 (or L^2) norm, it has been proposed, *e.g.* take square or cubic roots of entries [Jiang et al., 2007] or to *binarize* at a fixed or adaptive threshold [Nowak et al., 2006]. Alternatively, per-bin weighting schemes, such as *bi-normal separation* [Forman, 2003] or *tf-idf* [Hiemstra, 2000], that originated in text processing have been applied to visual word histograms as well.

3.3.6 Spatial Pyramid Representations

The *bag of visual words* representation discards all information about spatial structure from the image, as we have seen previously for other histogram representations. However, spatial information can be a valuable source of information, *e.g.* in image segmentation, where *sky* regions tend to occur much more frequently at the top of the image than at the bottom. Consequently, the idea of *local histograms* has proved useful in this setup as well. Instead of one global visual word histogram, a number of local histograms are formed, typically in a pyramid structure from coarse to fine. Each subhistogram has K bins and counts how many descriptors with center point in the corresponding pyramid cell have a specific codebook vector as nearest neighbor. Subsequently, either all local histograms are concatenated into a single larger histogram [Lazebnik et al., 2006], or separate kernel functions are applied for each level and cell, and the resulting kernel values combined into a single *spatial pyramid* score, *e.g.* by a weighted sum [Bosch et al., 2007]:

$$k_{SP}(x, x') = \sum_{l=1}^{L} \beta_l \sum_{k=1}^{K_l} k\left(h_{(l,k)}, h'_{(l,k)}\right), \tag{3.16}$$

where L is the number of levels and β_l is a per-level weight factor. K_l is the number of cells in the l-th level, and $h_{(l,k)}$ and $h'_{(l,k)}$ are the local histograms of x and x' respectively. The base kernel k is typically chosen from the same selection of histogram kernels as above, with or without separate histogram normalization.

4

Classification

Classification with support vector machines (SVMs) is the most popular application of kernel methods in computer vision. However, in contrast to the discussion in the *Introduction*, realistic classification problems are often not just binary decisions, but involve multiple possible class labels. In the following, we discuss extensions of SVMs that are capable of working with multi-class data. Subsequently, we give examples from the recent computer vision literature where kernel methods have successfully been applied to tasks like *object classification*, *scene classification* and *action classification.*

4.1 Multiclass Support Vector Machines

The hyperplane based classifier that forms the basis of the SVM formulation is only suitable for binary decision problems. However, several variants of SVMs have been developed that can handle more than two output classes by decomposing the multi-class decision step into a set of binary decisions.

4.1.1 One-versus-rest Multiclass SVM

The simplest form of multi-class classification by SVMs is the *one-versus-rest* formulation [Vapnik, 1998]. For a classification problem with M classes, we train M support vector machines $f_1, \dots, f_M$, each using the examples of one class as positive training examples and the examples of all other classes as negative training examples. To classify a test example, all SVMs are evaluated, and the class label of the SVM with largest value of the decision functions is selected:

$$c(x) = \operatorname*{argmax}_{m=1,\dots,M} f_m(x). \tag{4.1}$$

One-versus-rest multiclass SVMs achieve good classification performance [Rifkin and Klautau, 2004], and they are very popular in computer vision because their simple setup makes them easy to implement and evaluate.

4.1.2 One-versus-one Multiclass SVM

When the number of classes is large, the binary problems that one-versus-rest SVMs have to solve become very unbalanced. An alternative setup that avoids this problem are *one-versus-one* multiclass SVMs [Hsu and Lin, 2002]. Their underlying idea is to train one SVM per pair of class labels $f_{i,j}$ for $1 \leq i < j \leq M$, always using the samples of one class as positive and the samples of the other class as negative training examples. To classify a test sample, all SVMs are evaluated, and a voting is performed, which class label was selected most often.

$$c(x) = \operatorname*{argmax}_{m=1,\dots,M} \#\{i \in \{1, \dots, M\} : f_{m,i}(x) > 0\}, \tag{4.2}$$

where for convenience we used the notation $f_{j,i} = -f_{i,j}$ for $j > i$ and $f_{j,j} = 0$. If the argmax is not unique, *i.e.* there are several classes having the same maximal number of positive SVM decisions, some way of tie breaking is required, typically by just randomly choosing one of the classes. Although the one-vs-one setup requires many more SVMs to be trained and evaluated, each is based on a smaller training set, thus making the one-vs-one multiclass SVM approximately equally fast as the one-vs-rest multiclass SVM.

4.1.3 Crammer-Singer Multiclass SVM

While the one-vs-rest and one-vs-one SVMs consist of several independently trained binary classifiers, the multiclass SVM proposed by Crammer and Singer [Crammer and Singer, 2001] uses a training procedure that find solution hyperplanes $w_1, \dots, w_M$ for all classes from a joint optimization formulation:

$$\min_{\substack{w_1,\dots,w_M \in \mathcal{H} \\ \xi_1,\dots,\xi_n \in \mathbb{R}^+}} \sum_{m=1}^{M} \|w_m\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi_i \tag{4.3}$$

subject to

$$\langle w_{y_i}, \varphi(x_i) \rangle - \langle w_{y'}, \varphi(x_i) \rangle \geq 1 - \xi_i, \qquad \text{for } y' \neq y_i \tag{4.4}$$

for $i = 1, \dots, n$. The decision function

$$c(x) = \operatorname*{argmax}_{m=1,\dots,M} f_m(x) \tag{4.5}$$

with $f(x) = \langle w, \varphi(x_i) \rangle_{\mathcal{H}}$ is of the same form as in the one-vs-rest situation.

Analyzing the optimization problem shows that the Crammer-Singer SVM differs from M independent one-vs-rest SVMs only in the constraints (4.4). Ordinary SVM training enforces $y_i f(x_i) \geq 1$, *i.e.* each sample x_i should lie on the right side of the decision hyperplane and have a margin of at least 1 from the wrong side. The Crammer-Singer SVM enforces $f_m(x_i) - f_{m'}(x_i) \geq 1$ for each $m' \neq m$, *i.e.* each training sample should be classified correctly under the decision rule (4.5), and have a margin of 1 from being classified as any of the wrong classes. One can check that for $M = 2$, the optimization problem will yield $w_1 = -w_2$, which reduces the Crammer-Singler SVM to the ordinary, binary, SVM.

4.1.4 Other Multiclass SVM

Several other multiclass SVM variants have been developed for special situations: Platt et al. build a *directed acyclic graph (DAG)* out of the $M(M-1)/2$ binary SVMs of the *one-versus-one* setup [Platt et al., 1999]. By traversing along the DAG, only $(M - 1)$ SVM evaluations are required to come to a decision, thus speeding up the multiclass prediction (4.2).

In order to make the decision process more robust, multiclass SVMs based on *error-correcting codes* have been developed [Dietterich and Bakiri, 1995]. They introduce a certain redundancy into the decomposition of the multi-class decision problem into several binary decisions, By arranging the binary decisions in an *error-correcting code*, robustness to mistakes in the outcome of any of the binary decisions is achieved.

Overall, none of the multiclass SVM formulation has proven clearly superior to the others in terms of performance, and it depends on the problem at hand, which training and test setup is most appropriate.

4.2 Example: Optical Character Recognition

B. Schölkopf, C. Burges, and V. Vapnik: *"Extracting Support Data for a Given Task"*, Knowledge Discovery and Data Mining, 1995.

In one of the earliest application of a kernel-based classifier to a computer vision problem, [Schölkopf et al., 1995] use a support vector machine with polynomial, Gaussian or sigmoid kernel to classify handwritten digits in a a one-versus-rest setup. Apart from reporting improved error-rates, the authors show that the number of support vectors is only 4% of the dataset, making the evaluation procedure more efficient that, *e.g.*, a nearest neighbor classifier.

4.3 Example: Object Classification

K. Grauman and T. Darrell: *"The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features"*, ICCV 2005.

In this work, [Grauman and Darrell, 2005] introduce the *pyramid match kernel* (3.11) to compare two image that are represented by different numbers of local feature descriptors. The kernel is used in a one-versus-rest multi-class support vector machine for object classification on the ETH80 and Caltech101 datasets. Compared to the *matching* kernel (3.10), the pyramid match kernel is shown to be faster to compute and to achieve higher classification accuracy.

4.4 Example: Scene Classification

S. Lazebnik, C. Schmid, and J. Ponce: *"Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories"*, TR 2005

The authors introduce the *Spatial Pyramid Matching* (3.16) kernel that is based on representing images as a pyramids of increasingly localized bag of visual word histograms [Lazebnik et al., 2006]. The histograms of each pyramid level and cell are compared by the histogram intersection kernel 3.12 and combined into an overall kernel value by a weighted linear combination.

As part of the experimental evaluation, the authors test their setup on a dataset of 15 indoor and outdoor scene categories. A one-versus-rest multi-class SVM with spatial pyramid kernel performs consistently better than flat bag of visual kernels and improves over previous results for the dataset that had been obtained using Latent Dirichlet Allocation.

J. Yang, Y.-G. Jiang, A. G. Hauptmann, C.-W. Ngo: *"Evaluating Bag-of-Visual-Words Representations in Scene Classification"*, MIR 2007

This work [Yang et al., 2007] compares many different feature and preprocessing operations for scene classification. The classifier itself is chosen as one-versus-rest support vector machine with linear or Gaussian kernel. Experiments on three datasets for scene classification show that the Gaussian kernel achieves superior classification rates for small and medium sized bag-of-word codebooks, whereas for very large codebooks (larger than 10000 entries), linear kernels perform equally well or better.

M. Boutell and J. Luo: *"Beyond Pixels: Exploiting Camera Metadata for Photo Classification"*, CVPR 2004

The authors propose a system to integrate non-image cues, into a scene classification process [Boutell and Luo, 2005]. The baseline classification system is a support vector machine based on color and texture features. The experiments show that a combination of the SVM's output with camera meta-data, *e.g.* exposure time and the use of a flash, by a Bayesian network architecture, improves the classification accuracy over only using low-level features.

4.5 Example: Action Classification

P. Dollár, V. Rabaud, G. Cottrell, and S. Belongie: *"Behavior Recognition via Sparse Spatio-Temporal Features"*, CVPR 2006

The authors introduce a new spatio-temporal interest operator and local image descriptor that is conceptually similar to SIFT [Dollar et al., 2005]. Subsequently, they create a descriptor codebook by k-means clustering and represent each video as a *bag of spatio-temporal word* histogram. The experimental evaluation on the dataset of *facial expressions* and *human activities* showed that a one-versus-rest support vector machine with Gaussian kernel achieved superior classification performance to a 1-nearest neighbor classifier.

Interestingly, [Nowozin et al., 2007] achieved a significantly higher classification performance of 85.2% instead of 80.7% when repeating the experiments on the human activity dataset, using exactly the same extracted features, but performing cross-validation to determine relevant parameters for feature preprocessing and SVM regularization. Using a χ^2-kernel instead of a Gaussian kernel further improved the results to 87.0% accuracy. These results show that, even though SVMs are relatively robust against parameters changes, model selection should not be neglected as part of the training procedure in order to achieve the highest possible accuracy.

5

Outlier Detection

Data intensive computer vision tasks like image retrieval or video surveillance typically work with many thousand images, but not always do they have access to reliable class labels in order to train a classifier. Instead, *outlier* or *anomaly detection* are frequently applied techniques: outliers detection can help to "clean" a dataset by removing examples that are too different from all other samples. A typical example would be mislabeled examples returned by a search query. In anomaly detection, it is the non-typical samples that are the main objects of interest. This is a common preprocessing step, *e.g.* in surveillance tasks, where no new information is gained by processing scenes that are very similar to already previously seen ones. Methodically, both methods are basically identical, as they have the goal of identifying data points that are different from most other ones. In the following, we introduce two related techniques that are applicable to both tasks: *Support Vector Data Description (SVDD)* and the *One-Class Support-Vector-Machine (OC-SVM).*

5.1 Outlier Detection in $\mathbb{R}^d$

An intuitive way for outlier or anomaly detection is to estimate the density of data points in the feature space. Samples in regions of high density are *typical*, whereas samples in low-density regions are not. However, density estimation in high-dimensional spaces is a notoriously difficult problem[1]. Instead, we adopt a more *geometric* approach: we assume that the high density region consists of only a single sphere with unknown location and size. Outlier detection then becomes the problem of finding the center $c \in \mathbb{R}^d$ and radius $R \in \mathbb{R}^+$ of the sphere such that most sample points $x_1, \dots, x_n$ lie inside the sphere, *i.e.* they fulfill $\|x_i - c\| \leq R$. This intuition is formalized by the following optimization problem:

$$\min_{\substack{R\in\mathbb{R}, c\in\mathbb{R}^d \\ \xi_1,\dots,\xi_n\in\mathbb{R}^+}} R^2 + \frac{1}{\nu n}\sum_{i=1}^{n} \xi_i \tag{5.1}$$

subject to

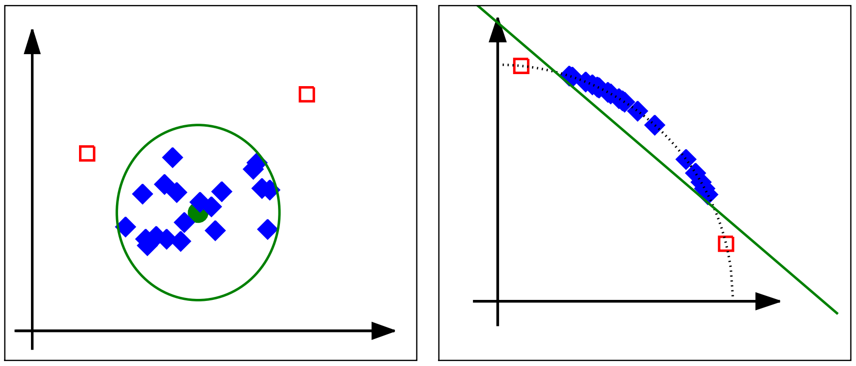
$$\|x_i - c\|^2 \leq R^2 + \xi_i \quad \text{for } i = 1, \dots, n. \tag{5.2}$$

As in the case of support vector machines, we use slack variables ξ_i to allow some points, namely the outliers, to lie outside of the sphere, but this is penalized linearly in the objective function. A parameter $\nu \in (0, 1]$ controls how many outliers are possible: the smaller ν the fewer points will be outliers, since violating the constraint (5.2) will be penalized more heavily. A more detailed analysis shows that νn is in fact a lower bound on number of outliers over the training set [Schölkopf and Smola, 2002].

For new samples, we decide based on their distance to the sphere's center: if their distance to the center is larger an R, they are considered outliers or anomalies, otherwise they are considered regular samples. A suitable decision function is given by

$$f(x) = R^2 - \|x - c\|^2. \tag{5.3}$$

[1] This makes kernel-based outlier detection a classical instance of *Vapnik's principle*: "Don't solve a problem by solving a more difficult problem as intermediate step." [Vapnik, 2000]



(a) Minimal enclosing ball (b) Separating hypersphere

Figure 5.1 Outlier Detection: *Support Vector Data Description* (left) finds the smallest sphere (green) that contains all samples (blue) except for some outliers (red). The *One-Class SVM* (right) separates inliers (blue) from outlier (red) by finding a hyperplane (green) of maximal distance from the origin. Both approaches are equivalent if all samples lie at the same distance from the origin and are linearly separable from it.

5.2 Support Vector Data Description

Support Vector Data Description (SVDD) [Tax and Duin, 2004] performs outlier detection in a latent feature space by kernelizing Equations (5.1) and (5.2). For a given kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with induced Hilbert space $\mathcal{H}$ and feature map $\varphi : \mathcal{X} \to \mathcal{H}$, the problem becomes

$$\min_{\substack{R\in\mathbb{R}, c\in\mathcal{H}\\ \xi_1,\dots,\xi_n\in\mathbb{R}^+}} R^2 + \frac{1}{\nu n}\sum_{i=1}^{n} \xi_i \tag{5.4}$$

subject to

$$\|\varphi(x_i) - c\|^2_{\mathcal{H}} \leq R^2 + \xi_i \quad \text{for } i = 1, \dots, n. \tag{5.5}$$

This is a *convex* optimization problem and can be solved efficiently even for several thousand data points. By use of the *Representer Theorem*, one obtains that the sphere's center point can be expressed as a linear combination $c = \sum_i \alpha_i \varphi(x_i)$ of the embedded data points. This allows an expression for the decision function depending only on kernel evaluation:

$$\begin{aligned} f(x) &= R^2 - \|\varphi(x) - c\|^2_{\mathcal{H}} \\ &= R^2 - k(x,x) + \sum_{i=1}^{n} \alpha_i k(x, x_i) - \sum_{i,j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) \end{aligned} \tag{5.6}$$

where the last sum does not depend on x and can be merged with the constant R^2.

5.3 One-Class Support Vector Machine

A quick geometric check shows that if all data points have the same feature space norm and can be separated linearly from the origin, finding the minimal enclosing sphere is equivalent to finding a maximal margin hyperplane between the data points and the origin.

The *One-Class Support Vector Machine (OC-SVM)* [Schölkopf et al., 2001] relies on this linear construction, solving the corresponding optimization problem in the kernel-induced feature space:

$$\min_{\substack{R\in\mathbb{R}, w\in\mathcal{H} \\ \xi_1,\dots\xi_n\in\mathbb{R}^+}} \|w\|^2_{\mathcal{H}} + \frac{1}{\nu n}\sum_{i=1}^{n} \xi - \rho \tag{5.7}$$

subject to

$$\langle w, \varphi(x_i)\rangle_{\mathcal{H}} \geq \rho - \xi_i, \qquad \text{for } i = 1, \dots, n. \tag{5.8}$$

Since the problem is very similar to the support vector machine optimization, it can be solved efficient by optimization techniques originally developed for binary classification with SVMs. The resulting decision function

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i) - \rho \tag{5.9}$$

is in fact identical to the decision function of an SVM (2.30) with bias term ρ.

Note that many kernels in computer vision, *e.g.* Gaussian and χ^2-kernel, have the properties that $k(x,x) = 1$ for all $x \in \mathcal{X}$. In this case, the underlying geometric problems of SVDD and OC-SVM are identical, and both methods learn the same decisions functions.

5.4 Example: Detection of Anomalous Image Regions

A. Banerjee, P. Burlina, and C. Diehl: *"A Support Vector Method for Anomaly Detection in Hyperspectral Imagery"*, IEEE Transactions on Geoscience and Remote Sensing, 2006

The authors study the problem of detecting pixels in hyperspectral images that differ significantly from the local background. They propose a method based on *support vector data description* with a Gaussian kernel [Banerjee et al., 2006]. To detect anomalous regions, for each pixel SVDD is trained using a local window region surrounding the pixel as input. Subsequently, SVDD is evaluated on the center pixel to determine if it is anomalous using two different threshold criteria. Comparisons to a previous method of anomaly detection shows improved accuracy. Additionally, SVDD is shown to be well-suited to the situation with many spectral bands, because, due to the use of kernels, training and test times scale only linearly with the feature dimensionality.

5.5 Example: Image Retrieval

Y. Chen, X. Zhou, and T. S. Huang: *"One-Class SVM for Learning in Image Retrieval"*, ICIP 2001

The authors propose a one-class SVM setup to rank images for retrieval tasks based on user feedback [Chen et al., 2001]. They argue that one-class models are well suited for such search-like scenarios, because relevant images typically have very similar feature representations, since users typically know well what they are looking for. In contrast, the images that the user is not interested in lie spread all over the feature space. Two-class classification does not handle this situation well, when only very few labeled images from user feedback are available.

Testing their setup on a dataset of 500 images, the authors show that the one-class SVM with linear kernel is not able to improve over the state-of-the-art, but using a Gaussian kernel clearly improves the performance over competing methods.

5.6 Example: Steganalysis

S. Lyu and H. Farid: *Steganalysis Using Color Wavelet Statistics and One-Class Support Vector Machines*, Proceedings of SPIE, 2004

Steganalysis is the task of detecting whether a given image contains a hidden signal that has been embedded steganographically. In this work, [Lyu and Farid, 2004] propose the use of a one-class SVM for this purpose. It is trained using steganography-free images in a multi-scale multi-orientation wavelet representation. Afterwards, images that the OC-SVM classifies as outliers are considered to potentially contain a hidden steganographic signal. A one-class SVM is preferable over a two-class classifier in this setup in order to detect all kinds of steganographic processes, not only known ones for which training data is available.

6

Regression

Regression problems occur frequently in data analysis, and the mathematics underlying *linear regression* techniques is well-understood. However, for high-dimensional problems as they occur in Computer Vision the performance of linear regression methods is often not satisfactory. By *kernelizing* linear regression techniques, non-linear functions can be included into the model class, while still retaining the largest part of the theoretical background. This opens new possibilities for the use of regression methods also in Computer Vision applications, *e.g.* for *pose estimation* or in *image superresolution*.

In the following, we introduce *kernel ridge regression* and *support vector regression* as the most important kernel-based regression techniques, and give examples of their successful application for state-of-the-art computer vision tasks[1].

[1] We do not include *Gaussian Process (GP) Regression* in our discussion, as its interpretation of the kernel is more as a covariance function than as the inner product in a high-dimensional feature space. See [Rasmussen and Williams, 2006] for an excellent introduction to Gaussian Processes.

6.1 Kernel Ridge Regression

Linear regression in its simplest form deals with the problem of predicting a linear function $f : \mathbb{R}^d \to \mathbb{R}$ from samples $x_i \in \mathbb{R}^d$ with corresponding target values $y_i \in \mathbb{R}$. Because f is assumed linear, it can be parameterized by a weight vector $w \in \mathbb{R}^d$ as

$$f(x) = \langle w, x \rangle + b, \tag{6.1}$$

showing that the problem is closely related to the problem of classification studied before. As for classification, we will simplify the notation by assuming that a component of constant value 1 has been appended to the vector x such the *bias term* b can be absorbed into the *weight vector* w.

In contrast to classification, were we only need to learn a function f that fulfills $f(x_i) > 0$ for the samples of one class and $f(x_i) < 0$ for the others, in regression we need to solve the more difficult problem of achieving $f(x_i) \approx y_i$ for all training samples. Different methods for linear regression differ in how they penalize errors, *i.e.* points x_i for which $f(x_i) \neq y_i$. A classical choice is *least-squares regression* that determines the weight vector w by the following minimization problem:

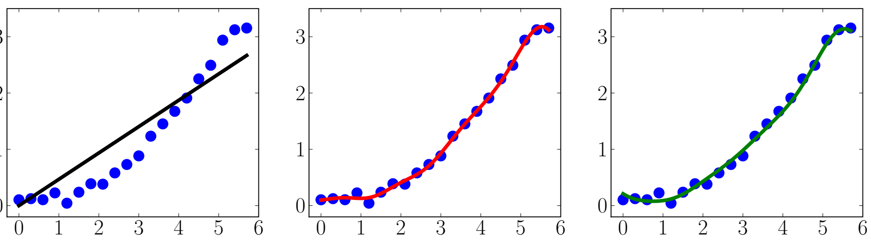
$$\min_{w \in \mathbb{R}^d, b \in \mathbb{R}} \frac{1}{2} \sum_{i=1}^{n} (\langle w, x_i \rangle - y_i)^2 + \lambda \|w\|^2, \tag{6.2}$$

i.e. deviations between $f(x_i)$ and y_i, called the *residua*, are penalized quadratically. The regularization term $\lambda \|w\|^2$ it optional, but typically it improves prediction performance as it penalizes too large weights. Linear least-squares regression is particularly popular because Equation (6.2) has a closed form solution

$$w = X(\lambda Id_n + X^T X)^{-1} y \tag{6.3}$$

where Id_n is the n-dimensional identity matrix, $X = (x_1, \dots, x_n)^t$ is the matrix formed by the sample points and $y = (y_1, \dots, y_n)^t$ is the vector containing the target values.

Kernel ridge regression (KRR) [Saunders et al., 1998] is obtained by *kernelizing* Equation 6.2: for a given kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, we replace x_i by the induced $\varphi(x_i)$. The closed form solution formula is valid in this situation as well, and if we replace X by $\Phi =$


(a) regression without outliers

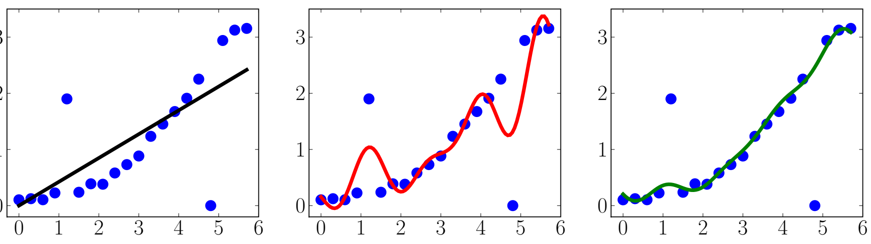

(b) regression with outliers

Figure 6.1 Linear regression (left) is not able to fit non-linear functions. kernel ridge regression (middle) can fit non-linear functions, but is susceptible to outliers. Support vector regression (right) can also perform non-linear regression and is more robust against outliers.

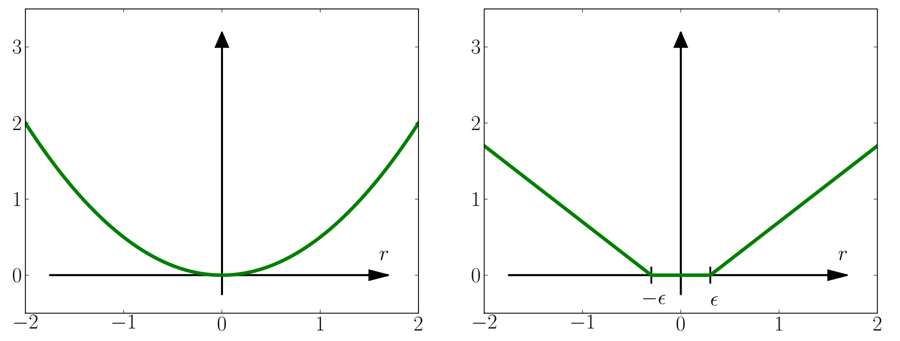
$(\varphi(x_1), \dots, \varphi(x_n))$ in Equation (6.3), we obtain the optimal weight vector $w = \Phi(\lambda Id_n + \Phi^T\Phi)^{-1}y$. As this is a vector in the only implicitly defined Hilbert space $\mathcal{H}$, we cannot represent w itself. However, we can calculate the kernelized prediction function $f(x)$, which is the inner product of w with the image of a test point x under the feature map φ:

$$f(x) = \langle w, \varphi(x) \rangle_{\mathcal{H}} = y(\lambda Id_n + K)^{-1}\kappa(x), \tag{6.4}$$

where $K = (k(x_i, x_j))_{i,j=1}^n$ is the kernel matrix of the training data and $\kappa(x) = (k(x_1, x), \dots, k(x_n, x))^t$ is the vector of kernel values for the test point. Equation (6.4) provides a closed form expression for doing non-linear regression with arbitrary kernel functions.

6.2 Support Vector Regression

Because kernel ridge regression is derived from linear least squared regression it suffers from some of the same drawback. In particular,



(a) Quadratic penalization function (b) ε-insensitive linear penalization function

Figure 6.2 Penalization of residua: kernel ridge regression (left) uses a quadratic cost $\frac{1}{2}r^2$ function to penalize a residual of r. Support vector regression (right) uses a piece-wise linear penalization $\max(0, |r| - \varepsilon)$ that causes residua of up to $|r| \leq \varepsilon$ not to be penalized at all.

the quadratic penalization of the residua makes it very sensitive to outliers. Also, the inverse matrix in Equation (6.4) is generally dense. This means that we have to store all $n(n+1)/2$ unique entries and we have to multiply with a full $n \times n$ matrix to make a prediction. This becomes computationally prohibitive when the number of training samples grows.

As a more robust and sparse non-linear regression technique, *support vector regression (SVR)* has been developed [Drucker et al., 1996]. It can be seen as an adaption of support vector machines to a regression setup. Like kernel ridge regression, support vector regression performs linear regression in an implicit feature space $\mathcal{H}$. However, it uses a different method to penalize the residua: up to a certain tolerance $\varepsilon > 0$, differences between $f(x_i)$ and y_i are not penalized at all. Larger differences are penalized only by their absolute value, not by a quadratic term as Equation (6.2), see Figure 6.2. This makes SVR more robust against outliers.

Mathematically, support vector regression can be termed as the following minimization problem:

$$\min_{\substack{w\in\mathcal{H},\\ \xi_1,\dots,\xi_n\in\mathbb{R}^+,\\ \xi'_1,\dots,\xi'_n\in\mathbb{R}^+}} \quad \|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi'_i) \tag{6.5}$$

subject to

$$y_i - \langle w, \varphi(x_i)\rangle \leq \varepsilon + \xi_i, \qquad \text{for } i = 1, \dots, n, \tag{6.6}$$

$$\langle w, \varphi(x_i)\rangle - y_i \leq \varepsilon + \xi_i', \qquad \text{for } i = 1, \dots, n. \tag{6.7}$$

By use of the Representer Theorem, we obtain $w = \sum_{i=1}^{n} \alpha_i \varphi(x_i)$, which allows us to express the system (6.5)–(6.7) without need of φ only in terms of kernel function evaluations. A detailed derivation can be found *e.g.* in [Schölkopf and Smola, 2002].

6.3 Example: Blind Image Deconvolution

D. Li, R. M. Mersereau, and S. Simske: *Blind Image Deconvolution Using Support Vector Regression* ICASSP 2005.

This work performs blind image deconvolution, *i.e.* it removes the effect of an unknown blur kernel from images. [Li et al., 2007] propose a method relying on support vector regression with a Gaussian kernel function. The training samples consist of small image patches that have been artificially blurred by different filters, and the target value is the original grayvalue of the target pixel location. With the resulting regression function, the value of each pixel position is predicted independently. The authors show that the proposed method perform much better than the popular Lucy-Richardson maximum likelihood method, in particular for noisy input images.

6.4 Example: Single Image Superresolution

K. S. Ni, S. Kumar, N. Vasconcelos, T. Q. Nguyen: *Single Image Superresolution Based on Support Vector Regression*, ICASSP 2006.

A problem related to blind image deconvolution is *single image superresolution*, *i.e.* the task of converting a low-resolution image into a high-resolution one. [Ni and Nguyen, 2007] use DCT coefficients of downscaled and original sized image patches as training data for support vector regression with a Gaussian kernel. They show that the resulting system performs better than previous approaches, including support vector regression in the pixel domain. In related work, [Kim and Kwon, 2008] observed that the ϵ-robustness property of support vector regression is not really necessary in typical superresolution tasks and showed that the simpler to compute kernel-ridge regression is as well or better suited to the problem of single image superresolution.

6.5 Example: Pose Estimation

Y. Li, S. Gong, J. Sherrah, and H. Liddell: *"Support Vector Machine based Multi-view Face Detection and Recognition"*, Image and Vision Computing, 2004.

The paper uses support vector regression to estimate the pose (yaw and tilt) of faces from single images [Li et al., 2004]. The system is trained with face images of known pose that are represented by PCA-coefficients of normalized pixel intensities. At test time, the method exhaustively scans the image, evaluating the support vector regression for each candidate location. Depending on the estimated yaw and tilt, different classifiers (frontal or profile face detectors) are used to decide if a face is present at the current location. For faces in frontal view, also face recognition is performed. The authors observe that by first estimating the faces' pose it is possible to *a priori* choose the right face detector. This avoids the problem of having to combine several classifiers responses that possibly contradict each other, as it would be necessary by just evaluating the face detectors for all poses for every candidate region.

7

Dimensionality Reduction

Data representations in computer vision are often high-dimensional. An image treated as a vector can have millions of dimensions and even the most compact histogram representations often still have several hundred entries. This introduces a number of problems for practical applications, *e.g.* high storage and computational requirements. Having a large number of non-informative signal dimensions also makes it more difficult to learn robust classification or regression functions. High-dimensional data spaces are also problematic, because humans have problems to build an intuitive understanding of such data source, when no suitable visualization techniques is available. Dimensionality reduction techniques can overcome all of these problems by reducing the number of signal dimensions while preserving as much as possible of the original information. This has made them valuable tools in computer vision.

7.1 Kernel Principal Component Analysis

The best known technique for dimensionality reduction is (linear) *principal component analysis (PCA)*. It identifies those subspaces in the feature space in which the data varies strongest. By projecting to these subspaces, the data is transfered into a low-dimensional coordinate system while retaining the largest possible amount of signal contents.

Mathematically, PCA can be formulated as a maximization problem. We are given a set of datapoints $x_1, \dots, x_n$ that we assume to have zero mean, *i.e.* $\frac{1}{n}\sum_i x_i = 0$ (we will discuss the general case of $\frac{1}{n}\sum_i x_i \neq 0$ in the later *Data Centering* section). The first PCA coordinate direction w_1 is defined by

$$w_1 = \operatorname*{argmin}_{w \in \mathbb{R}^d,\ \|w\|=1} \sum_{i=1}^{n} \|x_i - \langle x_i, w\rangle w\|^2. \tag{7.1}$$

The squared norm term measures the distortion introduced by projecting a data point x_i to the direction defined by w. Therefore, Equation (7.1) identifies the projection direction w_1 causing the minimal distortion over the whole dataset. To find subsequent PCA projection directions, we solve the same optimization problem as above, but with the additional constraint that any direction w_{k+1} must be orthogonal to all previous $w_1, \dots, w_k$. A classical analysis shows that the first K PCA directions are just the eigenvectors of the K largest eigenvalues of the data covariance matrix $C = \frac{1}{n}\sum_i x_i x_i^t$ [Jolliffe, 1986].

7.1.1 A inner product formulation of PCA

In order to kernelize PCA, we rewrite the problem in terms of inner products. Inserting $C = \frac{1}{n}\sum_i x_i x_i^t$ into the eigenvalue equation $\lambda_k w_k = C w_k$, where λ_k is the k-th eigenvalue of C, we obtain

$$w_k = \frac{1}{n\lambda_k} \sum_{i=1}^{n} \langle x_i, w_k\rangle x_i, \tag{7.2}$$

which shows that any w_k can be written as a linear combination $\sum_i \alpha_i^k x_i$ of the data points, and w_k is completely determined by the coefficient vector $\alpha^k = (\alpha_1^k, \dots, \alpha_n^k)$ built from the inner product values $\alpha_i^k := \frac{1}{n\lambda_k}\langle x_i, w_k\rangle$ for $i = 1, \dots, n$. Inserting w_k back into this expression for α^k, we obtain

$$n\lambda_k \alpha_i^k = \sum_{j=1}^{n} \alpha_j^k \langle x_i, x_j\rangle. \tag{7.3}$$

Since the coefficient α_i^k completely determine the projection direction, we have found a formulation of PCA that only requires information about inner products between data points.

7.1.2 Kernelization

We can now kernelize PCA by replacing all occurrences of the data points x_i by their nonlinearly transformed image $\varphi(x_i) \in \mathcal{H}$, induced by a kernel function k. In result, we obtain a method that perform linear PCA in $\mathcal{H}$, which is non-linear with respect to the original data space. Using $k(x_i, x_j) = \langle\varphi(x_i), \varphi(x_j)\rangle_{\mathcal{H}}$, Equation (7.3) becomes

$$n\lambda_k \alpha_i^k = \sum_{j=1}^{n} \alpha_j^k k(x_i, x_j). \tag{7.4}$$

showing that $(\alpha^k)_{i=1,\dots,n}$ is an eigenvector of the kernel matrix $K = (k(x_i, x_j))_{i,j=1}^n$ with corresponding eigenvalue $n\lambda_k$. Having computed $\alpha_1, \dots, \alpha_K$, we obtain the lower-dimensional coordinates $\tilde{x}_i \in \mathbb{R}^K$ for any x_i by the projection coefficient of $\varphi(x_i)$ to the kernelized $w_1, \dots, w_k$. By means of $w_k = \sum_j \alpha_j^k \varphi(x_j)$ this is

$$[\tilde{x}_i]_k = \langle\varphi(x_i), w_k\rangle \;=\; \sum_{j=1}^{n} \alpha_j^k \langle\varphi(x_i), \varphi(x_j)\rangle \;=\; n\lambda_k \alpha_i^k, \tag{7.5}$$

where in the last step we have made use of Equation (7.4).

Equation (7.5) shows that, despite the rather technical derivation, kernel PCA is very easy to apply in practice: the low-dimensional data matrix $\tilde{X} = [\tilde{x}_1, \dots, \tilde{x}_n]^t \in \mathbb{R}^{n\times K}$ can be calculated by finding the top K eigenvectors of the kernel matrix $K \in \mathbb{R}^{n\times n}$ and multiplying each eigenvector by its corresponding eigenvalue. As an additional bonus, kernel-PCA allows us to derive vector representation of input data that does not have a linear structure by itself, *e.g.* graph, sequences or images.

7.1.3 Data Centering

For the sake of simplicity, we have ignored in the previous discussion the condition of our original PCA formulation that the data points must have a mean of 0. If the data coordinates are explicitly available, this condition is easily established by *centering* the data, *i.e.* subtracting the data mean $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$ from each data point. Centering corresponds to a global translation of all points and therefore does not affect our intuition of identifying the directions of maximal data variance.

In the kernelized situation, we do not have access to an explicit coordinate representation of the data points $\varphi(x_i)$, and we can therefore not modify them individually to make the data set have zero mean. However, we can calculate the kernel values k^c that correspond to the inner product values between the embedded data points if they had been centered by subtracting their mean $\mu = \frac{1}{n}\sum_{i=1}^{n} \varphi(x_i)$:

$$\begin{aligned}
\overline{k}(x_i, x_j) &= \langle \varphi(x_i) - \mu, \varphi(x_j) - \mu \rangle && (7.6)\\
&= \langle \varphi(x_i) - \frac{1}{n}\sum_{k=1}^{n} \varphi(x_k)\,, \varphi(x_j) - \frac{1}{n}\sum_{l=1}^{n} \varphi(x_l) \rangle && (7.7)\\
&= k(x_i, x_j) - \frac{1}{n}\sum_{k=1}^{n} k(x_k, x_j) - \frac{1}{n}\sum_{l=1}^{n} k(x_i, x_l) + \frac{1}{n^2}\sum_{k=1}^{n}\sum_{l=1}^{n} k(x_k, x_l). && (7.8)
\end{aligned}$$

Thus, we can perform kernel-PCA for any kernel matrix K by first computing the *centered kernel matrix*

$$\overline{K} = K - \frac{1}{n}\mathbf{1}_n K - \frac{1}{n} K\,\mathbf{1}_n + \frac{1}{n^2}\mathbf{1}_n K\,\mathbf{1}_n \qquad (7.9)$$

where $\mathbf{1}_n$ is the $n \times n$ matrix of all ones, and applying the eigenvector decomposition of the previous section to it.

7.2 Kernel Discriminant Analysis

As an unsupervised technique, PCA only uses the variance of the data as an indicator what projection directions are most informative. If *label information* is available as part of the training data, *linear discriminant analysis (LDA)* is often a better choice. It identifies the K-dimensional subspace that best separates points of different class membership while grouping together points that belong to the same class.

Mathematically, LDA maximizes the ratio of *between-class scatter* and *within-class scatter*. For a dataset consisting of a part $X^+ =$

$(x_1^+, \dots, x_{n^+}^+)$ with positive class label and a part $X^- = (x_1^-, \dots, x_{n^-}^-)$ with negative class label, the first LDA projection direction w_1 is given by

$$w_1 = \operatorname*{argmax}_{w \in \mathbb{R}^d,\ \|w\|=1} \frac{w^t \Sigma_b w}{w^t \Sigma_w w} \tag{7.10}$$

where $\Sigma_b = (\mu^+ - \mu^-)(\mu^+ - \mu^-)^t$ is the between-class scatter matrix, $\Sigma_w = \sum_{i=1}^{n^+}(x_i^+ - \mu^+)(x_i^+ - \mu^+)^t + \sum_{i=1}^{n^-}(x_i^- - \mu^-)(x_i^- - \mu^-)^t$ is the sum of within-class scatter matrices, and $\mu^\pm = \frac{1}{n_\pm}\sum_{i=1}^{n^\pm} x_i^\pm$ denote the positive and negative class means [McLachlan, 1992]. The optimization problem (7.10) can be solved efficiently by means of a generalized eigenvalue problem.

As for PCA, more projection directions can be found by optimizing the same expression with additional constraints that any new direction must be orthogonal to all previous ones.

7.2.1 Kernelization

LDA can be kernelized by similar calculations as were required for PCA resulting in *kernel discriminant analysis (KDA)* [Mika et al., 1999]. Instead of Σ_b, we need to form $M = (M_+ - M_-)(M_+ - M_-)^t$ with $[M_\pm]_j = \frac{1}{n^\pm}\sum_{i=1}^{\pm} k(x_j, x_i^\pm)$ for $j = 1, \dots, n$ and slightly abusing the notation such that $X^+ \cup X^- = \{x_1, \dots, x_n\}$. The role of Σ_w is taken over by $N = K_+(Id_{n^+} - \frac{1}{n^+}\mathbf{1}_{n^+})K_+^t + K_-(Id_{n^-} - \frac{1}{n^-}\mathbf{1}_{n^-})K_-^t$ where $(K_\pm)_{ij} = k(x_i, x_j^\pm)$ for $i = 1, \dots, n$ and $j = 1, \dots, n^\pm$. With this notation, the solution $\alpha = (\alpha_1, \dots, \alpha_n)$ of the maximization problem

$$\operatorname*{argmax}_{\alpha \in \mathbb{R}^d} \frac{\alpha^t M \alpha}{\alpha^t N \alpha} \tag{7.11}$$

yields the coefficients of the expansion $w_1 = \sum_{i=1}^{n} \alpha_i^k \varphi(x_i)$. As in the case of PCA, more projection directions can be found by maximizing equation (7.11) with orthogonality constraints.

7.3 Kernel Canonical Correlation Analysis

canonical correlation analysis (CCA) generalizes PCA to the situation of *paired data*, *i.e.* a dataset $X = \{x_1, \dots, x_n\}$ where all samples are available in (at least) two representation at the same time: $\{x_1^1, \dots, x_n^1\} \subset \mathcal{X}^1$ and $\{x_1^2, \dots, x_n^2\} \subset \mathcal{X}^2$ with known correspondences between x_i^1 and x_i^2 for $i = 1, \dots, n$. CCA learns separate projections from each data domain $P^1 : \mathcal{X}^1 \to \mathbb{R}^K$ and $P^2 : \mathcal{X}^2 \to \mathbb{R}^K$ such that the low-dimensional representations $\{P^1(x_i^1)\}_{i=1,\dots,n}$ and $\{P^2(x_i^2)\}_{i=1,\dots,n}$ are maximally correlated. CCA can also be thought of as a generalization of LDA, where instead of a label y_i only a second data representation x_i^2 is available.

7.3.1 Kernelization

A direct kernelization of linear CCA yields degenerate projection directions if either the kernel matrices K_1 or K_2 are invertible. One can however regularize the problem and obtains the maximization problem

$$\operatorname*{argmax}_{\alpha\in\mathbb{R}^n, \beta\in\mathbb{R}^n} \frac{\alpha^t K_1 K_2 \beta}{\sqrt{\alpha^t K_1^2 \alpha + \epsilon_1 \alpha^t K_1 \alpha}\sqrt{\beta^t K_2^2 \beta + \epsilon_2 \beta^t K_2 \beta}}, \tag{7.12}$$

where $\epsilon_1, \epsilon_2 > 0$ are regularization parameters [Lai and Fyfe, 2000].

As for LDA, the solutions to Equation (7.12) can be found by a generalized eigenvalue problem, which yields projection directions of maximal correction: $w_1 = \sum_{i=1}^n \alpha_i \varphi_1(x_i^1)$ and $w_2 = \sum_{i=1}^n \beta_i \varphi_2(x_i^2)$. The lower-dimensional representations can also be calculated directly by $P(x) = \sum_{i=1}^n \alpha_i k_x(y, y_i)$ for $x \in \mathcal{X}$ and $Q(y) = \sum_{i=1}^n \beta_i k_y(y, y_i)$ for $y \in \mathcal{Y}$. As before, further projection directions can be computed by searching for solutions orthogonal to the earlier α and β. This procedure is equivalent to searching for the generalized eigenvectors corresponding to the largest K eigenvalues.

7.4 Example: Image Denoising

S. Mika, B. Schölkopf, A. Smola, K.-R. Müller, M. Scholz, G. Rätsch: *"Kernel PCA and De-Noising in Feature Spaces"*, NIPS 1999

[Mika et al., 1999] propose kernel PCA for image denoising. In contrast to denoising with linear PCA this task is non-trivial, because the low-dimensional representations found by kernel PCA are not images but vectors in the implicit feature space. To obtain a denoised version of an image, the authors therefore solve a *pre-image problem*, *i.e.* they find the image that best approximates the feature space representation. Examples from the USPS dataset of handwritten digits show superior performance of the proposed method compared to linear PCA.

It is noteworthy that [Kwok and Tsang, 2004] later introduced a different optimization technique to solve the preimage problem of kernel-PCA for certain kernel types, achieving improved denoising results and higher numerical stability.

K. I. Kim, M. O. Franz, and B. Schölkopf: *"Iterative Kernel Principal Component Analysis for Image Modeling"*, PAMI 2005

[Kim et al., 2005] study kernel PCA for the denoising of natural images, in particular face images. Since this requires very large data sets, they introduce the *kernel Hebbian algorithm*, a new iterative training technique that allows one to perform kernel PCA for larger dataset than could be handled previously, and they show improved performance for denoising and single image superresolution.

7.5 Example: Image Annotation

D. R. Hardoon, C. Saunders, S. Szedmak, and J. Shawe-Taylor: *"A Correlation Approach for Automatic Image Annotation"*, International Conference on Advanced Data Mining and Applications (ADMA), 2006.

[Hardoon et al., 2006] propose a method to label images with keywords. Using a training set that consists of images with corresponding text captions, the authors use kernel-CCA between the image and the text domain to find directions in feature space that are most correlated with the keywords. For a new image, the low dimensional image representation is calculated and the keywords most correlated to it are selected. Experiments on a standard dataset of images with captions show that kernel-CCA achieves significantly higher precision and recall than *Latent Semantic Analysis* [Deerwester et al., 1990] and a vector-valued SVM variant.

7.6 Example: fMRI Image Analysis

D. R. Hardoon, J. Mourão-Miranda, M. Brammer, and J. Shawe-Taylor: *"Unsupervised Analysis of fMRI Data using Kernel Canonical Correlation"*, Neuroimage (37) 4, 2007

Hardoon et al. [Hardoon et al., 2007] propose a method for fMRI image analysis by applying kernel-CCA to the pairs of image stimuli and resulting fMRI images, both represented by SIFT features. The authors test their approach in a classification task of pleasant versus unpleasant image stimuli. They show that the first projection direction found by KCCA without explicit training labels achieves nearly as good classification results as training a support vector machine from labeled examples.

7.7 Example: Shape Representation

S. Dambreville, Y. Rathi and A. Tannenbaum: *"Shape-Based Approach to Robust Image Segmentation using Kernel PCA"*, CVPR 2006

Image segmentation benefits strongly if one knows the shape of objects contained in the image. [Dambreville et al., 2006] integrate a shape prior into level set segmentations using a kernel PCA representation with Gaussian kernel for *signed distance functions* descriptions of training shapes. In four different experiments the authors show that their proposed method allows robust shape detection and tracking even in noisy or blurry images and videos.

L. Wang, and D. Suter: *"Recognizing Human Activities from Silhouettes: Motion Subspace and Factorial Discriminative Graphical Model"*, CVPR 2007

[Wang and Suter, 2007] use kernel PCA with a Gaussian kernel to compactly represent peoples' silhouettes. In combination with a factorial conditional random field, this allows them to perform action classification in videos more reliably than previous approaches that were based on pixel-wise silhouette representation.

8

Clustering

Clustering is a classical method for the detection of structure in large and unlabeled datasets. It partitions the data into a small number of groups of samples, typically based on their distances to a centroid or on pairwise similarities. In the following we introduce three clustering methods that rely on kernels for this procedure: *kernel-PCA clustering*, *kernel vector quantization* and *support vector clustering*.

8.1 Kernel-PCA Clustering

Possibly the most popular clustering technique is the K-means algorithm [MacQueen, 1967] that identifies a set of K centroids $c_1, \dots, c_K$ with the goal to minimize the L^2 distortion caused if each sample is replaced by the centroid it lies closest to:

$$\operatorname*{argmin}_{c_1,\dots,c_K \in \mathbb{R}^d} \quad \sum_{i=1}^{n} \min_{j=1,\dots,K} \|x_i - c_j\|^2 \tag{8.1}$$

Because K-means clustering requires only calculations of the distance between samples and prototype vectors and of the mean vector for finite sets of samples, it can be kernelized in a straight forward way, see [Girolami, 2002]. In contrast to the situation in $\mathbb{R}^d$, we cannot represent the centroids explicitly, because they are elements of the implicit Hilbert space induced by the kernel function. However, we do know their representation as linear combinations of the data points: $c_k = \sum_j \alpha_j^k \varphi(x_j)$. Each distance calculation between a prototype and a data sample therefore requires the use of a large part of the kernel matrix $K_{ij} = k(x_i, x_j)$. Due to the iterative nature of the K-means procedure, many such calculations are required before convergence, which makes kernel K-means a computationally expensive procedure.

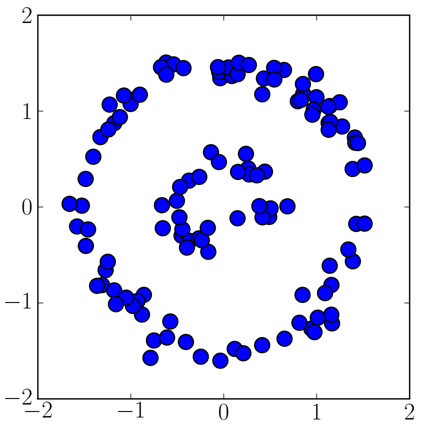
As a more efficient approximation, *kernel-PCA clustering* has been proposed. It makes use of the fact that *kernel-PCA* allows us to find an explicit vector representation of the dataset that approximates the kernel-defined inner product, and thereby the distances between samples. This allows us to come up with an efficient two-step clustering procedure:

Kernel PCA Clustering. Let $X = \{x_1, \dots, x_n\} \subset \mathcal{X}$ be a dataset to be clustered and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a kernel function.

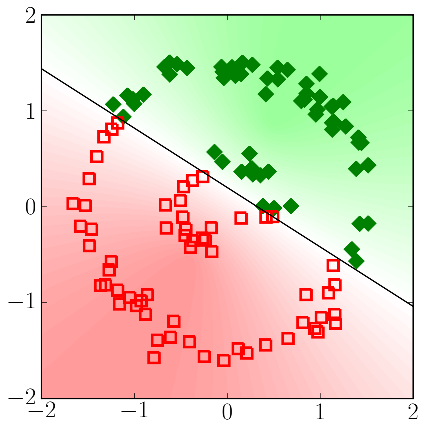
(1) Perform kernel-PCA on the dataset, obtaining a vector representation $\hat{x}_1, \dots, \hat{x}_n \in \mathbb{R}^d$.
(2) Apply ordinary K-means clustering to the vectors $\hat{x}_1, \dots, \hat{x}_n$.

Because *kernel-PCA clustering* is very easy to implement and computationally efficient, it has become one of the most popular methods for clustering with a non-linear distance functions[1]. Figure 8.2 illustrates the working principle of *kernel-PCA clustering*. The top row shows a dataset consisting of two ring shaped clusters that cannot be separated linearly. Ordinary K-means therefore fails to identify the cluster structure. Preprocessing the dataset with *kernel-PCA* (here with a graph-based kernel) results in a distribution of datapoints with well separated components. These are also identified by K-means, resulting in an error free clustering in the original coordinate system.

[1] The most popular method would be *Spectral Clustering* that uses *Laplacian Eigenmaps* for dimensionality reduction. *Spectral Clustering* can be shown to be kernel PCA clustering for a special choice of kernel, but it is better understood by a graph-based interpretation that is only indirectly related to kernel methods.

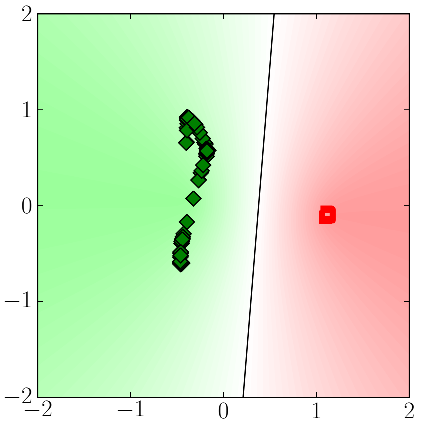


(a) *Two Rings* dataset

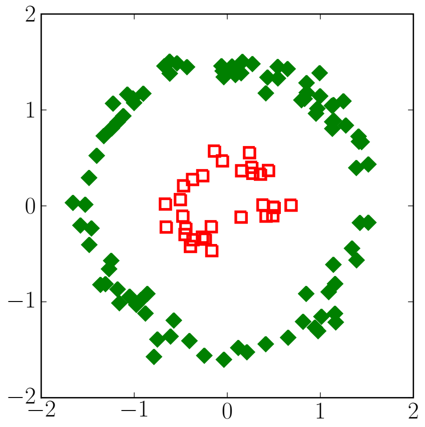


(b) Result of K-means clustering

Figure 8.1 K-means fails to clusters if they are linearly separable (left), because it can only model linear decision boundaries between clusters (right).



(a) K-means in kernel-PCA space



(b) resulting kernel-PCA clustering

Figure 8.2 Preprocessing the *two rings* dataset with *kernel-PCA* (here with *commute distance* kernel) creates a representation of the dataset with clearly separated cluster structure (left). Clustering this representation with K-means results in a perfect separation of the intended data clusters (right).

8.2 Kernel Vector Quantization

In *K-means* and *kernel-PCA clustering* the number of clusters is a free parameter that needs to be specified *a priori*. The algorithms then determines the centroids minimizing the total distortion if all samples are quantized to the centroid closest to them. *Learned Vector Quantization (LVQ)* is a clustering method taking a dual approach to this: for a fixed maximal distortion value θ, it determines how many and which centroids are necessary such that the distortion between any samples and its nearest centroid does not exceed the value θ. By choosing the centroids as a subset of the data points instead of linear combinations, one can formulate the problems as a boolean optimization problem:

$$\operatorname*{argmin}_{z_1,\dots,z_n \in \{0,1\}} \sum_{i=1}^{n} z_i \tag{8.2}$$

subject to

$$\sum_{i=1}^{n} z_j d_{ij} \geq 1 \qquad \text{for } j = 1, \dots, n, \tag{8.3}$$

where $d_{ij} = 1$ if $\|x_i - x_j\| \leq \theta$ and $d_{ij} = 0$ otherwise. The resulting variables z_i are indicators, whether the data point x_i is required as a centroid or not. Clusters are formed subsequently by assigning each data point to its closest cluster centroid.

An interesting aspect of the above optimization problem is the fact that the samples x_i only appear in it through their binarized pairwise distances d_{ij}. This allows us to kernelize LVQ by replacing the Euclidean distance by the distance in a Hilbert space $\mathcal{H}$ induced by a kernel function k. Such distances can be expressed by only kernel evaluations:

$$\begin{aligned} \|\varphi(x_i) - \varphi(x_j)\|^2_{\mathcal{H}} &= \langle \varphi(x_i) - \varphi(x_j), \varphi(x_i) - \varphi(x_j) \rangle_{\mathcal{H}} \\ &= \langle \varphi(x_i), \varphi(x_i) \rangle_{\mathcal{H}} + \langle \varphi(x_j), \varphi(x_j) \rangle_{\mathcal{H}} - 2\langle \varphi(x_i), \varphi(x_j) \rangle_{\mathcal{H}} \\ &= k(x_i, x_i) + k(x_j, x_j) - 2k(x_i, x_j), \end{aligned}$$

which allows us to calculate a kernelized version of the matrix d_{ij} directly from the entries of the kernel matrix. The resulting procedure is called *kernel vector quantization (KVQ)*.

To allow the solution of large problems, one typically convexifies the optimization problem by relaxing the constraints $z_i \in \{0, 1\}$ to $z_i \in [0, 1]$. The result is a *linear program*, which can be solved efficiently

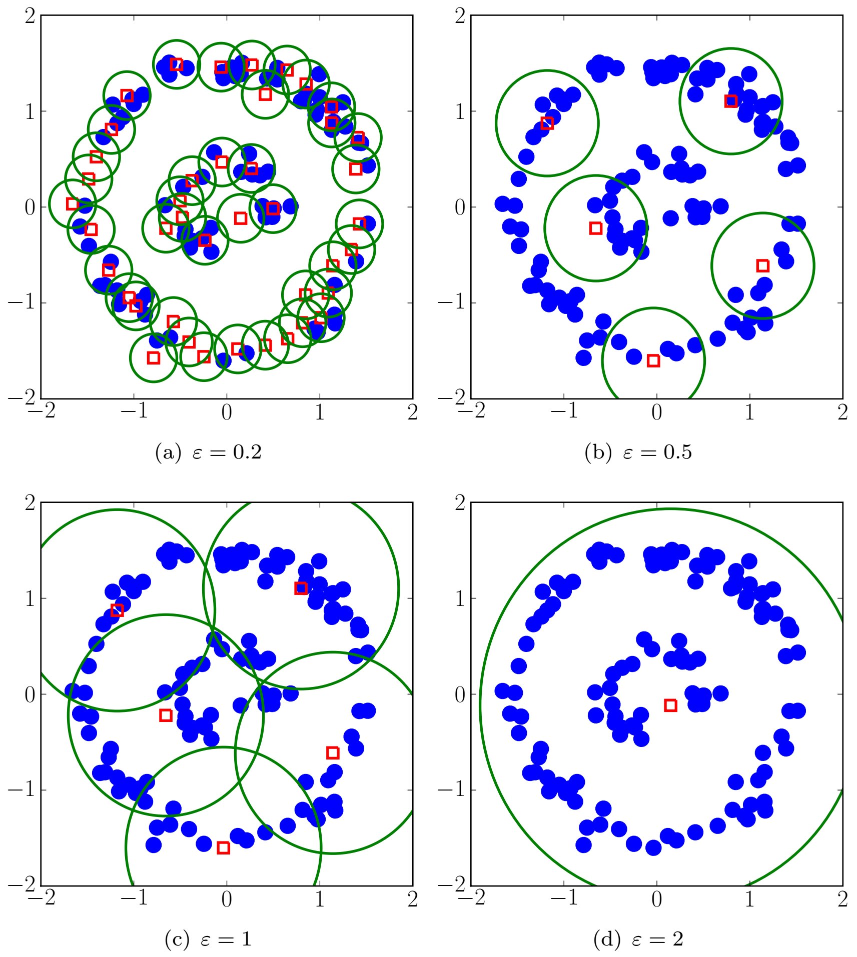


Figure 8.3 (Kernel) vector quantization. For any value $\theta > 0$, one identifies prototypes that provide a minimal covering of the data set by balls of radius θ. The larger the radius, the fewer prototypes are required.

even for tens of thousands of samples. However, the result might have more non-zero z_i, and therefore centroids, than necessary. In practice, postprocessing operations and iterative schemes have been proposed to bring the related solution closer to the optimal binary one [Tipping and Schölkopf, 2001; Weston et al., 2003].

8.3 Support Vector Clustering

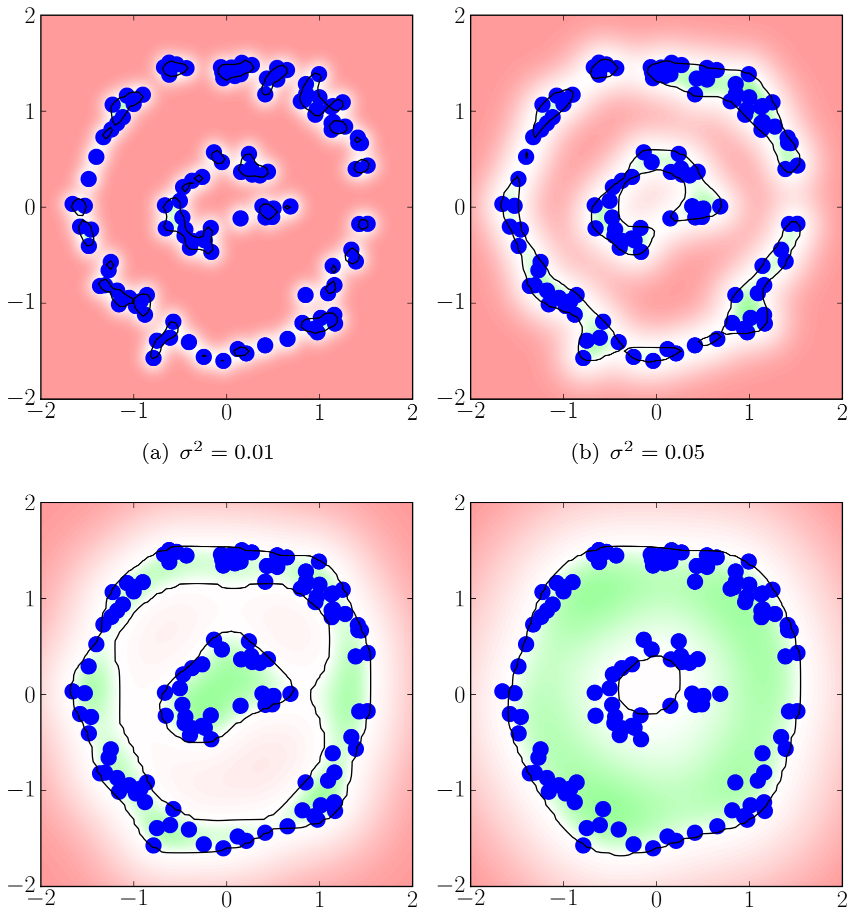
Support vector clustering (SVC) [Ben-Hur et al., 2002] differs from *K-means* and *vector quantization* based techniques in that it does not define clusters by a set of prototypes. Instead, it relies on the fact that the feature mapping $\varphi : \mathcal{X} \to \mathcal{H}$ induced by a Gaussian kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is topologically non-trivial, *i.e.* φ can map regions that are far apart in the original space to nearby locations in the kernel induced feature space $\mathcal{H}$. SVC works by first applying *support vector data description* with a Gaussian kernel on the dataset. While in feature space, the resulting *inlier region* is just a sphere S, the corresponding region $\varphi^{-1}(S)$ in the original data space is non-linearly distorted and often consists of multiple components. SVC defines each connected component of the inlier region to be one data cluster, *i.e.* two data points are assigned the same cluster label, if there is a path in the original data space connecting the two points without leaving the inlier region $\varphi^{-1}(S)$.

By this construction, the number of clusters that SVD identifies is not determined a priori, but it follows from the distribution of data points in the data space. However, the Gaussian's bandwidth parameter can be used to influence on the number of clusters, as a smaller bandwidth causes the preimage $\varphi^{-1}(S)$ to be more irregular and less connected. By use of SVDD's ν-parameter, SVD is one of the few clustering methods that can deal with outliers in the data, as it assigns no cluster label to those points.

8.4 Example: Unsupervised Object Categorization

T. Tuytelaars, C. H. Lampert, M. B. Blaschko, W. Buntine: *"Unsupervised Object Discovery: A comparison"*, IJCV 2009.

Tuytelaars et al. [2009] compare different centroid-based clustering methods with probabilistic topic models for the automatic detection of object categories in images collections. An experimental evaluation is performed on two large-scale datasets in bag of visual word representations where the goal is the unsupervised detection of 20 or 23 object categories. The results show that linear dimensionality reduc-



(a) $\sigma^2 = 0.01$ (b) $\sigma^2 = 0.05$

(c) $\sigma^2 = 0.2$ (d) $\sigma^2 = 0.5$

Figure 8.4 Support vector clustering (Gaussian kernel, $\nu = 0.05$). For small bandwidth parameters σ^2, many small clusters are identified. When σ^2 is increased, the decision boundary becomes smoother and fewer but larger clusters are formed.

tion decreases the quality of clustering found by a subsequent K-means clustering step, whereas *kernel-PCA* with χ^2-kernels improves the clustering performance. The same holds for *Laplacian Eigenmaps* [Belkin and Niyogi, 2003]. Both setups are shown to be superior to the tested generative topic models for small and medium size codebooks (< 10000 entries).

M. B. Blaschko, and C. H. Lampert: *"Correlational Spectral Clustering"*, CVPR 2008.

It is common in computer vision to have data in more than one modality, *e.g.* images with text captions. [Blaschko and Lampert, 2008a] introduce *Correlational Spectral Clustering (CSC)*, a technique for finding clusters in such data that comes in multiple representations. Instead of kernel-PCA, CSC uses kernel-CCA to find a lower-dimensional representation of the data that jointly reflects all channels of information of the training data. This is followed by K-means for the clustering step. Experiments on different image datasets with text captions show that the method is superior to techniques based on PCA or kernel-PCA of the single modalities and to preprocessing with linear CCA.

8.5 Example: fMRI Activation Detection

D. Wang, L. Shi, D. S. Yeung, P.-A. Heng, T.-T. Wong, and E. C. C. Tsang: *"Support Vector Clustering for Brain Activation Detection"*, Medical Image Computing and Computer-Assisted Intervention, MICCAI 2005.

Modern medical imaging techniques, such as fMRI, produce large amounts of image data in each measurement, that need to be analyzed automatically for efficient further processing. [Wang et al., 2005] propose a technique to categorize regions of neural activity in the brain using support vector clustering with a Gaussian kernel on top of an image representation by Fourier coefficients. Experiments on simulated and real fMRI data show clearly superior performance of SVC compared to K-means clustering with PCA or ICA preprocessing. A particular advantage of the proposed methods it its robustness against outliers due to SVC's ν-property.

9

Non-Classical Kernel Methods

Support vector clustering in the *"Clustering"* chapter was the first example we saw of a kernel method that did not have a linear counterpart, but explicitly required a non-linear feature mapping in order to give meaningful results. In this section, we study two more concepts that crucially rely on the rich and often infinite-dimensional structure of induced Hilbert spaces, the *prediction of structured objects* and the *empirical Hilbert-Schmidt independence criterion (HSIC)*.

9.1 Structured Prediction

Classical kernel methods like classification and regression learn mappings $\mathcal{X} \to \{1, \dots, K\}$ and $\mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is an arbitrary set, provided we can define a kernel over it. However, many realistic problems require decisions on several quantities at once, *e.g.* in pixel-wise image segmentation, or in the prediction of richly structured objects, *e.g.* interaction graphs. Such tasks can be subsumed under the label of *structured prediction*, where the task is to learn a prediction function $f : \mathcal{X} \to \mathcal{Y}$ for any suitable $\mathcal{Y}$.

9.1.1 Structured Support Vector Machines

The *structured (output) support vector machine (S-SVM)* provides an extension of SVM classification to structured prediction problems [Tsochantaridis et al., 2005]. Its key idea is to define a *joint kernel function*

$$k_{joint} : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}. \tag{9.1}$$

between two *(sample,label)*-pairs, and learn a kernelized *compatibility* function $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ based on this. Subsequently, a structured prediction function $F : \mathcal{X} \to \mathcal{Y}$ is obtained by maximizing the compatibility between any test input x and all possible outputs y:

$$F(x) := \operatorname*{argmax}_{y \in \mathcal{Y}} f(x, y). \tag{9.2}$$

9.1.2 Structured SVM Training

Structured support vector machines are trained using a maximum-margin principle similar to the ordinary SVM training procedure. For a given training set $\{(x_1, y_1), \dots, (x_n, y_n)\}$, one solves the convex optimization problem

$$\min_{\substack{w \in \mathcal{H}, \\ \xi_1, \dots, \xi_n \in \mathbb{R}^+}} \quad \|w\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi_i \tag{9.3}$$

subject to the margin constraints[1]

$$\langle w, \Phi(x_i, y_i) \rangle_{\mathcal{H}} - \langle w, \Phi(x_i, y) \rangle_{\mathcal{H}} \geq \Delta(y, y_i) - \xi_i, \quad \text{for all } y \neq y_i, \tag{9.4}$$

for $i = 1, \dots, n$, where $\mathcal{H}$ is the feature space belonging to k_{joint} and $\Phi : \mathcal{X} \times \mathcal{Y} \to \mathcal{H}$ is the induced implicit feature map. Similarly to the

[1] More precisely, the constraints (9.4) define a structured SVM *with margin-rescaling*. The variant of *slack rescaling*, which is also proposed in [Tsochantaridis et al., 2005], uses the constraints

$$\langle w, \Phi(x_i, y_i) \rangle_{\mathcal{H}} - \langle w, \Phi(x_i, y) \rangle_{\mathcal{H}} \geq \Delta(y, y_i)(1 - \xi_i) \quad \text{for all } y \neq y_i,$$

for $i = 1, \dots, n$. This has some theoretical advantages but is used less frequently in practice, because the resulting optimization problem is more difficult to solve.

Crammer-Singer multiclass SVM (4.4), the constraints (9.4) enforce a margin for each training example x_i between the correct label y_i and any other label $y \neq y_i$. However, in contrast to the multiclass situation, the margin is not of constant size 1, but it is given by a loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$. This allows the integration of the fact that in a structured output space, *e.g.* the space of all sequences, some output values y even if not completely correct, are still "less wrong" than others and should therefore not be penalized as strongly. The joint kernel function k_{joint} also generalizes of the multiclass situation, which can be recovered by setting

$$k_{multiclass}(\,(x,y)\,,(x',y')\,) = k_x(x,x')\delta_{y=y'} \tag{9.5}$$

for any kernel k_x over $\mathcal{X}$. The greater flexibility in choosing k_{joint} allows the formulation of many computer vision problems in terms of a structured SVM that before were mainly treated by heuristic or only locally optimal techniques.

9.2 Dependency Estimation

An important question in data analysis is to determine if two measured quantities are independent of each other. Mathematically, one studies this by considering the two sets of measurements $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ as realization of random variables $\mathbf{x}$ and $\mathbf{y}$ and tries to analyze if their joint probability distribution function factorizes, *i.e.* if $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x})\,p(\mathbf{y})$.

Elementary techniques for this measure dependence by calculating *data moments*, *e.g.* the sample *covariance* or *correlation* matrices. However, while correlation between two random quantities implies their statistical dependence, the converse is not true: two uncorrelated random variables can still have strong dependencies between them, see Figure 9.1 for an illustration.

One way to overcome this problem is the introduction of non-linearity: a classical result by Rényi [Rényi, 1959] states that two random variable $\mathbf{x}$ and $\mathbf{y}$ are independent if and only if all transformed variable $f(x)$ and $g(y)$ are uncorrelated, where f and g range over all bounded continuous function $f : \mathcal{X} \to \mathbb{R}$ and $g : \mathcal{Y} \to \mathbb{R}$.

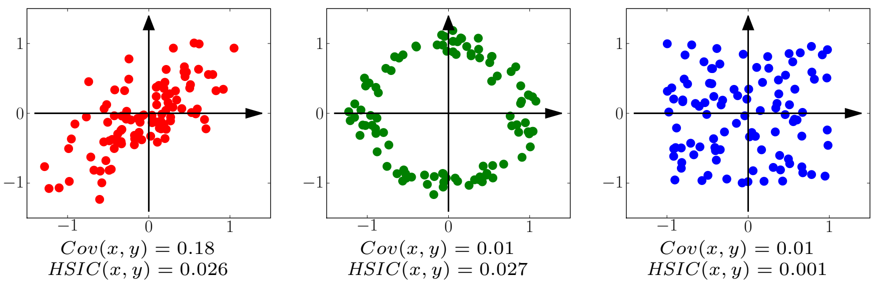


Figure 9.1 Covariance vs. independence: The samples in the left plot have correlated, and therefore dependent, x and y coordinates, which is reflected by non-zero values of both the *covariance coefficient* $Cov(x, y)$ and by the empirical *Hilbert-Schmidt independence criterion* $HSIC(x, y)$. The coordinates of the samples in the center plot are dependent, because for each x, only certain y values occur. The *covariance coefficient* of these sample, however, is almost 0, whereas *HSIC* correctly picks up on the dependence and yields a as big as in the correlated case. The samples on the right have independent x and y coordinates. Both, *covariance* and *HSIC*, nearly vanish.

9.2.1 Hilbert-Schmidt Independence Criterion

Clearly, it is not possible to calculate $Cov(f(\mathbf{x}), g(\mathbf{y}))$ for all possible functions f and g. However, in the following we will see that by using kernels we can do so *implicitly*.

Definition 9.1. Given two *characteristic*[2] kernel functions $k_x : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $k_y : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, an estimate of the statistical dependency between two datasets X and Y is given by the *empirical Hilbert-Schmidt independence criterion (HSIC)* [Fukumizu et al., 2008]:

$$HSIC_{emp}(Z; k_x, k_y) := \frac{1}{(n-1)^2} \operatorname{trace} \bar{K}_x \bar{K}_y \tag{9.6}$$

where $\bar{K}_x$ and $\bar{K}_y$ are the centered versions of the kernel matrices $K_x = k_x(x_i, x_j)_{i,j=1,\dots,n}$, $K_y = k_y(y_i, y_j)_{i,j=1,\dots,n}$.

The relevance of the empirical HSIC score follows from the fact that $HSIC_{emp}(Z; k_x, k_y)$ is a consistent estimator of the Hilbert Schmidt

[2] The definition of *characteristic* kernels is rather technical, see [Fukumizu et al., 2008]. However, many kernels popular in computer vision are of this class, *e.g.* Gaussian and χ^2-kernels and all universal kernels over compact domains.

Independence Criterion $HSIC(p(\mathbf{x},\mathbf{y})\,;k_x,k_y)$ for the probability density $p(\mathbf{x},\mathbf{y})$ according to which the data sets X and Y were generated. $HSIC(p(\mathbf{x},\mathbf{y})\,;k_x,k_y)$ can be shown to vanish if and only if $\mathbf{x}$ and $\mathbf{y}$ are independent random variables [Fukumizu et al., 2008].

The derivation of HSIC and the proof of the statements are beyond the scope of this work, but its consequences will prove useful also for practical application in computer vision. In particular, *HSIC* provides us with a tool to compare two datasets in terms of the *randomness* of their pairing without need for a similarity measure that compares elements $x \in \mathcal{X}$ with elements $y \in \mathcal{Y}$.

9.3 Example: Image Segmentation

D. Anguelov, B. Taskar, and V. Chatalbashev: *"Discriminative Learning of Markov Random Fields for Segmentation of 3D Scan Data"*, CVPR 2005.

[Anguelov et al., 2005] used structured prediction to segmentation 3D point clouds. The task is formulated as a graph labeling problem, where each sample point of a 3D scan is a node in a graph, to which a region label has to be assigned, *e.g. ground, building, tree, and shrubbery* in the case of *terrain classification.* The authors apply the framework of *Maximum Margin Markov Networks (M*^{3}*N)* [Taskar et al., 2003], which can be seen as a structured SVM with special choice of the loss function. Training and MAP-inference are done by *min-cut with* α*-expansion* [Boykov and Kolmogorov, 2004]. By integrating neighborhood information into the structured model, the proposed method achieves a higher segmentation accuracy than a per-site SVM, or an SVM with a post-processing step of label smoothing.

[Szummer et al., 2008] later took a similar approach for 2D image segmentation. They used the pixel grid structure of an image to form a neighborhood graph and train a structured SVM for the resulting graph labeling problem, achieving better segmentation accuracy than with a probabilistically trained *conditional random fields* [Lafferty et al., 2001].

9.4 Example: Object Localization

M. B. Blaschko, C. H. Lampert: *Learning to Localize Objects by Structured Output Regression*, ECCV 2008.

[Blaschko and Lampert, 2008b] use structured prediction to predict the location of objects in an image, *i.e.* the input are single images and the output are bounding box coordinate tuples. Using a standard representations of image regions by visual word histograms, they introduce a joint kernel function called *restriction kernel* in order to compare images with bounding box annotation. Based a linear kernel for bag of visual word histograms, they make use of a branch-and-bound technique [Lampert et al., 2008] for S-SVM training and prediction. By experiments on standard datasets for object localization the authors show that the structured SVM yields more accurate localization than previous approaches that used the decision function of a binary SVM for sliding window localization.

9.5 Example: Stereo Reconstruction

Y. Li and D. P. Huttenlocher: *"Learning for Stereo Vision Using the Structured Support Vector Machine"*, CVPR 2008.

Stereo reconstruction, *i.e.* the problem of predicting a depth map from a stereo image pair can be seen as a structured prediction task. [Li and Huttenlocher, 2008] propose a corresponding approach based on a structured SVM with linear kernel. The output is a discrete depth map with a special value indicating *occlusions*. Because the graph structure chosen has loops and long-range connections, an exact solution of the training optimization problem is infeasible. Instead, the authors use *loopy belief propagation (LBP)* [Murphy et al., 1999] for an approximation. Experiments on a standard dataset for stereo reconstruction show that the proposed model performs comparable to the state of the art.

9.6 Example: Graph and Shape Matching

T. S. Caetano, L. Cheng, Q. V. Le, and A. J. Smola: *"Learning Graph Matching"*, ICCV 2007.

Shape matching is the task of find the geometric alignment between two shapes. In practice, shape are typically represented by landmark points, and shape matching becomes the problem of finding correspondences between the landmarks. [Caetano et al., 2007] propose a structure learning formulation based on the S-SVM for this task, where the two sets of landmark points act as input while the correspondence assignment between the sets is the output. This makes the prediction step (9.2) into a linear or quadratic assignment problem, and the structured learning framework determines optimal weighting parameter for this step. As the authors demonstrate in their experiments, the resulting method allows, *e.g.*, wide baseline stereo matching or the consistent tracking of body parts of people in video sequences.

An extension of the framework that allows the inclusion geometrical constraints for the shapes was later proposed in [McAuley et al., 2008].

9.7 Example: Hierarchy Learning

M. B. Blaschko, and A. Gretton: *"Learning Taxonomies by Dependence Maximization"*, NIPS 2008

The partitioning of a dataset into several clusters is more informative if a relation between the resulting clusters can be inferred, *e.g.* a hierarchy. [Blaschko and Gretton, 2008] propose a method to learn the clustering of a dataset and a hierarchical relation between the clusters in a single step. Given a dataset X with kernel matrix K_x, their method relies on treating the unknown cluster labels as set Y and learning a block structured kernel matrix K_y that causes the maximal empirical HSIC score (9.6) between the X and Y. By approximating K_y with a tree metric, a hierarchy between clusters can be inferred.

9.8 Example: Image Montages

N. Quadrianto, L. Song and A. J. Smola: *"Kernelized Sorting"*, NIPS 2008.

Arranging images in two- or three-dimensional layouts is a useful tool for image screening and visualization. [Quadrianto et al., 2008] propose *Kernelized Sorting*, a method that allows the optimal arrangement of image datasets in an arbitrary output pattern, *e.g.* on a regular grid, with respect to a kernel function. The method works by finding the matching between images and grid points that maximizes the empirical *HSIC* score between the image set and the grid points. Since this requires the solution of an NP-hard assignment problem, an approximate scheme is derived based on DC programming [Horst and Thoai, 1999].

10

Learning the Kernel

All kernel methods we have seen so far make use of a single fixed kernel function. However, because different kernel function induce different feature space embeddings and are therefore differently well suited for a given problem, it is natural to ask the question which is the *best kernel function*. Ultimately, this is task dependent, because the *quality* of a kernel is determined by how well the trained kernel method performs in the task at hand, *e.g.* in the case of a classifier by the accuracy on future data points. Because we cannot use this quantity directly at training time, many estimators for the generalization error have been developed and used for parameter selection, *e.g. cross-validation* and *bootstrapping*, that work by iterating between training and test procedures on different parts of the training set.

In some situation, however, it is not possible to exhaustively explore which kernels or parameters work best, and algorithmic proxies have been developed: for kernels that come from parameterized families, *e.g.* Gaussian kernels with adjustable bandwidth parameter, *kernel target alignment* provides a way to determine good parameters. Alternatively, if a finite set of possible kernels has been preselected, *e.g.* by a human expert, and we want to find a reasonable subset and weights to use for a linear combination of kernels, then *multiple kernel learning* provides an answer. The latter case thereby generalizes *feature selection*, which is the special case that each base kernel functions depend on only one of several available features.

For simplicity of the discussion, in the following we will restrict ourselves to classification problems.

10.1 Kernel Target Alignments

The idea of *kernel target alignments (KTA)* [Cristianini et al., 2001] is that the values of a good kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ should resembled the values of a (hypothetical) kernel $l : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that has access to the data labels, *i.e.* $l(x, x') = 1$ for any samples x, x' having the same class label, and $l(x, x') = -1$ otherwise. To measure how close k is to the optimal l, we introduce an kernel alignment score

$$A(Z; k, l) = \frac{\operatorname{trace}(KL)}{n\sqrt{\operatorname{trace} KK}} \tag{10.1}$$

where $Z = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ is a labeled training set, K is the kernel matrix of k on Z, and L is the kernel matrix induced by the labels[1].

To select one kernel function out of a set of alternatives, we choose the kernel function that maximizes $A(Z; k, l)$. Since this procedure does not require to train and evaluate a classifier, it is in particular faster than, *e.g.*, multiple cross-validation runs. A further advantage of the kernel alignment score is its differentiability with respect to the kernel function k. For kernels that depend smoothly on real-valued parameters, it it therefore possible to find locally optimal parameters combinations by gradient-ascent optimization.

10.2 Multiple Kernel Learning

As we saw in the *Kernels for Computer Vision* part, it is often easy to come up with many reasonable kernel functions, but it is *a priori*

[1] A comparison with Equation (9.6) shows that Equation (10.1) resembles a normalized HSIC measure of dependence between the samples x_i and the labels y_i, but without prior centering of the kernel matrices.

unclear which one is best suitable for the problem, or if there even is a single best one. In computer vision, this is often related to the choice of representation and features: depending on the application, or even on the specific class of a multi-class problem, color, texture, or edge orientation might be the most relevant cue. Most often, however, one finds that different aspects are important at the same time, and one would like to find a kernel function that reflects the aspects of several kernels at the same time.

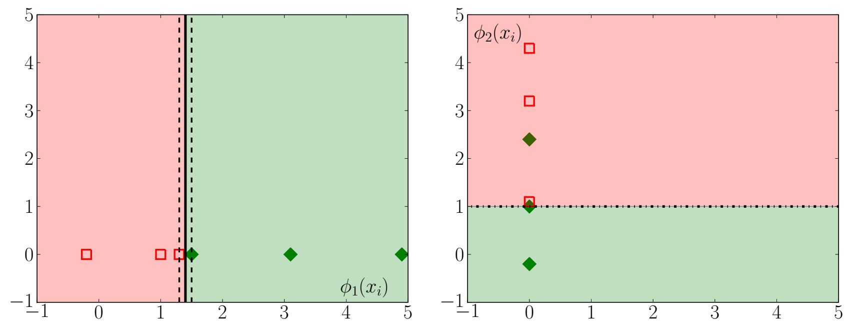
Kernel methods in general as well suitable for such feature combinations: as we saw in the Section *"Constructing Kernel Functions"*, the sum and product of existing kernels are kernels again, equally reflecting the properties of all base kernel. However, in situations, where we believe that some kernels are more important than others, we might prefer a *weighted linear combination* of kernel instead of just their unweighted sum. *Multiple kernel learning (MKL)* allows us to find the weights of such a linear combination[2]. It is based on the idea that each kernel or combination of kernels gives rise to a *margin* when used in the training of a support vector machine, and due to the linear kernel construction, we can find an explicit expression for the size of the margin. Because the concept of maximum margin learning tells us to prefer classifiers with a large margin between the classes, the MKL procedure jointly finds the SVM weight vector and the linear combination weights of the kernel functions that realized the generalized linear classifier of maximal *margin*, see Figure 10.1 for an illustration.

10.2.1 Linear Kernel Combinations

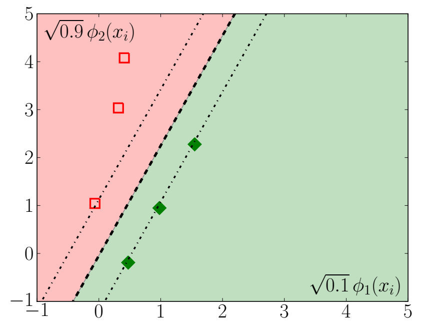
Let $k_1, \dots, k_m$ be kernel functions, $k_i : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, with induced Hilbert spaces $\mathcal{H}_i$ and feature maps $\varphi_i : \mathcal{X} \to \mathcal{H}_i$. We are interested in finding the best SVM classifier for any kernel

$$k(x, x') = \sum_{j=1}^{m} \beta_j k_j(x, x'). \tag{10.2}$$

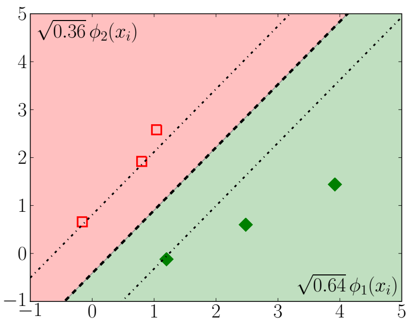
[2] *Kernel target alignment* has also been applied to the problem of learning linear kernel combinations, see *e.g.* [Lanckriet et al., 2004]. However multiple kernel learning has been found to yield better results for practical applications.



(a) In the representation $\varphi_1(x_i)$ (left) the classification problem is solvable only with very small margin. Using the representation $\varphi_2(x_i)$ (right) the data points are even non-separable.



(b) Represented by a combination of φ_1 and φ_2, the dataset becomes much better separable.



(c) A mixing coefficient of $\beta = 0.64$ leads to the maximal possible margin.

Figure 10.1 Example of multiple kernel learning. (a): given two kernels k_1 and k_2 with features maps φ_1 and φ_2, neither is well suited to solve the resulting classification problem. (b): a linear kernel combination $k(x,x') = \beta\, k_1(x,x') + (1-\beta)\, k_2(x,x')$ with induced feature map $(\sqrt{\beta}\varphi_1, \sqrt{1-\beta}\,\varphi_2)$ results in a better linear classifier. (c): solving the optimization (10.4) we can calculate the optimal mixing coefficients.

with $\beta_j \geq 0$.

For fixed coefficient $\beta_1, \dots, \beta_m$, we can think of k as inducing the product Hilbert space $\mathcal{H} = \mathcal{H}_1 \times \cdots \times \mathcal{H}_m$ by means of the stacked feature map $\varphi(x) = (\sqrt{\beta_1}\varphi_1(x), \dots, \sqrt{\beta_m}\varphi_m(x))$, because this construction induces the same scalar product as k:

$$\langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}} = \sum_{j=1}^{m} \beta_j \langle \varphi_j(x), \varphi_j(x') \rangle_{\mathcal{H}_j} = k(x,x'). \tag{10.3}$$

By decomposing and reparameterizing the weight vector as $w = (\frac{1}{\sqrt{\beta_1}}v_1, \dots, \frac{1}{\sqrt{\beta_m}}v_m)$ the SVM objective (2.33) becomes

$$\min_{\substack{(v_1,\dots,v_m)\in\mathcal{H}_1\times\dots\times H_m\\ \xi_1,\dots,\xi_n\in\mathbb{R}^+}} \sum_{j=1}^{m} \frac{1}{\beta_j}\|v_j\|_{\mathcal{H}_j^2} + \frac{C}{n}\sum_{i=1}^{n}\xi_i \tag{10.4}$$

subject to

$$y_i \sum_{j=1}^{m} \langle v_j, \varphi(x_i)\rangle_{\mathcal{H}_j} \geq 1 - \xi_i, \qquad \text{for } i = 1, \dots n. \tag{10.5}$$

In order to find the best coefficients for the linear combination kernel, we optimize (10.4) over all kernel choices, *i.e.* over $(\beta_1, \dots, \beta_m)$. However, changing β influences not only the margin, but by means of the feature map φ also the distribution of points in the feature space. Specifically, multiplying k with a constant will increase the margin, but it will also scale the point coordinates such that, geometrically, the situation is unaffected. To avoid this degenerate behavior, we bound the norm of β by either $\|\beta\|_{L^1} = 1$ (L^1-*MKL*) or $\|\beta\|_{L^2} = 1$ (L^2-*MKL* [Kloft et al., 2008])[3]. This optimization problem is *jointly-convex* in the vectors v_j and in the kernel coefficients β_j. Consequently, there is a unique global minimum and we can find it efficiently using standard optimization techniques. Note that, because in the case of an L^1 constraint, $\sum_{j=1}^{m}\beta_j = 1$, the coefficients β_j will typically become sparse, *i.e.* many β_j will be 0. This distinguishes relevant from irrelevant kernels and therefore allows the interpretation of multiple kernel learning as a feature selection technique.

A recent extension, *infinite kernel learning* [Gehler and Nowozin, 2009b] combines the advantages of *kernel target alignment* and *multiple kernel learning*: it allows the learning of a linear kernel combination while at the same time adjusting the kernel parameters.

[3] Interestingly, the third natural choice, $\|\beta\|_{L^\infty} = 1$, would results in the data-independent solution $\beta_1, \dots, \beta_m = 1$, *i.e.* summing up all base kernels.

10.3 Example: Image Classification

Y.-Y. Lin, T.-L. Liu, and C.-S. Fuh: *"Local Ensemble Kernel Learning for Object Category Recognition"*, CVPR 2007

The authors propose two methods to learn the best features and resulting kernel function for a given image classification task. Their *global ensemble kernel* is based on *kernel target alignment* to determine the coefficients $\beta_1, \dots, \beta_K$ of a convex kernel combination $k(x,x') = \sum_j \beta_j k_j(x,x')$ with $\beta_j \geq 0$ and $\sum_j \beta_j = 1$, as proposed by [Lanckriet et al., 2004; Bach et al., 2004]. Subsequently, they generalize the concept by introducing *local ensemble kernels* that learn individual mixing coefficients for each data point based on the local sample neighborhood. By experiments on a mixture of several standard datasets the authors show that the resulting method is able to handle very diverse classification tasks, such as texture classification and natural object recognition, in a single learned classifier.

G. Bakır, M. Wu and J. Eichhorn: *"Maximum-Margin Feature Combination for Detection and Categorization"*, Technical Report 2005

Bakır et al. [Bakır et al., 2005] first recognized the potential of multiple kernel learning to find the best features for image classification tasks. They define base kernels by splitting the image into a grid of cells and calculating χ^2-kernels for local color histograms, Bhattacharyya-kernels for SIFT features points, and Gaussian kernels for local edge maps. Subsequently they learn a combined kernel by one-versus-rest SVMs with multiple kernel learning. Experimental results for face recognition and object classification show that the combination of kernels can lead to higher classification performance than each individual kernel. Because one kernel combination is learned per class, the resulting coefficients can be interpreted as the importance of the individual feature types for the detection of the specific class.

Similar setups were later also proposed independently by Kumar and Sminchisescu [Kumar and Sminchisescu, 2007], who use MKL for object classification with different *pyramid match kernels* and *spatial* *pyramid kernels*, and by Varma and Ray [Varma and Ray, 2007], who develop a variant of MKL with different regularization, achieving good classification results in object and texture classification tasks.

Note, however, that all of these works compare only against the use of single kernels as baselines, but not against simpler methods of kernels combination. Recent evidence in [Gehler and Nowozin, 2009a] indicates that in image classification tasks MKL is only superior to kernel averaging if some base kernels are completely uninformative. If all base kernels contain meaningful information, simpler method for kernel combination work as well as MKL.

10.4 Example: Multi-Class Object Detection

C. H. Lampert and M. B. Blaschko: *“A Multiple Kernel Learning Approach to Joint Multi-Class Object Detection”*, DAGM 2008

In object classification in natural images, it is known that the background and the presence of other objects in the image can be strong cues for the presence of objects. [Lampert and Blaschko, 2008] propose a method to make use of such context in a one-vs-rest multiclass setup. They define kernels for each object category to be detected and learn a linear combination of these base kernels for each target class by L^1-MKL. The authors show that the kernel combinations that include other object regions as context perform better than kernels based on a single object region, and that the coefficients learned by MKL reflect inherent dependencies between object that for implicit subcategories, *e.g. animals* or *vehicles*.

Bibliography

Anguelov, D., B. Taskar, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng (2005), 'Discriminative learning of markov random fields for segmentation of 3D scan data'. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 169–176.

Bach, F. R., G. R. G. Lanckriet, and M. I. Jordan (2004), 'Multiple kernel learning, conic duality, and the SMO algorithm'. In: *International Conference on Machine Learning (ICML)*.

Bakır, G. H., M. Wu, and J. Eichhorn (2005), 'Maximum-margin feature combination for detection and categorization'. Technical report, Max Planck Institute for Biological Cybernetics, Tübingen, Germany.

Banerjee, A., P. Burlina, and C. Diehl (2006), 'A support vector method for anomaly detection in hyperspectral imagery'. *IEEE Transactions on Geoscience and Remote Sensing* **44**(8), 2282–2291.

Bay, H., T. Tuytelaars, and L. Van Gool (2006), 'SURF: Speeded up robust features'. In: *European Conference on Computer Vision (ECCV)*. pp. 404–417.

Belkin, M. and P. Niyogi (2003), 'Laplacian eigenmaps for dimensionality reduction and data representation'. *Neural computation* **15**(6), 1373–1396.

Ben-Hur, A., D. Horn, H. T. Siegelmann, and V. Vapnik (2002), 'Support vector clustering'. *Journal of Machine Learning Research* **2**, 125–137.

Bishop, C. M. (1995), *Neural networks for pattern recognition.* Oxford University Press, USA.

Blaschko, M. B. and A. Gretton (2008), 'Learning taxonomies by dependence maximization'. In: *Advances in Neural Information Processing Systems (NIPS)*, Vol. 22.

Blaschko, M. B. and C. H. Lampert (2008a), 'Correlational spectral clustering'. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR).*

Blaschko, M. B. and C. H. Lampert (2008b), 'Learning to localize objects with structured output regression'. In: *European Conference on Computer Vision (ECCV).*

Bosch, A., A. Zisserman, and X. Munoz (2007), 'Representing shape with a spatial pyramid kernel'. In: *International Conference on Image and Video Retrieval (CIVR).* pp. 401–408.

Boutell, M. and J. Luo (2005), 'Beyond pixels: exploiting camera metadata for photo classification'. *Pattern recognition* **38**(6), 935–946.

Boyd, S. P. and L. Vandenberghe (2004), *Convex optimization.* Cambridge University Press.

Boykov, Y. and V. Kolmogorov (2004), 'An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision'. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **26**(9), 1124–1137.

Breiman, L. (1998), *Classification and regression trees.* Chapman & Hall/CRC.

Caetano, T. S., L. Cheng, Q. V. Le, and A. J. Smola (2007), 'Learning graph matching'. In: *IEEE International Conference on Computer Vision (ICCV).*

Chapelle, O. (2007), 'Training a support vector machine in the primal'. *Neural Computation* **19**(5), 1155–1178.

Chen, Y., X. S. Zhou, and T. S. Huang (2001), 'One-class SVM for learning in image retrieval'. In: *IEEE International Conference on Image Processing (ICIP).* pp. 34–37.

Crammer, K. and Y. Singer (2001), 'On the algorithmic implementation of multiclass kernel-based vector machines'. *Journal of Machine Learning Research* **2**, 265–292.

Cristianini, N., J. Shawe-Taylor, A. Elisseeff, and J. Kandola (2001), 'On kernel-target alignment'. In: *Advances in Neural Information Processing Systems (NIPS)*, Vol. 15. pp. 367–373.

Dambreville, S., Y. Rathi, and A. Tannenbaum (2006), 'Shape-based approach to robust image segmentation using kernel PCA'. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 977–984.

Decoste, D. and B. Schölkopf (2002), 'Training invariant support vector machines'. *Machine Learning* **46**(1), 161–190.

Deerwester, S., S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman (1990), 'Indexing by latent semantic analysis'. *Journal of the American Society for Information Science* **41**(6), 391–407.

Dietterich, T. G. and G. Bakiri (1995), 'Solving multiclass learning problems via error-correcting output codes'. *Journal of Artificial Intelligence Research* **2**, 263–286.

Dollar, P., V. Rabaud, G. Cottrell, and S. Belongie (2005), 'Behavior recognition via sparse spatio-temporal features'. In: *IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance (VS-PETS)*. pp. 65–72.

Drucker, H., C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik (1996), 'Support vector regression machines'. In: *Advances in Neural Information Processing Systems (NIPS)*, Vol. 10. pp. 155–161.

Efron, B. and G. Gong (1983), 'A leisurely look at the bootstrap, the jackknife, and cross-validation'. *American Statistician* **37**(1), 36–48.

Forman, G. (2003), 'An extensive empirical study of feature selection metrics for text classification'. *Journal of Machine Learning Research* **3**, 1289–1305.

Fukumizu, K., A. Gretton, X. Sun, and B. Schölkopf (2008), 'Kernel measures of conditional dependence'. In: *Advances in Neural Information Processing Systems (NIPS)*, Vol. 22. pp. 489–496.

Gehler, P. and S. Nowozin (2009a), 'Feature Combination Methods for Kernel Classifiers'. In: *(currently under review)*.

Gehler, P. and S. Nowozin (2009b), 'Let the Kernel Figure it Out;

Principled Learning of Pre-processing for Kernel Classifiers'. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.

Girolami, M. (2002), 'Mercer kernel-based clustering in feature space'. *IEEE Transactions on Neural Networks* **13**(3), 780–784.

Grauman, K. and T. Darrell (2005), 'The pyramid match kernel: discriminative classification with sets of image features'. In: *IEEE International Conference on Computer Vision (ICCV)*. pp. 1458–1465.

Gusfield, D. (1997), *Algorithms on strings, trees, and sequences*. Cambridge University Press, New York.

Haasdonk, B. and H. Burkhardt (2007), 'Invariant kernel functions for pattern analysis and machine learning'. *Machine Learning* **68**(1), 35–61.

Hardoon, D., C. Saunders, S. Szedmak, and J. Shawe-Taylor (2006), 'A correlation approach for automatic image annotation'. In: *International Conference on Advanced Data Mining and Applications (ADMA)*. pp. 681–692.

Hardoon, D. R., J. Mourão-Miranda, M. Brammer, and J. Shawe-Taylor (2007), 'Unsupervised analysis of fMRI data using kernel canonical correlation'. *Neuroimage* **37**(4), 1250–1259.

Harris, C. and M. Stephens (1988), 'A combined corner and edge detector'. In: *Alvey Vision Conference*. pp. 147–151.

Hein, M. and O. Bousquet (2005), 'Hilbertian metrics and positive definite kernels on probability measures'. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Hiemstra, D. (2000), 'A probabilistic justification for using tf×idf term weighting in information retrieval'. *International Journal on Digital Libraries* **3**(2), 131–139.

Hinton, G. E., S. Osindero, and Y. W. Teh (2006), 'A fast learning algorithm for deep belief nets'. *Neural Computation* **18**(7), 1527–1554.

Horst, R. and N. V. Thoai (1999), 'DC programming: overview'. *Journal of Optimization Theory and Applications* **103**(1), 1–43.

Hsu, C. and C. Lin (2002), 'A comparison of methods for multiclass support vector machines'. *IEEE Transactions on Neural Networks* **13**(2), 415–425.

Jiang, Y. G., C. W. Ngo, and J. Yang (2007), 'Towards optimal bag-of-features for object categorization and semantic video retrieval'. In: *International Conference on Image and Video Retrieval (CIVR)*. pp. 494–501.

Joachims, T. (2006), 'Training linear SVMs in linear time'. In: *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. pp. 217–226.

Jolliffe, I. T. (1986), 'Principal component analysis'. *Springer, New York*.

Kim, K. and Y. Kwon (2008), 'Example-Based Learning for Single-Image Super-Resolution'. In: *Symposium of the German Pattern Recognition Society (DAGM)*. pp. 456–465.

Kim, K. I., M. O. Franz, and B. Schölkopf (2005), 'Iterative kernel principal component analysis for image modeling'. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **27**(9), 1351–1366.

Kloft, M., U. Brefeld, P. Laskov, and S. Sonnenburg (2008), 'Non-sparse multiple kernel learning'. In: *NIPS Workshop on Kernel Learning: Automatic Selection of Optimal Kernels*.

Kohavi, R. (1995), 'A study of cross-validation and bootstrap for accuracy estimation and model selection'. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. pp. 1137–1145.

Kumar, A. and C. Sminchisescu (2007), 'Support kernel machines for object recognition'. In: *IEEE International Conference on Computer Vision (ICCV)*.

Kwok, J. T. Y. and I. W. H. Tsang (2004), 'The pre-image problem in kernel methods'. *IEEE Transactions on Neural Networks* **15**(6), 1517–1525.

Lafferty, J. D., A. McCallum, and F. C. N. Pereira (2001), 'Conditional random fields: probabilistic models for segmenting and labeling sequence data'. In: *International Conference on Machine Learning (ICML)*. pp. 282–289.

Lai, P. L. and C. Fyfe (2000), 'Kernel and nonlinear canonical correlation analysis'. *Internation Journal of Neural Systems* **10**(5), 365–378.

Lampert, C. H. and M. B. Blaschko (2008), 'A multiple kernel learning approach to joint multi-class object detection'. In: *Symposium of the German Pattern Recognition Society (DAGM)*. pp. 31–40.

Lampert, C. H., M. B. Blaschko, and T. Hofmann (2008), 'Beyond sliding windows: object localization by efficient subwindow search'. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR).*

Lanckriet, G. R. G., N. Cristianini, P. Bartlett, L. El Ghaoui, and M. I. Jordan (2004), 'Learning the kernel matrix with semidefinite programming'. *Journal of Machine Learning Research* **5**, 27–72.

Lazebnik, S., C. Schmid, and J. Ponce (2006), 'Beyond bags of features: spatial pyramid matching for recognizing natural scene categories'. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR).* pp. 2169–2178.

Li, D., R. M. Mersereau, and S. J. Simske (2007), 'Blind image deconvolution through support vector regression'. *IEEE Transactions on Neural Networks* **18**(3), 931–935.

Li, Y., S. Gong, J. Sherrah, and H. Liddell (2004), 'Support vector machine based multi-view face detection and recognition'. *Image and Vision Computing* **22**(5), 413–427.

Li, Y. and D. P. Huttenlocher (2008), 'Learning for stereo vision using the structured support vector machine'. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR).*

Linden, A. and J. Kindermann (1989), 'Inversion of multilayer nets'. In: *International Joint Conference on Neural Networks (IJCNN).* pp. 425–430.

Loupias, E., N. Sebe, S. Bres, and J. M. Jolion (2000), 'Wavelet-based salient points for image retrieval'. In: *IEEE International Conference on Image Processing (ICIP).* pp. 518–521.

Lowe, D. G. (2004), 'Distinctive image features from scale-invariant keypoints'. *International Journal of Computer Vision* **60**(2), 91–110.

Lyu, S. and H. Farid (2004), 'Steganalysis using color wavelet statistics and one-class support vector machines'. In: *SPIE Symposium on Electronic Imaging.* pp. 35–45.

MacQueen, J. (1967), 'Some Methods for Classification and Analysis of Multivariate Observations'. In: *Fifth Berkeley Symposium on Mathematics, Statistics and Probability*, Vol. 1. pp. 281–296.

McAuley, J., T. Caetano, and A. J. Smola (2008), 'Robust near-isometric matching via structured learning of graphical models'. In:

Advances in Neural Information Processing Systems (NIPS), Vol. 22.

McLachlan, G. J. (1992), *Discriminant analysis and statistical pattern recognition*. Wiley New York.

Mika, S., G. Rätsch, J. Weston, B. Schölkopf, and K. R. Müller (1999), 'Fisher discriminant analysis with kernels'. In: *IEEE Workshop on Neural Networks for Signal Processing*. pp. 41–48.

Mikolajczyk, K. and C. Schmid (2004), 'Scale and affine invariant interest point detectors'. *International Journal of Computer Vision* **60**(1), 63–86.

Murphy, K., Y. Weiss, and M. I. Jordan (1999), 'Loopy belief propagation for approximate inference: an empirical study'. In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. pp. 467–475.

Nene, S. A., S. K. Nayar, and H. Murase (1996), 'Columbia object image library (COIL-100)'. Technical report, Department of Computer Science Columbia University, New York.

Ni, K. S. and T. Q. Nguyen (2007), 'Image superresolution using support vector regression'. *IEEE Transactions on Image Processing* **16**(6), 1596–1610.

Nowak, E., F. Jurie, and B. Triggs (2006), 'Sampling strategies for bag-of-features image classification'. In: *European Conference on Computer Vision (ECCV)*. pp. 490–503.

Nowozin, S., G. Bakır, and K. Tsuda (2007), 'Discriminative subsequence mining for action classification'. In: *IEEE International Conference on Computer Vision (ICCV)*.

Platt, J. C. (1999), 'Fast training of support vector machines using sequential minimal optimization'. In: *Advances in Kernel Methods: Support Vector Learning*. MIT Press, pp. 185–208.

Platt, J. C., N. Cristianini, and J. Shawe-Taylor (1999), 'Large margin DAGs for multiclass classification'. In: *Advances in Neural Information Processing Systems (NIPS)*, Vol. 13. pp. 547–553.

Quadrianto, N., L. Song, and A. J. Smola (2008), 'Kernelized sorting'. In: *Advances in Neural Information Processing Systems (NIPS)*, Vol. 22.

Quinlan, J. R. (1986), 'Induction of decision trees'. *Machine Learning* **1**(1), 81–106.

Rasmussen, C. E. and C. Williams (2006), *Gaussian processes for machine learning*. MIT Press.

Rényi, A. (1959), 'On measures of dependence'. *Acta Mathematica Hungarica* **10**(3), 441–451.

Rifkin, R. and A. Klautau (2004), 'In defense of one-vs-all classification'. *Journal of Machine Learning Research* **5**, 101–141.

Saunders, C., A. Gammerman, and V. Vovk (1998), 'Ridge regression learning algorithm in dual variables'. In: *International Conference on Machine Learning (ICML)*. pp. 515–521.

Schapire, R. and Y. Freund (1997), 'A decision theoretic generalization of on-line learning and an application to boosting'. *Journal of Computer and System Sciences* **55**(1), 119–139.

Schölkopf, B., C. Burges, and V. Vapnik (1995), 'Extracting support data for a given task'. In: *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. pp. 252–257.

Schölkopf, B., J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson (2001), 'Estimating the support of a high-dimensional distribution'. *Neural Computation* **13**(7), 1443–1471.

Schölkopf, B. and A. J. Smola (2002), *Learning with Kernels*. MIT Press.

Shalev-Shwartz, S., Y. Singer, and N. Srebro (2007), 'Pegasos: Primal Estimated sub-GrAdient SOlver for SVM'. In: *International Conference on Machine Learning (ICML)*. pp. 807–814.

Shawe-Taylor, J. and N. Cristianini (2004), *Kernel methods for pattern analysis*. Cambridge University Press.

Sietsma, J. and R. J. F. Dow (1991), 'Creating artificial neural networks that generalize'. *Neural Networks* **4**(1), 67–79.

Simard, P., Y. LeCun, J. S. Denker, and B. Victorri (1999), 'Transformation invariance in pattern recognition – tangent distance and tangent propagation'. In: *Neural Networks: Tricks of the Trade*. Springer, pp. 239–274.

Smith, J. R. and S. F. Chang (1996), 'Tools and techniques for color image retrieval'. In: *Storage and Retrieval for Image and Video Databases (SPIE)*. pp. 426–437.

Swain, M. J. and D. H. Ballard (1991), 'Color indexing'. *International Journal of Computer Vision* **7**(1), 11–32.

Szummer, M., P. Kohli, and D. Hoiem (2008), 'Learning CRFs using graph cuts'. In: *European Conference on Computer Vision (ECCV)*. pp. 582–595.

Taskar, B., C. Guestrin, and D. Koller (2003), 'Max-margin markov networks'. In: *Advances in Neural Information Processing Systems (NIPS)*, Vol. 17.

Tax, D. M. J. and R. P. W. Duin (2004), 'Support vector data description'. *Machine Learning* **54**(1), 45–66.

Tipping, M. and B. Schölkopf (2001), 'A kernel approach for vector quantization with guaranteed distortion bounds'. In: *Artificial Intelligence and Statistics (AISTATS)*. pp. 129–134.

Tsochantaridis, I., T. Joachims, T. Hofmann, and Y. Altun (2005), 'Large margin methods for structured and interdependent output variables'. *Journal of Machine Learning Research* **6**, 1453–1484.

Tuytelaars, T., C. Lampert, M. Blaschko, and W. Buntine (2009), 'Unsupervised object discovery: a comparison'. *under review*.

Tuytelaars, T. and K. Mikolajczyk (2008), *Local invariant feature detectors: A survey*. Now Publishers.

Vapnik, V. N. (1998), *Statistical learning theory*. Wiley.

Vapnik, V. N. (2000), *The nature of statistical learning theory*. Springer.

Varma, M. and D. Ray (2007), 'Learning the discriminative power-invariance trade-off'. In: *IEEE International Conference on Computer Vision (ICCV)*.

Wang, D., L. Shi, D. S. Yeung, P.-A. Heng, T.-T. Wong, and E. C. C. Tsang (2005), 'Support vector clustering for brain activation detection'. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. pp. 572–579.

Wang, L. and D. Suter (2007), 'Recognizing human activities from silhouettes: motion subspace and factorial discriminative graphical model'. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.

Weston, J., A. Elisseeff, B. Schölkopf, and M. Tipping (2003), 'Use of the zero norm with linear models and kernel methods'. *Journal of Machine Learning Research* **3**, 1439–1461.

Yang, J., Y. G. Jiang, A. G. Hauptmann, and C. W. Ngo (2007), 'Evaluating bag-of-visual-words representations in scene classification'. In: *International Workshop on Multimedia Information Retrieval (MIR)*. pp. 197–206.

Young, R. A. (1987), 'The Gaussian derivative model for spatial vision: I. Retinal mechanisms'. *Spatial Vision* **2**(4), 273–293.

Zhang, J., M. Marszalek, S. Lazebnik, and C. Schmid (2007), 'Local features and kernels for classification of texture and object categories: A comprehensive study'. *International Journal of Computer Vision* **73**(2), 213–238.