Chapter 14
Kernels

14.1 Introduction

So far in this book, we have been assuming that each object that we wish to classify or cluster or process in anyway can be represented as a fixed-size feature vector, typically of the form $\boldsymbol{x}_i \in \mathbb{R}^D$. However, for certain kinds of objects, it is not clear how to best represent them as fixed-sized feature vectors. For example, how do we represent a text document or protein sequence, which can be of variable length? or a molecular structure, which has complex 3d geometry? or an evolutionary tree, which has variable size and shape?

One approach to such problems is to define a generative model for the data, and use the inferred latent representation and/or the parameters of the model as features, and then to plug these features in to standard methods. For example, in Chapter 28 TODO, we discuss deep learning, which is essentially an unsupervised way to learn good feature representations.

Another approach is to assume that we have some way of measuring the similarity between objects, that doesnt require preprocessing them into feature vector format. For example, when comparing strings, we can compute the edit distance between them. Let $\kappa(\boldsymbol{x},\boldsymbol{x}') \geq 0$ be some measure of similarity between objects $\kappa(\boldsymbol{x},\boldsymbol{x}') \in \mathcal{X}$, we will call κ a **kernel function**. Note that the word kernel has several meanings; we will discuss a different interpretation in Section 14.7.1 TODO.

In this chapter, we will discuss several kinds of kernel functions. We then describe some algorithms that can be written purely in terms of kernel function computations. Such methods can be used when we dont have access to (or choose not to look at) the inside of the objects $\boldsymbol{x}$ that we are processing.

14.2 Kernel functions

Definition 14.1. A **kernel function**[23] is a real-valued function of two arguments, $\kappa(\boldsymbol{x},\boldsymbol{x}') \in \mathbb{R}$. Typically the function is symmetric (i.e., $\kappa(\boldsymbol{x},\boldsymbol{x}') = \kappa(\boldsymbol{x}',\boldsymbol{x})$, and non-negative (i.e., $\kappa(\boldsymbol{x},\boldsymbol{x}') \geq 0$).

[23] http://en.wikipedia.org/wiki/Kernel_function

We give several examples below.

14.2.1 RBF kernels

The **Gaussian kernel** or **squared exponential kernel**(SE kernel) is defined by

$$\kappa(\boldsymbol{x},\boldsymbol{x}') = \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}')^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{x}')\right) \quad (14.1)$$

If $\boldsymbol{\Sigma}$ is diagonal, this can be written as

$$\kappa(\boldsymbol{x},\boldsymbol{x}') = \exp\left(-\frac{1}{2}\sum_{j=1}^{D}\frac{1}{\sigma_j^2}(x_j - x_j')^2\right) \quad (14.2)$$

We can interpret the σ_j as defining the **characteristic length scale** of dimension j.If $\sigma_j = \infty$, the corresponding dimension is ignored; hence this is known as the **ARD kernel**. If $\boldsymbol{\Sigma}$ is spherical, we get the isotropic kernel

$$\kappa(\boldsymbol{x},\boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2}\right) \quad (14.3)$$

Here σ^2 is known as the **bandwidth**. Equation 14.4 is an example of a **radial basis function** or **RBF** kernel, since it is only a function of $\|\boldsymbol{x}-\boldsymbol{x}'\|^2$.

14.2.2 TF-IDF kernels

$$\kappa(\boldsymbol{x},\boldsymbol{x}') = \frac{\phi(\boldsymbol{x})^T\phi(\boldsymbol{x}')}{\|\phi(vecx)\|_2\|\phi(\boldsymbol{x}')\|_2} \quad (14.4)$$

where $\phi(\boldsymbol{x}) = \text{tf-idf}(\boldsymbol{x})$.

14.2.3 Mercer (positive definite) kernels

If the kernel function satisfies the requirement that the **Gram matrix**, defined by

$$K \triangleq \begin{pmatrix} \kappa(x_1,x_2) & \cdots \kappa(x_1,x_N) \\ \vdots & \vdots & \vdots \\ \kappa(x_N,x_1) & \cdots \kappa(x_N,x_N) \end{pmatrix} \quad (14.5)$$

be positive definite for any set of inputs $\{x_i\}_{i=1}^N$. We call such a kernel a **Mercer kernel**,or **positive definite kernel**.

If the Gram matrix is positive definite, we can compute an eigenvector decomposition of it as follows

$$K = U^T \Lambda U \quad (14.6)$$

where Λ is a diagonal matrix of eigenvalues $\lambda_i > 0$. Now consider an element of K:

$$k_{ij} = (\Lambda^{\frac{1}{2}} U_{:,i})^T (\Lambda^{\frac{1}{2}} U_{:,j}) \quad (14.7)$$

Let us define $\phi(x_i) = \Lambda^{\frac{1}{2}} U_{:,i}$, then we can write

$$k_{ij} = \phi(x_i)^T \phi(x_j) \quad (14.8)$$

Thus we see that the entries in the kernel matrix can be computed by performing an inner product of some feature vectors that are implicitly defined by the eigenvectors U. In general, if the kernel is Mercer, then there exists a function ϕ mapping $x \in \mathcal{X}$ to $\mathbb{R}^D$ such that

$$\kappa(x,x') = \phi(x)^T \phi(x') \quad (14.9)$$

where ϕ depends on the eigen functions of κ(so D is a potentially infinite dimensional space).

For example, consider the (non-stationary) **polynomial kernel** $\kappa(x,x') = (\gamma x x' + r)^M$, where $r > 0$. One can show that the corresponding feature vector $\phi(x)$ will contain all terms up to degree M. For example, if $M = 2, \gamma = r = 1$ and $x, x' \in \mathbb{R}^2$, we have

$$\begin{aligned}(xx'+1)^2 &= (1 + x_1 x_1' + x_2 + x_2')^2 \\ &= 1 + 2x_1x_1' + 2x_2x_2' + (x_1x_1')^2 + (x_2x_2')^2 x_1x_1'x_2x_2' \\ &= \phi(x)^T \phi(x') \end{aligned}$$

where $\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$

In the case of a Gaussian kernel, the feature map lives in an infinite dimensional space. In such a case, it is clearly infeasible to explicitly represent the feature vectors.

In general, establishing that a kernel is a Mercer kernel is difficult, and requires techniques from functional analysis. However, one can show that it is possible to build up new Mercer kernels from simpler ones using a set of standard rules. For example, if κ_1 and κ_2 are both Mercer, so is $\kappa(x,x') = \kappa_1(x,x') + \kappa_2(x,x') =$. See e.g., (Schoelkopf and Smola 2002) for details.

14.2.4 Linear kernels

$$\kappa(x,x') = x^T x' \quad (14.10)$$

14.2.5 Matern kernels

The **Matern kernel**, which is commonly used in Gaussian process regression (see Section 15.2), has the following form

$$\kappa(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}r}{\ell}\right)^\nu K_\nu \frac{\sqrt{2\nu}r}{\ell} \quad (14.11)$$

where $r = \|x - x'\|$, $\nu > 0$, $\ell > 0$, and K_ν is a modified Bessel function. As $\nu \to \infty$, this approaches the SE kernel. If $\nu = \frac{1}{2}$, the kernel simplifies to

$$\kappa(r) = \exp(-r/\ell) \quad (14.12)$$

If $D = 1$, and we use this kernel to define a Gaussian process (see Chapter 15 TODO), we get the **Ornstein-Uhlenbeck process**, which describes the velocity of a particle undergoing Brownian motion (the corresponding function is continuous but not differentiable, and hence is very jagged).

14.2.6 String kernels

Now let $\phi_s(x)$ denote the number of times that substrings appears in string x. We define the kernel between two strings x and x' as

$$\kappa(x,x') = \sum_{s \in \mathcal{A}^*} w_s \phi_s(x) \phi_s(x') \quad (14.13)$$

where $w_s \geq 0$ and $\mathcal{A}^*$ is the set of all strings (of any length) from the alphabet $\mathcal{A}$(this is known as the Kleene star operator). This is a Mercer kernel, and be computed in $O(|x| + |x'|)$ time (for certain settings of the weights $\{ws\}$) using suffix trees (Leslie et al. 2003; Vishwanathan and Smola 2003; Shawe-Taylor and Cristianini 2004).

There are various cases of interest. If we set $w_s = 0$for $|s| > 1$ we get a bag-of-characters kernel. This defines $\phi(x)$ to be the number of times each character in $\mathcal{A}$ occurs in x.If we require s to be bordered by white-space, we get a bag-of-words kernel, where $\phi(x)$ counts how many times each possible word occurs. Note that this is a very sparse vector, since most words will not be present. If we only consider strings of a fixed lengthk, we get the **k-spectrum** kernel. This has been used to classify proteins into SCOP superfamilies (Leslie et al. 2003).

14.2.7 Pyramid match kernels

14.2.8 Kernels derived from probabilistic generative models

Suppose we have a probabilistic generative model of feature vectors, $p(\boldsymbol{x}|\boldsymbol{\theta})$. Then there are several ways we can use this model to define kernel functions, and thereby make the model suitable for discriminative tasks. We sketch two approaches below.

14.2.8.1 Probability product kernels

$$\kappa(\boldsymbol{x}_i,\boldsymbol{x}_j)=\int p(\boldsymbol{x}|\boldsymbol{x}_i)^{\rho}p(\boldsymbol{x}|\boldsymbol{x}_j)^{\rho}\mathrm{d}\boldsymbol{x} \tag{14.14}$$

where $\rho>0$, and $p(\boldsymbol{x}|\boldsymbol{x}_i)$ is often approximated by $p(\boldsymbol{x}|\hat{\boldsymbol{\theta}}(\boldsymbol{x}_i))$,where $\hat{\boldsymbol{\theta}}(\boldsymbol{x}_i)$ is a parameter estimate computed using a single data vector. This is called a **probability product kernel**(Jebara et al. 2004).

Although it seems strange to fit a model to a single data point, it is important to bear in mind that the fitted model is only being used to see how similar two objects are. In particular, if we fit the model to $\boldsymbol{x}_i$ and then the model thinks $\boldsymbol{x}_j$ is likely, this means that $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are similar. For example, suppose $p(\boldsymbol{x}|\boldsymbol{\theta})\sim\mathcal{N}(\boldsymbol{\mu},\sigma^2\boldsymbol{I})$, where σ^2 is fixed. If $\rho=1$, and we use $\hat{\boldsymbol{\mu}}(\boldsymbol{x}_i)=\boldsymbol{x}_i$ and $\hat{\boldsymbol{\mu}}(\boldsymbol{x}_j)=\boldsymbol{x}_j$, we find (Jebara et al. 2004, p825) that

$$\kappa(\boldsymbol{x}_i,\boldsymbol{x}_j)=\frac{1}{(4\pi\sigma^2)^{D/2}}\exp\left(-\frac{1}{4\sigma^2}\|\boldsymbol{x}_i-\boldsymbol{x}_j\|^2\right) \tag{14.15}$$

which is (up to a constant factor) the RBF kernel.

It turns out that one can compute Equation 14.14 for a variety of generative models, including ones with latent variables, such as HMMs. This provides one way to define kernels on variable length sequences. Furthermore, this technique works even if the sequences are of real-valued vectors, unlike the string kernel in Section 14.2.6. See (Jebara et al. 2004) for further details

14.2.8.2 Fisher kernels

A more efficient way to use generative models to define kernels is to use a **Fisher kernel** (Jaakkola and Haussler 1998) which is defined as follows:

$$\kappa(\boldsymbol{x}_i,\boldsymbol{x}_j)=g(\boldsymbol{x}_i)^T\boldsymbol{F}^{-1}g(\boldsymbol{x}_j) \tag{14.16}$$

where g is the gradient of the log likelihood, or **score vector**, evaluated at the MLE $\hat{\boldsymbol{\theta}}$

$$g(\boldsymbol{x})\triangleq\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\log p(\boldsymbol{x}|\boldsymbol{\theta})|_{\hat{\boldsymbol{\theta}}} \tag{14.17}$$

and $\boldsymbol{F}$ is the Fisher information matrix, which is essentially the Hessian:

$$\boldsymbol{F}\triangleq\left[\frac{\partial^2}{\partial\theta_i\partial\theta_j}\log p(\boldsymbol{x}|\boldsymbol{\theta})\right]|_{\hat{\boldsymbol{\theta}}} \tag{14.18}$$

Note that $\hat{\boldsymbol{\theta}}$ is a function of all the data, so the similarity of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is computed in the context of all the data as well. Also, note that we only have to fit one model.

14.3 Using kernels inside GLMs

14.3.1 Kernel machines

We define a **kernel machine** to be a GLM where the input feature vector has the form

$$\phi(\boldsymbol{x})=(\kappa(\boldsymbol{x},\boldsymbol{\mu}_1),\cdots,\kappa(\boldsymbol{x},\boldsymbol{\mu}_K)) \tag{14.19}$$

where $\boldsymbol{\mu}_k\in\mathcal{X}$ are a set of K centroids. If κ is an RBF kernel, this is called an **RBF network**. We discuss ways to choose the $\boldsymbol{\mu}_k$ parameters below. We will call Equation 14.19 a **kernelised feature vector**. Note that in this approach, the kernel need not be a Mercer kernel.

We can use the kernelized feature vector for logistic regression by defining $p(y|\boldsymbol{x},\boldsymbol{\theta})=\mathrm{Ber}(y|\boldsymbol{w}^T\phi(\boldsymbol{x}))$. This provides a simple way to define a non-linear decision boundary. For example, see Figure 14.1.

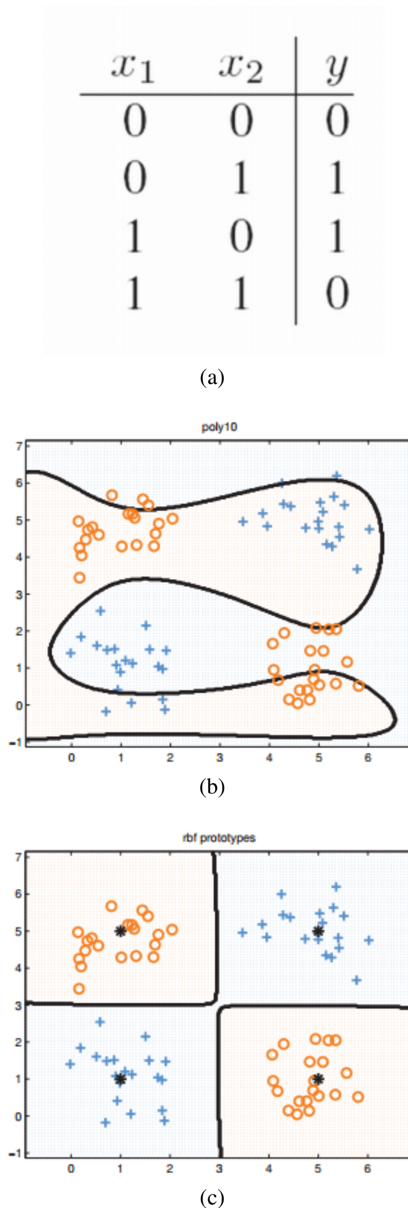
We can also use the kernelized feature vector inside a linear regression model by defining $p(y|\boldsymbol{x},\boldsymbol{\theta})=\mathcal{N}(y|\boldsymbol{w}^T\phi(\boldsymbol{x}),\sigma^2)$.

14.3.2 L1VMs, RVMs, and other sparse vector machines

The main issue with kernel machines is: how do we choose the centroids $\boldsymbol{\mu}_k$? We can use **sparse vector machine**, **L1VM**, **L2VM**, **RVM**, **SVM**.

14.4 The kernel trick

Rather than defining our feature vector in terms of kernels, $\phi(\boldsymbol{x})=(\kappa(\boldsymbol{x},\boldsymbol{x}_1),\cdots,\kappa(\boldsymbol{x},\boldsymbol{x}_N))$, we can instead work with the original feature vectors $\boldsymbol{x}$, but modify the algorithm so that it replaces all inner products of the form



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 14.1: (a) xor truth table. (b) Fitting a linear logistic regression classifier using degree 10 polynomial expansion. (c) Same model, but using an RBF kernel with centroids specified by the 4 black crosses.

$< \boldsymbol{x}_i, \boldsymbol{x}_j >$ with a call to the kernel function, $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$. This is called the **kernel trick**. It turns out that many algorithms can be kernelized in this way. We give some examples below. Note that we require that the kernel be a Mercer kernel for this trick to work.

14.4.1 Kernelized KNN

The Euclidean distance can be unrolled as

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| = < \boldsymbol{x}_i, \boldsymbol{x}_i > + < \boldsymbol{x}_j, \boldsymbol{x}_j > -2 < \boldsymbol{x}_i, \boldsymbol{x}_j > \tag{14.20}$$

then by replacing all $< \boldsymbol{x}_i, \boldsymbol{x}_j >$ with $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ we get Kernelized KNN.

14.4.2 Kernelized K-medoids clustering

K-medoids algorothm is similar to K-means(see Section 11.4.4), but instead of representing each clusters centroid by the mean of all data vectors assigned to this cluster, we make each centroid be one of the data vectors themselves. Thus we always deal with integer indexes, rather than data objects.

This algorithm can be kernelized by using Equation 14.20 to replace the distance computation.

14.4.3 Kernelized ridge regression

Applying the kernel trick to distance-based methods was straightforward. It is not so obvious how to apply it to parametric models such as ridge regression. However, it can be done, as we now explain. This will serve as a good warm up for studying SVMs.

14.4.3.1 The primal problem

we rewrite Equation 7.22 as the following

$$J(\boldsymbol{w}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|^2 \tag{14.21}$$

and its solution is given by Equation 7.23.

14.4.3.2 The dual problem

Equation 14.21 is not yet in the form of inner products. However, using the matrix inversion lemma (Equation 4.107 TODO) we rewrite the ridge estimate as follows

$$\boldsymbol{w} = \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I}_N)^{-1}\boldsymbol{y} \tag{14.22}$$

which takes $O(N^3 + N^2D)$ time to compute. This can be advantageous if D is large. Furthermore, we see that we can partially kernelize this, by replacing $\boldsymbol{X}\boldsymbol{X}^T$ with the Gram matrix $\boldsymbol{K}$. But what about the leading $\boldsymbol{X}^T$ term?

Let us define the following **dual variables**:

$$\alpha = (K + \lambda I_N)^{-1} y \quad (14.23)$$

Then we can rewrite the **primal variables** as follows

$$w = X^T \alpha = \sum_{i=1}^{N} \alpha_i x_i \quad (14.24)$$

This tells us that the solution vector is just a linear sum of the N training vectors. When we plug this in at test time to compute the predictive mean, we get

$$y = f(x) = \sum_{i=1}^{N} \alpha_i x_i^T x = \sum_{i=1}^{N} \alpha_i \kappa(x_i, x) \quad (14.25)$$

So we have succesfully kernelized ridge regression by changing from primal to dual variables. This technique can be applied to many other linear models, such as logistic regression.

14.4.3.3 Computational cost

The cost of computing the dual variables α is $O(N^3)$, whereas the cost of computing the primal variables w is $O(D^3)$. Hence the kernel method can be useful in high dimensional settings, even if we only use a linear kernel (c.f., the SVD trick in Equation 7.24). However, prediction using the dual variables takes $O(ND)$ time, while prediction using the primal variables only takes $O(D)$ time. We can speedup prediction by making α sparse, as we discuss in Section 14.5.

14.4.4 Kernel PCA

TODO

14.5 Support vector machines (SVMs)

In Section 14.3.2, we saw one way to derive a sparse kernel machine, namely by using a GLM with kernel basis functions, plus a sparsity-promoting prior such as ℓ_1 or ARD. An alternative approach is to change the objective function from negative log likelihood to some other loss function, as we discussed in Section 6.4.5. In particular, consider the ℓ_2 regularized empirical risk function

$$J(w, \lambda) = \sum i = 1^N L(y_i, \hat{y}_i) + \lambda \|w\|^2 \quad (14.26)$$

where $\hat{y}_i = w^T x_i + w_0$.

If L is quadratic loss, this is equivalent to ridge regression, and if L is the log-loss defined in Equation 6.3, this is equivalent to logistic regression.

However, if we replace the loss function with some other loss function, to be explained below, we can ensure that the solution is sparse, so that predictions only depend on a subset of the training data, known as **support vectors**. This combination of the kernel trick plus a modified loss function is known as a **support vector machine** or **SVM**.

Note that SVMs are very unnatural from a probabilistic point of view.

- First, they encode sparsity in the loss function rather than the prior.
- Second, they encode kernels by using an algorithmic trick, rather than being an explicit part of the model.
- Finally, SVMs do not result in probabilistic outputs, which causes various difficulties, especially in the multi-class classification setting (see Section 14.5.2.4 TODO for details).

It is possible to obtain sparse, probabilistic, multi-class kernel-based classifiers, which work as well or better than SVMs, using techniques such as the L1VM or RVM, discussed in Section 14.3.2. However, we include a discussion of SVMs, despite their non-probabilistic nature, for two main reasons.

- First, they are very popular and widely used, so all students of machine learning should know about them.
- Second, they have some computational advantages over probabilistic methods in the structured output case; see Section 19.7 TODO.

14.5.1 SVMs for classification

14.5.1.1 Primal form

Representation

$$\mathcal{H} : y = f(x) = \text{sign}(wx + b) \quad (14.27)$$

Evaluation

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 \quad (14.28)$$

$$s.t. \quad y_i(wx_i + b) \geqslant 1, i = 1, 2, \ldots, N \quad (14.29)$$

14.5.1.2 Dual form

Representation

$$\mathcal{H}: y = f(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i (\boldsymbol{x} \cdot \boldsymbol{x}_i) + b\right) \tag{14.30}$$

Evaluation

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j) - \sum_{i=1}^{N} \alpha_i \tag{14.31}$$

$$s.t. \quad \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{14.32}$$

$$\alpha_i \geqslant 0, i = 1, 2, \ldots, N \tag{14.33}$$

14.5.1.3 Primal form with slack variables

Representation

$$\mathcal{H}: y = f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}\boldsymbol{x} + b) \tag{14.34}$$

Evaluation

$$\min_{\boldsymbol{w},b} \quad C\sum_{i=1}^{N} \xi_i + \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{14.35}$$

$$s.t. \quad y_i(\boldsymbol{w}\boldsymbol{x}_i + b) \geqslant 1 - \xi_i \tag{14.36}$$

$$\xi_i \geqslant 0, \quad i = 1, 2, \ldots, N \tag{14.37}$$

14.5.1.4 Dual form with slack variables

Representation

$$\mathcal{H}: y = f(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i (\boldsymbol{x} \cdot \boldsymbol{x}_i) + b\right) \tag{14.38}$$

Evaluation

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_i \cdot \boldsymbol{x}_j) - \sum_{i=1}^{N} \alpha_i \tag{14.39}$$

$$s.t. \quad \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{14.40}$$

$$0 \leqslant \alpha_i \leqslant C, i = 1, 2, \ldots, N \tag{14.41}$$

$$\alpha_i = 0 \Rightarrow y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geqslant 1 \tag{14.42}$$

$$\alpha_i = C \Rightarrow y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \leqslant 1 \tag{14.43}$$

$$0 < \alpha_i < C \Rightarrow y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) = 1 \tag{14.44}$$

14.5.1.5 Hinge Loss

Linear support vector machines can also be interpreted as hinge loss minimization:

$$\min_{\boldsymbol{w},b} \sum_{i=1}^{N} L(y_i, f(\boldsymbol{x}_i)) + \lambda\|\boldsymbol{w}\|^2 \tag{14.45}$$

where $L(y, f(\boldsymbol{x}))$ is a **hinge loss function**:

$$L(y, f(\boldsymbol{x})) = \begin{cases} 1 - yf(x), & 1 - yf(x) > 0 \\ 0, & 1 - yf(x) \leqslant 0 \end{cases} \tag{14.46}$$

Proof. We can write equation 14.45 as equations 14.35 $\sim$ 14.37.

Define slack variables

$$\xi_i \triangleq 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b), \xi_i \geqslant 0 \tag{14.47}$$

Then $\boldsymbol{w}, b, \xi_i$ satisfy the constraints 14.35 and 14.36. And objective function 14.37 can be written as

$$\min_{\boldsymbol{w},b} \sum_{i=1}^{N} \xi_i + \lambda\|\boldsymbol{w}\|^2$$

If $\lambda = \frac{1}{2C}$, then

$$\min_{\boldsymbol{w},b} \frac{1}{C}\left(C\sum_{i=1}^{N} \xi_i + \frac{1}{2}\|\boldsymbol{w}\|^2\right) \tag{14.48}$$

It is equivalent to equation 14.35.

14.5.1.6 Optimization

QP, SMO

14.5.2 SVMs for regression

14.5.2.1 Representation

$$\mathcal{H}: y = f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b \tag{14.49}$$

14.5.2.2 Evaluation

$$J(\boldsymbol{w}) = C\sum_{i=1}^{N} L(y_i, f(\boldsymbol{x}_i)) + +\frac{1}{2}\|\boldsymbol{w}\|^2 \tag{14.50}$$

where $L(y, f(\boldsymbol{x}))$ is a **epsilon insensitive loss function**:

$$L(y, f(\boldsymbol{x})) = \begin{cases} 0 & , |y - f(\boldsymbol{x})| < \varepsilon \\ |y - f(\boldsymbol{x})| - \varepsilon & , \text{otherwise} \end{cases} \quad (14.51)$$

and $C = 1/\lambda$ is a regularization constant.

This objective is convex and unconstrained, but not differentiable, because of the absolute value function in the loss term. As in Section 13.4 TODO, where we discussed the lasso problem, there are several possible algorithms we could use. One popular approach is to formulate the problem as a constrained optimization problem. In particular, we introduce **slack variables** to represent the degree to which each point lies outside the tube:

$$y_i \leq f(\boldsymbol{x}_i) + \varepsilon + \xi_i^+$$
$$y_i \geq f(\boldsymbol{x}_i) - \varepsilon - \xi_i^-$$

Given this, we can rewrite the objective as follows:

$$J(\boldsymbol{w}) = C \sum_{i=1}^{N} (\xi_i^+ + \xi_i^-) + + \frac{1}{2} \|\boldsymbol{w}\|^2 \quad (14.52)$$

This is a standard quadratic problem in $2N + D + 1$ variables.

14.5.3 Choosing C

SVMs for both classification and regression require that you specify the kernel function and the parameter C. Typically C is chosen by cross-validation. Note, however, that C interacts quite strongly with the kernel parameters. For example, suppose we are using an RBF kernel with precision $\gamma = \frac{1}{2\sigma^2}$. If $\gamma = 5$, corresponding to narrow kernels, we need heavy regularization, and hence small C(so $\lambda = 1/C$is big). If $\gamma = 1$, a larger value of Cshould be used. So we see that γ and C are tightly coupled. This is illustrated in Figure 14.2, which shows the CV estimate of the 0-1 risk as a function of C and γ.

The authors of libsvm recommend (Hsu et al. 2009) using CV over a 2d grid with values $C \in \{2^5, 2^3, \cdots, 2^{15}\}$ and $\gamma \in \{2^{15}, 2^{13}, \cdots, 2^3\}$. In addition, it is important to standardize the data first, for a spherical Gaussian kernel to make sense.

To choose C efficiently, one can develop a path following algorithm in the spirit of lars (Section 13.3.4 TODO). The basic idea is to start with λ large, so that the margin $1/\|\boldsymbol{w}(\lambda)\|$ is wide, and hence all points are inside of it and have $\alpha_i = 1$. By slowly decreasing λ, a small set of points will move from inside the margin to outside, and their α_i values will change from 1 to 0, as they cease to be support vectors. When λ is maximal, the function is completely smoothed, and no support vectors remain. See (Hastie et al. 2004) for the details.

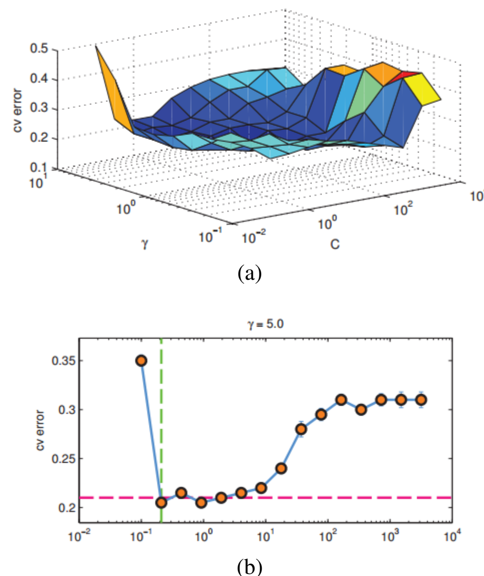


Fig. 14.2: (a) A cross validation estimate of the 0-1 error for an SVM classifier with RBF kernel with different precisions $\gamma = 1/(2\sigma^2)$ and different regularizer $\gamma = 1/C$, applied to a synthetic data set drawn from a mixture of 2 Gaussians. (b) A slice through this surface for $\gamma = 5$ The red dotted line is the Bayes optimal error, computed using Bayes rule applied to the model used to generate the data. Based on Figure 12.6 of (Hastie et al. 2009).

14.5.4 A probabilistic interpretation of SVMs

TODO see MLAPP Section 14.5.5

14.5.5 Summary of key points

Summarizing the above discussion, we recognize that SVM classifiers involve three key ingredients: the kernel trick, sparsity, and the large margin principle. The kernel trick is necessary to prevent underfitting, i.e., to ensure that the feature vector is sufficiently rich that a linear classifier can separate the data. (Recall from Section 14.2.3 that any Mercer kernel can be viewed as implicitly defining a potentially high dimensional feature vector.) If the original features are already high dimensional (as in many gene expression and text classification problems), it suffices to use a linear kernel, $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}'$, which is equivalent to working with the original features.

The sparsity and large margin principles are necessary to prevent overfitting, i.e., to ensure that we do not use all

the basis functions. These two ideas are closely related to each other, and both arise (in this case) from the use of the hinge loss function. However, there are other methods of achieving sparsity (such as ℓ_1), and also other methods of maximizing the margin(such as boosting). A deeper discussion of this point takes us outside of the scope of this book. See e.g., (Hastie et al. 2009) for more information.

14.6 Comparison of discriminative kernel methods

We have mentioned several different methods for classification and regression based on kernels, which we summarize in Table 14.1. (GP stands for Gaussian process, which we discuss in Chapter 15 TODO.) The columns have the following meaning:

- Optimize $\boldsymbol{w}$: a key question is whether the objective $J(\boldsymbol{w} = -\log p(\mathcal{D}|\boldsymbol{w}) - \log p(\boldsymbol{w}))$ is convex or not. L2VM, L1VM and SVMs have convex objectives. RVMs do not. GPs are Bayesian methods that do not perform parameter estimation.
- Optimize kernel: all the methods require that one tune the kernel parameters, such as the bandwidth of the RBF kernel, as well as the level of regularization. For methods based on Gaussians, including L2VM, RVMs and GPs, we can use efficient gradient based optimizers to maximize the marginal likelihood. For SVMs, and L1VM, we must use cross validation, which is slower (see Section 14.5.3).
- Sparse: L1VM, RVMs and SVMs are sparse kernel methods, in that they only use a subset of the training examples. GPs and L2VM are not sparse: they use all the training examples. The principle advantage of sparsity is that prediction at test time is usually faster. In addition, one can sometimes get improved accuracy.
- Probabilistic: All the methods except for SVMs produce probabilistic output of the form $p(y|\boldsymbol{x})$. SVMs produce a confidence value that can be converted to a probability, but such probabilities are usually very poorly calibrated (see Section 14.5.2.3 TODO).
- Multiclass: All the methods except for SVMs naturally work in the multiclass setting, by using a multinoulli output instead of Bernoulli. The SVM can be made into a multiclass classifier, but there are various difficulties with this approach, as discussed in Section 14.5.2.4 TODO.
- Mercer kernel: SVMs and GPs require that the kernel is positive definite; the other techniques do not.

Method	Opt. $\boldsymbol{w}$	Opt.	Sparse	Prob.	Multiclass	Non-Mercer	Section
L2VM	Convex	EB	No	Yes	Yes	Yes	14.3.2
L1VM	Convex	CV	Yes	Yes	Yes	Yes	14.3.2
RVM	Not convex	EB	Yes	Yes	Yes	Yes	14.3.2
SVM	Convex	CV	Yes	No	Indirectly	No	14.5
GP	N/A	EB	No	Yes	Yes	No	15

Table 14.1: Comparison of various kernel based classifiers. EB = empirical Bayes, CV = cross validation. See text for details

14.7 Kernels for building generative models

TODO