

13

Kernel Machines

Kernel machines are maximum margin methods that allow the model to be written as a sum of the influences of a subset of the training instances. These influences are given by application-specific similarity kernels, and we discuss “kernelized” classification, regression, ranking, outlier detection and dimensionality reduction, and how to choose and use kernels.

13.1 Introduction

We now discuss a different approach for linear classification and regression. We should not be surprised to have so many different methods even for the simple case of a linear model. Each learning algorithm has a different inductive bias, makes different assumptions, and defines a different objective function and thus may find a different linear model.

The model that we will discuss in this chapter, called the *support vector machine* (SVM), and later generalized under the name *kernel machine*, has been popular in recent years for a number of reasons:

1. It is a **discriminant-based method** and uses Vapnik’s principle to never solve a more complex problem as a first step before the actual problem (Vapnik 1995). For example, in classification, when the task is to learn the discriminant, it is not necessary to estimate where the class densities $p(\mathbf{x}|C_i)$ or the exact posterior probability values $P(C_i|\mathbf{x})$; we only need to estimate where the class boundaries lie, that is, \mathbf{x} where $P(C_i|\mathbf{x}) = P(C_j|\mathbf{x})$. Similarly, for outlier detection, we do not need to estimate the full density $p(\mathbf{x})$; we only need to find the boundary separating those \mathbf{x} that have low $p(\mathbf{x})$, that is, \mathbf{x} where $p(\mathbf{x}) < \theta$, for some threshold $\theta \in (0, 1)$.

2. After training, the parameter of the linear model, the weight vector, can be written down in terms of a subset of the training set, which are the so-called *support vectors*. In classification, these are the cases that are close to the boundary and as such, knowing them allows knowledge extraction: Those are the uncertain or erroneous cases that lie in the vicinity of the boundary between two classes. Their number gives us an estimate of the generalization error, and, as we see below, being able to write the model parameter in terms of a set of instances allows kernelization.
3. As we will see shortly, the output is written as a sum of the influences of support vectors and these are given by *kernel functions* that are application-specific measures of similarity between data instances. Previously, we talked about nonlinear basis functions allowing us to map the input to another space where a linear (smooth) solution is possible; the kernel function uses the same idea.
4. Typically in most learning algorithms, data points are represented as vectors, and either dot product (as in the multilayer perceptrons) or Euclidean distance (as in radial basis function networks) is used. A kernel function allows us to go beyond that. For example, G_1 and G_2 may be two graphs and $K(G_1, G_2)$ may correspond to the number of shared paths, which we can calculate without needing to represent G_1 or G_2 explicitly as vectors.
5. Kernel-based algorithms are formulated as convex optimization problems, and there is a single optimum that we can solve for analytically. Therefore we are no longer bothered with heuristics for learning rates, initializations, checking for convergence, and such. Of course, this does not mean that we do not have any hyperparameters for model selection; we do—any method needs them, to match the algorithm to the data at hand.

We start our discussion with the case of classification, and then generalize to regression, ranking, outlier (novelty) detection, and then dimensionality reduction. We see that in all cases basically we have the similar quadratic program template to maximize the separability, or *margin*, of instances subject to a constraint of the smoothness of solution. Solving for it, we get the support vectors. The kernel function defines the space according to its notion of similarity and a kernel function is good if we have better separation in its corresponding space.


13.2 Optimal Separating Hyperplane

Let us start again with two classes and use labels $-1/+1$ for the two classes. The sample is $\mathcal{X} = \{\mathbf{x}^t, r^t\}$ where $r^t = +1$ if $\mathbf{x}^t \in C_1$ and $r^t = -1$ if $\mathbf{x}^t \in C_2$. We would like to find \mathbf{w} and w_0 such that

$$\mathbf{w}^T \mathbf{x}^t + w_0 \geq +1 \quad \text{for } r^t = +1$$

$$\mathbf{w}^T \mathbf{x}^t + w_0 \leq -1 \quad \text{for } r^t = -1$$

which can be rewritten as



$$(13.1) \quad r^t (\mathbf{w}^T \mathbf{x}^t + w_0) \geq +1$$

Note that we do not simply require

$$r^t (\mathbf{w}^T \mathbf{x}^t + w_0) \geq 0$$

MARGIN Not only do we want the instances to be on the right side of the hyperplane, but we also want them some distance away, for better generalization. The distance from the hyperplane to the instances closest to it on either side is called the *margin*, which we want to maximize for best generalization.

Very early on, in section 2.1, we talked about the concept of the margin when we were talking about fitting a rectangle, and we said that it is better to take a rectangle halfway between S and G , to get a breathing space. This is so that in case noise shifts a test instance slightly, it will still be on the right side of the boundary.

OPTIMAL SEPARATING
HYPERPLANE

Similarly, now that we are using the hypothesis class of lines, the *optimal separating hyperplane* is the one that maximizes the margin.

We remember from section 10.3 that the distance of \mathbf{x}^t to the discriminant is

$$\frac{|\mathbf{w}^T \mathbf{x}^t + w_0|}{\|\mathbf{w}\|}$$

which, when $r^t \in \{-1, +1\}$, can be written as

$$\frac{r^t (\mathbf{w}^T \mathbf{x}^t + w_0)}{\|\mathbf{w}\|}$$

and we would like this to be at least some value ρ :

$$(13.2) \quad \frac{r^t (\mathbf{w}^T \mathbf{x}^t + w_0)}{\|\mathbf{w}\|} \geq \rho, \forall t$$

We would like to maximize ρ but there are an infinite number of solutions that we can get by scaling \mathbf{w} and for a unique solution, we fix $\rho\|\mathbf{w}\| = 1$ and thus, to maximize the margin, we minimize $\|\mathbf{w}\|$. The task can therefore be defined (see Cortes and Vapnik 1995; Vapnik 1995) as to

$$(13.3) \quad \min \frac{1}{2} \|\mathbf{w}\|^2 \text{ subject to } r^t(\mathbf{w}^T \mathbf{x}^t + w_0) \geq +1, \forall t$$

This is a standard quadratic optimization problem, whose complexity depends on d , and it can be solved directly to find \mathbf{w} and w_0 . Then, on both sides of the hyperplane, there will be instances that are $1/\|\mathbf{w}\|$ away from the hyperplane and the total margin will be $2/\|\mathbf{w}\|$.

We saw in section 10.2 that if the problem is not linearly separable, instead of fitting a nonlinear function, one trick is to map the problem to a new space by using nonlinear basis functions. It is generally the case that this new space has many more dimensions than the original space, and, in such a case, we are interested in a method whose complexity does not depend on the input dimensionality.

In finding the optimal hyperplane, we can convert the optimization problem to a form whose complexity depends on N , the number of training instances, and not on d . Another advantage of this new formulation is that it will allow us to rewrite the basis functions in terms of kernel functions, as we will see in section 13.5.

To get the new formulation, we first write equation 13.3 as an unconstrained problem using Lagrange multipliers α^t :

$$(13.4) \quad \begin{aligned} L_p &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{t=1}^N \alpha^t [r^t(\mathbf{w}^T \mathbf{x}^t + w_0) - 1] \\ &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_t \alpha^t r^t(\mathbf{w}^T \mathbf{x}^t + w_0) + \sum_t \alpha^t \end{aligned}$$

This should be minimized with respect to \mathbf{w} , w_0 and maximized with respect to $\alpha^t \geq 0$. The saddle point gives the solution.

This is a convex quadratic optimization problem because the main term is convex and the linear constraints are also convex. Therefore, we can equivalently solve the dual problem, making use of the Karush-Kuhn-Tucker conditions. The dual is to maximize L_p with respect to α^t , subject to the constraints that the gradient of L_p with respect to \mathbf{w} and w_0 are 0

and also that $\alpha^t \geq 0$:

$$(13.5) \quad \frac{\partial L_p}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_t \alpha^t r^t \mathbf{x}^t$$

$$(13.6) \quad \frac{\partial L_p}{\partial w_0} = 0 \Rightarrow \sum_t \alpha^t r^t = 0$$

Plugging these into equation 13.4, we get the dual

$$\begin{aligned} L_d &= \frac{1}{2}(\mathbf{w}^T \mathbf{w}) - \mathbf{w}^T \sum_t \alpha^t r^t \mathbf{x}^t - w_0 \sum_t \alpha^t r^t + \sum_t \alpha^t \\ &= -\frac{1}{2}(\mathbf{w}^T \mathbf{w}) + \sum_t \alpha^t \\ (13.7) \quad &= -\frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s (\mathbf{x}^t)^T \mathbf{x}^s + \sum_t \alpha^t \end{aligned}$$

which we maximize with respect to α^t only, subject to the constraints

$$\sum_t \alpha^t r^t = 0, \text{ and } \alpha^t \geq 0, \forall t$$

This can be solved using quadratic optimization methods. The size of the dual depends on N , sample size, and not on d , the input dimensionality. The upper bound for time complexity is $\mathcal{O}(N^3)$, and the upper bound for space complexity is $\mathcal{O}(N^2)$.

Once we solve for α^t , we see that though there are N of them, most vanish with $\alpha^t = 0$ and only a small percentage have $\alpha^t > 0$. The set of \mathbf{x}^t whose $\alpha^t > 0$ are the *support vectors*, and as we see in equation 13.5, \mathbf{w} is written as the weighted sum of these training instances that are selected as the support vectors. These are the \mathbf{x}^t that satisfy

$$r^t(\mathbf{w}^T \mathbf{x}^t + w_0) = 1$$

and lie on the margin. We can use this fact to calculate w_0 from any support vector as

$$(13.8) \quad w_0 = r^t - \mathbf{w}^T \mathbf{x}^t$$

For numerical stability, it is advised that this be done for all support vectors and an average be taken. The discriminant thus found is called the *support vector machine* (SVM) (see figure 13.1).

The majority of the α^t are 0, for which $r^t(\mathbf{w}^T \mathbf{x}^t + w_0) > 1$. These are the \mathbf{x}^t that lie more than sufficiently away from the discriminant,

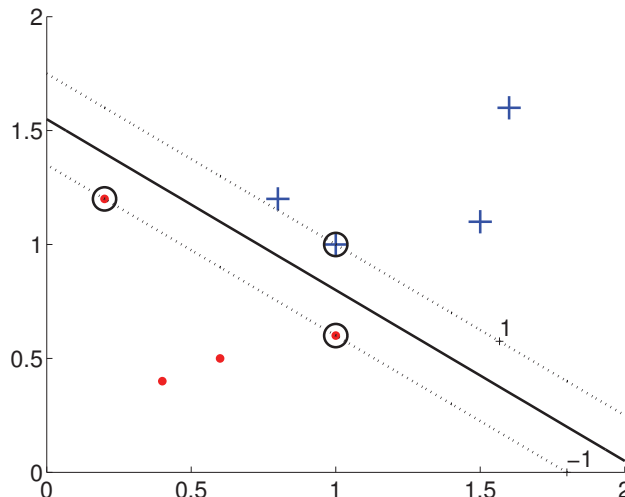


Figure 13.1 For a two-class problem where the instances of the classes are shown by plus signs and dots, the thick line is the boundary and the dashed lines define the margins on either side. Circled instances are the support vectors.

and they have no effect on the hyperplane. The instances that are not support vectors carry no information; even if any subset of them are removed, we would still get the same solution. From this perspective, the SVM algorithm can be likened to the condensed nearest neighbor algorithm (section 8.5), which stores only the instances neighboring (and hence constraining) the class discriminant.

Being a discriminant-based method, the SVM cares only about the instances close to the boundary and discards those that lie in the interior. Using this idea, it is possible to use a simpler classifier before the SVM to filter out a large portion of such instances, thereby decreasing the complexity of the optimization step of the SVM (exercise 1).

During testing, we do not enforce a margin. We calculate $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$, and choose according to the sign of $g(\mathbf{x})$:

Choose C_1 if $g(\mathbf{x}) > 0$ and C_2 otherwise

13.3 The Nonseparable Case: Soft Margin Hyperplane

If the data is not linearly separable, the algorithm we discussed earlier will not work. In such a case, if the two classes are not linearly separable such that there is no hyperplane to separate them, we look for the one that incurs the least error. We define *slack variables*, $\xi^t \geq 0$, which store the deviation from the margin. There are two types of deviation: An instance may lie on the wrong side of the hyperplane and be misclassified. Or, it may be on the right side but may lie in the margin, namely, not sufficiently away from the hyperplane. Relaxing equation 13.1, we require

$$(13.9) \quad r^t(\mathbf{w}^T \mathbf{x}^t + w_0) \geq 1 - \xi^t$$

If $\xi^t = 0$, there is no problem with \mathbf{x}^t . If $0 < \xi^t < 1$, \mathbf{x}^t is correctly classified but in the margin. If $\xi^t \geq 1$, \mathbf{x}^t is misclassified (see figure 13.2). The number of misclassifications is $\#\{\xi^t > 1\}$, and the number of non-

SOFT ERROR

$$\sum_t \xi^t$$

and add this as a penalty term:


$$(13.10) \quad L_p = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_t \xi^t$$

subject to the constraint of equation 13.9. C is the penalty factor as in any regularization scheme trading off complexity, as measured by the L_2 norm of the weight vector (similar to weight decay in multilayer perceptrons; see section 11.9 and 11.10), and data misfit, as measured by the number of nonseparable points. Note that we are penalizing not only the misclassified points but also the ones in the margin for better generalization, though these latter would be correctly classified during testing.

Adding the constraints, the Lagrangian of equation 13.4 then becomes

$$(13.11) \quad L_p = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_t \xi^t - \sum_t \alpha^t [r^t(\mathbf{w}^T \mathbf{x}^t + w_0) - 1 + \xi^t] - \sum_t \mu^t \xi^t$$

where μ_t are the new Lagrange parameters to guarantee the positivity of ξ^t . When we take the derivatives with respect to the parameters and set them to 0, we get

$$(13.12) \quad \frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_t \alpha^t r^t \mathbf{x}^t = 0 \Rightarrow \mathbf{w} = \sum_t \alpha^t r^t \mathbf{x}^t$$


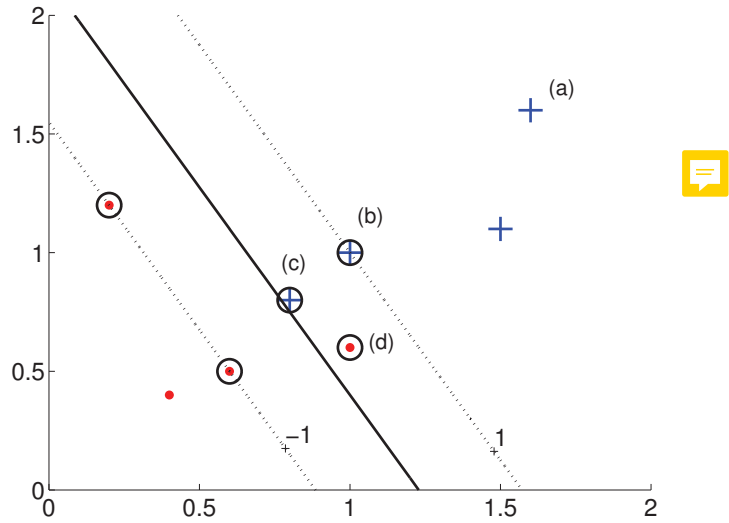


Figure 13.2 In classifying an instance, there are four possible cases: In (a), the instance is on the correct side and far away from the margin; $r^t g(\mathbf{x}^t) > 1$, $\xi^t = 0$. In (b), $\xi^t = 0$; it is on the right side and on the margin. In (c), $\xi^t = 1 - g(\mathbf{x}^t)$, $0 < \xi < 1$; it is on the right side but is in the margin and not sufficiently away. In (d), $\xi^t = 1 + g(\mathbf{x}^t) > 1$; it is on the wrong side—this is a misclassification. **All cases except (a) are support vectors.** In terms of the dual variable, in (a), $\alpha^t = 0$; in (b), $\alpha^t < C$; in (c) and (d), $\alpha^t = C$.

$$(13.13) \quad \frac{\partial L_p}{\partial w_0} = \sum_t \alpha^t r^t = 0$$

$$(13.14) \quad \frac{\partial L_p}{\partial \xi^t} = C - \alpha^t - \mu^t = 0$$

Since $\mu^t \geq 0$, this last implies that $0 \leq \alpha^t \leq C$. Plugging these into equation 13.11, we get the dual that we maximize with respect to α^t :

$$(13.15) \quad L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s (\mathbf{x}^t)^T \mathbf{x}^s$$

subject to

$$\sum_t \alpha^t r^t = 0 \text{ and } 0 \leq \alpha^t \leq C, \forall t$$

Solving this, we see that as in the separable case, instances that lie on the correct side of the boundary with sufficient margin vanish with their $\alpha^t = 0$ (see figure 13.2). The support vectors have their $\alpha^t > 0$ and they define \mathbf{w} , as given in equation 13.12. Of these, those whose $\alpha^t < C$ are the ones that are on the margin, and we can use them to calculate w_0 ; they have $\xi^t = 0$ and satisfy $r^t(\mathbf{w}^T \mathbf{x}^t + w_0) = 1$. Again, it is better to take an average over these w_0 estimates. Those instances that are in the margin or misclassified have their $\alpha^t = C$.

The nonseparable instances that we store as support vectors are the instances that we would have trouble correctly classifying if they were not in the training set; they would either be misclassified or classified correctly but not with enough confidence. We can say that the number of support vectors is an upper-bound estimate for the expected number of errors. And, actually, Vapnik (1995) has shown that the expected test error rate is

$$E_N[P(\text{error})] \leq \frac{E_N[\# \text{ of support vectors}]}{N}$$

where $E_N[\cdot]$ denotes expectation over training sets of size N . The nice implication of this is that it shows that the error rate depends on the number of support vectors and not on the input dimensionality.

Equation 13.9 implies that we define error if the instance is on the wrong side or if the margin is less than 1. This is called the *hinge loss*. If $y^t = \mathbf{w}^T \mathbf{x}^t + w_0$ is the output and r^t is the desired output, hinge loss is defined as

$$(13.16) \quad L_{\text{hinge}}(y^t, r^t) = \begin{cases} 0 & \text{if } y^t r^t \geq 1 \\ 1 - y^t r^t & \text{otherwise} \end{cases}$$

In figure 13.3, we compare hinge loss with 0/1 loss, squared error, and cross-entropy. We see that unlike 0/1 loss, hinge loss also penalizes instances in the margin even though they may be on the correct side, and the loss increases linearly as the instance moves away on the wrong side. This is different from the squared loss that therefore is not as robust as the hinge loss. We see that cross-entropy minimized in logistic discrimination (section 10.7) or by the linear perceptron (section 11.3) is a good continuous approximation to the hinge loss.

C of equation 13.10 is the regularization parameter fine-tuned using cross-validation. It defines the trade-off between margin maximization and error minimization: If it is too large, we have a high penalty for nonseparable points, and we may store many support vectors and overfit.

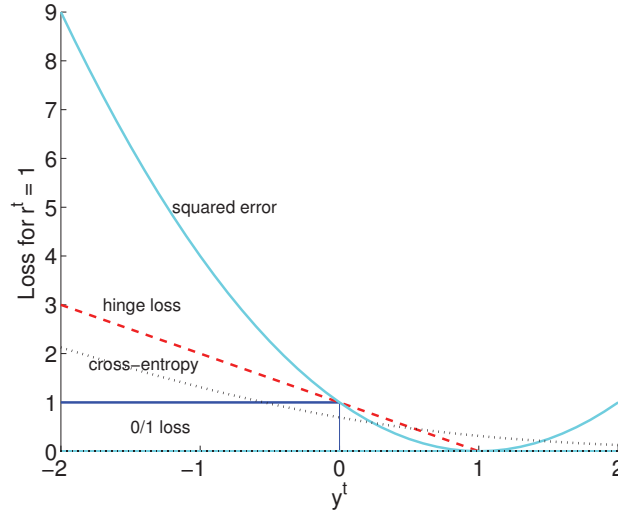


Figure 13.3 Comparison of different loss functions for $r^t = 1$: 0/1 loss is 0 if $y^t = 1$, 1 otherwise. Hinge loss is 0 if $y^t > 1$, $1 - y^t$ otherwise. Squared error is $(1 - y^t)^2$. Cross-entropy is $\log(1/(1 + \exp(-y^t)))$.

If it is too small, we may find too simple solutions that underfit. Typically, one chooses from $[10^{-6}, 10^{-5}, \dots, 10^{+5}, 10^{+6}]$ in the log scale by looking at the accuracy on a validation set.

13.4 ν -SVM

There is another, equivalent formulation of the soft margin hyperplane that uses a parameter $\nu \in [0, 1]$ instead of C (Schölkopf et al. 2000). The objective function is

$$(13.17) \quad \min \frac{1}{2} \|\mathbf{w}\|^2 - \nu \rho + \frac{1}{N} \sum_t \xi^t$$

subject to

$$(13.18) \quad r^t(\mathbf{w}^T \mathbf{x}^t + w_0) \geq \rho - \xi^t, \quad \xi^t \geq 0, \quad \rho \geq 0$$

ρ is a new parameter that is a variable of the optimization problem and scales the margin: The margin is now $2\rho/\|\mathbf{w}\|$. ν has been shown to be

a lower bound on the fraction of support vectors and an upper bound on the fraction of instances having margin errors ($\sum_t \#\{\xi^t > 0\}$). The dual is

$$(13.19) \quad L_d = -\frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s (\mathbf{x}^t)^T \mathbf{x}^s$$

subject to

$$\sum_t \alpha^t r^t = 0, \quad 0 \leq \alpha^t \leq \frac{1}{N}, \quad \sum_t \alpha^t \geq \nu$$

When we compare equation 13.19 with equation 13.15, we see that the term $\sum_t \alpha^t$ no longer appears in the objective function but is now a constraint. By playing with ν , we can control the fraction of support vectors, and this is advocated to be more intuitive than playing with C .

13.5 Kernel Trick



Section 10.2 demonstrated that if the problem is nonlinear, instead of trying to fit a nonlinear model, we can map the problem to a new space by doing a nonlinear transformation using suitably chosen basis functions and then use a linear model in this new space. The linear model in the new space corresponds to a nonlinear model in the original space. This approach can be used in both classification and regression problems, and in the special case of classification, it can be used with any scheme. In the particular case of support vector machines, it leads to certain simplifications that we now discuss.

Let us say we have the new dimensions calculated through the basis functions

$$\mathbf{z} = \boldsymbol{\phi}(\mathbf{x}) \text{ where } z_j = \phi_j(\mathbf{x}), j = 1, \dots, k$$

mapping from the d -dimensional \mathbf{x} space to the k -dimensional \mathbf{z} space where we write the discriminant as

$$(13.20) \quad \begin{aligned} g(\mathbf{z}) &= \mathbf{w}^T \mathbf{z} \\ g(\mathbf{x}) &= \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) \\ &= \sum_{j=1}^k w_j \phi_j(\mathbf{x}) \end{aligned}$$

where we do not use a separate w_0 ; we assume that $z_1 = \phi_1(\mathbf{x}) \equiv 1$. Generally, k is much larger than d and k may also be larger than N , and there

lies the advantage of using the dual form whose complexity depends on N , whereas if we used the primal it would depend on k . We also use the more general case of the soft margin hyperplane here because we have no guarantee that the problem is linearly separable in this new space.

The problem is the same

$$(13.21) \quad L_p = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_t \xi^t$$

except that now the constraints are defined in the new space

$$(13.22) \quad r^t \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}^t) \geq 1 - \xi^t$$

The Lagrangian is

$$(13.23) \quad L_p = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_t \xi^t - \sum_t \alpha^t [r^t \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}^t) - 1 + \xi^t] - \sum_t \mu^t \xi^t$$

When we take the derivatives with respect to the parameters and set them to 0, we get

$$(13.24) \quad \frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} = \sum_t \alpha^t r^t \boldsymbol{\phi}(\mathbf{x}^t)$$

$$(13.25) \quad \frac{\partial L_p}{\partial \xi^t} = C - \alpha^t - \mu^t = 0$$

The dual is now

$$(13.26) \quad L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s \boldsymbol{\phi}(\mathbf{x}^t)^T \boldsymbol{\phi}(\mathbf{x}^s)$$

subject to

$$\sum_t \alpha^t r^t = 0 \text{ and } 0 \leq \alpha^t \leq C, \forall t$$

KERNEL FUNCTION The idea in *kernel machines* is to replace the inner product of basis functions, $\boldsymbol{\phi}(\mathbf{x}^t)^T \boldsymbol{\phi}(\mathbf{x}^s)$, by a *kernel function*, $K(\mathbf{x}^t, \mathbf{x}^s)$, between instances in the original input space. So instead of mapping two instances \mathbf{x}^t and \mathbf{x}^s to the \mathbf{z} -space and doing a dot product there, we directly apply the kernel function in the original space.

$$(13.27) \quad L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s K(\mathbf{x}^t, \mathbf{x}^s)$$

The kernel function also shows up in the discriminant

$$\begin{aligned}
 g(\mathbf{x}) &= \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) = \sum_t \alpha^t r^t \boldsymbol{\phi}(\mathbf{x}^t)^T \boldsymbol{\phi}(\mathbf{x}) \\
 (13.28) \quad &= \sum_t \alpha^t r^t K(\mathbf{x}^t, \mathbf{x})
 \end{aligned}$$

This implies that if we have the kernel function, we do not need to map it to the new space at all. Actually, for any valid kernel, there does exist a corresponding mapping function, but it may be much simpler to use $K(\mathbf{x}^t, \mathbf{x})$ rather than calculating $\boldsymbol{\phi}(\mathbf{x}^t)$, $\boldsymbol{\phi}(\mathbf{x})$ and taking the dot product. Many algorithms have been *kernelized*, as we will see in later sections, and that is why we have the name “kernel machines.”

KERNELIZATION

GRAM MATRIX

The matrix of kernel values, \mathbf{K} , where $\mathbf{K}_{ts} = K(\mathbf{x}^t, \mathbf{x}^s)$, is called the *Gram matrix*, which should be symmetric and positive semidefinite. Recently, it has become standard practice in sharing datasets to have available only the \mathbf{K} matrices without providing \mathbf{x}^t or $\boldsymbol{\phi}(\mathbf{x}^t)$. Especially in bioinformatics or natural language processing applications where \mathbf{x} (or $\boldsymbol{\phi}(\mathbf{x})$) has hundreds or thousands of dimensions, storing/downloading the $N \times N$ matrix is much cheaper (Vert, Tsuda, and Schölkopf 2004); this, however, implies that we can use only those available for training/testing and cannot use the trained model to make predictions outside this dataset.

13.6 Vectorial Kernels

The most popular, general-purpose kernel functions are

- *polynomials* of degree q :

$$(13.29) \quad K(\mathbf{x}^t, \mathbf{x}) = (\mathbf{x}^T \mathbf{x}^t + 1)^q$$

where q is selected by the user. For example, when $q = 2$ and $d = 2$,

$$\begin{aligned}
 K(\mathbf{x}, \mathbf{y}) &= (\mathbf{x}^T \mathbf{y} + 1)^2 \\
 &= (x_1 y_1 + x_2 y_2 + 1)^2 \\
 &= 1 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 x_2 y_1 y_2 + x_1^2 y_1^2 + x_2^2 y_2^2
 \end{aligned}$$

corresponds to the inner product of the basis function (Cherkassky and Mulier 1998):

$$\boldsymbol{\phi}(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2]^T$$

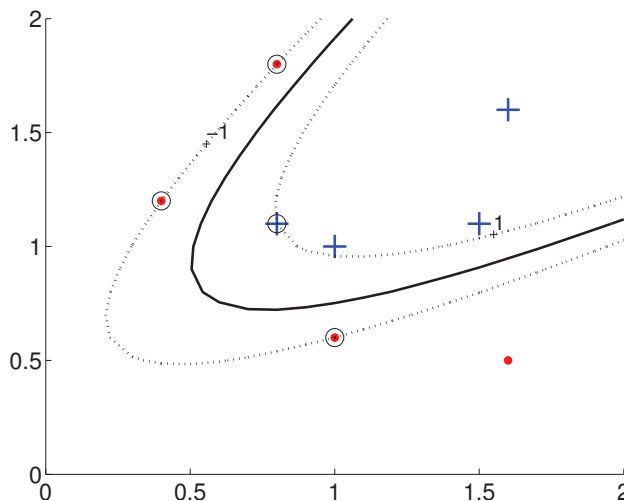


Figure 13.4 The discriminant and margins found by a polynomial kernel of degree 2. Circled instances are the support vectors.

An example is given in figure 13.4. When $q = 1$, we have the *linear kernel* that corresponds to the original formulation.

■ **radial-basis functions:**

$$(13.30) \quad K(\mathbf{x}^t, \mathbf{x}) = \exp \left[-\frac{\|\mathbf{x}^t - \mathbf{x}\|^2}{2s^2} \right]$$

defines a spherical kernel as in Parzen windows (chapter 8) where \mathbf{x}^t is the center and s , supplied by the user, defines the radius. This is also similar to radial basis functions that we discuss in chapter 12.

An example is shown in figure 13.5 where we see that larger spreads smooth the boundary; the best value is found by cross-validation. Note that when there are two parameters to be optimized using cross-validation, for example, here C and s^2 , one should do a grid (factorial) search in the two dimensions; we will discuss methods for searching the best combination of such factors in section 19.2.

One can have a Mahalanobis kernel, generalizing from the Euclidean distance:

$$(13.31) \quad K(\mathbf{x}^t, \mathbf{x}) = \exp \left[-\frac{1}{2} (\mathbf{x}^t - \mathbf{x})^T \mathbf{S}^{-1} (\mathbf{x}^t - \mathbf{x}) \right]$$

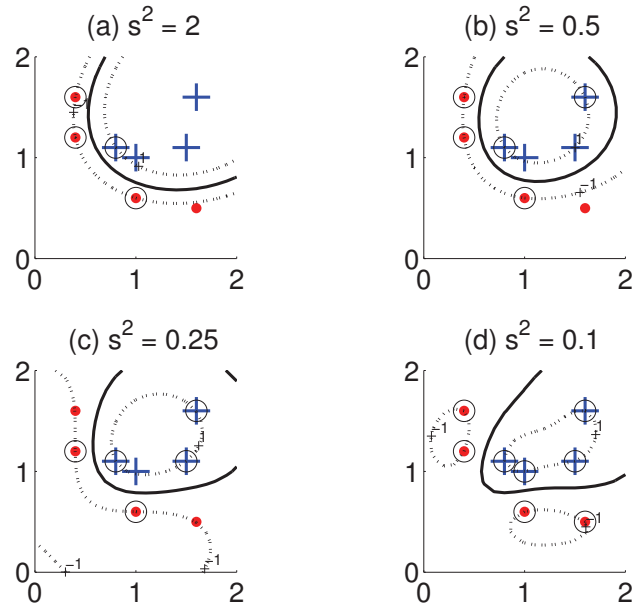


Figure 13.5 The boundary and margins found by the Gaussian kernel with different spread values, s^2 . We get smoother boundaries with larger spreads.

where \mathbf{S} is a covariance matrix. Or, in the most general case,

$$(13.32) \quad K(\mathbf{x}^t, \mathbf{x}) = \exp \left[-\frac{\mathcal{D}(\mathbf{x}^t, \mathbf{x})}{2s^2} \right]$$

for some distance function $\mathcal{D}(\mathbf{x}^t, \mathbf{x})$.

■ *sigmoidal functions:*

$$(13.33) \quad K(\mathbf{x}^t, \mathbf{x}) = \tanh(2\mathbf{x}^T \mathbf{x}^t + 1)$$

where $\tanh(\cdot)$ has the same shape with sigmoid, except that it ranges between -1 and $+1$. This is similar to multilayer perceptrons that we discussed in chapter 11.

13.7 Defining Kernels

It is also possible to define application-specific kernels. Kernels are generally considered to be measures of similarity in the sense that $K(\mathbf{x}, \mathbf{y})$ takes a larger value as \mathbf{x} and \mathbf{y} are more “similar,” from the point of view of the application. This implies that any prior knowledge we have regarding the application can be provided to the learner through appropriately defined kernels—“kernel engineering”—and such use of kernels can be seen as another example of a “hint” (section 11.8.4).

There are string kernels, tree kernels, graph kernels, and so on (Vert, Tsuda, and Schölkopf 2004), depending on how we represent the data and how we measure similarity in that representation.

For example, given two documents, the number of words appearing in both may be a kernel. Let us say D_1 and D_2 are two documents and one possible representation is called *bag of words* where we predefine M words relevant for the application, and we define $\boldsymbol{\phi}(D_1)$ as the M -dimensional binary vector whose dimension i is 1 if word i appears in D_1 and is 0 otherwise. Then, $\boldsymbol{\phi}(D_1)^T \boldsymbol{\phi}(D_2)$ counts the number of shared words. Here, we see that if we directly define and implement $K(D_1, D_2)$ as the number of shared words, we do not need to preselect M words and can use just any word in the vocabulary (of course, after discarding uninformative words like “of,” “and,” etc.) and we would not need to generate the bag-of-words representation explicitly and it would be as if we allowed M to be as large as we want.

Sometimes—for example, in bioinformatics applications—we can calculate a *similarity score* between two objects, which may not necessarily be positive semidefinite. Given two strings (of genes), a kernel measures the *edit distance*, namely, how many operations (insertions, deletions, substitutions) it takes to convert one string into another; this is also called *alignment*. In such a case, a trick is to define a set of M templates and represent an object as the M -dimensional vector of scores to all the templates. That is, if $\mathbf{m}_i, i = 1, \dots, M$ are the templates and $s(\mathbf{x}^t, \mathbf{m}_i)$ is the score between \mathbf{x}^t and \mathbf{m}_i , then we define

$$\boldsymbol{\phi}(\mathbf{x}^t) = [s(\mathbf{x}^t, \mathbf{m}_1), s(\mathbf{x}^t, \mathbf{m}_2), \dots, s(\mathbf{x}^t, \mathbf{m}_M)]^T$$

EMPIRICAL KERNEL
MAP

and we define the *empirical kernel map* as

$$K(\mathbf{x}^t, \mathbf{x}^s) = \boldsymbol{\phi}(\mathbf{x}^t)^T \boldsymbol{\phi}(\mathbf{x}^s)$$

which is a valid kernel.

DIFFUSION KERNEL

Sometimes, we have a binary score function; for example, two proteins may interact or not, and we want to be able to generalize from this to scores for two arbitrary instances. In such a case, a trick is to define a graph where the nodes are the instances and two nodes are linked if they interact, that is, if the binary score returns 1. Then we say that two nodes that are not immediately linked are “similar” if the path between them is short or if they are connected by many paths. This converts pairwise local interactions to a global similarity measure, rather like defining a geodesic distance used in Isomap (section 6.10), and it is called the *diffusion kernel*.

If $p(\mathbf{x})$ is a probability density, then

$$K(\mathbf{x}^t, \mathbf{x}) = p(\mathbf{x}^t)p(\mathbf{x})$$

is a valid kernel. This is used when $p(\mathbf{x})$ is a generative model for \mathbf{x} measuring how likely it is that we see \mathbf{x} . For example, if \mathbf{x} is a sequence, $p(\mathbf{x})$ can be a hidden Markov model (chapter 15). With this kernel, $K(\mathbf{x}^t, \mathbf{x})$ will take a high value if both \mathbf{x}^t and \mathbf{x} are likely to have been generated by the same model. It is also possible to parametrize the generative model as $p(\mathbf{x}|\theta)$ and learn θ from data; this is called the *Fisher kernel* (Jaakkola and Haussler 1998).

FISHER KERNEL

13.8 Multiple Kernel Learning

It is possible to construct new kernels by combining simpler kernels. If $K_1(\mathbf{x}, \mathbf{y})$ and $K_2(\mathbf{x}, \mathbf{y})$ are valid kernels and c a constant, then

$$(13.34) \quad K(\mathbf{x}, \mathbf{y}) = \begin{cases} cK_1(\mathbf{x}, \mathbf{y}) \\ K_1(\mathbf{x}, \mathbf{y}) + K_2(\mathbf{x}, \mathbf{y}) \\ K_1(\mathbf{x}, \mathbf{y}) \cdot K_2(\mathbf{x}, \mathbf{y}) \end{cases}$$

are also valid.

Different kernels may also be using different subsets of \mathbf{x} . We can therefore see combining kernels as another way to fuse information from different sources where each kernel measures similarity according to its domain. When we have input from two representations A and B

$$(13.35) \quad \begin{aligned} K_A(\mathbf{x}_A, \mathbf{y}_A) + K_B(\mathbf{x}_B, \mathbf{y}_B) &= \boldsymbol{\phi}_A(\mathbf{x}_A)^T \boldsymbol{\phi}_A(\mathbf{y}_A) + \boldsymbol{\phi}_B(\mathbf{x}_B)^T \boldsymbol{\phi}_B(\mathbf{y}_B) \\ &= \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{y}) \\ &= K(\mathbf{x}, \mathbf{y}) \end{aligned}$$

where $\mathbf{x} = [\mathbf{x}_A, \mathbf{x}_B]$ is the concatenation of the two representations. That is, taking a sum of two kernels corresponds to doing a dot product in the concatenated feature vectors. One can generalize to a number of kernels

$$(13.36) \quad K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^m K_i(\mathbf{x}, \mathbf{y})$$

which, similar to taking an average of classifiers (section 17.4), this time averages over kernels and frees us from the need to choose one particular kernel. It is also possible to take a weighted sum and also learn the weights from data (Lanckriet et al. 2004; Sonnenburg et al. 2006):

$$(13.37) \quad K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^m \eta_i K_i(\mathbf{x}, \mathbf{y})$$

MULTIPLE KERNEL
LEARNING

subject to $\eta_i \geq 0$, with or without the constraint of $\sum_i \eta_i = 1$, respectively known as convex or conic combination. This is called *multiple kernel learning* where we replace a single kernel with a weighted sum (Gönen and Alpaydm 2011). The single kernel objective function of equation 13.27 becomes

$$(13.38) \quad L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s r^t r^s \sum_i \eta_i K_i(\mathbf{x}^t, \mathbf{x}^s)$$

which we solve for both the support vector machine parameters α^t and the kernel weights η_i . Then, the combination of multiple kernels also appear in the discriminant

$$(13.39) \quad g(\mathbf{x}) = \sum_t \alpha^t r^t \sum_i \eta_i K_i(\mathbf{x}^t, \mathbf{x})$$

After training, η_i will take values depending on how the corresponding kernel $K_i(\mathbf{x}^t, \mathbf{x})$ is useful in discriminating. It is also possible to localize kernels by defining kernel weights as a parameterized function of the input \mathbf{x} , rather like the gating function in mixture of experts (section 17.8)

$$(13.40) \quad g(\mathbf{x}) = \sum_t \alpha^t r^t \sum_i \eta_i(\mathbf{x}|\theta_i) K_i(\mathbf{x}^t, \mathbf{x})$$

and the gating parameters θ_i are learned together with the support vector machine parameters (Gönen and Alpaydm 2008).

When we have information coming from multiple sources in different representations or modalities—for example, in speech recognition where we may have both acoustic and visual lip image—the usual approach is to

feed them separately to different classifiers and then fuse the decisions; we will discuss methods for this in detail in chapter 17. Combining multiple kernels provides us with another way of integrating input from multiple sources, where there is a single classifier that uses different kernels for inputs of different sources, for which there are different notions of similarity (Noble 2004). The localized version can then be seen as an extension of this where we can choose between sources, and hence similarity measures, depending on the input.

13.9 Multiclass Kernel Machines

When there are $K > 2$ classes, the straightforward, *one-vs.-all* way is to define K two-class problems, each one separating one class from all other classes combined and learn K support vector machines $g_i(\mathbf{x})$, $i = 1, \dots, K$. That is, in training $g_i(\mathbf{x})$, examples of C_i are labeled $+1$ and examples of C_k , $k \neq i$ are labeled as -1 . During testing, we calculate all $g_i(\mathbf{x})$ and choose the maximum.

Platt (1999) proposed to fit a sigmoid to the output of a single (2-class) SVM output to convert to a posterior probability. Similarly, one can train one layer of softmax outputs to minimize cross-entropy to generate $K > 2$ posterior probabilities (Mayraz and Alpaydm 1999):

$$(13.41) \quad y_i(\mathbf{x}) = \sum_{j=1}^K v_{ij} f_j(\mathbf{x}) + v_{i0}$$

where $f_j(\mathbf{x})$ are the SVM outputs and y_i are the posterior probability outputs. Weights v_{ij} are trained to minimize cross-entropy. Note, however, that as in stacking (section 17.9), the data on which we train v_{ij} should be different from the data used to train the base SVMs $f_j(\mathbf{x})$, to alleviate overfitting.

Instead of the usual approach of building K two-class SVM classifiers to separate one from all the rest, as with any other classifier, one can build $K(K-1)/2$ *pairwise* classifiers (see also section 10.4), each $g_{ij}(\mathbf{x})$ taking examples of C_i with the label $+1$, examples of C_j with the label -1 , and not using examples of the other classes. Separating classes in pairs is normally expected to be an easier job, with the additional advantage that because we use less data, the optimizations will be faster, noting however that we have $\mathcal{O}(K^2)$ discriminants to train instead of $\mathcal{O}(K)$.

ERROR-CORRECTING
OUTPUT CODES

In the general case, both one-vs.-all and pairwise separation are special cases of the *error-correcting output codes* (ECOC) that decompose a multiclass problem to a set of two-class problems (Dietterich and Bakiri 1995) (see also section 17.6). SVMs being two-class classifiers are ideally suited to this (Allwein, Schapire, and Singer 2000), and it is also possible to have an incremental approach where new two-class SVMs are added to better separate pairs of classes that are confused, to ameliorate a poor ECOC matrix (Mayoraz and Alpaydın 1999).

Another possibility is to write a single *multiclass* optimization problem involving all classes (Weston and Watkins 1998):

$$(13.42) \quad \min \frac{1}{2} \sum_{i=1}^K \|\mathbf{w}_i\|^2 + C \sum_i \sum_t \xi_i^t$$

subject to

$$\mathbf{w}_{z^t} \mathbf{x}^t + w_{z^t 0} \geq \mathbf{w}_i \mathbf{x}^t + w_{i0} + 2 - \xi_i^t, \forall i \neq z^t \text{ and } \xi_i^t \geq 0$$

where z^t contains the class index of \mathbf{x}^t . The regularization terms minimize the norms of all hyperplanes simultaneously, and the constraints are there to make sure that the margin between the actual class and any other class is at least 2. The output for the correct class should be at least +1, the output of any other class should be at least -1, and the slack variables are defined to make up any difference.

Though this looks neat, the one-vs.-all approach is generally preferred because it solves K separate N variable problems whereas the multiclass formulation uses $K \cdot N$ variables.

~~13.10~~ Kernel Machines for Regression

Now let us see how support vector machines can be generalized for regression. We see that the same approach of defining acceptable margins, slacks, and a regularizing function that combines smoothness and error is also applicable here. We start with a linear model, and later on we see how we can use kernel functions here as well:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

In regression proper, we use the square of the difference as error:

$$e_2(r^t, f(\mathbf{x}^t)) = [r^t - f(\mathbf{x}^t)]^2$$

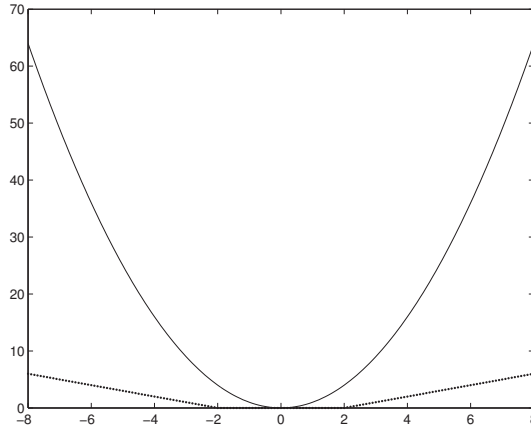


Figure 13.6 Quadratic and ϵ -sensitive error functions. We see that ϵ -sensitive error function is not affected by small errors and also is less affected by large errors and thus is more robust to outliers.

whereas in support vector regression, we use the ϵ -sensitive loss function:

$$(13.43) \quad e_{\epsilon}(r^t, f(\mathbf{x}^t)) = \begin{cases} 0 & \text{if } |r^t - f(\mathbf{x}^t)| < \epsilon \\ |r^t - f(\mathbf{x}^t)| - \epsilon & \text{otherwise} \end{cases}$$

which means that we tolerate errors up to ϵ and also that errors beyond have a linear effect and not a quadratic one. This error function is therefore more tolerant to noise and is thus more *robust* (see figure 13.6). As in the hinge loss, there is a region of no error, which causes sparseness.

ROBUST REGRESSION

Analogous to the soft margin hyperplane, we introduce slack variables to account for deviations out of the ϵ -zone and we get (Vapnik 1995)

$$(13.44) \quad \min \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_t (\xi_+^t + \xi_-^t)$$

subject to

$$\begin{aligned} r^t - (\mathbf{w}^T \mathbf{x} + w_0) &\leq \epsilon + \xi_+^t \\ (\mathbf{w}^T \mathbf{x} + w_0) - r^t &\leq \epsilon + \xi_-^t \\ \xi_+^t, \xi_-^t &\geq 0 \end{aligned}$$

where we use two types of slack variables, for positive and negative deviations, to keep them positive. Actually, we can see this as two hinges

added back to back, one for positive and one for negative slacks. This formulation corresponds to the ϵ -sensitive loss function given in equation 13.43. The Lagrangian is

$$\begin{aligned}
 L_p = & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_t (\xi_+^t + \xi_-^t) \\
 & - \sum_t \alpha_+^t [\epsilon + \xi_+^t - r^t + (\mathbf{w}^T \mathbf{x} + w_0)] \\
 & - \sum_t \alpha_-^t [\epsilon + \xi_-^t + r^t - (\mathbf{w}^T \mathbf{x} + w_0)] \\
 (13.45) \quad & - \sum_t (\mu_+^t \xi_+^t + \mu_-^t \xi_-^t)
 \end{aligned}$$

Taking the partial derivatives, we get

$$(13.46) \quad \frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{w} - \sum_t (\alpha_+^t - \alpha_-^t) \mathbf{x}^t = 0 \Rightarrow \mathbf{w} = \sum_t (\alpha_+^t - \alpha_-^t) \mathbf{x}^t$$

$$(13.47) \quad \frac{\partial L_p}{\partial w_0} = \sum_t (\alpha_+^t - \alpha_-^t) = 0$$

$$(13.48) \quad \frac{\partial L_p}{\partial \xi_+^t} = C - \alpha_+^t - \mu_+^t = 0$$

$$(13.49) \quad \frac{\partial L_p}{\partial \xi_-^t} = C - \alpha_-^t - \mu_-^t = 0$$

The dual is

$$\begin{aligned}
 L_d = & -\frac{1}{2} \sum_t \sum_s (\alpha_+^t - \alpha_-^t) (\alpha_+^s - \alpha_-^s) (\mathbf{x}^t)^T \mathbf{x}^s \\
 (13.50) \quad & -\epsilon \sum_t (\alpha_+^t + \alpha_-^t) + \sum_t r^t (\alpha_+^t - \alpha_-^t)
 \end{aligned}$$

subject to

$$0 \leq \alpha_+^t \leq C, 0 \leq \alpha_-^t \leq C, \sum_t (\alpha_+^t - \alpha_-^t) = 0$$

Once we solve this, we see that all instances that fall in the tube have $\alpha_+^t = \alpha_-^t = 0$; these are the instances that are fitted with enough precision (see figure 13.7). The support vectors satisfy either $\alpha_+^t > 0$ or $\alpha_-^t > 0$ and are of two types. They may be instances that are on the boundary of the tube (either α_+^t or α_-^t is between 0 and C), and we use these to calculate w_0 . For example, assuming that $\alpha_+^t > 0$, we have $r^t = \mathbf{x}^T \mathbf{x}^t + w_0 + \epsilon$. Instances that fall outside the ϵ -tube are of the second type; these are

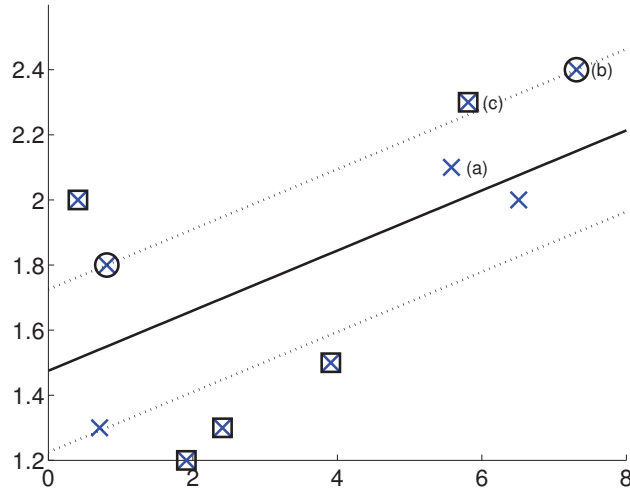


Figure 13.7 The fitted regression line to data points shown as crosses and the ϵ -tube are shown ($C = 10, \epsilon = 0.25$). There are three cases: In (a), the instance is in the tube; in (b), the instance is on the boundary of the tube (circled instances); in (c), it is outside the tube with a positive slack, that is, $\xi_+^t > 0$ (squared instances). (b) and (c) are support vectors. In terms of the dual variable, in (a), $\alpha_+^t = 0, \alpha_-^t = 0$, in (b), $\alpha_+^t < C$, and in (c), $\alpha_+^t = C$.

instances for which we do not have a good fit ($\alpha_+^t = C$), as shown in figure 13.7.

Using equation 13.46, we can write the fitted line as a weighted sum of the support vectors:

$$(13.51) \quad f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = \sum_t (\alpha_+^t - \alpha_-^t) (\mathbf{x}^t)^T \mathbf{x} + w_0$$

Again, the dot product $(\mathbf{x}^t)^T \mathbf{x}^s$ in equation 13.50 can be replaced with a kernel $K(\mathbf{x}^t, \mathbf{x}^s)$, and similarly $(\mathbf{x}^t)^T \mathbf{x}$ be replaced with $K(\mathbf{x}^t, \mathbf{x})$ and we can have a nonlinear fit. Using a polynomial kernel would be similar to fitting a polynomial (figure 13.8), and using a Gaussian kernel (figure 13.9) would be similar to nonparametric smoothing models (section 8.8) except that because of the sparsity of solution, we would not need the whole training set but only a subset.

There is also an equivalent ν -SVM formulation for regression (Schölkopf et al. 2000), where instead of fixing ϵ , we fix ν to bound the fraction of

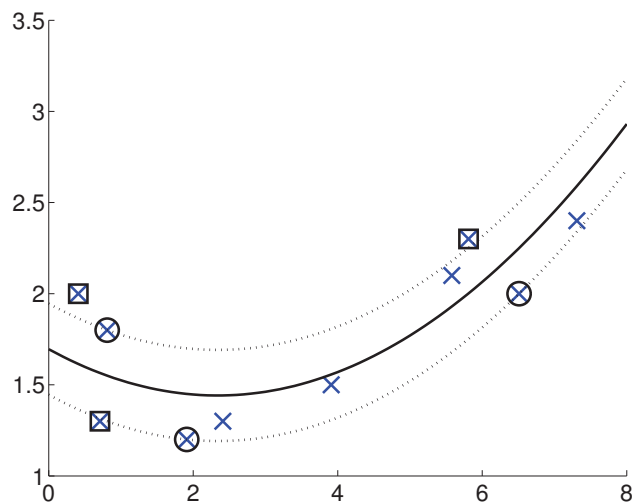


Figure 13.8 The fitted regression line and the ϵ -tube using a quadratic kernel are shown ($C = 10, \epsilon = 0.25$). Circled instances are the support vectors on the margins, squared instances are support vectors which are outliers.

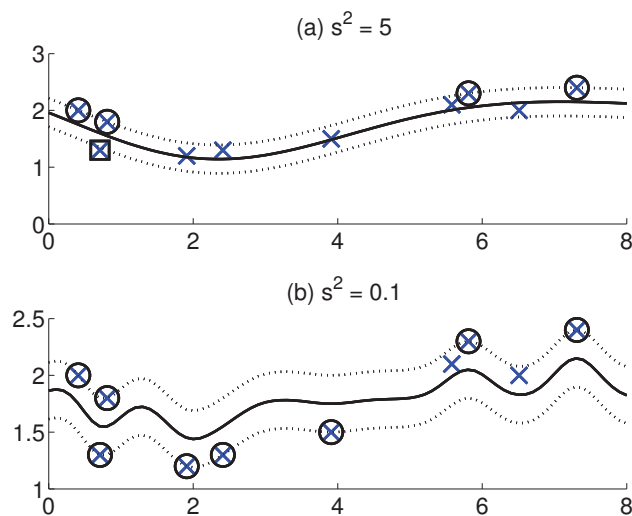


Figure 13.9 The fitted regression line and the ϵ -tube using a Gaussian kernel with two different spreads are shown ($C = 10, \epsilon = 0.25$). Circled instances are the support vectors on the margins, and squared instances are support vectors that are outliers.

support vectors. There is still a need for C though.

~~13.11~~ Kernel Machines for Ranking

Remember that in ranking, we have instances that need to be ordered in a certain way (Liu 2011). For example, we may have pairwise constraints such as $r^u < r^v$ which means that instance \mathbf{x}^u should generate a higher score than \mathbf{x}^v . In section 10.9, we discuss how we can train a linear model for this purpose using gradient descent. We now discuss how we can do the same using support vector machines.

We consider each pairwise constraint as one data instance $t : r^u < r^v$ and minimize

$$(13.52) \quad L_p = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_t \xi^t$$

subject to

$$(13.53) \quad \begin{aligned} \mathbf{w}^T \mathbf{x}^u &\geq \mathbf{w}^T \mathbf{x}^v + 1 - \xi^t, \text{ for each } t : r^u < r^v \\ \xi^t &\geq 0 \end{aligned}$$

Equation 13.53 requires that the score for \mathbf{x}^u be at least 1 unit more than the score for \mathbf{x}^v and hence defines a margin. If the constraint is not satisfied, the slack variable is nonzero and equation 13.52 minimizes the sum of such slacks and the complexity term, which again corresponds to making the width of the margin as large as possible (Herbrich, Obermayer, and Graepel 2000; Joachims 2002). Note that the second term of the sum of slacks is the same as the error used in equation 10.46 except for the 1 unit margin, and the complexity term, as we discussed before, can be interpreted as a weight decay term on the linear model (see section 11.10).

Note that there is one constraint for each pair where an ordering is defined, and hence the number of such constraints is $\mathcal{O}(N^2)$. The constraint of equation 13.53 can also be written as

$$\mathbf{w}^T (\mathbf{x}^u - \mathbf{x}^v) \geq 1 - \xi^t$$

That is, we can view this as a two-class classification of pairwise differences, $\mathbf{x}^u - \mathbf{x}^v$. So by calculating such differences and labeling them as $r^t \in \{-1, +1\}$ depending on whether $r^v < r^u$ or $r^u < r^v$ respectively, any two-class kernel machine can be used to implement ranking. But this is

not the most efficient way to implement, and faster methods have been proposed (Chapelle and Keerthi 2010).

The dual is

$$(13.54) \quad L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s (\mathbf{x}^u - \mathbf{x}^v)^T (\mathbf{x}^k - \mathbf{x}^l)$$

subject to $0 \leq \alpha^t \leq C$. Here, t and s are two pairwise constraints, such as $t : r^u < r^v$ and $s : r^k < r^l$. Solving this, for the constraints that are satisfied, we have $\xi^t = 0$ and $\alpha^t = 0$; for the ones that are satisfied but are in the margin, we have $0 < \xi^t < 1$ and $\alpha^t < C$; and for the ones that are not satisfied (and are misranked), we have $\xi^t > 1$ and $\alpha^t = C$.

For new test instance \mathbf{x} , the score is calculated as

$$(13.55) \quad g(\mathbf{x}) = \sum_t \alpha^t (\mathbf{x}^u - \mathbf{x}^v)^T \mathbf{x}$$

It is straightforward to write the kernelized version of the primal, dual, and score functions, and this is left to the reader (see exercise 7).

13.12 One-Class Kernel Machines

Support vector machines, originally proposed for classification, are extended to regression by defining slack variables for deviations around the regression line, instead of the discriminant. We now see how SVM can be used for a restricted type of unsupervised learning, namely, for estimating regions of high density. We are not doing a full density estimation; rather, we want to find a boundary (so that it reads like a classification problem) that separates volumes of high density from volumes of low density (Tax and Duin 1999). Such a boundary can then be used for *novelty* or *outlier detection*. This is also called *one-class classification*.

OUTLIER DETECTION
ONE-CLASS
CLASSIFICATION

We consider a sphere with center \mathbf{a} and radius R that we want to enclose as much as possible of the density, measured empirically as the enclosed training set percentage. At the same time, trading off with it, we want to find the smallest radius (see figure 13.10). We define slack variables for instances that lie outside (we only have one type of slack variable because we have examples from one class and we do not have any penalty for those inside), and we have a smoothness measure that is proportional to the radius:

$$(13.56) \quad \min R^2 + C \sum_t \xi^t$$

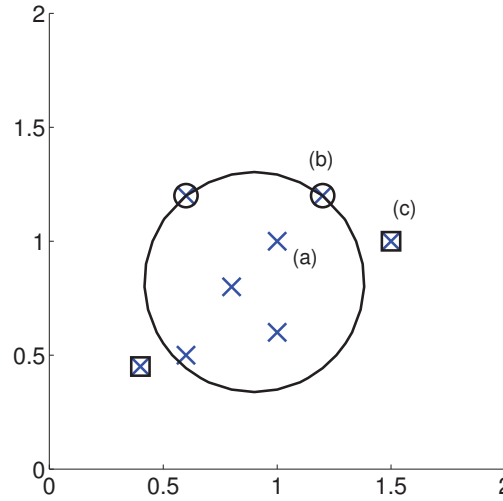


Figure 13.10 One-class support vector machine places the smoothest boundary (here using a linear kernel, the circle with the smallest radius) that encloses as much of the instances as possible. There are three possible cases: In (a), the instance is a typical instance. In (b), the instance falls on the boundary with $\xi^t = 0$; such instances define R . In (c), the instance is an outlier with $\xi^t > 0$. (b) and (c) are support vectors. In terms of the dual variable, we have, in (a), $\alpha^t = 0$; in (b), $0 < \alpha^t < C$; in (c), $\alpha^t = C$.

subject to

$$\|\mathbf{x}^t - \mathbf{a}\|^2 \leq R^2 + \xi^t \text{ and } \xi^t \geq 0, \forall t$$

Adding the constraints, we get the Lagrangian, which we write keeping in mind that $\|\mathbf{x}^t - \mathbf{a}\|^2 = (\mathbf{x}^t - \mathbf{a})^T (\mathbf{x}^t - \mathbf{a})$:

$$(13.57) \quad L_p = R^2 + C \sum_t \xi^t - \sum_t \alpha^t \left(R^2 + \xi^t - [(\mathbf{x}^t)^T \mathbf{x}^t - 2\mathbf{a}^T \mathbf{x}^t + \mathbf{a}^T \mathbf{a}] \right) - \sum_t \gamma^t \xi^t$$

with $\alpha^t \geq 0$ and $\gamma^t \geq 0$ being the Lagrange multipliers. Taking the derivative with respect to the parameters, we get

$$(13.58) \quad \frac{\partial L}{\partial R} = 2R - 2R \sum_t \alpha^t = 0 \Rightarrow \sum_t \alpha^t = 1$$

$$(13.59) \quad \frac{\partial L}{\partial \mathbf{a}} = \sum_t \alpha^t (2\mathbf{x}^t - 2\mathbf{a}) = 0 \Rightarrow \mathbf{a} = \sum_t \alpha^t \mathbf{x}^t$$

$$(13.60) \quad \frac{\partial L}{\partial \xi^t} = C - \alpha^t - \gamma^t = 0$$

Since $\gamma^t \geq 0$, we can write this last as the constraint: $0 \leq \alpha^t \leq C$. Plugging these into equation 13.57, we get the dual that we maximize with respect to α^t :

$$(13.61) \quad L_d = \sum_t \alpha^t (\mathbf{x}^t)^T \mathbf{x}^t - \sum_t \sum_s \alpha^t \alpha^s (\mathbf{x}^t)^T \mathbf{x}^s$$

subject to

$$0 \leq \alpha^t \leq C \text{ and } \sum_t \alpha^t = 1$$

When we solve this, we again see that most of the instances vanish with their $\alpha^t = 0$; these are the typical, highly likely instances that fall inside the sphere (figure 13.10). There are two type of support vectors with $\alpha^t > 0$: There are instances that satisfy $0 < \alpha^t < C$ and lie on the boundary, $\|\mathbf{x}^t - \mathbf{a}\|^2 = R^2$ ($\xi^t = 0$), which we use to calculate R . Instances that satisfy $\alpha^t = C$ ($\xi^t > 0$) lie outside the boundary and are the outliers. From equation 13.59, we see that the center \mathbf{a} is written as a weighted sum of the support vectors.

Then given a test input \mathbf{x} , we say that it is an outlier if

$$\|\mathbf{x} - \mathbf{a}\|^2 > R^2$$

or

$$\mathbf{x}^t \mathbf{x} - 2\mathbf{a}^T \mathbf{x} + \mathbf{a}^T \mathbf{a} > R^2$$

Using kernel functions, allow us to go beyond a sphere and define boundaries of arbitrary shapes. Replacing the dot product with a kernel function, we get (subject to the same constraints):

$$(13.62) \quad L_d = \sum_t \alpha^t K(\mathbf{x}^t, \mathbf{x}^t) - \sum_t \sum_s \alpha^t \alpha^s K(\mathbf{x}^t, \mathbf{x}^s)$$

For example, using a polynomial kernel of degree 2 allows arbitrary quadratic surfaces to be used. If we use a Gaussian kernel (equation 13.30), we have a union of local spheres. We reject \mathbf{x} as an outlier if

$$K(\mathbf{x}, \mathbf{x}) - 2 \sum_t \alpha^t K(\mathbf{x}, \mathbf{x}^t) + \sum_t \sum_s \alpha^t \alpha^s K(\mathbf{x}^t, \mathbf{x}^s) > R^2$$

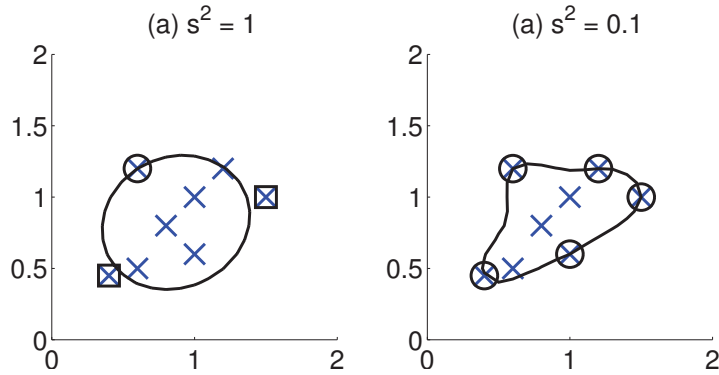


Figure 13.11 One-class support vector machine using a Gaussian kernel with different spreads.

The third term does not depend on \mathbf{x} and is therefore a constant (we use this as an equality to solve for R where \mathbf{x} is an instance on the margin). In the case of a Gaussian kernel where $K(\mathbf{x}, \mathbf{x}) = 1$, the condition reduces to

$$\sum_t \alpha^t K_G(\mathbf{x}, \mathbf{x}^t) < R_c$$

for some constant R_c , which is analogous to the kernel density estimator (section 8.2.2)—except for the sparseness of the solution—with a probability threshold R_c (see figure 13.11).

There is also an alternative, equivalent ν -SVM type of formulation of one-class support vector machines that uses the canonical $(1/2)\|\mathbf{w}\|^2$ type of smoothness (Schölkopf et al. 2001).

~~13.13 Large Margin Nearest Neighbor Classifier~~

In chapter 8, we discussed nonparametric methods where instead of fitting a global model to the data we interpolate from a subset of neighboring instances, and specifically in section 8.6, we covered the importance of using a good distance measure. We now discuss a method to learn a distance measure from the data. Strictly speaking, this is not a kernel machine, but it uses the idea of keeping a margin in ranking, as we noted in section 13.11.

The basic idea is to view k -nearest neighbor classification (section 8.4) as a ranking problem. Let us say the k -nearest neighbors of \mathbf{x}^i contains two instances \mathbf{x}^j and \mathbf{x}^l such that \mathbf{x}^i and \mathbf{x}^j are of the same class and \mathbf{x}^l belongs to another class. In such a case, we want a distance measure such that the distance between \mathbf{x}^i and \mathbf{x}^l is more than \mathbf{x}^i and \mathbf{x}^j . Actually, we not only require that it be more but that there be a one-unit margin between them and if this is not satisfied, we have a slack variable for the difference:

$$\mathcal{D}(\mathbf{x}^i, \mathbf{x}^l) \geq \mathcal{D}(\mathbf{x}^i, \mathbf{x}^j) + 1 - \xi^{ijl}$$

The distance measure works as a score function in a ranking problem, and each $(\mathbf{x}^i, \mathbf{x}^j, \mathbf{x}^l)$ triple defines one ranking constraint as in equation 13.53.

LARGE MARGIN
NEAREST NEIGHBOR

This is the basic idea behind the *large margin nearest neighbor* (LMNN) algorithm (Weinberger and Saul 2009). The error function minimized is

$$(13.63) \quad (1 - \mu) \sum_{i,j} \mathcal{D}(\mathbf{x}^i, \mathbf{x}^j) + \mu \sum_{i,j,l} (1 - y_{il}) \xi_{ijl}$$

subject to

$$(13.64) \quad \begin{aligned} \mathcal{D}(\mathbf{x}^i, \mathbf{x}^l) &\geq \mathcal{D}(\mathbf{x}^i, \mathbf{x}^j) + 1 - \xi^{ijl}, \text{ if } \mathbf{r}^i = \mathbf{r}^j \text{ and } \mathbf{r}^i \neq \mathbf{r}^l \\ \xi^{ijl} &\geq 0 \end{aligned}$$

Here, \mathbf{x}^j is one of the k -nearest neighbors of \mathbf{x}^i and they are of the same class: $\mathbf{r}^i = \mathbf{r}^j$ —it is a *target* neighbor. \mathbf{x}^l is also one of the k -nearest neighbors of \mathbf{x}^i ; if they are of the same label, then y_{il} is set to 1 and we incur no loss; if they are of different classes, then \mathbf{x}^l is an *impostor*, y_{il} is set to 0, and if the condition 13.64 is not satisfied, the slack defines a cost. The second term of equation 13.63 is the sum of such slacks. The first term is the total distance to all target neighbors and minimizing that has an effect of regularization—we want to keep the distances as small as possible.

In LMNN, Mahalanobis distance is used as the distance measure model:

$$(13.65) \quad \mathcal{D}(\mathbf{x}^i, \mathbf{x}^j | \mathbf{M}) = (\mathbf{x}^i - \mathbf{x}^j)^T \mathbf{M} (\mathbf{x}^i - \mathbf{x}^j)$$

and \mathbf{M} matrix is the parameter that is to be optimized. Equation 13.63 defines a convex (more specifically, positive semi-definite) problem and hence has a unique minimum.

When the input dimensionality is high and there are few data, as we discuss in equation 8.21, we can regularize by factoring \mathbf{M} as $\mathbf{L}^T\mathbf{L}$ where \mathbf{L} is $k \times d$ with $k < d$:

$$(13.66) \quad \mathcal{D}(\mathbf{x}^i, \mathbf{x}^j | \mathbf{L}) = \|\mathbf{L}\mathbf{x}^i - \mathbf{L}\mathbf{x}^j\|^2$$

$\mathbf{L}\mathbf{x}$ is the k -dimensional projection of \mathbf{x} , and Mahalanobis distance in the original d -dimensional \mathbf{x} space corresponds to the (squared) Euclidean distance in the new k -dimensional space—see figure 8.7 for an example. If we plug equation 13.66 into equation 13.63 as the distance measure, we get the *large margin component analysis* (LMCA) algorithm (Torresani and Lee 2007); unfortunately, this is no longer a convex optimization problem, and if we use gradient descent, we get a locally optimal solution.

LARGE MARGIN
COMPONENT ANALYSIS

~~13.14 Kernel Dimensionality Reduction~~

We know from section 6.3 that principal components analysis (PCA) reduces dimensionality by projecting on the eigenvectors of the covariance matrix Σ with the largest eigenvalues, which, if data instances are centered ($E[\mathbf{x}] = 0$), can be written as $\mathbf{X}^T\mathbf{X}$. In the kernelized version, we work in the space of $\phi(\mathbf{x})$ instead of the original \mathbf{x} and because, as usual, the dimensionality d of this new space may be much larger than the dataset size N , we prefer to work with the $N \times N$ matrix $\mathbf{X}\mathbf{X}^T$ and do feature embedding instead of working with the $d \times d$ matrix $\mathbf{X}^T\mathbf{X}$. The projected data matrix is $\Phi = \phi(\mathbf{X})$, and hence we work with the eigenvectors of $\Phi^T\Phi$ and hence the kernel matrix \mathbf{K} .

KERNEL PCA

Kernel PCA uses the eigenvectors and eigenvalues of the kernel matrix and this corresponds to doing a linear dimensionality reduction in the $\phi(\mathbf{x})$ space. When \mathbf{c}_i and λ_i are the corresponding eigenvectors and eigenvalues, the projected new k -dimensional values can be calculated as

$$\mathbf{z}_j^t = \sqrt{\lambda_j} \mathbf{c}_j^t, j = 1, \dots, k, t = 1, \dots, N$$

An example is given in figure 13.12 where we first use a quadratic kernel and then decrease dimensionality to two (out of five) using kernel PCA and implement a linear SVM there. Note that in the general case (e.g., with a Gaussian kernel), the eigenvalues do not necessarily decay and there is no guarantee that we can reduce dimensionality using kernel PCA.

What we are doing here is multidimensional scaling (section 6.7) using kernel values as the similarity values. For example, by taking $k = 2$,

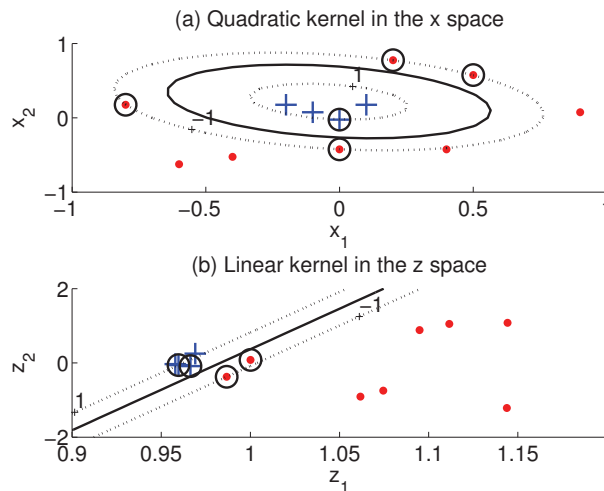


Figure 13.12 Instead of using a quadratic kernel in the original space (a), we can use kernel PCA on the quadratic kernel values to map to a two-dimensional new space where we use a linear discriminant (b); these two dimensions (out of five) explain 80 percent of the variance.

one can visualize the data in the space induced by the kernel matrix, which can give us information as to how similarity is defined by the used kernel. Linear discriminant analysis (LDA) (section 6.8) can similarly be kernelized (Müller et al. 2001). The kernelized version of canonical correlation analysis (CCA) (section 6.9) is discussed in Haroon, Szedmak, Shawe-Taylor 2004.

In chapter 6, we discussed nonlinear dimensionality reduction methods, Isomap and LLE. In fact, by viewing the elements of the cost matrix in equation 6.58 as kernel evaluations for pairs of inputs, LLE can be seen as kernel PCA for a particular choice of kernel. The same also holds for Isomap when a kernel function is defined as a function of the geodesic distance on the graph.

13.15 Notes

The idea of generalizing linear models by mapping the data to a new space through nonlinear basis functions is old, but the novelty of sup-

DUAL
REPRESENTATION

port vector machines is that of integrating this into a learning algorithm whose parameters are defined in terms of a subset of data instances (the so-called *dual representation*), hence also without needing to explicitly evaluate the basis functions and thereby also limiting complexity by the size of the training set; this is also true for Gaussian processes where the kernel function is called the covariance function (section 16.9).

The sparsity of the solution shows the advantage over nonparametric estimators, such as k -nearest neighbor and Parzen windows, or Gaussian processes, and the flexibility to use kernel functions allows working with nonvectorial data. Because there is a unique solution to the optimization problem, we do not need any iterative optimization procedure as we do in neural networks. Because of all these reasons, support vector machines are now considered to be the best, off-the-shelf learners and are widely used in many domains, especially bioinformatics (Schölkopf, Tsuda, and Vert 2004) and natural language processing applications, where an increasing number of tricks are being developed to derive kernels (Shawe-Taylor and Cristianini 2004).

The use of kernel functions implies a different data representation; we no longer define an instance (object/event) as a vector of attributes by itself, but in terms of how it is similar to or differs from other instances; this is akin to the difference between multidimensional scaling that uses a matrix of distances (without any need to know how they are calculated) and principal components analysis that uses vectors in some space.

The support vector machine is currently considered to be the best off-the-shelf learning algorithm and has been applied successfully in various domains. The fact that we are solving a convex problem and hence optimally and the idea of kernels that allow us to code our prior information has made it quite popular. There is a huge literature on the support vector machine and all types of kernel machines. The classic books are by Vapnik (1995, 1998) and Schölkopf and Smola (2002). Burges 1998 and Smola and Schölkopf 1998 are good tutorials on SVM classification and regression, respectively. Many free software packages are also available, and the ones that are most popular are SVMlight (Joachims 2008) and LIBSVM (Chang and Lin 2011).

13.16 Exercises

1. Propose a filtering algorithm to find training instances that are very unlikely to be support vectors.

SOLUTION: Support vectors are those instances that are close to the boundaries. So if there is an instance surrounded by a large number of instances all of the same class, it will very probably not be chosen as a support vector. So, for example, we can do an 11-nearest neighbor search for all instances and if all its 11 neighbors are of the same class, we can prune that instance from the training set.

2. In equation 13.31, how can we estimate \mathbf{S} ?

SOLUTION: We can calculate the covariance matrix of the data and use that as \mathbf{S} . Another possibility is to have a local \mathbf{S}^t for each support vector, and we can use a number of neighborhood data points to estimate it; we may need to take measures in such a case to make sure that \mathbf{S} is not singular or decrease dimensionality in some way.

3. In the empirical kernel map, how can we choose the templates?

SOLUTION: The easiest and most frequently used approach is to use all the training instances, and in such a case $\boldsymbol{\phi}(\cdot)$ is N -dimensional. We can decrease complexity and make the model more efficient by choosing a subset; we can use a randomly chosen subset, do some clustering, and use the cluster centers as templates (as in vector quantization), or use a subset that covers the input space well using as few instances as possible.

4. In the localized multiple kernel of equation 13.40, propose a suitable model for $\eta_i(\mathbf{x}|\theta_i)$ and discuss how it can be trained.
5. In kernel regression, what is the relation, if any, between ϵ and noise variance?
6. In kernel regression, what is the effect of using different ϵ on bias and variance?

SOLUTION: ϵ is a smoothing parameter. When it is too large, we smooth too much, which reduces variance but risks increasing bias. If it is too small, the variance may be large and bias would be small.

7. Derive the kernelized version of the primal, dual, and the score functions for ranking..

SOLUTION: The primal is

$$L_p = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_t \xi^t$$

subject to

$$\begin{aligned} \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}^u - \mathbf{x}^v) &\geq 1 - \xi^t \\ \xi^t &\geq 0 \end{aligned}$$

The dual is

$$L_d = \sum_t \alpha^t - \frac{1}{2} \sum_t \sum_s \alpha^t \alpha^s K(\mathbf{x}^u - \mathbf{x}^v, \mathbf{x}^k - \mathbf{x}^l)$$

where $K(\mathbf{x}^u - \mathbf{x}^v, \mathbf{x}^k - \mathbf{x}^l) = \boldsymbol{\phi}(\mathbf{x}^u - \mathbf{x}^v)^T \boldsymbol{\phi}(\mathbf{x}^k - \mathbf{x}^l)$.

For new test instance \mathbf{x} , the score is calculated as

$$g(\mathbf{x}) = \sum_t \alpha^t K(\mathbf{x}^u - \mathbf{x}^v, \mathbf{x})$$

8. How can we use one-class SVM for classification?

SOLUTION: We can use a separate one-class SVM for each class and then combine them to make a decision. For example, for each class C_i , we fit a one-class SVM to find parameters α_i^t :

$$\sum_t \alpha_i^t K_G(\mathbf{x}, \mathbf{x}^t)$$

and this then can be taken as an estimator for $p(\mathbf{x}|C_i)$. If the priors are more or less equal, we can simply choose the class having the largest value; otherwise we can use Bayes' rule for classification.

9. In a setting such as that in figure 13.12, use kernel PCA with a Gaussian kernel.
10. Let us say we have two representations for the same object and associated with each, we have a different kernel. How can we use both to implement a joint dimensionality reduction using kernel PCA?

13.17 References

- Allwein, E. L., R. E. Schapire, and Y. Singer. 2000. "Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers." *Journal of Machine Learning Research* 1:113-141.
- Burges, C. J. C. 1998. "A Tutorial on Support Vector Machines for Pattern Recognition." *Data Mining and Knowledge Discovery* 2:121-167.
- Chang, C.-C., and C.-J. Lin. 2011. *LIBSVM: A Library for Support Vector Machines*. *ACM Transactions on Intelligent Systems and Technology* 2: 27:1-27:27.
- Chapelle, O., and S. S. Keerthi. 2010. "Efficient Algorithms for Ranking with SVMs." *Information Retrieval* 11:201-215.
- Cherkassky, V., and F. Mulier. 1998. *Learning from Data: Concepts, Theory, and Methods*. New York: Wiley.
- Cortes, C., and V. Vapnik. 1995. "Support Vector Networks." *Machine Learning* 20:273-297.

- Dietterich, T. G., and G. Bakiri. 1995. "Solving Multiclass Learning Problems via Error-Correcting Output Codes." *Journal of Artificial Intelligence Research* 2: 263–286.
- Gönen, M., and E. Alpaydm. 2008. "Localized Multiple Kernel Learning." In *25th International Conference on Machine Learning*, ed. A. McCallum and S. Roweis, 352–359. Madison, WI: Omnipress.
- Gönen, M., and E. Alpaydm. 2011. "Multiple Kernel Learning Algorithms." *Journal of Machine Learning Research* 12:2211–2268.
- Hardoon, D. R., S. Szedmak, J. Shawe-Taylor. 2004. "Canonical Correlation Analysis: An Overview with Application to Learning Methods." *Neural Computation* 16:2639–2664.
- Herbrich, R., K. Obermayer, and T. Graepel. 2000. "Large Margin Rank Boundaries for Ordinal Regression." In *Advances in Large Margin Classifiers*, ed. A. J. Smola, P. Bartlett, B. Schölkopf and D. Schuurmans, 115–132. Cambridge, MA: MIT Press.
- Jaakkola, T., and D. Haussler. 1999. "Exploiting Generative Models in Discriminative Classifiers." In *Advances in Neural Information Processing Systems 11*, ed. M. J. Kearns, S. A. Solla, and D. A. Cohn, 487–493. Cambridge, MA: MIT Press.
- Joachims, T. 2002. "Optimizing Search Engines using Clickthrough Data." In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 133–142. New York, NY: ACM.
- Joachims, T. 2008. *SVMLight*, <http://svmlight.joachims.org>.
- 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.
- Liu, T.-Y. 2011. *Learning to Rank for Information Retrieval*. Heidelberg: Springer.
- Mayoraz, E., and E. Alpaydm. 1999. "Support Vector Machines for Multiclass Classification." In *Foundations and Tools for Neural Modeling, Proceedings of IWANN'99, LNCS 1606*, ed. J. Mira and J. V. Sanchez-Andres, 833–842. Berlin: Springer.
- Müller, K. R., S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. 2001. "An Introduction to Kernel-Based Learning Algorithms." *IEEE Transactions on Neural Networks* 12:181–201.
- Noble, W. S. 2004. "Support Vector Machine Applications in Computational Biology." In *Kernel Methods in Computational Biology*, ed. B. Schölkopf, K. Tsuda, and J.-P. Vert, 71–92. Cambridge, MA: MIT Press.

- Platt, J. 1999. "Probabilities for Support Vector Machines." In *Advances in Large Margin Classifiers*, ed. A. J. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, 61–74. Cambridge, MA: MIT Press.
- Schölkopf, B., J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. 2001. "Estimating the Support of a High-Dimensional Distribution." *Neural Computation* 13:1443–1471.
- Schölkopf, B., and A. J. Smola. 2002. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press.
- Schölkopf, B., A. J. Smola, R. C. Williamson, and P. L. Bartlett. 2000. "New Support Vector Algorithms." *Neural Computation* 12:1207–1245.
- Schölkopf, B., K. Tsuda, and J.-P. Vert, eds. 2004. *Kernel Methods in Computational Biology*. Cambridge, MA: MIT Press.
- Shawe-Taylor, J., and N. Cristianini. 2004. *Kernel Methods for Pattern Analysis*. Cambridge, UK: Cambridge University Press.
- Smola, A., and B. Schölkopf. 1998. *A Tutorial on Support Vector Regression*, NeuroCOLT TR-1998-030, Royal Holloway College, University of London, UK.
- Sonnenburg, S., G. Rätsch, C. Schäfer, and B. Schölkopf. 2006. "Large Scale Multiple Kernel Learning." *Journal of Machine Learning Research* 7:1531–1565.
- Tax, D. M. J., and R. P. W. Duin. 1999. "Support Vector Domain Description." *Pattern Recognition Letters* 20:1191–1199.
- Torresani, L., and K. C. Lee. 2007. "Large Margin Component Analysis." In *Advances in Neural Information Processing Systems 19*, ed. B. Schölkopf, J. Platt, and T. Hoffman, 1385–1392. Cambridge, MA: MIT Press.
- Vapnik, V. 1995. *The Nature of Statistical Learning Theory*. New York: Springer.
- Vapnik, V. 1998. *Statistical Learning Theory*. New York: Wiley.
- Vert, J.-P., K. Tsuda, and B. Schölkopf. 2004. "A Primer on Kernel Methods." In *Kernel Methods in Computational Biology*, ed. B. Schölkopf, K. Tsuda, and J.-P. Vert, 35–70. Cambridge, MA: MIT Press.
- Weinberger, K. Q., and L. K. Saul. 2009. "Distance Metric Learning for Large Margin Classification." *Journal of Machine Learning Research* 10:207–244.
- Weston, J., and C. Watkins. 1998. "Multiclass Support Vector Machines." *Technical Report CSD-TR-98-04*, Department of Computer Science, Royal Holloway, University of London.