CHAPTER

5 Support vector machines

Support vector machines (SVMs) are a variant of linear models that attempt to explicitly identify a decision function that places the training data as far as possible from the decision boundary. Such models are sometimes called "large margin" learners, since they encourage a margin of safety around the decision boundary. We first derive linear SVMs for linearly separable data sets, and discuss optimization of the resulting learning problem. We then relax the separability requirement and discuss the resulting "soft margin" SVM. Finally, we show how the linear SVM formulation can be extended to provide nonlinear and even nonparametric models using a technique called the *kernel trick*.

5.1 Linear SVMs

Suppose that we have a data set $D = \{x^{(i)}, y^{(i)}\}$, with $y^{(i)} \in \{-1, +1\}$ a binary class variable (class negative versus positive). If D is linearly separable, then there exists a linear classifier that correctly classifies the data points, i.e., the linear response $r(x;\theta) = x \odot \theta^T = \theta_0 + \theta_1 x_1 + \ldots$ has the correct sign on all the training data: $r(x^{(i)};\theta) > 0 \Leftrightarrow y^{(i)} > 0$. Typically, if D is linearly separable, there will be many such linear classifiers. (We postpone discussing what we can do if the data are not linearly separable to Section 5.3.) One method to choose among these linear decision boundaries is to select one that places each $x^{(i)}$ as far away from the decision boundary as possible, i.e., maximizes the distance of the point $x^{(i)}$ that is *closest* to some point x on the decision boundary.

We illustrate this point in Figure 5.1. The data set in the figure is linearly separable; there are many hyperplanes (lines, for 2D data) that separate the data from the two classes, including the two shown in the figure. In addition to the decision boundary (solid line), we also show the distance to the nearest data point(s), illustrated by the parallel dashed lines. Intuitively, this *margin* captures the stability of our decisions to small changes in the data: if the margin is large (no data are near the decision boundary), new data points that are close to the training data will also be correctly classified. On the other hand, if the margin is small, new data points have more chance of falling on the other side of the boundary and being misclassified.

More formally, we can define the margin $\mathcal{M}$ as,

$$\mathcal{M}(\theta) = 2 \min_i \min_{x: x\odot\theta^T = 0} \|x^{(i)} - x\|$$

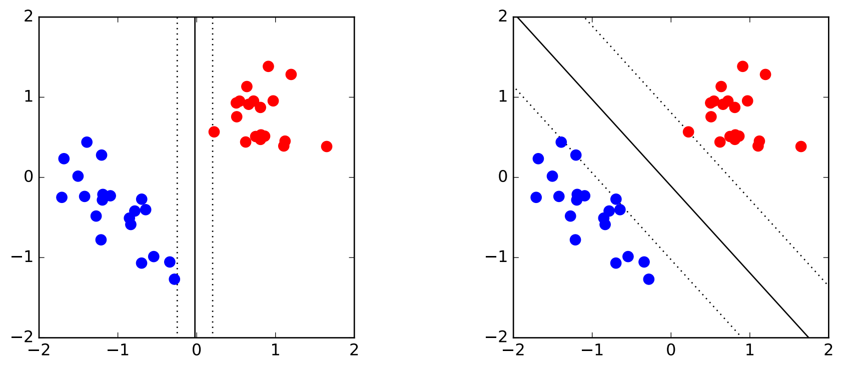


Figure 5.1: Margin of a linear classifier. For a linear hyperplane that separates (correctly classifies) all the data, we can measure the *margin*, or closest distance of any data point to the decision boundary. Intuitively, we would like the margin to be large, as this means that data near the training points will also be classified correctly.

and the linear classifier with the largest margin as

$$\theta_{svm} = \arg\max_{\theta}\ \mathcal{M}(\theta) \qquad \text{s.t.} \quad \forall i, \quad \text{sign}(x^{(i)} \odot \theta^T) = y^{(i)}. \tag{5.1}$$

As framed, this objective function does not seem like an easy thing to optimize; it involves minimizing and maximizing together, over both continuous-valued vectors (x, θ) and discrete data indices (i). However, we can reframe the same task more carefully in a much more convenient form.

A key observation is that the decision boundary, $\{x : x \odot \theta^T = 0\}$, is scale invariant to θ – if we multiply θ by a constant, its decision boundary does not change. Let us use this scale, then, to try to capture the margin's size. Instead of simply requiring that the response $r(x;\theta)$ have the correct sign, let us also require that it be "sufficiently large", i.e.,

$$\begin{aligned} y^{(i)} &= +1 \Rightarrow (x^{(i)} \odot \theta^T) \geq +1 \\ y^{(i)} &= -1 \Rightarrow (x^{(i)} \odot \theta^T) \leq -1 \end{aligned} \tag{5.2}$$

Given any θ that correctly classifies all the training points, so that the decision boundary hyperplane $\{x : x \odot \theta^T = 0\}$ lies between the positive and negative points, we can simply scale θ to ensure that these margin conditions hold, so that there are also no data between the hyperplanes defined by $x \odot \theta^T = +1$ and $x \odot \theta^T = -1$. The "margin constraints" in (5.2) thus define a band in the feature space in which no data are allowed to lie. Thus, the optimization in (5.1) can be equivalently framed as

$$\theta_{svm} = \arg\max_{\theta}\ \mathcal{M}_{\pm 1}(\theta) \qquad \text{s.t.} \quad \forall i, \quad y^{(i)}\,(x^{(i)} \odot \theta^T) \geq 1. \tag{5.3}$$

where

$$\mathcal{M}_{\pm 1}(\theta) = \min_{x^+, x^-} \|x^+ - x^-\| \qquad \text{s.t.} \quad x^+ \odot \theta^T = +1, \quad x^- \odot \theta^T = -1$$

is the distance between the $+1$ and -1 hyperplanes, and the positive and negative margin constraints have been combined into a single form for convenience.

Next, let us see that $\mathcal{M}_{\pm 1}$ can be expressed succinctly in terms of the parameter values of θ. For convenience, we will also make a notational change for our previous linear classifier form, explicitly breaking out the constant term and using the parameters w and b. In this notation, our linear response is expressed as,

$$r(x) = b + x \odot w^T.$$

The constant term, b, is sometimes called the *bias* term[1], and the entries of the vector w are called the *weights*, one per feature.

Now let us measure the distance $\mathcal{M}_{\pm 1}$ between the ± 1 hyperplanes. Let $x^{(-)}$ be any point with $r(x^{(-)}) = -1$, and $x^{(+)}$ be the point that is as near as possible to $x^{(-)}$ with $r(x^{(+)}) = +1$. It is easy to see that the vector $x^{(+)} - x^{(-)}$ is a multiple of w, since w is perpendicular to the decision boundary $x \odot w^T + b = 0$; if two points x, x' are on the boundary, then

$$x \odot w^T + b = 0 \quad \text{and} \quad x' \odot w^T + b = 0 \qquad \Rightarrow \qquad (x - x') \odot w = 0$$

i.e., w is perpendicular to the decision boundary, and thus to the ± 1 hyperplanes as well. So,

$$x^{(+)} \odot w^T + b = +1 \qquad x^{(-)} \odot w^T + b = -1 \qquad x^{(+)} - x^{(-)} = \gamma w$$

for some value of γ. Rearranging, we can solve for the scalar γ and see that the margin width is

$$\mathcal{M}_{\pm 1} = \frac{2}{\|w\|} = 2\Big(\sum_j w_j^2\Big)^{-\frac{1}{2}}$$

This frames the maximum margin classifier as a problem of finding the value of $\theta = (w, b)$ with the largest value of $1/\|w\|$ that still satisfies the margin constraints. Since we are interested in the values of (w, b) that attain the maximum margin, and not in the value of the objective itself, we can instead equivalently find the value of (w, b) that *minimize* $\|w\|$ subject to the constraints, or,

$$\theta^* = \arg \min_{\theta=(w,b)} \sum_j w_j^2 \qquad \text{subject to} \qquad y^{(i)}(b + x^{(i)} \odot w^T) \geq +1 \quad \forall i \tag{5.4}$$

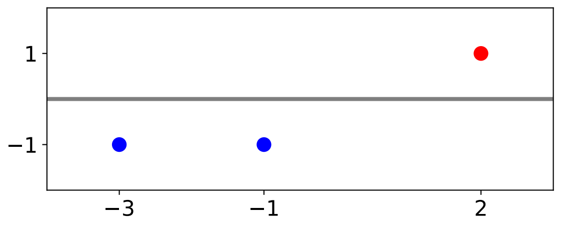
We note again that this expression of the constraints in (5.4) assumes $y^{(i)} \in \{-1, +1\}$.

The optimization (5.4) is a special type of optimization called a *quadratic program*, in which we optimize a quadratic function subject to linear constraints. Quadratic programs are highly studied in the field of optimization, with many different types of algorithms to solve them efficiently.

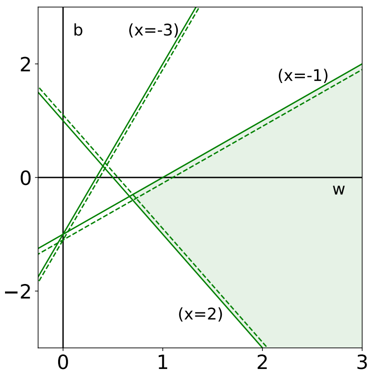
Example 5-1 : Toy SVM

We can illustrate the constrained optimization problem of an SVM on a simple problem consisting of three data points with a single (scalar) feature x. Each data point $x^{(i)}$ constrains the values of w and b, since the response $x^{(i)} \odot w^T + b$ must have the correct sign, $y^{(i)}$, and sufficient magnitude. For example, a data point $x = -1$, $y = -1$ induces the constraint $w(-1) + b \leq -1$, or equivalently, $b \leq w - 1$.

[1]Not to be confused with statistical bias, as arises in under- and over-fitting.



(a) 1-D Data



(b) SVM Constraints

Our three observed data points are shown at left, located at $x = -3$, $x = -1$, and $x = 2$, with $y = \pm 1$. This induces three constraints on w and b, e.g., $wx + b \leq -1$ for $x = -3$, and so on, shown at right; labels show which constraint is associated with which data point, while dashed lines show on which side of each constraint (w, b) must lie. The valid (w, b) (satisfying all constraints) are thus in the shaded area. Finally, within the shaded area, the value with the largest margin (and thus the smallest value $\|w\|$) is the leftmost corner of the shaded region, at $w = \frac{2}{3}$, $b = -\frac{1}{3}$.

We can also see that this point exactly satifies two of the constraints, so that the rightmost negative data point has response $r(x = -1) = -1$, the positive data point has response $r(x = 2) = +1$, and the decision boundary is at the midpoint of this region between the classes, at $x = \frac{1}{2}$. If either of these two constraints were removed from the optimization, we would be able to reduce $\|w\|$ further; thus, these constraints are called *active* at the solution, and their associated data points are called the *support vectors* (for reasons discussed in the sequel). In contrast, the third data point's constraint is not active; removing this data point and its associated constraint would not change the solution.

5.2 Lagrangian Optimization and Duality

Although the optimization (5.4) has a simple objective function and simple linear constraints, it is not easy to optimize directly. To see why, consider an algorithm that follows the gradient of the objective until it runs into one of the constraints. To initialize such an algorithm, we would need to select a value for (w, b) that satisfies the constraints – or, in other words, a linear classifier that correctly separates the data. This is nontrivial – just finding such a classifier was the entire purpose of the perceptron algorithm!

A more practical approach is to use the Lagrangian technique to solve for θ. Lagrangian optimization is a powerful framework for solving constrained optimization problems. We will denote the optimization (5.4) as the *primal* optimization problem, and define a related optimization problem called the *Lagrangian* in which the constraints have been incorporated into the objective function itself, scaled by coefficients α termed the *Lagrange multipliers*.

Lagrangian Optimization

Consider a generic constrained optimization problem with inequality constraints,

$$\theta^* = \arg\min_{\theta} f(\theta) \qquad \text{s.t.} \qquad \forall_i \;\; g_i(\theta) \leq 0 \tag{5.5}$$

where $f(\theta)$ is the objective and the constraints are given by the $\{g_i\}$; here f and the g_i are assumed to be continuous and differentiable. We can define an equivalent optimization problem as,

$$\theta^* = \arg\min_\theta f_\infty(\theta) \qquad\qquad f_\infty(\theta) = \begin{cases} f(\theta) & \forall_i\, g_i(\theta) < 0 \\ \infty & \text{otherwise} \end{cases} \tag{5.6}$$

We then re-define f_∞ using its own optimization problem,

$$f_\infty(\theta) = \max_{\alpha \geq 0} f(\theta) + \sum_i \alpha_i\, g_i(\theta).$$

Examining f_∞, we see that at any point θ that satisfies all $g_i(\theta) \leq 0$, the maximization over α will ensure that $f + \sum_i \alpha_i\, g_i = f$, since if $g_i < 0$ the maximum over α_i will be attained by $\alpha_i = 0$. On the other hand, if θ does *not* satisfy some constraint g_i, then the optimal value of $\alpha_i \to \infty$, and $f_\infty(\theta) = \infty$. Thus the constrained optimization of $f(\theta)$ can be reframed as a minimax optimization problem of θ and α,

$$f(\theta^*) = \min_\theta f_\infty(\theta) = \min_\theta \max_{\alpha \geq 0} f(\theta) + \sum_i \alpha_i\, g_i(\theta). \tag{5.7}$$

The α_i are termed the *Lagrange multipliers*, with one parameter per constraint of the original form. Our new framing has only simple (nonnegativity) constraints, and we can optimize it using local, gradient-based search. As we attempt to decrease the objective with respect to θ, if we begin to violate some constraint g_i, our gradient steps will increase the corresponding α_i, increasing the objective function and making that value of θ less appealing.

This intuition is formalized by the Karush-Kuhn-Tucker or KKT conditions, which state that any saddle point of the Lagrangian (5.7) corresponds to a local optimum of the original constrained optimization (5.5). Additionally, at such points we have that $\alpha_i\, g_i(\theta) = 0$; this condition is called *complementary slackness* since it means that $\alpha_i > 0$ (i.e., its non-negativity constraint is not active) if and only if $g_i(\theta) = 0$ (i.e., the constraint g_i is active).

To apply this framework to the margin optimization in (5.4), we can take

$$f(w, b) = \sum_j w_j^2 \qquad\qquad g_i(\theta) = 1 - y^{(i)}\,(b + x^{(i)} \odot w^T)$$

It is then easy to search for a saddle point using local gradient updates. We can initialize θ arbitrariliy, and $\alpha = 0$, and then update each by moving along the gradient direction to decrease or increase the objective, respectively.

Lagrangian Duality

A closely related optimization problem is the *dual* optimziation problem, given by reversing the order of the min and max operations in the Lagrangian (5.7). This gives a lower bound on the objective:

$$\begin{aligned} \min_\theta f_\infty(\theta) &= \min_\theta \max_{\alpha \geq 0} f(\theta) + \sum_i \alpha_i\, g_i(\theta) \\ &\geq \max_{\alpha \geq 0} \min_\theta f(\theta) + \sum_i \alpha_i\, g_i(\theta) \quad = \quad \max_{\alpha \geq 0} \tilde{f}(\alpha) \end{aligned} \tag{5.8}$$

The dual objective $\tilde{f}$ is now a function of only the Lagrange multipliers α.

In the case of the margin optimization (5.4), for fixed α the minimization over θ is a simple, unconstrained quadratic, and can thus be solved in closed form by taking the derivative, setting it equal to zero, and solving. Doing this for the weights w gives,

$$w^* = \sum_i \alpha_i \, y^{(i)} \, x^{(i)}, \tag{5.9}$$

while the derivative with respect to the bias b will be zero if and only if we enforce the constraint, $\sum_i \alpha_i \, y^{(i)} = 0$.

Notice that the solution to w^* expresses the weights as a linear combination of data points $x^{(i)}$, multiplied by α_i. This again illustrates the point that the solution depends only on the data points where $\alpha_i \neq 0$, which are also the points where the margin constraint g_i is enforced with equality ($g_i(\theta) = 0$), i.e., the support vectors. Once we have solved for w^*, we can compute the value of b^* using this relation, since for any data point with $\alpha_i > 0$ we can solve the margin constraint for b, e.g., $b = y^{(i)} - w^* \odot x^{(i)}$.

The Lagrangian dual form gives an optimization problem over only the α_i, i.e., over m variables, with relatively simple constraints (positivity of the α_i, and an equality constraint):

$$\max_{\alpha \geq 0} \tilde{f}(\alpha) = \max_{\alpha \geq 0} \sum_i \Big[\alpha_i - \frac{1}{2} \sum_j \alpha_i \, \alpha_j \, y^{(i)} \, y^{(j)} \, x^{(i)} \odot x^{(j)} \Big] \qquad \text{s.t.} \quad \sum_i \alpha_i \, y^{(i)} = 0$$

where $x^{(i)} \odot x^{(j)}$ is the inner product between data points i and j. Like the primal optimization (5.4), it is a quadratic program (quadratic objective with linear constraints). However, this dual quadratic program may be easier to solve than the original, primal form, particularly if the data x are few and high dimensional ($m \ll n$). We will see how to use this form to our advantage in Section 5.4.

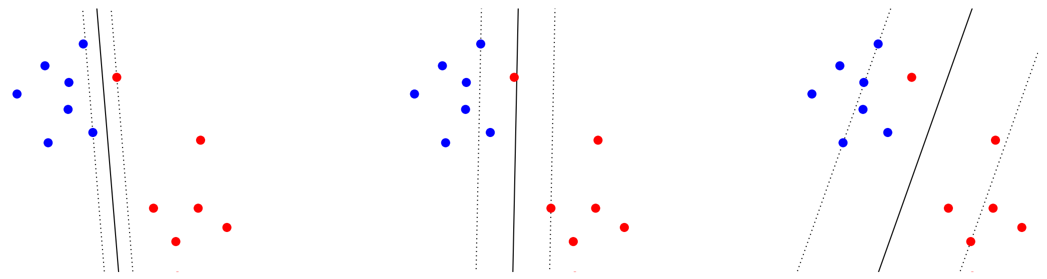
Lagrangian duality is a general approach that allows us to construct a convex lower bound to the original constrained optimization problem, even when that original (primal) problem is non-convex. In this case, the optimal solutions may not coincide, and there may be a *duality gap*, i.e., $\min_\theta f_\infty(\theta) - \max_\alpha \tilde{f}(\alpha) > 0$. We use the term *strong duality* to indicate when this gap is zero, meaning that the solutions to the two problems are equivalent. For convex primal problems, strong duality is guaranteed to hold under a number of conditions. Of particular relevance to the quadratic program that defines the support vector machine model (a convex objective with linear constraints), a variant of Slater's condition guarantees strong duality [Boyd and Vandenberghe, 2004] if a solution to the primal form exists.

5.3 Soft Margin SVMs

What can we do about data that are not linearly separable, so that there exist no values of θ that satisfy the margin constraints? Conceptually, we might like to optimize the margin of "most" of the data, while allowing a small number of data to violate the constraints. Inherently, this involves some trade-off between the two desiderata: if we allow more data to violate the margin, we can by definition increase the margin's width. We therefore need to balance two different costs.

The standard way of doing this for SVMs is called the *soft-margin* SVM, and balances the size of the margin, as measured by $\|w\|^2$, with the degree to which the data violate this margin. To measure the latter, we introduce an additional "slack" variable $\epsilon^{(i)}$ for each data point into the optimization. We allow data point i to violate its margin constraint by the amount $\epsilon^{(i)}$, but penalize this in the objective function:

$$\theta^* = \arg \min_{\theta=(w,b)} \min_{\epsilon} \sum_j w_j^2 + R \sum_i \epsilon^{(i)} \qquad \text{subject to} \quad \begin{aligned} & y^{(i)}(b + x^{(i)} \odot w^T) \geq 1 - \epsilon^{(i)}, \\ & \epsilon^{(i)} \geq 0 \end{aligned} \tag{5.10}$$



(a) $R = R_0$ (b) $R = R_0 \cdot 10^{-2}$ (c) $R = R_0 \cdot 10^{-4}$

Figure 5.2: Soft-margin linear SVM example. (a) Large values of R mimic the hard-margin SVM, which maximizes the margin (dashed lines) subject to separating the training data. (b)–(c) As R decreases, the solution emphasizes a larger "margin", paying a cost for each data point that violates it. This can allow the learner to ignore outliers like the single red point and select boundaries further from the other data.

The constant R determines the relative balance between the cost of violating the margins, $\sum_i \epsilon^{(i)}$, and the (inverse squared) width of the margin, $\sum_j w_j^2$. When R is very large, as few data as possible with be allowed to violate the margin; if R is very small, more data will violate a wider margin.

This general effect is illustrated in Figure 5.2. At a relatively large value, R_0, the optimization prefers a decision boundary with a small margin an no violations, equivalent to the hard-margin SVM found previously. As R decreases, the margin is wider, with more data falling inside the ± 1 band, or even on the wrong side of the decision boundary; these data have $\epsilon^{(i)} > 0$.

The Hinge Loss

A useful side effect of the soft-margin SVM formulation is that it is now easy to initialize θ and ϵ to satisfy the constraints. Although finding a value of θ that separates the data is difficult, given any value of θ it is possible to choose a value of ϵ to satisfy the constraints in (5.10), by taking

$$\epsilon^{(i)} = \max\left[0, 1 - y^{(i)}(b + x^{(i)} \odot w^T)\right] \quad := \quad J^{(i)}_{\text{hinge}}(\theta = (w, b))$$

Returning to (5.10), plugging in this value of ϵ, and dividing the objective by R gives the optimization,

$$\theta^* = \arg \min_{\theta=(w,b)} \frac{1}{R} \sum_j w_j^2 + \sum_i J^{(i)}_{\text{hinge}}(\theta), \tag{5.11}$$

which looks like an optimization of a decomposible loss function $J^{(i)}$ with an ℓ_2 regularization over the weights w. From this perspective, $\frac{1}{R}$ determines the amount of regularization; when R is small, we regularize the model more strongly, increasing its training error but reducing its propensity for overfitting (see again illustration in Figure 5.2).

5.4 The Kernel Trick

We saw with earlier linear models (linear regression, perceptrons) that we could increase the flexibility and representational capacity of our model by artificially expanding the features of the data using a *feature transform* $\Phi(x)$, e.g.,

$$\hat{y}(x) = \text{sign}(\, w \odot \Phi(x)^T + b\,).$$

The dual form of the SVM reveals an interesting property when viewed in this light. The dual optimization, applying a feature transform Φ, takes the form,

$$\alpha^* = \arg\max_{\alpha \geq 0} \sum_i \Big[\alpha_i - \frac{1}{2} \sum_j \alpha_i \, \alpha_j \, y^{(i)} \, y^{(j)} \; \Phi(x^{(i)}) \odot \Phi(x^{(j)}) \Big] \qquad \text{s.t.} \quad \sum_i \alpha_i \, y^{(i)} = 0$$

i.e., it depends on the inner product between the transformed data $\Phi(x^{(i)}) \odot \Phi(x^{(j)})$.

Consider an example degree-two polynomial transform of x,

$$\Phi(x) = \big[\, 1 \,,\, \sqrt{2}x_1 \,, \ldots ,\, \sqrt{2}x_n \,,\, x_1^2 \,,\, \ldots \,,\, x_n^2 \,,\, \sqrt{2}x_1x_2 \,,\, \ldots \,,\, \sqrt{2}x_{n-1}x_n \,\big]$$

where the scaling of $\sqrt{2}$ is for convenience in the sequel. Then, consider the relevant dot product between two data points $a = x^{(i)}$ and $b = x^{(j)}$:

$$\begin{aligned}
\Phi(a) \odot \Phi(b) &= 1 + 2a_1b_1 + \ldots + 2a_nb_n + a_1^2b_1^2 + \ldots + 2a_{n-1}a_nb_{n-1}b_n \\
&= 1 + \sum_k 2a_kb_k + \sum_k a_k^2b_k^2 + \sum_k \sum_{l>k} 2a_ka_lb_kb_l \\
&= \Big(1 + \sum_k a_kb_k\Big)^2 \\
&= K_{\text{poly}}(a, b\,;\,2)
\end{aligned}$$

Practically speaking, this identity tells us that although computing the transform Φ and then taking the dot product nominally requires $O(n^2)$ work (the number of features in the degree-two expansion), we can actually sidestep this computation and calculate the same quantity using a nonlinear function of a and b that requires only $O(n)$ work.

The function K is called a *kernel*, and can be viewed as measuring the similarity between the two points a, b. The standard dot product $a \odot b^T$ can be viewed as a similarity measure of the vectors a,b; it is largest when a,b point in the same direction, zero when they are orthogonal, and negative if their relative angle is larger. The function $K_{\text{poly}}(a, b\,;\,2)$ measures this similarity in the vector space embedded by Φ, but without explicitly transforming the data.

Moreover, having identified the kernel K_{poly}, we can even compute predictions at new points without applying Φ, since

$$\begin{aligned}
\hat{y}(x) = \text{sign}(\, w^* \odot \Phi(x)^T + b^* \,) &= \text{sign}\Big(\sum_i \alpha_i \, y^{(i)} \; \Phi(x^{(i)}) \odot \Phi(x) + b^* \Big) \\
&= \text{sign}\Big(\sum_i \alpha_i \, y^{(i)} \; K_{\text{poly}}(x^{(i)}, x) + b^* \Big). \qquad (5.12)
\end{aligned}$$

Thus we can use even high-dimensional transforms Φ if they correspond to efficient-to-compute kernel functions K.

Mercer Kernels

We found the function K_{poly} by first (carefully) choosing a feature transform Φ, and then showing that K evalutes $\Phi(a) \odot \Phi(b)$. To use this idea, we could design Φ and then look for an efficient K to match it. But what about the converse? Since we do not need Φ once we have identified K, what functions K can we choose that correspond to some feature transform Φ?

The solution to this question lies in Mercer's theorem [Mercer, 1909], which tells us that any continuous and symmetric function $K(a,b) = K(b,a)$ can be expressed as the inner product of some feature transform Φ if K is positive semi-definite, i.e.,

$$\sum_i \sum_j K(x^{(i)}, x^{(j)}) g_i g_j \geq 0 \qquad \forall D = \{x^{(i)}\} \quad \text{and} \quad \forall \{g_i \in \mathbb{R}\}$$

We can interpret this as requiring that the matrix $\mathbf{K}$ defined by $\mathbf{K}_{ij} = K(x^{(i)}, x^{(j)})$ be a symmetric positive semi-definite matrix for any possible collection of points $D = \{x^{(i)}\}$.

Note that this proof does not guarantee that we can calculate Φ easily (in fact, the features Φ correspond to the eigenfunctions of K), and Φ may even be infinite dimensional! But, since we do not need to calculate Φ, even such implicit or computationally intractable transforms can be used with the dual form of the SVM.

Common Kernel Functions

The kernel perspective of the support vector machine gives us a novel way to define a nonlinear model – instead of selecting features Φ that we believe will be linearly related to the target, we can instead define a similarlity function K that captures which data are most relevant for predicting a new point x. The predictive model (5.12) can be viewed as a weighted average, where the weights depend on the Lagrange multipliers α_i and the similarity of $x^{(i)}$ to x. One can therefore view the kernel (or dual) SVM model as a type of *instance-based* learner, where our learner memorizes some or all of the training data and compares them to the test point. Contrast this with the linear (primal) SVM model, which stores no data points explicitly but instead stores a fixed-dimensional parameter vector θ (a *parameter-based* model).

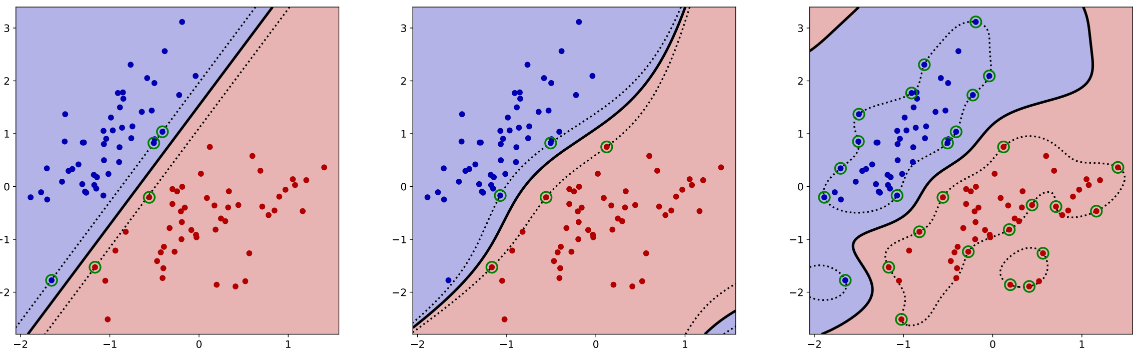
There are a number of common, popular choices for similarity kernels K, including

$$\begin{aligned} K_{\text{rbf}}(a, b\,;\,\sigma) &= \exp\Big[-\|a-b\|^2/\sigma^2\Big] && \text{(Radial Basis)} \\ K_{\text{poly}}(a, b\,;\,d) &= \Big(1 + a \odot b^T\Big)^d && \text{(Polynomial)} \\ K_{\text{sat}}(a, b\,;\,(c,h)) &= \tanh\Big(c\,a \odot b^T + h\Big) && \text{(Saturating)} \end{aligned}$$

Each type of kernel typically involves some hyperparameters, such as the polynomial degree d, or the scale parameter σ (sometimes called the *bandwidth* of the RBF kernel). Often these parameters influence the flexibility of the model, and hence its propensity to overfit. For example, larger values of d correspond to higher-dimensional transformations Φ, or equivalently, more flexibile decision boundaries (i.e., the solution to a degree-d polynomial), making it easier for the learner to overfit. **Example later?**

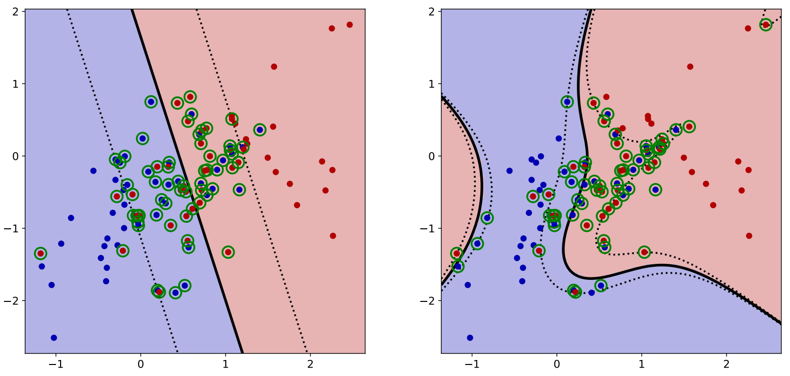
It is worth noting that we do not even *need* to require that K be a Mercer kernel – the condition is required in order to make the connection to a linear model in some (implicit, high-dimensional) feature space, but in a practical sense we can apply the same optimization and prediction procedures with some arbitrary symmetric function K. Indeed, in some settings we may want to design a special similarity function for some particular domain (for example, evaluating similarity between protein sequences using string matching scores [e.g., Saigo et al., 2004]). In fact, despite its popularity the saturating, $\tanh$ kernel K_{sat} is not a valid Mercer kernel for arbitrary parameters c, h [Smola et al., 2000].

Compared to other instance-based learners, like k-nearest neighbors, we can view the SVM optimization as identifying which data points are "easy" and thus unnecessary to retain. In certain cases, the number of support vectors can be much smaller than the number of data m, making the predictor significantly easier to store and faster to evaluate. For example, in Figure 5.3(b), the degree-4 polynomial



(a) Linear (b) Degree-4 Polynomial (c) Radial Basis

Figure 5.3: Decision boundaries (solid) and margins (dashed) for (a) a linear SVM, (b) a degree-4 polynomial, and (c) a radial basis (Gaussian kernel) SVM on easily separable data. The support vectors are indicated by green circles.



(a) Linear (b) Degree-4 Polynomial

Figure 5.4: Although fewer than m support vectors may be required, in practice many problems (particularly those with significant class overlap) may still result in a large number of support vectors.

kernel solution requires five support vectors, while the radial basis kernel solution uses about 30 of the 100 data points as support vectors[2]. Unfortunately, although in some examples the number of support vectors can be very small, in practice for high-dimensional or non-separable data SVMs tend to result in a fairly large fraction of the data being retained as support vectors (see Figure 5.4).

Sequential Minimal Optimization (SMO)

Traditional quadratic program solvers are often not very efficient for SVM optimization. The most common method to optimize the SVM dual is a coordinate descent algorithm called *Sequential Minimal Optimization*, or SMO [Platt, 1998].

SMO updates pairs of Lagrange multipliers α_i, α_j at each step; since the overall sum, $\sum_i \alpha_i \, y^{(i)} = 0$, is constrained, we can write α_i as a deterministic function of α_j and solve a one-dimensional quadratic optimization at each step. The pair of constraints $\alpha_i, \alpha_j \in [0, R]$ gives a range for $\alpha_j \in [L, U]$, and

[2]This also illustrates that the dual form may be more efficient than the primal in some settings; the degree-4 polynomial primal solution θ has 15 parameters, compared to 5 2D support vectors. In the RBF case, the primal features are infinite dimensional.

Algorithm 5.1 SMO(D, K): Optimize the SVM dual

```
**Input**: A data set $D = (X, Y)$ and kernel $K(\cdot, \cdot)$.
**Output**: Lagrange multipliers $\alpha$.
Initialize $\alpha = 0$
while $\exists i$ violating the margin constraints do
    Select some $j \neq i$
    Compute constraints $L \leq \alpha_j \leq U$
    Solve for $\alpha_j$
    Update bias $b$
end while
```

we can simply solve the unconstrained quadratic and truncate it if it falls beyond this range. The overall algorithm is sketched in Algorithm 5.1.

More here More specifically, suppose we have selected our two data points i, j, and define the linear response given the current values of α to be,

$$r(x\,;\,\alpha) = \sum_i \alpha_i\, y^{(i)} x^{(i)} \odot x^T. \tag{5.13}$$

and let $c_{ij} = y^{(i)}\alpha_i + y^{(j)}\alpha_j$ be the contribution of these data to the equality constraint. If $y^{(i)} = y^{(j)}$, then we have $\alpha_i + \alpha_j$ constant, and the constraints

$$\alpha_j \in [0, R] \cap [\alpha_i + \alpha_j - R, \alpha_i + \alpha_j] \quad = \quad [L, U].$$

Conversely, if $y^{(i)} \neq y^{(j)}$, then we have $\alpha_j - \alpha_i$ constant, and

$$\alpha_j \in [0, R] \cap [\alpha_j - \alpha_i, R + \alpha_j - \alpha_i] \quad = \quad [L, U].$$

The solution to the unconstrained quadratic function is

$$\alpha_j \leftarrow \alpha_j - y^{(j)} \frac{r(x^{(i)}) - r(x^{(j)})}{2(x^{(i)} \odot x^{(j)T}) - \|x^{(i)}\|^2 - \|x^{(j)}\|^2}$$

and we simply check whether this value falls in the range $[L, U]$, and if not, set it to the limiting value. Finally, we set α_i in order to keep c_{ij} constant given the new α_j:

$$\alpha_i \leftarrow c_{ij} - y^{(i)} y^{(j)} \alpha_j.$$

Much of the efficiency of SMO comes from developing heuristics to select which pairs of data points i, j to update at each step. The standard SMO algorithm Platt [1998] makes passes through the data to pick i, initially focusing on data points in the interior of the constraint set, $0 < \alpha_i < R$. Given such a point i, we can first try the j which maximizes (or approximately maximizes) the value $|r(x^{(i)}) - r(x^{(j)})|$, in the hopes that this will produce the largest update; maintaining a cache of the r values allows this to be done efficiently. If this j does not make progress, the algorithm begins trying other data points j; when no j makes progress with any interior-value i, we begin trying α_i on the constraints, and so on. The algorithm stops when the margin constraints are deemed sufficently close to being satisfied.

Update rule for b

5.5 Multiclass SVMs and structured losses

So far we have considered only binary classification problems. How can we extend our margin intuition and optimization to multiple classes? And, what if some class errors are more serious than others – for example, misclassifying an image of a dog by labelling it with the wrong breed might be less serious than labelling it as a type of cat, or as a bicycle.

We can easily extend our linear classifier decision function by using the joint feature-class transform representation, $\Phi(x, y)$, in which we have features associated with each class value y, and use prediction rule

$$\hat{y}(x) = f(x\,;\,\theta) = \arg\max_{y} \theta \odot \Phi(x, y).$$

explain more? explained in linear classifier section?

In this form, our slack variable representation of the soft SVM can be defined as

$$\begin{gathered}(w^*, b^*) = \arg\min_{w,b,\epsilon} \sum_j w_j^2 + R \sum_i \epsilon^{(i)} \\ \text{subject to } \; w^T\Phi(x^{(i)}, y^{(i)}) - w^T\Phi(x^{(i)}, y) \geq 1 - \epsilon^{(i)} \quad \forall y \neq y^{(i)}\end{gathered} \tag{5.14}$$

If our classes admit some structure, we can also introduce class-specific losses, i.e., penalize some types of mistakes more than others, by adjusting the required margin on those classes:

$$\begin{gathered}(w^*, b^*) = \arg\min_{w,b,\epsilon} \sum_j w_j^2 + R \sum_i \epsilon^{(i)} \\ \text{subject to } \; w^T\Phi(x^{(i)}, y^{(i)}) - w^T\Phi(x^{(i)}, y) \geq \Delta(y^{(i)}, y) - \epsilon^{(i)} \quad \forall y \neq y^{(i)}\end{gathered} \tag{5.15}$$

Here the function $\Delta(y, y')$ tries to make sure that class pairs y, y' that are very different are very far from one another; if we take $\Delta(y, y') = \mathbb{1}[y \neq y']$ this reduces to (5.14). Both (5.14) and (5.15) are quadratic programs with $m(c-1)$ linear constraints (one per data point and incorrect class). We can then optimize the resulting function either using a QP solver such as SMO, or a method such as SGD on a hinge-like loss function.

5.6 Kernelized Linear Regression

The connection between the kernel function $K(\cdot)$ and an implicit feature space Φ can be applied in a number of types of linear model. For example, we can develop a kernelized version of L_2-regularized linear regression to enable a non-linear regression model that is instance-based and local.

To derive this form, we start with the regularized solution to the mean-squared error linear regression model, and rearrange to obtain:

$$\begin{aligned} & \theta^* = \mathbf{y} \odot \mathbf{X} \odot (\mathbf{X}^T \odot \mathbf{X} + \omega I)^{-1} \\ \Rightarrow \quad & \theta^* \odot (\mathbf{X}^T \odot \mathbf{X} + \omega I) = \mathbf{y} \odot \mathbf{X} \\ \Rightarrow \quad & \omega\, \theta^* = (\mathbf{y} - \theta \odot \mathbf{X}^T) \odot \mathbf{X} \\ \Rightarrow \quad & \theta^* = \frac{1}{\omega}(\mathbf{y} - \theta^* \odot \mathbf{X}^T) \odot \mathbf{X} \quad = r \odot \mathbf{X} \end{aligned}$$

where the last line defines the quantity $r = \frac{1}{\omega}(\mathbf{y} - \theta^* \odot \mathbf{X}^T)$, the vector of error residuals scaled by the regularization parameter ω. From this definition of r, and applying the last line $\theta^* = r \odot \mathbf{X}$ gives an expression for r in terms of itself:

$$\omega\, r = \mathbf{y} - \theta^* \odot \mathbf{X}^T = \mathbf{y} - r \odot \mathbf{X} \odot \mathbf{X}^T$$

Rearranging and solving for r gives,

$$r = (\mathbf{X} \odot \mathbf{X}^T + \omega I)^{-1} \odot \mathbf{y} = (\mathbf{K} + \omega I)^{-1} \odot \mathbf{y}$$

where $\mathbf{K}$ is the Gram matrix of dot products between pairs of data points, $\mathbf{K}_{ij} = x^{(i)} \odot x^{(j)T}$. Then, $\mathbf{K}_{ij}$ can easily be replaced by a kernel function $K(x^{(i)}, x^{(j)})$ that evalutes inner products in an implicit feature space Φ.

Our prediction at a new test point $\tilde{x}$ is similarly representable using the kernel function K and the scaled error residual values r (which are analagous to the Lagrange multipliers in the SVM classifier, with one value per data point):

$$\tilde{y} = \theta^* \odot \tilde{x}^T = r \odot \mathbf{X} \odot \tilde{x}^T = \sum_i r_i \, (x^{(i)} \odot \tilde{x}^T) = \sum_i r_i \, K(x^{(i)}, \tilde{x})$$

Notice that, when ω becomes large, the scaled residual values r_i become small and our predictor approaches zero. Also, unlike the coefficients of the SVM classifier, the vector r in this formulation is *not* sparse; each prediction will depend on all m data (unless $K(x^{(i)}, \tilde{x}) = 0$ for some data i). For a version of linear regression that allows for a smaller (sparse) set of support vectors, we can replace the mean-squared error loss with a more hinge-like loss that does not penalize “small” errors in the regression estimates [Drucker et al., 1996]. In that setting, the data points at which the prediction is sufficiently accurate will have coefficients zero, and the predictor will only depend on data whose predictions are at or beyond the acceptable level of error.

Fix figure

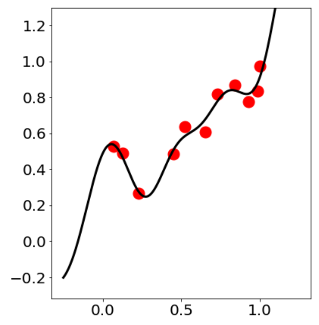


Figure 5.5: Kernel linear regression using an RBF kernel.