17 Support Vector Machines

We now discuss an influential and effective classification algorithm called Support Vector Machines (SVMs). In addition to their successes in many classification problems, SVMs are responsible for introducing and/or popularizing several important ideas to machine learning, namely, *kernel methods*, *maximum margin methods*, *convex optimization*, and *sparsity/support vectors*. Unlike the mostly-Bayesian treatment that we have given in this course, SVMs are based on some very sophisticated Frequentist arguments (based on a theory called Structural Risk Minimization and VC-Dimension) which we will not discuss here, although there are many close connections to Bayesian formulations.

17.1 Maximizing the margin

Suppose we are given N training vectors $\{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x} \in \mathbb{R}^D$, $y \in \{-1, 1\}$. We want to learn a classifier

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b \tag{1}$$

so that the classifier's output for a new $\mathbf{x}$ is $\operatorname{sign}(f(\mathbf{x}))$.

Suppose that our training data are linearly-separable in the feature space $\phi(\mathbf{x})$, i.e., as illustrated in Figure 1, the two classes of training exemplars are sufficiently well separated in the feature space that one can draw a hyperplane between them (e.g., a line in 2D, or plane in 3D). If they are linearly separable then in almost all cases there will be many possible choices for the linear decision boundary, each one of which will produce no classification errors on the training data. Which one should we choose? If we place the boundary very close to some of the data, there seems to be a greater danger that we will misclassify test data, especially when the training data are alsmot certainy noisy.

This motivates the idea of placing the boundary to maximize the **margin**, that is, the distance from the hyperplane to the closest data point in either class. This can be thought of having the largest "margin for error" — if you are driving a fast car between a scattered set of obstacles, it's safest to find a path that stays as far from them as possible.

More precisely, in a *maximum margin method*, we want to optimize the following objective function:

$$\max_{\mathbf{w},b} \min_i \operatorname{dist}(\mathbf{x}_i, \mathbf{w}, b) \tag{2}$$

$$\text{such that, for all } i,\ y_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 0 \tag{3}$$

where $\operatorname{dist}(\mathbf{x}, \mathbf{w}, b)$ is the Euclidean distance from the feature point $\phi(\mathbf{x})$ to the hyperplane defined by $\mathbf{w}$ and b. With this objective function we are maximizing the distance from the decision boundary $\mathbf{w}^T \phi(\mathbf{x}) + b = 0$ to the nearest point i. The constraints force us to find a decision boundary that classifies all training data correctly. That is, for the classifier a training point correctly y_i and $\mathbf{w}^T \phi(\mathbf{x}_i) + b$ should have the same sign, in which case their product must be positive.

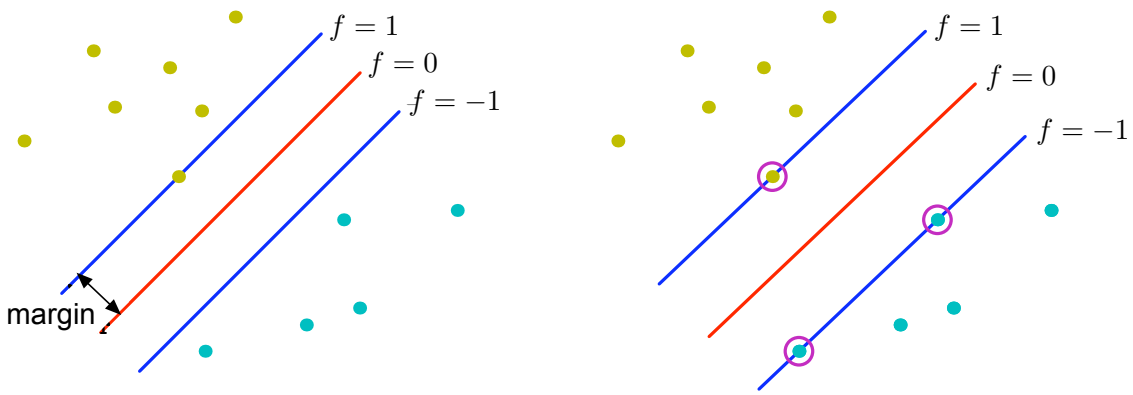

Figure 1: Left: the margin for a decision boundary is the distance to the nearest data point. Right: In SVMs, we find the boundary with maximum margin. (Figure from *Pattern Recognition and Machine Learning* by Chris Bishop.)

It can be shown that the distance from a point $\phi(\mathbf{x}_i)$ to a hyperplane $\mathbf{w}^T\phi(\mathbf{x}) + b = 0$ is given by $\frac{|\mathbf{w}^T\phi(\mathbf{x}_i)+b|}{||\mathbf{w}||}$, or, since y_i tells us the sign of $f(\mathbf{x}_i)$, $\frac{y_i(\mathbf{w}^T\phi(\mathbf{x}_i)+b)}{||\mathbf{w}||}$. This can be seen intuitively by writing the hyperplane in the form $f(\mathbf{x}) = \mathbf{w}^T(\phi(\mathbf{x}_i) - \mathbf{p})$, where $\mathbf{p}$ is a point on the hyperplane such that $\mathbf{w}^T\mathbf{p} = b$. The vector from $\phi(\mathbf{x}_i)$ to the hyperplane projected onto $\mathbf{w}/||\mathbf{w}||$ gives a vector from the hyperplane to the the point; the length of this vector is the desired distance.

Substituting this expression for the distance function into the above objective function, we get:

$$\max_{\mathbf{w},b} \min_i \frac{y_i(\mathbf{w}^T\phi(\mathbf{x}_i)+b)}{||\mathbf{w}||} \tag{4}$$

$$\text{such that, for all } i,\ y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 0 \tag{5}$$

Note that, because of the normalization by $||\mathbf{w}||$ in (4), the scale of $\mathbf{w}$ is arbitrary in this objective function. That is, if we were to multiply $\mathbf{w}$ and b by some real scalar α, the factors of α in the numerator and denominator will cancel one another. Now, suppose that we choose the scale so that the *nearest point* to the hyperplane, $\mathbf{x}_i$, satisfies $y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) = 1$. With this assumption the $\min_i$ in Eqn (4) becomes redundant and can be removed. Thus we can rewrite the objective function and the constraint as

$$\max_{\mathbf{w},b} \frac{1}{||\mathbf{w}||} \tag{6}$$

$$\text{such that, for all } i,\ y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 \tag{7}$$

Finally, as a last step, since maximizing $1/||\mathbf{w}||$ is the same as minimizing $||\mathbf{w}||^2/2$, we can re-express the optimization problem as

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 \tag{8}$$

$$\text{such that, for all } i,\ y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 \tag{9}$$

This objective function is a **quadratic program**, or QP, because the objective function is quadratic in the unknowns, and all of the constraints are linear in the unknowns. A QP has a single global minima, which can be found efficiently with current optimization packages.

In order to understand this optimization problem, we can see that the constraints will be "active" for only a few datapoints. That is, only a few datapoints will be close to the margin, thereby constraining the solution. These points are called the **support vectors**. Small movements of the other data points have no effect on the decision boundary. Indeed, the decision boundary is determined only by the support vectors. Of course, moving points to within the margin of the decision boundary will change which points are support vectors, and thus change the decision boundary. This is in constrast to the probabilistic methods we have seen earlier in the course, in which the positions of all data points affect the location of the decision boundary.

17.2 Slack Variables for Non-Separable Datasets

Many datasets will not be linearly separable. As a result, there will be no way to satisfy all the constraints in Eqn. (9). One way to cope with such datasets and still learn useful classifiers is to loosen some of the constraints by introducing **slack variables**.

Slack variables are introduced to allow certain constraints to be violated. That is, certain training points will be allowed to be within the margin. We want the number of points within the margin to be as small as possible, and of course we want their penetration of the margin to be as small as possible. To this end, we introduce a slack variable ξ_i, one for each datapoint i. (ξ is the Greek letter xi, pronounced "ksi."). The slack variable is introduced into the optmization problem in two ways. First, the slack variable ξ_i dictates the degree to which the constraint on the ith datapoint can be violated. Second, by adding the slack variable to the energy function we are aiming to simultaneously minimize the use of the slack variables.

Mathematically, the new optimization problem can be expressed as

$$\min_{\mathbf{w},b,\xi_{1:N}} \sum_i \xi_i + \lambda \frac{1}{2}||\mathbf{w}||^2 \tag{10}$$

$$\text{such that, for all } i,\ y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \tag{11}$$

As discussed above, we aim to both maximize the margin and minimize violation of the margin constraints. This objective function is still a QP, and so can be optimized with a QP library. However, it does have a much larger number of optimization variables, namely, one slack variable ξ must now be optimized for each datapoint. In practice, SVMs are normally optimized with special-purpose optimization procedures designed specifically for SVMs.

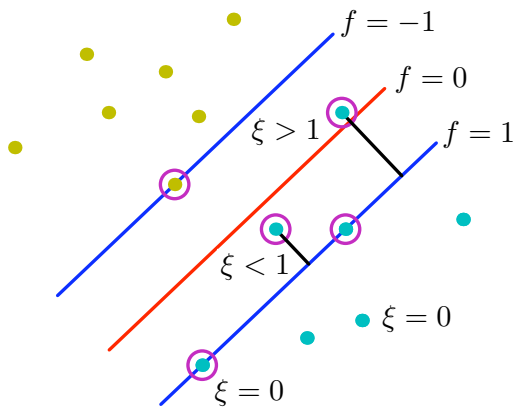

Figure 2: The slack variables $\xi_i \geq 1$ for misclassified points, and $0 < \xi_i < 1$ for points close to the decision boundary. (Figure from *Pattern Recognition and Machine Learning* by Chris Bishop.)++

17.3 Loss Functions

In order to better understand the behavior of SVMs, and how they compare to other methods, we will analyze them in terms of their **loss functions.**[1] In some cases, this loss function might come from the problem being solved: for example, we might pay a certain dollar amount if we incorrectly classify a vector, and the penalty for a false positive might be very different for the penalty for a false negative. The rewards and losses due to correct and incorrect classification depend on the particular problem being optimized. Here, we will simply attempt to minimize the total number of classification errors, using a penalty is called the **0-1 Loss:**

$$L_{0-1}(\mathbf{x}, y) = \begin{cases} 1 & yf(\mathbf{x}) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

(Note that $yf(\mathbf{x}) > 0$ is the same as requiring that y and $f(\mathbf{x})$ have the same sign.) This loss function says that we pay a penalty of 1 when we misclassify a new input, and a penalty of zero if we classify it correctly.

Ideally, we would choose the classifier to minimize the loss over the new test data that we are given; of course, we don't know the true labels, and instead we optimize the following surrogate objective function over the training data:

$$E(\mathbf{w}) = \sum_i L(\mathbf{x}_i, y_i) + \lambda R(\mathbf{w}) \tag{13}$$

[1] A loss function specifies a measure of the quality of a solution to an optimization problem. It is the penalty function that tell us how badly we want to penalize errors in a models ability to fit the data. In probabilistic methods it is typically the negative log likelihood or the negative log posterior.

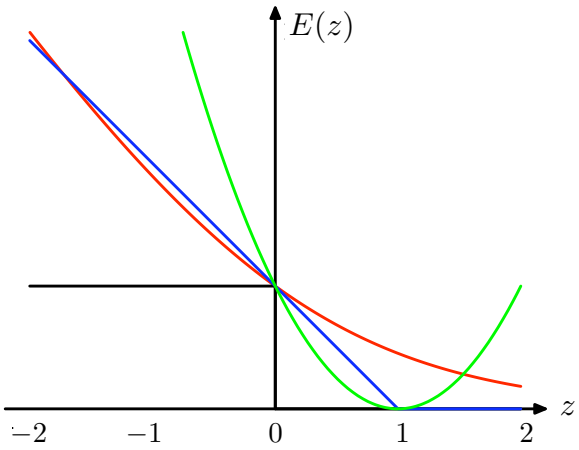

Figure 3: Loss functions, $E(z)$, for learning, for $z = y\,f(x)$. Black: 0-1 loss. Red: LR loss. Green: Quadratic loss ($(z-1)^2$). Blue: Hinge loss. (Figure from *Pattern Recognition and Machine Learning* by Chris Bishop.)

where $R(\mathbf{w})$ is a regularizer meant to prevent overfitting (and thus improve performance on future test data). The basic assumption is that loss on the training set should correspond to loss on the test set. If we can get the classifier to have small loss on the training data, while also being smooth, then the loss we pay on new data ought to not be too big either. This optimization framework is equivalent to MAP estimation as discussed previously[2]; however, here we are not at all concerned with probabilities. We only care about whether the classifier gets the right answers or not.

Unfortunately, optimizing a classifier for the 0-1 loss is very difficult: it is not differentiable everywhere, and, where it is differentiable, the gradient is zero everywhere. There are a set of algorithms called Perceptron Learning which attempt to do this; of these, the Voted Perceptron algorithm is considered one of the best. However, these methods are somewhat complex to analyze and we will not discuss them further. Instead, we will use other loss functions that approximate 0-1 loss.

We can see that maximum likelihood logistic regression is equivalent to optimization with the following loss function:

$$L_{\text{LR}} = \ln\left(1 + e^{-yf(\mathbf{x})}\right) \tag{14}$$

which is the negative log-likelihood of a single data vector. This function is a poor approximation to the 0-1 loss, and, if all we care about is getting the labels right (and not the class probabilities), then we ought to search for a better approximation.

SVMs minimize the slack variables, which, from the constraints, can be seen to give the **hinge loss**:

$$L_{hinge} = \begin{cases} 1 - yf(\mathbf{x}) & 1 - yf(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

[2]However, not all loss functions can be viewed as the negative log of a valid likelihood function, although all negative-log likelihoods can be viewed as loss functions for learning.

That is, when a data point is correctly classified and further from the decision boundary than the margin, then the loss is zero. In this way it is insensitive to correctly-classified points far from the boundary. But when the point is within the margin or incorrectly classified, then the loss is simply the magnitude of the slack variable, i.e. $\xi = 1 - yf(\mathbf{x})$, where $f(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) + b$. The hinge loss therefore increases linearly for misclassified points, which is not nearly as quickly as the LR loss.

17.4 The Lagrangian and the Kernel Trick

We now use the Lagrangian to transform the SVM problem in a way that will lead to a powerful generalization. For simplicity here we assume that the dataset is linearly separable, and so we drop the slack variables.

The Langrangian allows us to take the constrained optimization problem above in Eqn. (9) and re-express it as an unconstrained problem. The Lagrangian for the SVM objective function in Eqn. (9), with Lagrange multipliers $a_i \geq 0$, is:

$$L(\mathbf{w}, b, a_{1:N}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_i a_i \left(y_i \left(\mathbf{w}^T\phi(\mathbf{x}_i) + b\right) - 1\right) \tag{16}$$

The minus sign with the secon term is used because we are minimizing with respect to the first term, but maximizing the second.

Setting the derivative of $\frac{dL}{d\mathbf{w}} = 0$ and $\frac{dL}{db} = 0$ gives the following constraints on the solution:

$$\mathbf{w} = \sum_i a_i y_i \phi(\mathbf{x}_i) \tag{17}$$

$$\sum_i y_i a_i = 0 \tag{18}$$

Using (17) we can substitute for $\mathbf{w}$ in 16. Then simplifying the result, and making use of the next constraint (17), one can derive what is often called the **dual Lagrangian**:

$$L(a_{1:N}) = \sum a_i - \frac{1}{2}\sum_i \sum_j a_i a_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \tag{19}$$

While this objective function is actually more expensive to evaluate than the primal Lagrangian (i.e., 16), it does lead to the following modified form

$$L(a_{1:N}) = \sum a_i - \frac{1}{2}\sum_i \sum_j a_i a_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{20}$$

where $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ is called a **kernel function.** For example, if we used the basic linear features, i.e., $\phi(\mathbf{x}) = \mathbf{x}$, then $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j$.

The advantage of the kernel function representation is that it frees us from thinking about the features directly; the classifier can be specified solely in terms of the kernel. Any kernel that

satisfies a specific technical condition[3] is a valid kernel. For example, one of the most commonly-used kernels is the "RBF kernel":

$$k(\mathbf{x}, \mathbf{z}) = e^{-\gamma||\mathbf{x}-\mathbf{z}||^2} \tag{21}$$

which corresponds to a vector of features $\phi(\mathbf{x})$ with infinite dimensionality! (Specifically, each element of ϕ is a Gaussian basis function with vanishing variance).

Note that, just as most constraints in the Eq. (9) are not "active", the same will be true here. That is, only some constraints will be active (ie the support vectors), and for all other constraints, $a_i = 0$. Hence, once the model is learned, most of the training data can be discarded; only the support vectors and their a values matter.

The one final thing we need to do is estimate the bias b. We now know the values for a_i for all support vectors (i.e., for data constraints that are considered active), and hence we know $\mathbf{w}$. Accordingly, for all support vectors we know, by assumption above, that

$$f(\mathbf{x}_i) = \mathbf{w}^T\phi(\mathbf{x}_i) + b = 1 \; . \tag{22}$$

From this one can easily solve for b.

Applying the SVM to new data. For the kernel representation to be useful, we need to be able to classify new data without needing to evaluate the weights. This can be done as follows:

$$\begin{aligned} f(\mathbf{x}_{new}) &= \mathbf{w}^T\phi(\mathbf{x}_{new}) + b & (23)\\ &= \left(\sum_i a_i y_i \phi(\mathbf{x}_i)\right)^T \phi(\mathbf{x}_{new}) + b & (24)\\ &= \sum_i a_i y_i k(\mathbf{x}_i, \mathbf{x}_{new}) + b & (25) \end{aligned}$$

Generalizing the kernel representation to non-separable datasets (i.e., with slack variables) is straightforward, but will not be covered in this course.

17.5 Choosing parameters

To determine an SVM classifier, one must select:

- The regularization weight λ
- The parameters to the kernel function
- The type of kernel function

These values are typically selected either by hand-tuning or cross-validation.

[3]Specifically, suppose one is given N input points $\mathbf{x}_{1:N}$, and forms a matrix $\mathbf{K}$ such that $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. This matrix must be positive semidefinite (i.e., all eigenvalues non-negative) for all possible input sets for k to be a valid kernel.

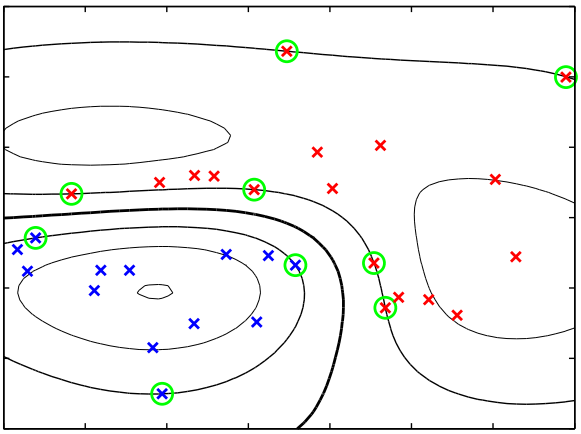

Figure 4: Nonlinear classification boundary learned using kernel SVM (with an RBF kernel). The circled points are the support vectors; curves are isocontours of the decision function (e.g., the decision boundary $f(x) = 0$, etc.) (Figure from *Pattern Recognition and Machine Learning* by Chris Bishop.)

17.6 Software

Like many methods in machine learning there is freely available software on the web. For SVM classification and regression there is well-known software developed by Thorsten Joachims, called SVMlight, (`URL: http://svmlight.joachims.org/`).

