CS181 Lecture 13 — Support Vector Machines

Support vector machines (SVMs) are an approach to classification that has received a lot of interest in the past few years. While they are more difficult to understand than decision trees or neural networks, they have been observed to have very good results. The purpose of these notes is not for you to understand the details of SVMs, but to know enough about them that you can apply them intelligently to a problem that you might encounter.

SVMs are based on three big ideas. The first is maximizing the margin. Intuitively this means that when we learn a linear separator, we should try to choose the decision boundary so as to maximize the distance to the points that are closest to the boundary. The second big idea is duality. This is an idea that is used many times in optimization problems. It allows one problem to be transformed into another problem that may be easier to solve. The third big idea is kernels. Kernels allow a set of features to be mapped into a higher-dimensional, and therefore more expressive feature space, without incurring the full computational cost one might expect.

1 Maximizing the margin

The “margin”, intuitively, means the distance of the decision boundary to the closest points. There are two definitions of margin. The first one, called the geometric margin, is more intuitive, but the second one, called the functional

margin, is the one that is actually used in practice.

Definition 1.1: Let H be a hyperplane that divides the space into positive and negative regions, and x be a point with classification c. The geometric margin of the point x is the distance from x to H, if it is correctly classified, otherwise it is the negative distance.

Definition 1.2: The geometric margin of a training set is the minimum geometric margin of points in the training set.

Figure 1(a) shows the geometric margin of a given set of points. One thing to note is that there are several closest points to the hyperplane. This often happens in practice with support vector machines.

SVMs find a linear separator (although using kernels they might find a linear separator in a higher-dimensional feature space). The goal of SVMs is to find the linear separator that maximizes the geometric margin. Figure 1(b) shows a

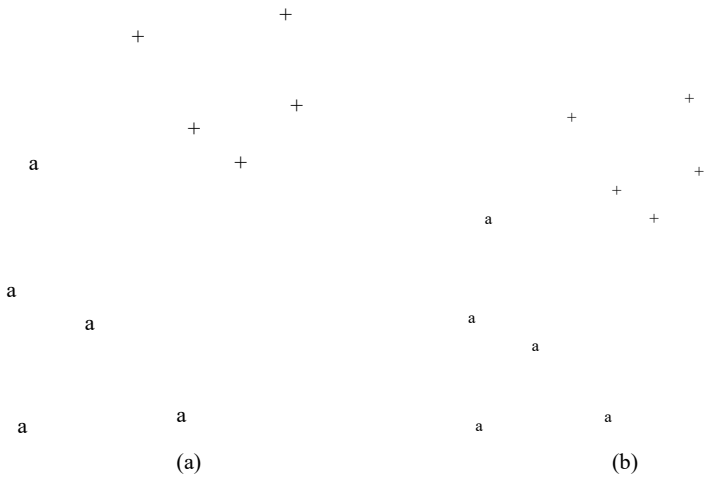

Figure 1: (a) The geometric margin of a set of points. (b) A separator with a poor margin.

linear separator with a smaller geometric margin than Figure 1(a). Intuitively this linear separator is less good, because it has less margin for error in classifying the points. We might therefore expect it to generalize less well to unseen points.

We can make this intuition formal by providing error bounds on the generalization error given a margin.

Theorem 1.3: Let R be the maximum norm of any instance X. Let N be the number of training examples. With probability at least $1 - \delta$, a hypothesis with margin $\geq \gamma$ has error ϵ no more than

$$\frac{2}{N}\left(\frac{64R^2}{\gamma^2}\log\frac{eN\gamma}{4R}\log\frac{128R^2}{\gamma^2} + \log\frac{4}{\delta}\right)$$

provided $N > 2/\epsilon$ and $N > 64R^2/\gamma^2$

The surprising thing about this bound is that the dimension of the instance space does not appear explicitly (although implicitly, the more dimensions the instance space has, the larger we will expect R to be, presuming the scale of points in each dimension is the same).

While the geometric margin is a good thing to maximize, it turns out to be easier to work with the functional margin.

Definition 1.4: Given a hyperplane $(w \cdot x) + b = 0$, the functional margin of an instance (x_i, y_i) is $y_i((w \cdot x_i)+b)$. $(w \cdot x_i)$ denotes the inner product between w and x_i, i.e. $w_1 x_{i\,1} + \ldots + w_n x_{i\,n}$.

Definition 1.5: The functional margin of a training set $(x_1, y_1), \ldots, (x_N, y_N)$ is $\min(y_i((w \cdot x_i) + b))$

The functional margin is related to the geometric margin. When $\|w\| = 1$, the functional margin is the geometric margin. In general, given a hyperplane $(w \cdot x) + b = 0$, we can divide w and b by $\|w\|$, to get an equivalent hyperplane with $\|w\| = 1$. This is called normalization.

We can view this the other way round. Suppose we fix the functional margin at 1, and the norm is $\|w\|$. Then to get the geometric margin, we normalize, i.e. divide by $\|w\|$. Since the functional margin was 1, the geometric margin will be $1/\|w\|$. This shows that to maximize the geometric margin, we can minimize $\|w\|$, over all hyperplanes whose functional margin is 1. This can be expressed as the following program:

$$\text{minimize} \quad 1/2(w \cdot w) \tag{1}$$

$$\text{subject to } y_i((w \cdot x_i) + b) \geq 1 \tag{2}$$

How do we understand this program? $< w \cdot w >$ is just $\|w\|_2$, and the 1/2 is thrown in for convenience. What about the constraints. They say that for every instance, the functional margin of the instance is at least 1. Since this must be true for all instances, the functional margin of the training set will be at least 1. Clearly these constraints have to hold if the functional margin is to be 1. But how do they force the functional margin to be exactly 1 and not something greater? The answer is that they don't do that directly, but in any optimal solution to the program some of the constraints will be tight. If there is a potential solution whose functional margin will be greater than one, there will be a better solution with functional margin equal to one, which will be the same hyperplane with smaller norm. Therefore, for any solution, the functional margin will be one.

2 Duality

How do we solve this optimization problem? The answer is to use duality. Duality is a general principle that is used to transform difficult optimization problems into something simpler. The idea is to put the constraints into the objective function. Each constraint is associated with a slack variable that indicates how important the constraint is in the solution. The general approach starts with a program

$$\begin{aligned} &\min_{w \in \Omega} && f(w) \\ &\text{subject to } && g_i(w) \leq 0 \ \forall i = 1, ..., k \\ &&& h_j(w) = 0 \ \forall j = 1, ..., m \end{aligned}$$

The first step is to construct the Lagrangian function

$$L(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{j=1}^{m} \beta_j h_j(w)$$

In this function the constraints are made part of the function, with Lagrange

multipliers α_i and β_j.

Theorem 2.1: [Kuhn-Tucker] Given such a problem with Ω convex, f convex and g and h affine (i.e. of the form h(w) = Aw + B), necessary and sufficient conditions for w_* to be an optimum are the existence of α_*, β_* such that

$$\begin{aligned}
&\partial L(w_*,\alpha_*,\beta_*)/\partial w = 0 \\
&\partial L(w_*,\alpha_*,\beta_*)/\partial \beta = 0 \\
&\alpha_{*_i} g_i(w_*) && = 0 \ \forall i = 1, ..., k \\
&g_i(w_*) && \leq 0 \ \forall i = 1, ..., k \\
&\alpha_{*_i} && \geq 0 \ \forall i = 1, ..., k
\end{aligned}$$

For our program (6), we introduce Lagrange or slack variables α_i for each training instance i associated with constraint $y_i((w \cdot x_i) + b) - 1 \geq 0$. The intuitive meaning of α_i is: how important is instance i in forming the final solution. $\alpha_i = 0$ means the instance is not important at all. The Lagrangian function for our program looks like:

$$L(w, b, \alpha)=1/2(w \cdot w) - \sum_{i=1}^{N} \alpha_i[y_i((w \cdot x_i) + b) - 1] \quad (3)$$

In a solution to the program, the derivative of the Lagrangian with respect to w and b is 0. So we differentiate the Lagrangian with respect to these variables, and set the derivative to 0.

$$\begin{aligned}
\partial L/\partial w &= w - \textstyle\sum_i \alpha_i y_i x_i = 0 \\
\partial L/\partial b &= - \textstyle\sum_i \alpha_i y_i = 0
\end{aligned}$$

We get

$$w = \sum_i \alpha_i y_i x_i \quad (4)$$

$$0 = \sum_i \alpha_i y_i \quad (5)$$

We plug these back into the Lagrangian (3) to get

$$\begin{aligned}
L(w, b, \alpha) = 1/2 \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) - \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) \\
-b \sum_{i=1}^{N} \alpha_i y_i + \sum_{i=1}^{N} \alpha_i \\
= \sum_{i=1}^{N} \alpha_i - 1/2 \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j)
\end{aligned}$$

The second line follows because $\sum_i \alpha_i y_i = 0$. This leads to a new program, in which the variables are the slack variables α_i. The objective function is the expression we have just derived. The constraint is $\sum_{i=1}^{N} \alpha_i y_i = 0$. This is duality: We have taken a problem in terms of one set of variables, and restated it in terms of the dual variables. In applying duality, we also convert minimization

problems into maximization problems. The new program is as follows: To restate the new program, we have

$$\begin{array}{ll} \max_{\alpha} & \sum_{i=1}^{N} \alpha_i - 1/2 \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j (x_i \cdot x_j) \\ \text{subject to } \sum_{i=1}^{N} & \alpha_i y_i = 0 \end{array} \tag{6}$$

This is a quadratic program. A quadratic program is one in which the objective function is quadratic in the variables (in this case α) and the constraints are linear. This can be solved using any quadratic programming algorithm.

When we solve the program we obtain an optimal α_*. From this, we plug into Equation 4 to obtain

$$w_* = \sum_i \alpha_{*i} y_i x_i \tag{7}$$

b is not mentioned in the dual program. To get b_*, it turns out we can use the following expression, derived from the primal program:

$$b_* = -1/2(\max_{y_i=-1}(< w \cdot x_i >) + \min_{y_i=1}(< w \cdot x_i >)$$

How do we use the solution to classify? Going back to Theorem 2.1, we see that in a solution, for each constraint, either the constraint is tight or the Lagrange variable is 0. I.e.

$$\alpha_{*i}(y_i(< w.x_i > + b - 1)) = 0$$

This can be interpreted as follows: when $\alpha_{*i} = 0$, the constraint is slack and the instance (x_i,y_i) does not impact the value of w. When $\alpha_{*i} > 0$, the constraint is tight and the instance does impact the value of w. An instance (x_i,y_i), for which $\alpha_{*i} > 0$ is called a support vector. Going back to the idea of maximizing the margin, the support vectors are precisely those points that are at a minimum distance from the hyperplane. In other words, these are the points on the margin. Looking at Figure 1(a), the support vectors are the points for which there is an arrow between them and the line. We can classify any point using only the support vectors: To classify, use signf(x) where

$$f(x) = \sum_{i:\alpha_{*i}>0} y_i \alpha_{*i} (x_i \cdot x) + b_* \tag{8}$$

The sum is over support vectors. Note that it requires computing the inner product of each support vector with the instance to be classified. The classification process compares the new instance with each of the support vectors. $(x_i \cdot x)$ measures how similar the new instance x is to the training instance x_i. α_{*i} measures the contribution of x_i, i.e. α_{*i} measures how important the given support vector is. We multiply by y_i to take into account the influence of the given support vector on the classification.

3 Kernels

So far, the formulation of SVMs has relied on finding a linear separator. But in general the data may not be linearly separable. It may be the case, however, that if we map the data into a higher dimensional space it will be linearly separable. Kernels are a clever way of mapping data into a higher dimensional feature space.

First, notice the form of the dual problem. It contains instances of the data multiplied with each other through an inner product. This inner product between instances allows us to use a kernel function.

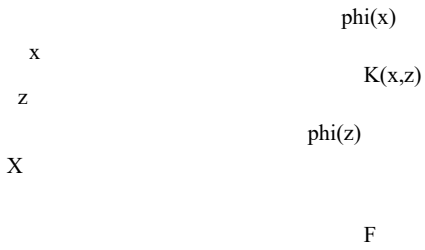

Figure 2: Picture of a kernel function

Definition 3.1: A kernel is a function K such that $K(x, z) = (\phi(x) \cdot \phi(z))$, where ϕ is a mapping from X to an inner product feature space F.

Figure 2 illustrates this definition. ϕ maps points in the input space X into the feature space F. So ϕ takes x and produces $\phi(x)$, and similarly takes z and produces $\phi(z)$. Now we can take inner products in F — as I said, the inner product is a measure of the distance between points. But with a kernel function, we don't actually have to do the mapping and compute the inner product in F; we can compute the kernel function K directly on points in X. So when we compute K(x, z), that immediately tells us the inner product between $\phi(x)$ and $\phi(z)$ — but we don't actually have to compute $\phi(x)$ and $\phi(z)$ to compute K(x, z). This is the important property of kernels — we can map inputs into a higher dimensional feature space, without doing all the computations in that higher dimensional space.

Here are some examples of kernel functions:

Example 3.2:

$$K(x, z) = (x \cdot z)$$

This is the trivial kernel function. The input space X and the feature space F are the same, and the kernel simply computes the inner product in X.

Example 3.3: Let X be R_n, and F be R_{n^2}. $\phi(x)$ maps a point $x_1, ..., x_n$ to $x_1^2, x_1x_2, x_1x_3, ..., x_1x_n, x_2x_1, ..., x_2x_n, ..., x_2^n$

$$\begin{aligned} K(x, z) &= ((x \cdot z))^2 \\ &= \left(\sum_{i=1}^{n} x_i z_i\right)^2 \\ &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_j z_j\right) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j z_i z_j \\ &= \sum_{(i,j)=(1,1)}^{(n,n)} (x_i x_j)(z_i z_j) \\ &= (\phi(x) \cdot \phi(z)) \end{aligned}$$

We see here the advantage of a kernel. Instead of having to compute the entire feature map (which is quadratic in this case) and then computing the inner product directly, you only have to compute the kernel function, which is $((x \cdot z))^2$, and takes linear time.

Example 3.4: The previous example generalizes to polynomial kernel functions:

$$K(x, z)=((x \cdot z) + c)^d$$

Example 3.5:

The Guassian kernel is

$$K(x, z) = e^{(-\|x-z\|^2/\sigma^2)}$$

While this doesn't look much like an inner product, it is a legal kernel. It compares the distances between two points, with the importance of z to x

decaying exponentially with the distance from z to x.

A key point is that we can describe this kernel without describing the inner product space explicitly. When we come to computing with this kernel, we will be comparing training instances and computing their distance from each other.

The bottom line is that you can plug the kernel function in wherever the inner product appears earlier. So you can use it to classify as in Equation 8. It is used to define the program expressed in Program 6 and to compute the solution as in Equations 7 and ??. Because it is a kernel function, you know it is an inner product in some space. That means that you know you are solving the optimization problem to find the maximum margin classifier in some high-dimensional space.

4 Properties of SVMs

- Accurate classification

- With appropriate kernel function, can express very complex hypotheses
- No problem with local minima (quadratic objective function)
- Hypothesis directly represented as set of support vectors: independent of all other non-support vectors. This means that points that are classified very well have no impact on the decision boundary. Is this a good or a bad thing?
- Very slow training. This is particularly a problem for large training sets. The objective function to the quadratic program contains the inner product of all pairs of instances.
- Relatively slow classification. We need to compare a point to all the support vectors. In high dimensions there may be many.
- Exactly separating points in a high-dimensional space can lead to strange looking decision boundaries. Can this lead to overfitting?

What if the training dats is not linearly separable, even with a high dimen-

sional feature space? The method still tries to maximize the margin, which means making the worst misclassified points as least bad as possible. There are also “soft margin” methods that optimize the distribution of the marginas while still allowing for points to be misclassified.