CS269: Machine Learning Theory
Lecture 16: SVMs and Kernels
November 17, 2010

Lecturer: Jennifer Wortman Vaughan
Scribe: Jason Au, Ling Fang, Kwanho Lee

Today, we will continue on the topic of support vector machines, finishing our derivation of the dual optimization problem. Next, we will go over how to solve the dual optimization and examine the case in which the data is not linearly separable. For dealing with such a case, we introduce two approaches: the Soft Margin Approach and the Kernel Trick.

1 The Primal and Dual Problems

When we talked about the analysis of boosting, we found that having a large margin led to small generalization error. In the last lecture, we showed that we could maximize the margin directly by framing the maximization as a convex optimization problem. In section 1.1, we will derive the dual of this problem, which is more convenient to compute and can be used with kernels.

In the previous lecture, we introduced the standard form of the optimization problem:

$$\begin{aligned} \min_{\vec{w},b} \quad & \frac{1}{2}||\vec{w}||^2 \\ \text{s.t.} \quad & 1 - y_i(\vec{w} \cdot \vec{x}_i + b) \leq 0, i = 1, ..., m \end{aligned}$$

In the previous lecture, we showed that solving a convex optimization was equivalent to solving the following optimization of the Lagrangian function:

$$\min_{\vec{w},b} \max_{\vec{\alpha};\alpha_i \geq 0} L(\vec{w}, b, \vec{\alpha}) \qquad \text{where L is the Lagrangian function}$$

We dropped the $\vec{\beta}$ in the Lagrangian function from the previous lecture because $\vec{\beta}$ is the vector of the equality constraints, but here, we are only dealing with inequalities, which corresponds to $\vec{\alpha}$.

We previously showed that the solution to the primal problem is equivalent to the solution to the dual problem if they satisfy the following primal-dual equivalence conditions. First, we need a convex objective function and in our case, it is $\frac{1}{2}||\vec{w}||^2$. Second, we need convex inequality constraints g_i, which are $1 - y_i(\vec{w} \cdot \vec{x}_i + b)$ for $i = 1, ..., m$. The last condition states that for each inequality g_i, there exists some value of the variables that make g_i strictly less than 0. Since these conditions are satisfied for the optimization problem we defined, we know that we can solve the dual version of the problem instead of the primal problem and the solutions will be equivalent.

1.1 Deriving The Dual Problem

The dual problem is defined as follows:

$$\max_{\vec{\alpha};\alpha_i \geq 0} \min_{\vec{w},b} L(\vec{w}, b, \vec{\alpha})$$

We will derive a more convenient form in three steps. The first step is to calculate the value of the Lagrangian function by plugging in our objective function and constraints:

$$\begin{aligned} L(\vec{w}, b, \vec{\alpha}) = & \underbrace{f(\vec{w}, b)}_{\text{objective function}} + \sum_{i=1}^{m} \alpha_i \underbrace{g_i(\vec{w}, b)}_{\text{inequality constraints}} \\ & = \frac{1}{2}||\vec{w}||^2 + \sum_{i=1}^{m} \alpha_i (1 - y_i(\vec{w} \cdot \vec{x}_i + b)) \end{aligned}$$

The next step is to find the minimum value of the Lagrangian. Because the Lagrangian is convex, the minimum occurs when the partial derivatives are equal to 0. Thus, we can derive the following:

$$\begin{aligned} \nabla_{\vec{w}} L = 0 \rightarrow \vec{w} - \sum_{i=1}^{m} \alpha_i y_i \vec{x}_i = 0 \rightarrow \vec{w} = \sum_{i=1}^{m} \alpha_i y_i \vec{x}_i \\ \frac{\partial L}{\partial b} = 0 \rightarrow \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned}$$

By plugging the values into our Lagrangian equation which we derived in the first step, we can verify:

$$\min_{\vec{w},b} L(\vec{w}, b, \vec{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \vec{x}_i \cdot \vec{x}_j$$

By maximizing this expression with respect to $\vec{\alpha}$ and applying the constraint we derived from the partial derivatives, we obtain the dual form of the problem.

$$\begin{aligned} \max_{\vec{\alpha}} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \vec{x}_i \cdot \vec{x}_j \\ \text{s.t.} \quad & \alpha_i \geq 0 \\ & \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned}$$

This version of the problem is easy to solve. Additionally, it only depends on the $\vec{x}_i$ through dot products of the form $\vec{x}_i \cdot \vec{x}_j$, so it can be used with the kernel trick, which we will introduce in the last section.

2 KKT Conditions

In the last class, we stated the KKT conditions. Recall that when the primal-dual equivalence conditions we mentioned earlier are satisfied, we are guaranteed that some set of values $\vec{w}^*, b^*, \vec{\alpha}^*$, is a solution to both the primal and dual problems if and only if the KKT conditions are met. For our problem, the KKT conditions imply the following restrictions on solutions:

1. Stationarity:

   From taking the partial derivatives above, we know this gives us:

   $$\vec{w}^* = \sum_{i=1}^{m} \alpha_i^* y_i \vec{x}_i$$
   $$\sum_{i=1}^{m} \alpha_i^* y_i = 0$$

   Therefore, by plugging in the value of $\vec{w}^*$, we can derive that the linear separator we output must be of the form

   $$h(\vec{x}) = \text{sign}(\vec{w}^* \cdot \vec{x} + b^*) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i^* y_i \vec{x}_i \cdot \vec{x} + b^*\right)$$

   Again, evaluating $h(\vec{x})$ depends only on the dot product $\vec{x}_i \cdot \vec{x}$, but does not depend on $\vec{x}_i$ or $\vec{x}$ in any other way. This will be important when we discuss kernels.

2. Primal Feasibility:

   The first primal feasibility condition we mentioned in the previous lecture is irrelevant here because there are no equality constraints for the original primal problem. The primal feasibility condition for the inequality constraints is:

   $$y_i(\vec{x}_i \cdot \vec{w}^* + b^*) \geq 1, \text{ for } i = 1, ..., m$$

   This ensures a margin of $\frac{1}{\|w^*\|}$.

3. Dual Feasibility:

   This constraint is the same as in the dual problem.

   $$\alpha_i^* \geq 0, \text{ for } i = 1, ..., m$$

4. Complementary Slackness:

   The implications of this condition are rather interesting. It implies that

   $$\alpha_i^*(y_i(\vec{w}^* \cdot \vec{x}_i + b) - 1) = 0, \text{ for } i = 1, ..., m$$

This means that for any input point $\vec{x}_i$, at least one of the primal or dual constraint is tight, i.e., $\alpha_i = 0$ or $y_i(\vec{x}_i \cdot \vec{w}^* + b^*) = 1$. When $y_i(\vec{x}_i \cdot \vec{w}^* + b^*) > 1$, its distance from the separator is greater than our minimum margin, and this condition ensures that $\alpha_i = 0$ so that point $\vec{x}_i$ does not contribute to our weight vector. On the other hand, if α_i is non-zero, $\vec{x}_i$ contributes to our weight vector, so its margin has to be minimum; we call these points the support vectors.

The number of support vectors is typically much smaller than the number of data points m.

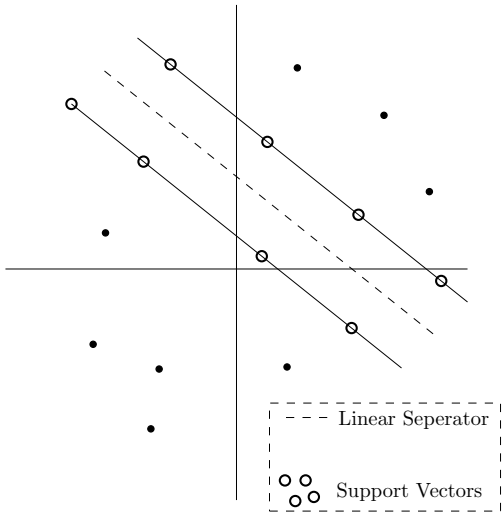

Figure 1: Only the support vectors contribute to our weight vector. If $\vec{x}_i$ is a support vector, we have α_i nonzero and $y_i(\vec{x}_i \cdot \vec{w}^* + b^*) = 1$

3 Solving The Dual Problem

There are many different algorithms that can be used to solve the dual problem. We can use any type of hill climbing technique. One popular technique is the Sequential Minimization Optimization (SMO) algorithm, which uses simple coordinate ascent with a few constraints that need to be worked around.

Suppose we want to solve an *unconstrained* optimization problem of the form

$$\max_{\vec{\alpha}} f(\vec{\alpha}) \qquad \text{where } f \text{ is concave.}$$

We could solve this problem by running a simple Coordinate Ascent algorithm. While there are many ways to specify the details of Coordinate Ascent, the outline of the algorithm is as follows:

initialize $\vec{\alpha}$
repeat until convergence
    choose some i
    set $\alpha_i = \underset{\hat{\alpha}}{argmax}\ f(\alpha_i, ..., \alpha_{i-1}, \hat{\alpha}, \alpha_{i+1}, ..., \alpha_m)$

Unfortunately, we can't apply Coordinate Ascent directly to our problem because of the stationarity condition we described above, which tells us that

$$\alpha_i = -\sum_{j \neq i}^{m} \alpha_j y_i y_j$$

We cannot just change one coordinate and keep this constraint satisfied. Using Sequential Minimization Optimization, we choose a *pair* of α_i points every time step and optimize them *together* while maintaining the constraint. We can easily test for convergence by checking if the KKT conditions are satisfied up to some tolerance parameter.

4 The Non Linearly Separable Case

Previously, we only looked at the case were the data was linearly separable. We can modify our problem so that the optimization still returns a reasonable linear separator when data is not strictly (but close to) linearly separable.

4.1 The Soft Margin Approach

In this approach, we introduce "slack variables" $\mu_1, \cdots, \mu_m$ to relax the margin constraint, not requiring the margin constraint to hold perfectly on every data point. Our optimization problem becomes:

$$\begin{aligned} \min_{\vec{w},b,\vec{\mu}} \quad & \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{m} \mu_i \\ \text{s.t.} \quad & y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \mu_i \text{ for } i = 1, ..., m \\ & \mu_i \geq 0 \text{ for } i = 1, ..., m \end{aligned}$$

where C is a parameter of the algorithm. This new optimization problem lets us occasionally fail to meet the margin condition, but we pay in the $C\sum_{i=1}^{m} \mu_i$ term when this happens.
It is easy to see that when the optimal solution is found, we have

$$\mu_i = \max(0, 1 - y_i(\vec{w} \cdot \vec{x}_i + b))$$

This is precisely the hinge loss for the ith data point! We can think of this problem as minimizing a weighted combination of loss and a regularization term, as we have done before. In fact, $||w||^2$ is the same regularizer we discussed when we talked about online gradient descent. However, now it is hinge loss that we are minimizing.

We can derive a new dual optimization from the SVM with soft margin:

$$\begin{aligned} \max_{\vec{\alpha}} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \vec{x}_i \cdot \vec{x}_j \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^{m} \alpha_i y_i = 0 \end{aligned}$$

The derivation is similar to the derivation above, and the dual problem itself is similar to the dual we derived without the slack variables. We now have the additional constraint of $\alpha_i \leq C$, but this problem is just as easy to solve. This is good news! To handle data that isn't linearly separable, we only have to slightly modify our original problem and the same hill climbing techniques work.

This dual optimization problem is what people typically mean when they talk about the SVM algorithm. This will give good results if the problem is close to being linearly separable.

5 The Kernel Trick

Although the soft margin approach may work for data which is close to linearly separable, it returns poor results when data is not close, such as in the following case:

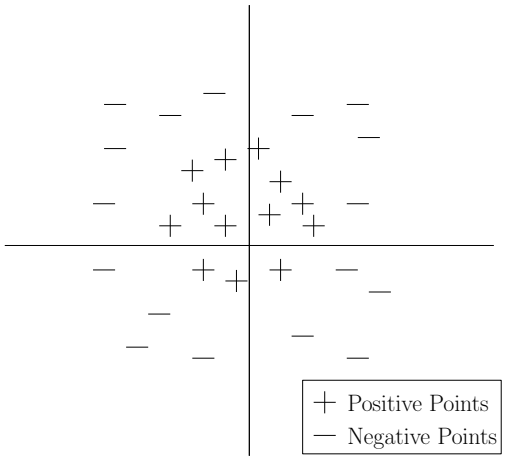

Figure 2: There is no linear separater that can approximately distinguish the positive data points from the negative data points.

Although this data is clearly not linearly separable, we can *transform* it into something that is. In particular, suppose that we transformed this data from its current form

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

to the following form, using the feature mapping function ϕ.

$$\phi(\vec{x}) = \begin{bmatrix} {x_1}^2 \\ {x_2}^2 \\ x_1 x_2 \end{bmatrix}$$

This new feature space is close to linearly separable! So we could run the SVM algorithm on these new features instead.

5.1 Kernel Functions

Given a mapping ϕ, we define the corresponding **Kernel function** K on data points $\vec{x}$ and $\vec{z}$ as:

$$K(\vec{x}, \vec{z}) = \phi(\vec{x}) \cdot \phi(\vec{z})$$

Instead of applying SVMs using the original data point $\vec{x}$, we may want to learn using $\phi(\vec{x})$. To do so, we simply replace each $\vec{x}$ with $\phi(\vec{x})$. Since the dual SVM problem can be written in terms of the dot product of data points, we can replace this with the Kernel function of those data points. Since both the SVM optimization problem and the resulting classifier only handle data through dot products of data points, even if computing $\phi(\vec{x})$ is inefficient, the SVM algorithm can still be run efficiently as long as we can compute the kernel function efficiently.

As an example, consider the kernel function $K(\vec{x}, \vec{z}) = (\vec{x} \cdot \vec{z})^2$. We can calculate this in time linear in the dimension of $\vec{x}$. We have that

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

where ϕ is the extension of the transformation we described above to n dimensions. That is,

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \rightarrow \phi(\vec{x}) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ \vdots \\ x_n x_{n-1} \\ x_n x_n \end{bmatrix}$$

We can consider another case involving a constant c:

$$\begin{aligned} K(\vec{x}, \vec{z}) &= (\vec{x} \cdot \vec{z} + c)^2 \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) + \sum_{i=1}^{n} (\sqrt{2c} x_i)(\sqrt{2c} z_i) + c^2 \end{aligned}$$

This corresponds to the following feature mapping, which includes linear terms in addition to quadratic terms.

$$\phi(\vec{x}) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ \vdots \\ x_nx_{n-1} \\ x_nx_n \\ \sqrt{2c}x_1 \\ \vdots \\ \sqrt{2c}x_n \\ c \end{bmatrix}$$

Instead of computing either version of $\phi(\vec{x})$, which takes $O(n^2)$ time, we can compute their corresponding Kernels $K(\vec{x}, \vec{z})$, taking only $O(n)$ time.

Generalizing further, the Kernel $K(\vec{x}, \vec{z}) = (\vec{x} \cdot \vec{z} + c)^d$ corresponds to a feature mapping to an $\binom{n+d}{d}$ feature space. However, despite working in an $O(n^d)$ dimensional space, computing the Kernel only takes $O(n)$ time.

There are kernels that correspond to infinite dimensional feature vectors that we can still work with efficiently (such as the radial basis function kernel). Running the SVM directly on these feature vectors will be impossible since the vectors are infinite, but we can still do it efficiently using the kernel trick.

5.2 A Note on Generalization Error

We might expect it to be a bad idea to use what are effectively infinite-dimensional feature vectors since the class of functions we are learning will become very complex, but in fact, this works well in practice. It can be shown that only the margin and number of support vectors matter.