CS260: Machine Learning Theory
Lecture 16: SVMs and Kernels
November 16, 2011

Lecturer: Jennifer Wortman Vaughan

1 Examining the Dual

In the last lecture, we derived a primal standard form convex optimization problem that can be solved to obtain the threshold function $h(\vec{x}) = \text{sign}(\vec{w} \cdot \vec{x} + b)$ that maximizes the margin of a set of data points $(\vec{x}_1, y_1), \cdots, (\vec{x}_m, y_m)$:

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

We also derived the dual of this problem:

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

Since the primal problem is convex (i.e., has a convex objective function and convex inequality constraints), we know that solving the primal is equivalent to solving the dual. We also know that any optimal solution $\vec{w}^*$, b^*, α^* must satisfy the KKT conditions. Let's see what the KKT conditions tell us about the solutions.

- **Stationarity:**

  Stationarity tells us that the partial derivatives of the Lagrangian with respect to $\vec{w}$ and b must be 0 at the optimal solution. As we saw in the last lecture, this implies that

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

  and

  $$\sum_{i=1}^{m} \alpha_i^* y_i = 0 \ .$$

All CS260 lecture notes build on the scribes' notes written by UCLA students in the Fall 2010 offering of this course. Although they have been carefully reviewed, it is entirely possible that some of them contain errors. If you spot an error, please email Jenn.

The first equation here tells us how to back out the optimal value $\vec{w}^*$ of the weights $\vec{w}$ once we have solved the dual to obtain the optimal value $\vec{\alpha}^*$ for $\vec{\alpha}$. In particular, the optimal weight vector will be a weighted combination of input points, similar to the weights output by the Perceptron algorithm.

(In order to obtain our classifier, we will need to back out the optimal value of b as well. Let's hold off on the question of how to do this for a moment, and examine what the other KKT conditions tell us about the solution to this problem...)

- **Primal Feasibility:**

  Primal feasibility tells us that all primal constraints must be satisfied. In this case, it implies that for all data points $i \in \{1, \cdots, m\}$,

  $$y_i(\vec{x}_i \cdot \vec{w}^* + b^*) \geq 1 \ ,$$

  or

  $$y_i \frac{\vec{x}_i \cdot \vec{w}^* + b^*}{||\vec{w}^*||} \geq \frac{1}{||\vec{w}^*||} \ .$$

  In other words, we must have a margin of at least $1/||w^*||$ on the training data.

- **Dual Feasibility:**

  Primal feasibility tells us that all dual constraints must be satisfied, or for all $i \in \{1, \cdots, m\}$, $\alpha_i^* \geq 0$. This makes sense. It tells us that our weight vector $\vec{w}^*$ will only ever put positive or zero weight on positively labeled points, and negative or zero weight on negatively labeled points.

- **Complementary Slackness:**

  The implications of this condition are rather interesting. In particular, we get that for all $i \in \{1, \cdots, m\}$,

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

  Stated another way, for every input point $\vec{x}_i$, at least one of the corresponding primal constraint and the corresponding dual constraint must be tight, i.e., we must have either $\alpha_i = 0$ or $y_i(\vec{x}_i \cdot \vec{w}^* + b^*) = 1$.

  This implies that the only input points that contribute to the weight vector are those with minimal margin, i.e., those points x_i such that $y_i(\vec{x}_i \cdot \vec{w}^* + b^*) = 1$. These points are referred to as *support vectors*. The number of support vectors is typically much smaller than the number of data points m, as in Figure 1.

  Note that this also gives us a way to solve for the optimal value b^*. We know that for any data point x_i with $\alpha_i > 0$, we must have

  $$b^* = 1/y_i - \vec{x}_i \cdot \vec{w}^* = y_i - \vec{x}_i \cdot \vec{w}^*.$$

  In practice, different points x_i might give slightly different values of b^*, so it is common to average over all of the support vectors.

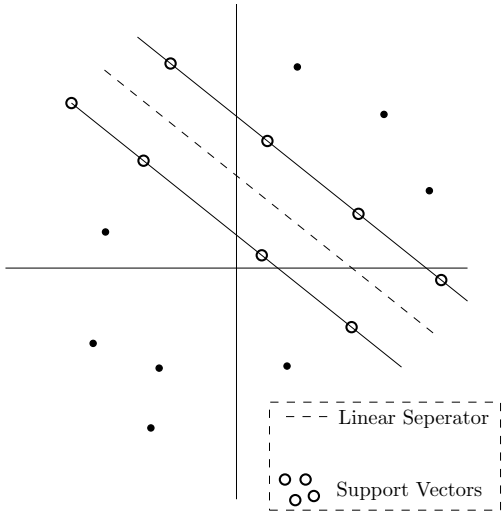

Figure 1: The support vectors are the input points with minimal margin. Only these points contribute to the weight vectors $\vec{w}^*$.

2 Solving The Dual Problem

Many different techniques can be used to solve the dual problem. We briefly discuss the main idea behind one common hill-climbing technique, Sequential Minimization Optimization (SMO). For full details on this algorithm, check out John Platt's book chapter, available for free online. [1]

First, suppose we want to solve an *unconstrained* optimization problem of the form $\max_{\vec{\alpha}} f(\vec{\alpha})$ for some concave function f. A common technique for doing this is to use a coordinate ascent algorithm. There are different ways to specify the details of coordinate ascent, but the general outline is as follows:

Initialize $\vec{\alpha}$
Repeat until convergence
    Choose some i
    Set $\alpha_i = \arg\max_{\hat{\alpha}} f(\alpha_1, ..., \alpha_{i-1}, \hat{\alpha}, \alpha_{i+1}, ..., \alpha_m)$

Unfortunately, we can't apply coordinate ascent directly to the dual problem because of the stationarity condition which tells us that

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

This condition tells us that there is only one possible value for the ith component of $\vec{\alpha}$ given all of the other components, so we can't optimize components separately.

The SMO algorithm gets around this problem by choosing a ***pair*** of components, α_i and α_j, at each time step, and optimizing them ***together*** while maintaining the stationarity constraint. We can easily test for convergence by checking if the KKT conditions are satisfied up to some tolerance parameter.

[1] `http://research.microsoft.com/apps/pubs/?id=68391`

3 No Perfect Target

So far we have assumed that the data is linearly separable, i.e., that it can be classified perfectly by a linear separator. It turns out that it is easy to modify the optimization so that it still gives us something reasonable even when there is no perfect target.

3.1 The Soft Margin Approach

In the soft margin approach, we introduce a set of "slack variables" $\mu_1, \cdots, \mu_m$, to relax the margin constraint. Our optimization problem becomes:

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

where C is a parameter of the algorithm. This allows for occasional failure of the margin condition (i.e., data points for which $y_i(\vec{w}\cdot\vec{x}_i + b) < 1$), but we pay a price in the objective function each time 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 (a scaled version of) the hinge loss for the ith data point. We can therefore think of this problem as minimizing a weighted combination of a loss term and a regularization term, as we have done before. However, now it is hinge loss that we are minimizing.

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

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

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 constraints that $\alpha_i \leq C$, 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 soft max dual optimization problem is what people typically mean when they refer to the SVM algorithm. This will give good results if the data is "almost" linearly separable.

4 The Kernel Trick

Although the soft margin approach works for data that is close to linearly separable, it returns poor results when data is not close, such as in Figure 2.

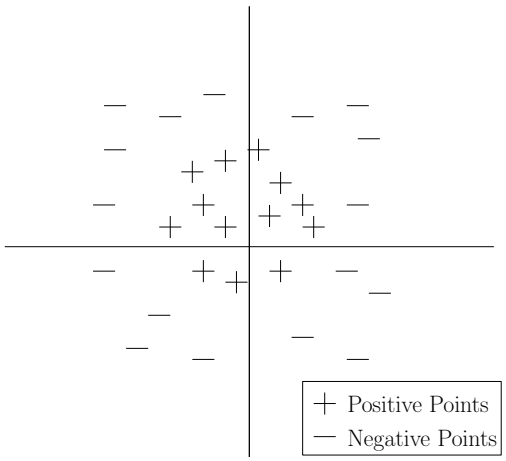

Figure 2: There is no linear separator that can approximately distinguish the positive data points from the negative data points.

Although this data is clearly not linearly separable, we can *transform* it into something that is. In particular, suppose that we represent a data point $\vec{x}$ as

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

We could (almost) separate this data with a function of the form

$$(x_1 - p)^2 + (x_2 - q)^2 \leq r^2 ,$$

or

$$x_1^2 - 2px_1 + x_2^2 - 2qx_2 \leq r^2 - p^2 - q^2 .$$

Suppose we transformed the original data point to the following form, using the feature mapping function ϕ:

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

We can see that this new feature space is (close to) linearly separable! So we could run the SVM algorithm on these new features instead.

4.1 Kernel Functions

Suppose that we wanted to apply the SVM algorithm to transformed data points $((\phi(\vec{x}_1), y_1), \cdots, (\phi(\vec{x}_m), y_m))$ for some mapping ϕ. Let's think about how the optimization would change. We'll consider the basic version for simplicity, but the same ideas apply for the soft margin version.

The dual optimization problem depends on the input points only through the dot products $\vec{x}_i \cdot \vec{x}_j$. We could replace these terms with $\phi(\vec{x}_i) \cdot \phi(\vec{x}_j)$.

What about our final classifier? The label of a new point $\vec{x}$ can be classified as

$$\begin{aligned}\text{sign}(\vec{w}^* \cdot \phi(\vec{x}) + b^*) &= \text{sign}\left(\sum_{i=1}^{m} \alpha_i^* y_i \phi(\vec{x}_i) \cdot \phi(\vec{x}) + y_\ell - \vec{w}^* \cdot \phi(\vec{x}_\ell)\right) \\ &= \text{sign}\left(\sum_{i=1}^{m} \alpha_i^* y_i \phi(\vec{x}_i) \cdot \phi(\vec{x}) + y_\ell - \sum_{i=1}^{m} \alpha_i^* y_i \phi(\vec{x}_i) \cdot \phi(\vec{x}_\ell)\right)\end{aligned}$$

for some support vector $\phi(\vec{x}_\ell)$. So in order to classify our data points, we also need only to be able to compute the dot product between pairs of transformed vectors.

It turns out that this observation is very powerful, and allows us to work with very large or even infinite vectors of transformed features. Given a mapping ϕ, we define the corresponding **Kernel function** K on data points $\vec{x}$ and $\vec{z}$ as

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

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.

We'll see some examples of kernels in the next lecture.