Lecture 14: Support Vector Machines (Kumar Chapter 8)

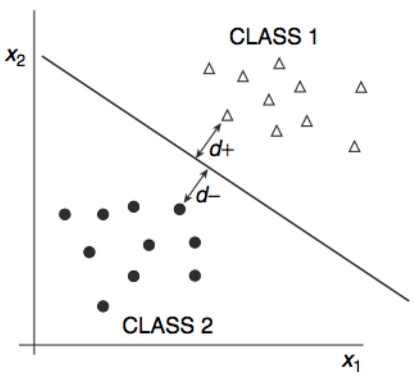
CS 167: Machine Learning

Support Vector Machines

Let's stop and look at some different decision surfaces on the white board.

Support Vector Machine Idea

Support Vector Machine Idea:
We want the separating hyperplane with the *maximum* possible *distance* to the nearest positive and negative data points



d_+ and d_- are referred to as the *margin*

Some Notation

We sometimes might write our inputs and weights as *vectors*

$$X = x_1, x_2, \ldots, x_n$$

$$W = w_1, w_2, \ldots w_n$$

The *dot product* of these two vectors is

$$W \cdot X = w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

So then the equation of a hyperplane would be

$$W \cdot X + w_0$$

Hey, this is a lot like Perceptrons

For a training example X_i, t_i (attributes X_i, target function t_i)
With a Perceptron unit,

$$\begin{aligned} W \cdot X_i + w_0 > 0, &\quad t_i = +1 \\ W \cdot X_i + w_0 < 0, &\quad t_i = -1 \end{aligned}$$

For SVM, we can choose W and w_0 so that

$$\begin{aligned} W \cdot X_i + w_0 \geq +1, &\quad t_i = +1 \\ W \cdot X_i + w_0 \leq -1, &\quad t_i = -1 \end{aligned}$$

or rewriting it as one line:

$$t_i(W \cdot X_i + w_0) - 1 \geq 0$$

Norms

Recall our hyperplane:

$$W \cdot X + w_0$$

The W vector is *normal* (perpendicular) to the hyperplane

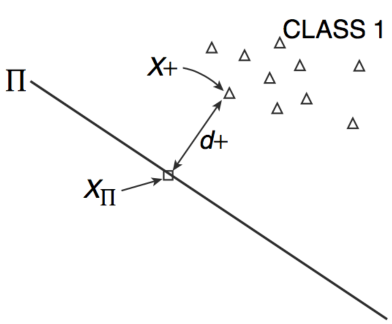
It's norm/length is

$$\|W\| = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2}$$

Distance between training points and hyperplane

Π: the hyperplane separator
X_+: closest training example on the positive side
X_Π: point on hyperplane closest to X_+

so
$W \cdot X_+ + w_0 = 1$
$W \cdot X_\Pi + w_0 = 0$



and we can then show that

$d_+ = X_+ - X_\Pi = \frac{1}{\|W\|}$

similarly, $d_- = \frac{1}{\|W\|}$

Do we want $\|W\|$ to be big or small?

The resulting optimization problem

That leaves us with the following optimization problem:

Minimize $\frac{1}{2}\|W\|^2$ (squaring to get rid of the nasty square root)

subject to the constraints

$$\begin{gathered} t_1(W \cdot X_1 + w_0) - 1 \geq 0 \\ t_2(W \cdot X_2 + w_0) - 1 \geq 0 \\ t_3(W \cdot X_3 + w_0) - 1 \geq 0 \\ \cdots \\ t_N(W \cdot X_N + w_0) - 1 \geq 0 \end{gathered}$$

for all N training examples

There's a whole field of mathematics/computer-science dedicated to solving *optimization problems*, and this one is actually *difficult* because of the quadratic terms - even after we get rid of that square root

Training and Prediction

We end up introducing some α_i multipliers so that

$$W = \sum_{i=1}^{N} \alpha_i t_i X_i$$

and transform to a *dual form* that looks like this:
Maximize (in α_i)

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j (X_i \cdot X_j)$$

s.t.

$$\alpha_i \geq 0, \qquad \sum_{i=1}^{N} \alpha_i t_i = 0$$

And our predictor is:

$$h(X) = sign\left(\sum_{i=1}^{N_s} t_i \alpha_i (X \cdot X_i) + w_0\right)$$

(and α is 0 for non-support vectors, so you don't have to keep those ones)

Kernel Tricks

Do we have to explicitly come up with the new features for kernel tricks ourselves? No

We replace the dot product in our training and prediction algorithms with a *kernel function* $K(X, Y)$ and work implicitly in the remapped space:
Maximize (in α_i)

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j K(X_i, X_j)$$

s.t.

$$\alpha_i \geq 0, \qquad \sum_{i=1}^{N} \alpha_i t_i = 0$$

And our predictor is:

$$h(X) = sign\left(\sum_{i=1}^{N_s} t_i \alpha_i K(X, X_i) + w_0\right)$$

Some common kernels

Polynomial:

$$K(X, Y) = (X \cdot Y + r)^d \quad \text{(for } d,\ r \text{ constants)}$$

Gaussian radial basis function:

$$K(X, Y) = \exp(-\gamma \|X - Y\|^2) \quad \text{(for } \gamma > 0\text{)}$$

Hyperbolic tangent (or sigmoid):

$$K(X, Y) = \tanh(\gamma X \cdot Y + r) \quad \text{(for } \gamma > 0,\ r < 0\text{)}$$

Softening the margin

If kernel still doesn't separate, you can add in *slack* variables to all this that *soften* the margin. End up basically bounding α_is by constant C:
Maximize (in α_i)

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j t_i t_j K(X_i, X_j)$$

s.t.

$$0 \leq \alpha_i \leq C, \qquad \sum_{i=1}^{N} \alpha_i t_i = 0$$

Large C: more consistent with training set
Small C: smoother decision boundary

Exercise

Determine the answer to these questions:

- Does `scikit-learn` have a support vector machine classifier?
- If so, can I tweak the value of C?
- If so, which kernels are supported?
- If any, which is the default?
- If so, can I create my own custom kernel?
- If so, can I tweak the values of γ, d, r, etc.?

Exercises:

- How does the linear kernel (i.e., just dot product) do with the iris data?
- Does it work well for the quadratic kernel (i.e., polynomial kernel with $d = 2$, $r = 1$)?
- How about on the LFW data set?