Classification (cont.), SVMs

Somani, Neel - Bao, Jason

March 20th, 2018

1 Logistic Regression Review

During last lecture, we derived logistic regression from MLE. In the process, we assumed:

$$Pr[Y = 1|X = x] = S(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

This isn't a perfect assumption, but it's generally pretty good.

We ultimately found the loss function:

$$J(w) = -\sum_{i=1}^{n} (y_i * log(s(w^T x)) + (1 - y_i) * log(S(1 - w^T x)))$$

It just so happens that this function is convex, as covered in earlier lectures. This means that the loss function is "bowl shaped." Conveniently, we can find the minimum of the function by repeatedly moving in the direction of steepest descent.

1.1 Application of Gradient Descent

The direction of steepest descent can be found by moving in the opposite direction of the derivative, or more generally, the opposite direction of the gradient. The *gradient* of a function represents how it changes with respect to each of its inputs. In this case, we want to represent how J changes with respect to the components of w. Note that $S'(x) = S(x) * (1 - S(x))$.

$$s_i := S(w^T x)$$

$$\nabla_w J = -\sum_{i=1}^{n} (\frac{y_i}{s_i} \nabla s_i - \frac{(1 - y_i)}{(1 - s_i)} \nabla s_i)$$

where $\nabla log(s_i) = \frac{\nabla s_i}{s_i}$ by the chain rule.

$$= -\sum_{i=1}^{n}(\frac{y_i}{s_i} - \frac{(1-y_i)}{(1-s_i)}) * s_i * (1-s_i) * X_i$$

$$= -\sum_{i=1}^{n}(y_i - s_i) * X_i$$
$$= -X^T(y - S(Xw))$$

We want to move w in the opposite direction of this gradient:

$$w := w - \epsilon \nabla_w J(w)$$

where we choose different values of ϵ to find the best one.

2 Cross Entropy

Cross entropy is a notion of how similar two distributions are. It's defined as:

$$H(p,q) = -\sum_x p(x) * log(q(x))$$

You can find that:

$$p = argmin_q H(p,q) = -\sum_x p(x) * log(q(x))$$

So rather than using something like Euclidean distance, like when finding the similarity between vectors, we use cross entropy to find the similarity between two probability distributions.

How can we use this to help us classify? Maybe we could have a binary classifier for each class, to determine the probability that a given element is in a class, versus not in a class.

One alternative is the *softmax* function:

$$P(y = z | X = x_j) = \frac{e^{w_z^T x_j}}{\sum_{i=1}^{k} e^{w_i^T * x_j}}$$

This defines a distribution over k classes. Notice that it's always greater than or equal to 0, and that the possible values of this distribution sum to 1.

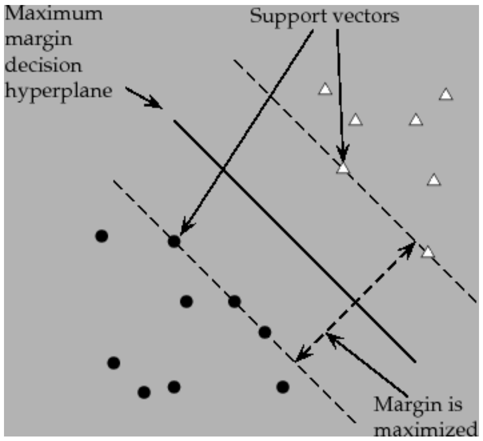
The softmax loss function:

$$-\sum_i (y_{ij} * log(\hat{y}_i))$$

where $y_{ij} = 1$ if x_j has label i, and $y_{ij} = 0$ otherwise. $\hat{y}_i$ is our guess of $P(Y|X)$.

3 Support Vector Machines

Consider a classifier that separates two sets of points with a line (in 2D) or with a plane (in 3D). If a point is above the line (the plane), then the classifier will classify that point to group A, and otherwise the point will be classified as group B. Intuitively, we want to pick the line (or the plane) that is as far as possible from all of the points that we're classifying.



To make this rigorous, we might point out that the line that maximizes the above margin is also the most "robust" to noise, meaning that if all of the points were perturbed by some amount, then this line is least likely to misclassify points.

To make the concepts of line or plane more general, we'll define the concept of a *hyperplane*. A hyperplane in dimension N is a subspace of dimension $N-1$ (e.g., a line is a 2D subspace in a 3D space). So we want to find a *hyperplane* that maximizes the distance from all of the points that it's classifying. To do so, we need to find the distance from a point to a hyperplane.

Let's consider the set of hyperplanes in R^2.

$$\{x : w^T x + \alpha = 0\}$$

First, let's note that w is orthogonal to the plane. Proof: consider two points x_0 and x_1 on the plane. The difference $x_1 - x_0$ is a vector on the plane. Then:

$$w^T(x_1 - x_0) = w^T x_1 - w^T x_0 = 0$$

Next lecture, we'll discuss how to use this information to derive the support vector machine.