Perceptron for Imbalanced Classes and Multiclass Classification

Piyush Rai
(CS5350/6350: Machine Learning)
November 1, 2011

We have already seen Perceptron for binary classification. In this note, we will see how we can modify the Perceptron algorithm for (1) dealing with the case when the classes are imbalanced (assuming binary classification), and (2) dealing with the case of multiclass classification.

1 Perceptron for Imbalanced Classes

For the imbalanced classes case, we will use a variance of the Perceptron algorithm with margins. Recall from homework 2 that the Perceptron with margins algorithm simply replaces the standard Perceptron mistake/update condition $y_n(w^\top x_n + b) < 0$ by $y_n(w^\top x_n + b) < \gamma$ where $\gamma$ is the minimum margin we want to have for each example (it’s a hyperparameter we have to give as input).

If the distance of an example $x_n$ is less than $\gamma$ we consider this a mistake and update the Perceptron using the update equations $w = w + y_n x_n$.

For the case of imbalanced data, instead of using the same margin $\gamma$ for both positive and negative classes, we will use different margins $\gamma_+$ and $\gamma_-$ for the positive and negative classes respectively and we will let the class having a smaller number of training examples have the larger margin (intuitively, because we want to be more careful about the minority class while training). For example, if the number of training examples in negative class dominates the positive class, say by a factor of $M$, we can set $\gamma_+ = M \gamma_-$.

Everything else stays the same as in the case of standard binary classification.

2 Perceptron for Multiclass Classification

The idea in multiclass variant of the Perceptron algorithm is pretty much the same as in the binary classification except for a few minor differences. In the multiclass classification with $K$ classes, we will maintain a set of $K$ weight vectors $w_1, \ldots, w_K$ (each weight vector is of size $D$ where $D$ is the number of features). The prediction (both at training and test time) would change to: $\hat{y}_n = \arg \max_k (w_k^\top x_n + b)$ which means that the predicted class will be the one
for which the weight vector gives the highest score. The update condition is given by (assuming that $y_n$ is the true label of $x_n$):

$$\text{if}(\hat{y}_n \neq y_n)$$

$$w_{\hat{y}_n} = w_{\hat{y}_n} - x_n$$

$$w_{y_n} = w_{y_n} + x_n$$

Basically, by doing these updates we are ensuring that the weight vector $w_{\hat{y}_n}$ does not give the highest score (since $\hat{y}_n \neq y_n$ suggests that it indeed shouldn’t have!). This update will make this score go down (it’s an easy exercise to verify) and will make the score of the true class $y_n$’s weight vector $w_{y_n}$ to go up. Also note that in the multiclass setting, we have 2 update equations instead of a single update equation $w = w + y_n x_n$ as we had in the case of binary classification setting.