8803 Machine Learning Theory

Maria-Florina Balcan Lecture 5: September 6, 2011

The Winnow Algorithm

We now turn to an algorithm called the *Winnow Algorithm* developed by Nick Littlestone that performs especially well when many of the features given to the learner turn out to be irrelevant. Like the Perceptron Training Procedure discussed in the previous lectures, Winnow uses linear threshold functions as hypotheses and performs incremental updates to its current hypothesis. Unlike the Perceptron Training Procedure, Winnow uses *multiplicative* rather than *additive* updates. Winnow was developed with the goal of providing a significant Mistake Bound improvement when $r \ll n$.

We will analyze the Winnow algorithm for learning the class of C of {monotone disjunctions of r variables}. Note that Winnow or generalizations of Winnow can handle other specific concept classes (e.g. non-monotone disjunctions, majority functions, linear spearators), but the analysis is simplest in the case of monotone disjunctions.

The Algorithm

Both Winnow and Perceptron Algorithms use the same classification scheme:

- $\mathbf{w} \cdot \mathbf{x} \geq \theta \Rightarrow$ positive classification
- $\mathbf{w} \cdot \mathbf{x} < \theta \Rightarrow$ negative classification

For convenience, we assume that $\theta = n$ and we initialize $\mathbf{w}$ to the "all ones" vector, i.e., $w_i = 1$ for all i[1] .

The Winnow Algorithm differs from the Perceptron Algorithm in its update scheme. When misclassifying a positive training example $\mathbf{x}$ (i.e. the predition was negative because $\mathbf{w} \cdot \mathbf{x}$ was too small):

$$\forall x_i = 1 : w_i \leftarrow 2w_i.$$

When misclassifying a negative training example $\mathbf{x}$ (i.e. prediction was positive because $\mathbf{w} \cdot \mathbf{x}$ was too large):

[1]Note that Perceptron used a threshold of 0 but here we use a threshold of n.

$$\forall x_i = 1 : w_i \leftarrow w_i/2.$$

Notice that because we are updating multiplicatively, all weights remain positive; so, to handle a non-monotone target concept we would need to transform the input space to have $\bar{x}_i$ be a new variable. Intuitively, Winnow does more with the training information than the traditional *list-and-cross-off* scheme, because now in order to predict positive there must be "enough" evidence; thus we make more progress when we make a mistake. As we will see later, this is why Winnow guarentees a better performance bound for learning disjunctions when r is small.

The Mistake Bound Analysis

We can show the following guarantee:

Theorem 1 *The Winnow Algorithm learns the class of monotone disjunctions in the Mistake Bound model, making at most* $2 + 3r(1 + \log n)$ *mistakes when the target concept is an OR of* r *variables.*

Proof: Let $X_r = \{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\}$ be the r relevant variables in our target concept. Let $W_r = \{w_{i_1}, w_{i_2}, \ldots, w_{i_r}\}$ be the weights of the relevant variables. Let $w(t)$ denote the value of weight w at time t and let $TW(t)$ be the Total Weight of the LTF (including both relevant and irrelevant variables) at time t.

We will first bound the number of mistakes that will be made on positive examples. Note first that any mistake made on a positive example must double at least one of the weights in the target function (the relevant weights). So if at time t we misclassify a positive example we have:

$$\exists w \in W_r \text{ such that } w(t+1) = 2w(t) \tag{1}$$

Moreover a mistake made on a negative example will *not* halve any of the relevant weights, by definition of a disjunction; so for all times t, we have:

$$\forall w \in W_r, w(t+1) \geq w(t) \tag{2}$$

Moreover, each of these weights can be doubled at most $1 + \log(n)$ times, since only weights that are less than n can ever be doubled. Combining this together with (1) and (2), we get that Winnow makes at most $M_+ \leq r(1 + \log(n))$ mistakes on positive examples.

We now bound the number of mistakes made on negative examples. Note first that a mistake made on a positive example increases the total weight by at most n. To see this assume that we made mistake on the positive example x at time t. We must have:

$$w_1(t)x_1 + ... + w_n(t)x_n < n$$

Since

$$TW(t+1) = TW(t) + (w_1(t)x_1 + ... + w_n(t)x_n),$$

we get

$$TW(t+1) \leq TW(t) + n. \tag{3}$$

Similarly, we can show that each mistake made on a negative example decreases the total weight by at least $n/2$. To see this assume that we made mistake on the negative example x at time t. We must have:

$$w_1(t)x_1 + ... + w_n(t)x_n \geq n.$$

Since

$$TW(t+1) = TW(t) - (w_1(t)x_1 + ... + w_n(t)x_n)/2,$$

we get

$$TW(t+1) \leq TW(t) - n/2. \tag{4}$$

Finally, the total weight never drops below zero, i.e., at all times:

$$TW(t) > 0 \tag{5}$$

Combining equations (4), (3), and (5) we get:

$$0 < TW(t) \leq TW(0) + nM_+ - (n/2)M_- \tag{6}$$

The total weight summed over all the variables is initially n since $\mathbf{w}(0) = \mathbf{1}$. Solving (6) we get $M_- \leq 2 + 2M_+$. Combining both the negative and positive mistakes, we get that Winnow obtains makes at most $2 + 3r(1 + \log n)$ mistakes when the target concept is an OR of r variables. ■

Note: One can easily show that Winnow makes $\Omega(r \log n)$ mistakes in the worst case.

Interesting Open Question: Is there a computationally efficient algorithm for learning *decision lists* in the mistake bound model with a mistake bound $poly(L, \log n)$, where L is length of target decision list? Note that the so called halving algorithm achieves this bound, but it is not a computationally efficient algorithm.

Winnow versus Perceptron One can generalize the basic analysis we did for Winnow to the case of learning linear separators; the guarantee depends on the L_1, L_∞ margin of the target. In particular, if the target vector w^* is a linear separator such that $w^* \cdot x > c$ on positives and $w^* \cdot x < c - \alpha$ on negatives, then the mistake bound of Winnow is

$$O((L_1(w^*)L_\infty(X)/\alpha)^2 \log(n)).$$

The quantity $\gamma = \alpha/[L_1(w^*)L_\infty(X)]$ is called the "L_1, L_∞" margin of the separator, and our bound is $O((1/\gamma^2) \cdot \log(n))$. On the other hand, the Perceptron algorithm has a mistake bound of $O(1/\gamma^2)$ where $\gamma = \alpha/[L_2(w^*)L_2(X)]$ (this is called the "L_2, L_2" margin of the separator).

Intuitively, if "n" is large but most features are irrelevant (i.e. target is sparse but examples are dense), then Winnow is better because adding irrelevant features increases $L_2(X)$ but not $L_\infty(X)$. On the other hand, if the target is dense and examples are sparse, then Perceptron is better.