15-850(C) Machine Learning Theory

Lecturer: Avrim Blum
Scribe: Belinda Thom

Lecture 7: October 10, 1995

---

1 The Winnow Algorithm

We now turn to an algorithm called the *Winnow Algorithm* developed by Littlestone [1] that performs especially well when many of the features given to the learner turn out to be irrelevant. Like the Perceptron Training Procedure discussed last time, 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.

We will analyze the Winnow algorithm under the following conditions:

- $C = \{\text{monotone disjunctions of } r \text{ variables}\}$.
- Training examples are consistent.

It should be noted that Winnow can handle other specific concept classes (e.g. conjunctions, non-monotone disjunctions, majority functions) but the analysis is simplest in the case of monotone disjunctions. Winnow was developed with the following goal in mind: to provide a significant Mistake Bound improvement when $r \ll n$.

2 The Algorithm

Both Winnow and Perceptron Algorithms use the same classification scheme:

- $\vec{w} \cdot \vec{x} > \theta \Rightarrow$ positive classification
- $\vec{w} \cdot \vec{x} \leq \theta \Rightarrow$ negative classification

For convenience, we will set the threshold θ to n, and initialize $\vec{w}$ to the "all ones" vector ($w_i = 1$ for all i).

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

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

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

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

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.

3 Winnow's Performance

How does the Winnow Algorithm perform?

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. Given all data is consistent with the target concept, Winnow provides the following guarentees.

For each misclassified positive example:

$$TW(t+1) \leq TW(t) + \theta \tag{1}$$

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

$$\forall w \in W_r, w(t+1) \geq w(t) \tag{3}$$

For each miscalssified negative example:

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

Finally, at all times:

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

The number of mistakes needed to learn positive examples (M_+) is bounded by equations (2) and (3):

$$M_+ \leq r \log(\theta)$$

The work needed to learn negative examples (M_-) is bounded by equations (1), (4) and (5):

$$0 < TW(t) \leq TW(0) + \theta M_+ - (\theta/2)M_-$$

Assuming $\theta = n$ and $\vec{w}(0) = \vec{1}$, we can solve for M_-:

$$M_- \leq (n + nr\log(\theta))/(n/2) = 2 + 2r\log(n)$$

Combining both negative and positive mistakes, Winnow obtains the Mistake Bound:

$$O(r\log(n)).$$

4 Incorporating Winnow Into The PAC Model

Recall that Mistake Bound algorithms provide the following PAC guarentee. One can report the current hypothesis h with confidence $1-\delta$ provided at least $1/\epsilon\log(M/\delta)$ consecutively perfect predictions have been made. Thus in the worst case (we need to make M mistakes), Winnow would need to see $O(M/\epsilon\log(M/\delta)) = O((r\log(n)/\epsilon)\log(r\log(n)/\delta))$ examples.

A better bound can be achieved if we modify our PAC simulation as follows:

1. Sample from the distribution for a while (take m_1 samples).
2. Run a Mistake Bound algorithm on this sample to generate at most M hypotheses $H = \{h_1, h_2, \ldots, h_M\}$ (we are assuming here, without loss of generality, that the mistake-bound algorithm is conservative).
3. Sample again (take m_2 samples).
4. Use this new sample to determine a good hypotheses in H.
5. Report this hypothesis.

This simulation gets its leverage from the fact that for most hypotheses whose error is greater than ϵ, one will not have to wait for fully $1/\epsilon\log(M/\delta)$ examples to see a mistake. This simulation allows one to first bound the number of examples needed to guarentee at least one $h \in H$ is good (m_1), and then bound the number of examples needed to distinguish at least one good hypothesis in H from all the bad ones (m_2). Deriving each of m_1 and m_2 so that each guarantee occurs with probability at least $1-\delta'$ for $\delta' = \delta/2$ allows us to provide a confidence of at least $1-\delta$ to the entire procedure.

To bound m_1: Let $\hat{M} = \max(M, 4\ln(1/\delta'))$. The claim is that $m_1 = 4\hat{M}/\epsilon$ examples are sufficient to ensure that with high probability, at least one hypothesis produced has error at most $\epsilon/2$. The reasoning is as follows. Suppose one repeatedly drew examples until the first time that a hypothesis of error less than $\epsilon/2$ is produced. Let us say that this is hypothesis

number $(i+1)$ for some $i \leq M$ (i.e., the first i hypotheses all have error at least $\epsilon/2$). We want to bound the probability that hypothesis $i+1$ occurs after m_1 examples. Another way to think of this is to think of flipping a sequence of coins, each with a bias greater than or equal to $\epsilon/2$, and asking how long we need to wait to get i heads (a head corresponds to a hypothesis making a mistake). It is clear that the worst case occurs when $i = M$ and each coin has bias exactly $\epsilon/2$; in that case, the expected number of coin flips needed is $M/(\epsilon/2)$. If we flip $4\hat{M}/\epsilon$ coins, then the expected number of heads is $2\hat{M}$. By Chernoff bounds, the probability that the number of heads is less than half the expectation is at most $e^{-(2\hat{M})/8}$ which is at most δ' by definition of $\hat{M}$. Therefore, with high probability the number of heads is at least $\hat{M} \geq M$. In other words, with high probability, the first "good" hypothesis we see will occur sometime within the first $m_1 = 4\hat{M}/\epsilon$ examples.

To bound m_2, it is sufficient to bound the number of examples needed to guarentee that if we select a hypothesis with observed error less than $3\epsilon/4$, then with high probability:

1. No hypothesis with true error greater than ϵ is chosen.
2. At least one hypothesis with true error less than $\epsilon/2$ is chosen.

This is a standard Chernoff bound argument and results in $m_2 = O(1/\epsilon[\log(M)+\log(1/\delta')])$.

The m_1 bound is the leading term, yielding a familiar PAC bound of $O(1/\epsilon[M + \log(1/\delta)])$ examples. In the case of Winnow, $O(1/\epsilon[r\log(n) + \log(1/\delta)])$ examples would suffice.

Using this bound and the Winnow Algorithm, one obtains the best bound known for learning r of n disjunctions, which is better than that achieved by the greedy set cover algorithm. It is interesting that an LTF hypothesis space acheives a better PAC bound than any known disjunctive hypothesis space.

References

[1] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.