8803 Machine Learning Theory

Maria-Florina Balcan Lecture 3: January 19, 2010

In this lecture we study the *online learning* protocol. In this setting, the following scenario is repeated indefinitely:

1. The algorithm receives an unlabeled example.
2. The algorithm predicts a classification of this example.
3. The algorithm is then told the correct answer.

We will call whatever is used to perform step (2), the algorithm's "current hypothesis."

1 The Perceptron Algorithm

For simplicity, we'll use a threshold of 0, so we're looking at learning functions like: $w_1x_1 + w_2x_2 + ... + w_nx_n > 0$. This is WLOG since we can simulate a nonzero threshold with a "dummy" input x_0 that is always 1, so this can be done without loss of generality.

The Perceptron Algorithm: Let's automatically scale all examples $\mathbf{x}$ to have Euclidean length 1, since this doesn't affect which side of the plane they are on.

1. Start with the all-zeroes weight vector $\mathbf{w}_1 = \mathbf{0}$, and initialize t to 1.
2. Given example $\mathbf{x}$, predict positive iff $\mathbf{w}_t \cdot \mathbf{x} > 0$.
3. On a mistake, update as follows:
   - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}$.
   - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}$.

   $t \leftarrow t + 1$.

The guarantee we showed for the Perceptron Algorithm is the following:

Theorem 1 *Let $\mathcal{S}$ be a sequence of labeled examples consistent with a linear threshold function $\mathbf{w}^* \cdot \mathbf{x} > 0$, where $\mathbf{w}^*$ is a unit-length vector. Then the number of mistakes M on $\mathcal{S}$ made by the online Perceptron algorithm is at most $(1/\gamma)^2$, where $\gamma = \min_{\mathbf{x} \in \mathcal{S}} \frac{|\mathbf{w}^* \cdot \mathbf{x}|}{||\mathbf{x}||}$.*

(I.e., if we scale examples to have Euclidean length 1, then γ is the minimum distance of any example to the plane $\mathbf{w}^* \cdot \mathbf{x} = 0$.)

The key idea of the proof is to show that: for all t, $\mathbf{w}_{t+1} \cdot \mathbf{w}^* \geq \mathbf{w}_t \cdot \mathbf{w}^* + \gamma$ (Claim 1) and $||\mathbf{w}_{t+1}||^2 \leq ||\mathbf{w}_t||^2 + 1$ (Claim 2). So after M mistakes we have both $\mathbf{w}_{M+1} \cdot \mathbf{w}^* \geq \gamma M$ and $||\mathbf{w}_{M+1}|| \leq \sqrt{M}$. Combining these together with $\mathbf{w}_t \cdot \mathbf{w}^* \leq ||\mathbf{w}_t||$, we get the desired upper bound on the number of mistakes $M \leq 1/\gamma^2$.

What if there is no perfect separator? What if only *most* of the data is separable by a large margin, or what if $\mathbf{w}^*$ is not perfect? We can see that the thing we need to look at is Claim 1. Claim 1 said that we make "γ amount of progress" on every mistake. Now it's possible there will be mistakes where we make very little progress, or even negative progress. One thing we can do is bound the total number of mistakes we make in terms of the total distance we would have to move the points to make them actually separable by margin γ. Let's call that TD_γ. Then, we get that after M mistakes, $\mathbf{w}_{M+1} \cdot \mathbf{w}^* \geq \gamma M - \text{TD}_\gamma$. So, combining with Claim 2, we get that $\sqrt{M} \geq \gamma M - \text{TD}_\gamma$. We could solve the quadratic, but this implies, for instance, that $M \leq 1/\gamma^2 + (2/\gamma)\text{TD}_\gamma$. The quantity $\frac{1}{\gamma}\text{TD}_\gamma$ is called the total *hinge-loss* of w^*.

So, this is not too bad: we can't necessarily say that we're making only a small multiple of the number of mistakes that $\mathbf{w}^*$ is (in fact, the problem of finding an approximately-optimal separator is NP-hard), but we can say we're doing well in terms of the "total distance" parameter.

1.1 Perceptron for approximately maximizing margins.

We saw that the perceptron algorithm makes at most $1/\gamma^2$ mistakes on any sequence of examples that is linearly-separable by margin γ (i.e., any sequence for which there exists a unit-length vector $\mathbf{w}^*$ such that all examples $\mathbf{x}$ satisfy $\ell(\mathbf{x})(\mathbf{w}^* \cdot \mathbf{x})/||\mathbf{x}|| \geq \gamma$, where $\ell(\mathbf{x}) \in \{-1, 1\}$ is the label of $\mathbf{x}$).

Suppose we are handed a set of examples $\mathcal{S}$ and we want to actually find a *large-margin* separator for them. One approach is to directly solve for the maximum-margin separator using convex programming (which is what is done in the SVM algorithm). However, if we only need to *approximately* maximize the margin, then another approach is to use Perceptron. In particular, suppose we cycle through the data using the Perceptron algorithm, updating not only on mistakes, but also on examples $\mathbf{x}$ that our current hypothesis gets correct by margin less than $\gamma/2$. Assuming our data is separable by margin γ, then we can show that this is guaranteed to halt in a number of rounds that is polynomial in $1/\gamma$. (In fact, we can replace $\gamma/2$ with $(1-\epsilon)\gamma$ and have bounds that are polynomial in $1/(\epsilon\gamma)$.)

The Margin Perceptron Algorithm(γ):

1. Assume again that all examples are normalized to have Euclidean length 1. Initialize $\mathbf{w}_1 = \ell(\mathbf{x})\mathbf{x}$, where $\mathbf{x}$ is the first example seen and initialize t to 1.

2. Predict positive if $\frac{\mathbf{w}_t \cdot \mathbf{x}}{||\mathbf{w}_t||} \geq \gamma/2$, predict negative if $\frac{\mathbf{w}_t \cdot \mathbf{x}}{||\mathbf{w}_t||} \leq -\gamma/2$, and consider an example to be a margin mistake when $\frac{\mathbf{w}_t \cdot x}{||\mathbf{w}_t||} \in (-\gamma/2, \gamma/2)$.

3. On a mistake (incorrect prediction or margin mistake), update as in the standard Perceptron algorithm: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \ell(\mathbf{x})\mathbf{x}$; $t \leftarrow t+1$.

Theorem 2 *Let $\mathcal{S}$ be a sequence of labeled examples consistent with a linear threshold function $\mathbf{w}^* \cdot \mathbf{x} > 0$, where $\mathbf{w}^*$ is a unit-length vector, and let*

$$\gamma = \min_{\mathbf{x} \in \mathcal{S}} \frac{|\mathbf{w}^* \cdot \mathbf{x}|}{||\mathbf{x}||}.$$

Then the number of mistakes (including margin mistakes) made by Margin Perceptron(γ) on $\mathcal{S}$ is at most $12/\gamma^2$.

Proof: The argument for this new algorithm follows the same lines as the argument for the original Perceptron algorithm.

As before, we can show that each update sincreases $\mathbf{w}_t \cdot \mathbf{w}^*$ by at least γ. What is now a little more complicated is to bound the increase in $||\mathbf{w}_t||$. For the original algorithm, we had: $||\mathbf{w}_{t+1}||^2 \leq ||\mathbf{w}_t||^2 + 1$, which implies $||\mathbf{w}_{t+1}|| \leq ||\mathbf{w}_t|| + \frac{1}{2||\mathbf{w}_t||}$.

For the new algorithm, we can show instead:

$$||\mathbf{w}_{t+1}|| \leq ||\mathbf{w}_t|| + \frac{1}{2||\mathbf{w}_t||} + \frac{\gamma}{2}. \tag{1}$$

To see this note that:

$$||\mathbf{w}_{t+1}||^2 = ||\mathbf{w}_t||^2 + 2l(x)\mathbf{w}_t \cdot \mathbf{x} + ||\mathbf{x}||^2 = ||\mathbf{w}_t||^2 \left(1 + \frac{2l(x)}{||\mathbf{w}_t||}\frac{\mathbf{w}_t \cdot \mathbf{x}}{||\mathbf{w}_t||} + \frac{1}{||\mathbf{w}_t||^2}\right)$$

Using the inequlaity $\sqrt{1+\alpha} \leq 1 + \frac{\alpha}{2}$ together with the fact $\frac{\mathbf{w}_t \cdot \mathbf{x}}{||\mathbf{w}_t||} \leq \frac{\gamma}{2}$ (since w_t made a mistake on x) we ge the desired upper bound on $||\mathbf{w}_{t+1}||$, namely: $||\mathbf{w}_{t+1}|| \leq ||\mathbf{w}_t|| + \frac{1}{2||\mathbf{w}_t||} + \frac{\gamma}{2}$.

Note that (1) implies that if $||\mathbf{w}_t|| \geq 2/\gamma$ then $||\mathbf{w}_{t+1}|| \leq ||\mathbf{w}_t|| + 3\gamma/4$. Note also that $||\mathbf{w}_{t+1}|| \leq ||\mathbf{w}_t|| + 1$. These two facts imply that after M updates we have:

$$||\mathbf{w}_{M+1}|| \leq 1 + 2/\gamma + 3M\gamma/4.$$

Solving $M\gamma \leq 1 + 2/\gamma + 3M\gamma/4$ we get $M \leq 12/\gamma^2$, as desired. ∎

2 The Mistake Bound model

Definition 1 *An algorithm A is said to* learn C in the mistake bound model *if for any concept $c \in C$, and for any ordering of examples consistent with c, the total number of mistakes ever made by A is bounded by $p(n, size(c))$, where p is a polynomial. We say that A is a* polynomial time *learning algorithm if its running time per stage is also polynomial in n and $size(c)$.*

Let us now examine a few problems that are learnable in the mistake bound model.

2.1 Conjunctions

Let us assume that we know that the target concept c will be a conjunction of a set of (possibly negated) variables, with an example space of n-bit strings.

Consider the following algorithm:

1. Initialize hypothesis h to be $x_1\overline{x_1}x_2\overline{x_2}\ldots x_n\overline{x_n}$.
2. Predict using $h(x)$.
3. If the prediction is **False** but the label is actually **True**, remove all the literals in h which are **False** in x. (So if the first mistake is on 1001, the new h will be $x_1\overline{x_2}\,\overline{x_3}\,x_4$.)
4. If the prediction is **True** but the label is actually **False**, then output "no consistent conjunction".
5. Return to step 2.

An invariant of this algorithm is that the set of literals in c will always be a subset of the set of literals in h.

The first mistake on a positive example will bring the size of h to n. Each subsequent such mistake will remove at least one literal from h, so that the maximum number of mistakes made will be at most $n + 1$.

Lower Bound: In fact no deterministic algorithm can achieve a mistake bound better than n in the worst case. This can be seen by considering the sequence of n examples in which the ith example has all bits except the ith bit set to 1. The target concept c will be a monotone conjunction constructed by including x_i only if the algorithm predicts the ith example to be **True** (in which case the ith example's label will be **False**). (If the algorithm predicts the ith example to be **False**, then the target concept will not include x_i, and so the true label will be **True**.) The algorithm will have made n mistakes by the time all of these n examples are processed.

2.2 Decision lists

As discussed last time a *Decision List* is a list of if-then rules where each condition is a literal (a variable or its negation). Formally,

Definition 2 *A* **decision list** *is a list of rules:*

$$A_1 \to B_1,\ A_2 \to B_2,\ \ldots,\ A_{l-1} \to B_{l-1},\ True \to B_l$$

where A_i are literals and $B_i \in \{0, 1\}$. The meaning is: given an example, we look at the first left-hand-side satisfied and output the corresponding right-hand-side.

We present here an algorithm for learning in the mistake bound model. Note that in this algorithm our hypotheses will *not* belong to C.

In particular, our hypothesis will be a list of *subsets* of $\{x_1 \Rightarrow \textbf{True}, x_1 \Rightarrow \textbf{False}, \overline{x_1} \Rightarrow \textbf{True}, \overline{x_1} \Rightarrow \textbf{False}, x_2 \Rightarrow \textbf{True}, \ldots \overline{x_n} \Rightarrow \textbf{False}, T \Rightarrow \textbf{True}, T \Rightarrow \textbf{False}\}$. The algorithm is the following.

1. Let h be the one-level list, whose level consists of the single set that includes all $4n+2$ possible terms.
2. Given an example x, look at the first set in h for which x satisfies the antecedents of at least one of the rules of the set. (I.e., select some rule that fires.) Predict based on this rule, breaking ties arbitrarily.
3. If the prediction is mistaken, move all the rules in that set which predicted incorrectly to the next set in the list.
4. Return to step 2.

As an example of running this algorithm, say that n is 2 and the target concept is:

$$x_1 \Rightarrow \textbf{False}, \text{ else } \overline{x_2} \Rightarrow \textbf{True}, \text{ else } \textbf{False}.$$

Initially h will be

$$\{ \text{ all rules } \}.$$

If the first example is 01 then the algorithm will see two possible predictions in the first set of h; say it chooses to predict positively. In fact, though, this is a negative example. So h will be changed to

$$\{x_1 \Rightarrow \textbf{True}, x_1 \Rightarrow \textbf{False}, \overline{x_1} \Rightarrow \textbf{False}, x_2 \Rightarrow \textbf{False}, \overline{x_2} \Rightarrow \textbf{True}, \overline{x_2} \Rightarrow \textbf{False}, T \Rightarrow \textbf{False}\},$$

$$\text{else } \{\overline{x_1} \Rightarrow \textbf{True}, x_2 \Rightarrow \textbf{True}, T \Rightarrow \textbf{True}\}.$$

Now say that we give an example of 00 to the algorithm. Then the algorithm might choose to predict that this will be negative, though in fact it is positive. So h will now be

$$\{x_1 \Rightarrow \textbf{True}, x_1 \Rightarrow \textbf{False}, x_2 \Rightarrow \textbf{False}, \overline{x_2} \Rightarrow \textbf{True}\},$$
$$\text{else } \{\overline{x_1} \Rightarrow \textbf{True}, \overline{x_1} \Rightarrow \textbf{False}, x_2 \Rightarrow \textbf{True}, \overline{x_2} \Rightarrow \textbf{False}, T \Rightarrow \textbf{True}, T \Rightarrow \textbf{False}\}.$$

And so on.

We can make several observations about this algorithm. First, with each mistake at least one term will move one level lower in h. Second, the ith rule of c will never move below the

ith level of h. And finally, each rule will fall at most $L + 1$ levels (where L is the length of target c). These three properties imply that with every mistake the algorithm always makes progress toward a correct solution. Furthermore, the number of mistaken predictions is bounded above by $(4n + 2)(L + 1)$. This is polynomial in n and the size of c, so this algorithm learns decision lists in the mistake bound model.