COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Arnold K Mong

Lecture #16
April 3, 2019

1 Analysis of RWMA

General Framework:
N= number of experts
For $t = 1, \ldots, T$
Observe ξ_i for $i \in \{1, 2, ..., N\}$
Predict $\hat{y} \in \{0, 1\}$
Observe $y \in \{0, 1\}$

Recap: In lecture #15, we covered the weighted majority algorithm (WMA) and the randomized weighted majority algorithm (RWMA). We will finish the analysis of RWMA.

New Notation:
$L_A = \mathbb{E}[$ # of mistakes of learner$]$
$L_i =$ # of mistakes of expert i

Theorem: Using RWMA,

$$L_A \leq a_\beta \min_i L_i + c_\beta \ln(N)$$

where $a_\beta = \frac{\ln(\frac{1}{\beta})}{1-\beta}$ and $c_\beta = \frac{1}{1-\beta}$.

Corollary: Given $\min_i L_i \leq K$, we can set β for RWMA such that

$$L_A \leq \min_i L_i + \sqrt{2K \ln(N)} + \ln(N)$$

1.1 Intermediate WMA

As mentioned in lecture #15, there are algorithms which make predictions that are between those of the WMA and RWMA algorithms, and which can achieve better mistake bounds. While not demonstrated in lecture, we can show

Theorem: It is possible to find a "middle-ground" between the WMA and RWMA algorithms as to achieve the following mistake bound:

$$L_A \leq a_\beta \min_i L_i + c_\beta \ln(N)$$

where $a_\beta = \frac{\ln(\frac{1}{\beta})}{2\ln(\frac{2}{1+\beta})}$ and $c_\beta = \frac{1}{2\ln(\frac{2}{1+\beta})}$.

Corollary: Given $\min_i L_i \leq K$, we can set β for the algorithm above such that

$$L_A \leq \min_i L_i + \sqrt{K \ln(N)} + \frac{\ln(N)}{2}$$

Extension: Usually, we can guarantee that $K \leq \frac{T}{2}$. For example, if one expert always predicts 0 and the other always predicts 1, then one or the other will be right at least half the time (making fewer than $\frac{T}{2}$ mistakes). Thus, we can (almost) without loss of generality assume $K \leq \frac{T}{2}$. Then the bound above reads:

$$L_A \leq \min_i L_i + \sqrt{\frac{T \ln(N)}{2}} + \frac{\lg(N)}{2}$$

Definition: The difference between L_A and $\min_i L_i$ is called *cumulative regret.*

Another interpretation of the bound above pertains to the mistake rate. Dividing the inequality above by T reads:

$$\frac{L_A}{T} \leq \frac{\min_i L_i}{T} + \sqrt{\frac{\ln(N)}{2T}} + \frac{\lg(N)}{2T}$$

Definition: The difference between $\frac{L_A}{T}$ and $\frac{\min_i L_i}{T}$ is called *average regret.*

In the result above, it is clear that as $T \to \infty$, the average regret $\to 0$, so the learner's mistake rate approaches the mistake rate of the best expert. This is called *no-regret learning.*

1.2 Lower bound on L_A

The mistake bound for the intermediate algorithm is essentially tight for large T for the dominant term of the regret. Currently, we only have the upper bound:

$$L_A \leq \min_i L_i + \sqrt{\frac{T \ln(N)}{2}} + \frac{\lg(N)}{2}$$

We will now prove a lower bound on the number of mistakes which nearly matches the upper bound, even up to the constants of the dominant term.

Theorem: For any learning algorithm A, there exists an adversary (who chooses ξ_i and y) such that

$$L_A \gtrapprox \min_i L_i + \sqrt{\frac{T \ln(N)}{2}}$$

Proof: For any A, we can construct the adversary which makes every expert predict 0 with 50% chance and 1 with 50% chance independently. Similarly, the outcomes y_t are 0 and 1 as the result of a coin flip. Then, since the outcomes and expert predictions are purely random, any algorithm A and each of the experts is expected to have 50% accuracy. With T rounds, taking the expectation over the random outcomes y_t and expert predictions $\xi_{i,t}$,

$$\mathbb{E}[L_A] = \frac{T}{2} \qquad \forall i,\ \mathbb{E}[L_i] = \frac{T}{2}$$

However, from random variations we expect the best expert to do better than just $\frac{T}{2}$. It can be shown that

$$\mathbb{E}[\min_i L_i] \approx \frac{T}{2} - \sqrt{\frac{T \ln(N)}{2}}$$

which implies

$$\mathbb{E}[L_A] \approx \mathbb{E}[\min_i L_i] + \sqrt{\frac{T\ln(N)}{2}}$$

So, the adversary can select some ξ_i, y for each round to create a scenario where

$$\text{Ł}_A \gtrapprox \min_i L_i + \sqrt{\frac{T\ln(N)}{2}} \quad \square$$

2 Predictions Using Groups of Experts

In the previous examples of online learning, the mistake bounds we found relate the number of mistakes the learner makes to the number of mistakes the *single* best expert makes. These bounds are useful under the assumption that there is at least one expert who achieves high accuracy alone. However, one could imagine a scenario where no single expert is very good, but some subset of the experts together make very accurate predictions. We now present a general algorithm skeleton and two specific algorithms for this scenario.

Change of Setting:
Let $\mathbf{x}_t$ be a vector representing the predictions of the N experts on round t. In this special case, $\mathbf{x}_t \in \{-1, +1\}^N$. But to be more general, we will allow $\mathbf{x}_t \in \mathbb{R}^N$. Next, the algorithm makes the prediction $\hat{y}_t \in \{-1, +1\}$, and finally observes the outcome $y_t \in \{-1, +1\}$ of round t.

"Good Group" Assumption
We will assume that $y_t = \text{sign}(\mathbf{u}\cdot\mathbf{x}_t)$, so the outcomes are a linear threshold function defined by some vector $\mathbf{u}$. So, we are assuming that for all t, y_t can be found via some weighted majority vote of experts with weights given by $\mathbf{u}$.

2.1 General Algorithm Outline

We now present a general framework encompassing the two algorithms (perceptron and winnow) which we will present in the upcoming sections. Both involve keeping a weight vector $\mathbf{w}$ which is updated each round after the algorithm sees the data, makes a prediction, and observes the true outcome.

Algorithm Framework:
Initialize $\mathbf{w}_1$.
For $t = 1, 2, \ldots, T$:
    Observe $\mathbf{x}_t \in \mathbb{R}^N$.
    Learner predicts $\hat{y}_t = \text{sign}(\mathbf{w}_t \cdot \mathbf{x}_t)$
    Observe $y_t \in \{-1, +1\}$.
    Update $\mathbf{w}_{t+1} = F(\mathbf{w}_t, \mathbf{x}_t, y_t)$

If we have the "good group" assumption, we want any specific algorithm to modify $\mathbf{w}_t$ to make predictions (almost) as good as those that $\mathbf{u}$ does as t increases. Moreover, since the y_t's are of the form $\text{sign}(\mathbf{u} \cdot \mathbf{x}_t)$ and our predictions $\hat{y}_t$ are of the form $\text{sign}(\mathbf{w}_t \cdot \mathbf{x}_t)$, the general algorithm is trying to learn a linear threshhold function on a hypercube (when

$\mathbf{x}_t \in \{-1, +1\}^N$), or, in the more general case that $\mathbf{x}_t \in \mathbb{R}^N$, we return to learning a linear threshhold function on our typical $\mathbb{R}^N$ domain.

3 Perceptron Algorithm

Next, we present an algorithm to learn the linear threshold function by filling in the general framework provided in the last section.

Initialize $\mathbf{w}_1 = \mathbf{0}$.
For $t = 1, 2, \ldots, T$:
Observe $\mathbf{x}_t \in \mathbb{R}^N$.
Learner predicts $\hat{y}_t$ =sign$(\mathbf{w}_t \cdot \mathbf{x}_t)$
Observe $y_t \in \{-1, +1\}$.
If $\hat{y}_t \neq y_t$, let $\mathbf{w}_{t+1} = \mathbf{w}_t + y_t \mathbf{x}_t$
Otherwise, let $\mathbf{w}_{t+1} = \mathbf{w}_t$

If the algorithm is wrong on input $\mathbf{x}_t$, the algorithm nudges $\mathbf{w}_t$ so that the classifying hyperplane is closer to classifying $\mathbf{x}_t$ correctly in the future. If the algorithm is correct on round t, then there is no update to the weight vector. Because of this, we call the perceptron algorithm above *conservative*.

3.1 Analysis

In order to analyze the effectiveness of the perceptron algorithm, we first make a few assumptions before presenting a bound on the number of mistakes the learner makes:

1. Without loss of generality, assume the algorithm makes a mistake on every round (since otherwise the algorithm does nothing to $\mathbf{w}_t$). In particular, in the analysis below, T will be the number of rounds with mistakes.

2. Without loss of generality, assume $\forall t, \|\mathbf{x}_t\|_2 \leq 1$.

3. Strong "Good Group" Assumption:
   $\exists \delta > 0, \exists \mathbf{u} \in \mathbb{R}^N$ with $\|\mathbf{u}\|_2 = 1$ such that $\forall t, y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq \delta$.
   This assumption sacrifices generality and is assuming much more than before. We are assuming that taking a weighted majority vote of experts is not only perfect, but also that this prediction has a minimum margin δ. In other words, we are assuming there is a linear threshold function (such as a committee of experts) which gives perfect predictions with some baseline confidence about those predictions.

Theorem: Under the assumptions above,

$$\#\text{ mistakes perceptron algorithm makes} \leq \frac{1}{\delta^2}$$

Proof: Let us keep track of the following quantity as t increases:

$$\Phi_t = \frac{\mathbf{w}_t \cdot \mathbf{u}}{\|\mathbf{w}_t\|_2}$$

The quantity Φ_t is called the *potential* and is the cosine of the angle between $\mathbf{w}_t$ and $\mathbf{u}$. Intuitively, if our algorithm is working, we would like $\mathbf{w}_t$ and $\mathbf{u}$ to be making similarly

accurate predictions as t increases. Φ_t is one way of measuring the similarity between $\mathbf{w}_t$ and $\mathbf{u}$, where Φ_t close to 1 indicates strong similarity in predictions.

We will find a lower bound on Φ_t which depends on the number of errors made (T) and use the fact that $\forall t, \Phi_t \leq 1$ to bound the total number of errors the perceptron algorithm makes.

Step 1: Lower Bounding Φ_t Numerator
Claim 1: $\mathbf{w}_{t+1} \cdot \mathbf{u} \geq T\delta$
Sub-Proof: Because we are assuming the algorithm makes a mistake (and hence updates $\mathbf{w}$) on every round,

$$\forall t, \mathbf{w}_{t+1} \cdot \mathbf{u} = (\mathbf{w}_t + y_t \mathbf{x}_t) \cdot \mathbf{u} = \mathbf{w}_t \cdot \mathbf{u} + y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq \mathbf{w}_t \cdot \mathbf{u} + \delta$$

Notice that the last inequality follows from the strong "good group" assumption — that the margin of $\mathbf{u}$'s predictions at least δ.
Combined with the fact that $\mathbf{w}_1 \cdot \mathbf{u} = \mathbf{0}$, we have that $\mathbf{w}_{t+1} \cdot \mathbf{u} \geq T\delta$. ✓

Step 2: Upper Bounding Φ_t Denominator
Claim 2: $\|\mathbf{w}_{T+1}\|_2 \leq \sqrt{T}$
Sub-Proof: Again, assuming the algorithm makes a mistake (and hence updates $\mathbf{w}$) on every round,

$$\forall t, \|\mathbf{w}_{t+1}\|_2^2 = \mathbf{w}_{t+1} \cdot \mathbf{w}_{t+1} = (\mathbf{w}_t + y_t \mathbf{x}_t) \cdot (\mathbf{w}_t + y_t \mathbf{x}_t) =$$
$$\|\mathbf{w}_t\|_2^2 + 2y_t(\mathbf{w}_t \cdot \mathbf{x}_t) + y_t^2 \mathbf{x}_t \cdot \mathbf{x}_t$$

Then, since the algorithm makes a mistake on round t, $y_t(\mathbf{w}_t \cdot \mathbf{x}_t) \leq 0$. Moreover, by our normalization assumption, $y_t^2(\mathbf{x}_t \cdot \mathbf{x}_t) \leq 1$. In all, we have that

$$\|\mathbf{w}_{t+1}\|_2^2 = \|\mathbf{w}_t\|_2^2 + 2y_t(\mathbf{w}_t \cdot \mathbf{x}_t) + y_t^2 \mathbf{x}_t \cdot \mathbf{x}_t \leq \|\mathbf{w}_t\|_2^2 + 1$$

Recalling $\mathbf{w}_1 = \mathbf{0}$, we have that $\|\mathbf{w}_{T+1}\|_2^2 \leq T \implies \|\mathbf{w}_{T+1}\|_2 \leq \sqrt{T}$. ✓

Step 3: Combining Bounds
Combining the two bounds above,

$$\Phi_{T+1} \geq \frac{T\delta}{\sqrt{T}} = \sqrt{T}\delta$$

Then, since $\forall t, \Phi_t \leq 1$, we have that

$$\sqrt{T}\delta \leq 1 \implies T \leq \frac{1}{\delta^2}$$

where T is the number of mistakes the algorithm makes. □

3.2 VC-Dimension Corollary

Recall from lecture #13 on SVMs, we made the following claim without proof:
Theorem: The VC-dimension of linear threshold functions (LTFs) containing the origin, with margin δ, such that $\forall i, \|\mathbf{x}_i\|_2 \leq 1$ is $\leq \frac{1}{\delta^2}$.
We now are able to provide a short proof.

Proof: The algorithm above provides a bound on the maximum number of mistakes for learning LTFs of the form above. From before, we therefore have that for any hypothesis space $\mathcal{H}$,

$$\text{VC-dim}(\mathcal{H}) \leq \text{Mistake bound for any algorithm } A \text{ learning } \mathcal{H}$$

From the theorem on the perceptron maximum error bound above, we therefore have that

$$\text{VC-dim}(\text{LTF with margin } \delta \text{ through origin for } \|\mathbf{x}_t\|_2 \leq 1) \leq \frac{1}{\delta^2} \quad \square$$

4 Winnow Algorithm

Why might we want a new algorithm? Consider the following setting:

Suppose the expert prediction vectors are of the form $\mathbf{x}_t \in \{-1, +1\}^N$ and that the vector $\mathbf{u}$ is the simple majority vote of k of the experts so $\mathbf{u} \in \{0, 1\}^N$ with k 1's. Let us assume k odd, so that for any $\mathbf{x}_t$, y_t, the margin of the expert predictions $y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq 1$. However, in order to apply the bound of the perceptron algorithm derived above, we must first normalize the vectors $\mathbf{x}_t$ and $\mathbf{u}$. So, the normalized $\mathbf{x'}_t \in \{\frac{-1}{\sqrt{N}}, \frac{+1}{\sqrt{N}}\}^N$ and likewise the normalized $\mathbf{u'} \in \{0, \frac{1}{\sqrt{k}}\}^N$. Then, to apply our perceptron analysis, the size of the margins in this setup is $y_t(\mathbf{u'} \cdot \mathbf{x'}_t) \geq \frac{1}{\sqrt{Nk}}$ which we will set to δ. The mistake bound derived in the previous section then reads that the number of mistakes of the perceptron algorithm is at most $\frac{1}{\delta^2} = Nk$.

While this is a good bound if N is small and there are only a few relevant experts, we would really like an algorithm which allows us to bound the error by a function which is $O(\lg(N))$. We do so with the winnow algorithm:

4.1 Winnow Algorithm Description

Now, we fill in the gaps of the general algorithm outline from section 2.1 and present the winnow algorithm. While the perceptron made additive updates to $\mathbf{w}_t$ on rounds where the algorithm made a mistake, the winnow algorithm revolves around a multiplicative update to the weight vector $\mathbf{w}_t$ on these rounds. Moreover, the winnow algorithm keeps $\mathbf{w}_t$ as a probability distribution over the experts so that for all t, $\mathbf{w}_t$ is non-negative and its components sum to 1.

Initialize $w_{1,i} = \frac{1}{N}$ and fix some $\eta > 0$.
For $t = 1, 2, ..., T$:
Learner predicts $\hat{y}_t = \text{sign}(\mathbf{w}_t \cdot \mathbf{x}_t)$
Observe y_t.
If $\hat{y}_t \neq y_t$, let $w_{t+1,i} = \dfrac{w_{t,i} e^{\eta y_i x_{t,i}}}{Z_t}$ where Z_t is a normalization factor so $\sum_{i=1}^N w_{t+1,i} = 1$.
Otherwise, let $\mathbf{w}_{t+1} = \mathbf{w}_t$

Notice that winnow, like the perceptron algorithm, is a conservative algorithm.

In the update step, if the prediction $\hat{y}_t \neq y_t$, we can also write the next weight vector as:

$$w_{t+1,i} = \frac{w_{t,i}}{Z_t} \begin{cases} e^{-\eta} & \text{if } y_t \neq x_{t,i} \\ e^{\eta} & \text{if } y_t = x_{t,i} \end{cases} = \frac{w_{t,i} \cdot e^{\eta}}{Z_t} \begin{cases} e^{-2\eta} & \text{if } y_t \neq x_{t,i} \\ 1 & \text{if } y_t = x_{t,i} \end{cases}$$

The last form should look familiar as it is the update rule for the *weighted majority algorithm* (if we set $\beta = e^{-2\eta}$). The winnow algorithm can be thought of as a recasting of WMA.

4.2 Analysis

For lecture #17.