COS 511: Foundations of Machine Learning

Rob Schapire
Scribe: Qing Wang

Lecture # 15
April 4, 2006

1 Perceptron Algorithm (continued)

1.1 Review

The goal of the Perceptron algorithm is to find a combination of expert predictions such that the performance of the learner is as close as possible to the best combination/subcommittee of experts. It is a conservative algorithm in the sense that it ignores samples that it classifies correctly. Given N experts, the algorithm works as follows:

Perceptron Algorithm:

- $\mathbf{w}_1 = 0$
- for $t = 1, \ldots, T$
  get $\mathbf{x}_t$
  predict $\hat{y}_t = sign(\mathbf{w}_t \cdot \mathbf{x}_t)$
  observe outcome $y_t = \{1, -1\}$
  update $\mathbf{w}_{t+1} = \begin{cases} \mathbf{w}_t + y_t \mathbf{x}_t, & \text{if } y_t \neq \hat{y}_t \\ \mathbf{w}_t & \text{else} \end{cases}$

For the Perceptron algorithm we have the following upper bound on the number of mistakes in terms of the margin δ:

Assumptions:

- $\|\mathbf{x}_t\|_2 = 1$
- $\exists \mathbf{u} \in \mathbb{R}^N$, $\delta > 0$, s.t. $y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq \delta > 0$, for $\forall t = 1, \ldots, T$
- $\|\mathbf{u}\|_2 = 1$

Theorem 1 *# mistakes made by the perceptron algorithm* $\leq 1/\delta^2$.

1.2 Example: Committee of Experts

Suppose we have N experts and a subcommittee of K experts give perfect predictions (meaning that the majority vote on these K experts always gives correct predictions).

$\mathbf{x}_t$ will have the form $\mathbf{x}_t = \frac{1}{\sqrt{N}}\langle +1, -1, -1, +1, \ldots, +1\rangle$ (constant for normalization) and $\mathbf{u}$ will have the form $\mathbf{u} = \frac{1}{\sqrt{K}}\langle 0, 0, 1, 1, \ldots, 0\rangle$ (constant for normalization), where $u_i = 1$ if i-th expert is on the perfect subcommittee; $u_i = 0$ otherwise.

Suppose K is an odd number. We have $y_t(\mathbf{w}_t \cdot \mathbf{x}_t) \geq \frac{1}{\sqrt{KN}}$. Then, with $\delta = \frac{1}{\sqrt{KN}}$, all of the assumptions for Theorem 1 are met. Therefore, if we use the Perceptron Algorithm to learn $\mathbf{u}$, by Theorem 1, we can be assured that

$$\#\text{ mistakes} \leq KN.$$

Weakness of the bound: We can view the experts as features or attributes. In practice, there are usually a large number of features available for classification purpose. An efficient learning algorithm is designed such that it will choose the most important or discriminant features (eg perfect sub-committee as in this example) and usually we should have $K \ll N$ and N can be very large. Then the above bound is not good since it is linear in N. Another problem is that the bound on the number of mistakes should depend on the initial condition $\|\mathbf{w}_1 - \mathbf{u}\|_2$. Consider the extreme case: $K = N$. If we set $\mathbf{w}_1 = \langle 1, 1, \ldots, 1 \rangle$, then # mistakes=0, whereas the bound $KN = N^2$.

2 Winnow Algorithm

Similar to Perceptron algorithm, Winnow algorithm is another conservative algorithm for learning a linear separating hyper-plane. The meaning of "winnow" is to get rid of something undesirable or unwanted. Winnow algorithm is designed such that we could get down quickly to the desired weight vector in spite of the astronomical size of the available features. As commented briefly in the previous section, a good learning algorithm should be immune to the increasing size of the feature library. In this respect, Winnow algorithm is more efficient than Perceptron algorithm.

Updating rule: One major difference between Perceptron and Winnow is in the updating step. Recall that in the Perceptron algorithm, $\mathbf{w}_{t+1} = \mathbf{w}_t$ if there is no mistake; otherwise, $\mathbf{w}_{t+1} = \mathbf{w}_t + y_t \mathbf{x}_t$. In other words, we add weight to expert i if expert i gives correct prediction. In Winnow algorithm, the weight is updated *multiplicatively instead of additively*. Namely, if no mistake, $\mathbf{w}_{t+1} = \mathbf{w}_t$; otherwise, $\mathbf{w}_{t+1,i} = \mathbf{w}_{t,i} e^{\eta y_t x_{t,i}} / Z_t$, for $\forall i$, where parameter $\eta > 0$ is the learning rate. In this algorithm, we punish incorrect experts and boost correct experts. To be more precise, Winnow algorithm works as follows:

Winnow Algorithm:

- Initialize: $w_{1,i} = 1/N$
- Update

  if no mistake, $\mathbf{w}_{t+1} = \mathbf{w}_t$;

  else, $\forall i, w_{t+1,i} = w_{t,i} \frac{e^{\eta y_t x_{t,i}}}{Z_t}$, where Z_t is the normalization factor.

Remarks: If $x_{t,i} \in \{-1, 1\}$, we have

$$w_{t+1,i} \propto w_{t,i} \begin{cases} e^{\eta}, & \text{if } x_{t,i} = y_t \\ e^{-\eta}, & \text{else.} \end{cases}$$

$$w_{t+1,i} \propto w_{t,i} \begin{cases} 1, & \text{if } x_{t,i} = y_t \\ \beta = e^{-2\eta}, & \text{else.} \end{cases}$$

So Winnow algorithm is equivalent to weighted majority algorithm (WMA) with $\beta = e^{-2\eta}$ (note that $e^{\eta} \to 1$, $e^{-\eta} \to \beta$). This updating rule also reminds us of Boosting. In fact, AdaBoosting was derived from WMA and Winnow algorithm. The "examples" in Boosting correspond to "experts" in Winnow and the "examples" in Winnow act as "weak hypotheses" in Boosting.

2.1 Performance Analysis

For Winnow, we make the following new assumptions:

- $\|\mathbf{x}_t\|_\infty \leq 1$
- $\exists \mathbf{u} \in \mathbb{R}^N$, $\delta > 0$, s.t. $y_t(\mathbf{u} \cdot \mathbf{x}_t) \geq \delta > 0$, for $\forall t = 1, \ldots, T$
- $\|\mathbf{u}\|_1 = 1$
- $u_i \geq 0$ (can be removed)

Remarks: Different from Perceptron algorithm, in Winnow algorithm, L_1 and L_∞ (dual of L_1) norms are used instead of L_2 norm.

Theorem 2

$$\#\ \textit{mistakes made by the Winnow Algorithm} \leq \frac{\ln N}{\eta\delta + \ln\left(\frac{2}{e^{\eta}+e^{-\eta}}\right)}. \tag{1}$$

Choosing η to minimize the upper bound yields

$$\#\ \textit{mistakes} \leq \frac{2\ln N}{\delta^2}, \textit{achieved when } \eta = \frac{1}{2}\ln\left(\frac{1+\delta}{1-\delta}\right). \tag{2}$$

Let us now apply Winnow Algorithm to the **Example in Section 1.2**. Note that δ will be changed due to the change in norms. $\mathbf{x}_t$ will have the form $\mathbf{x}_t = \langle +1, -1, -1, +1, \ldots, +1\rangle$ and $\mathbf{u}$ will have the form $\mathbf{u} = \frac{1}{K}\langle 0, 0, 1, 1, \ldots, 0\rangle$, such that $\|\mathbf{x}_t\|_\infty \leq 1$ and $\|\mathbf{u}\|_1 = 1$. Then $y_t(\mathbf{w}_t \cdot \mathbf{x}_t) \geq 1/K$. Using $1/K$ as δ, we apply Theorem 2 and obtain that

$$\#\ \text{mistakes made by Winnow Algorithm} \leq 2K^2 \ln N.$$

Note that this bound is logarithmic in N for the Winnow Algorithm as compared to linear in N in the Perceptron Algorithm. In the case where $K \ll N$, the Winnow Algorithm is much better than the Perceptron Algorithm. Another thing to note that, similar to Theorem 1, (1) does not take into account the initial condition, either. A tighter bound could be expressed in terms of $RE(\mathbf{u}\|\mathbf{w}_1)$ the relative entropy between $\mathbf{u}$ and $\mathbf{w}_1$ (not L_2 distance). Here both $\mathbf{u}$ and $\mathbf{w}_1$ are normalized vectors, so they could be considered as probability distributions.

Proof of Theorem 2:

Assume we make a mistake every trial. The idea of the proof is to keep track of the potential function

$$\Phi_t = RE(\mathbf{u}\|\mathbf{w}_t).$$

First, we are going to show that $\Phi_1 = RE(\mathbf{u}\|\mathbf{w}_1) \leq \ln N$. Then we prove that the decrease in potential every mistake is at least a constant c.

Step 1: $\Phi_1 = RE(\mathbf{u}\|\mathbf{w}_1) = \sum u_i \ln(u_i N) \leq \sum u_i \ln N = \ln N$.

Step 2: Now let us consider the drop in the potential (for simplicity $\mathbf{w}' = \mathbf{w}_{t+1}, \mathbf{w} = \mathbf{w}_t$ and we drop all subscripts t):

$$\begin{aligned}
\Phi_{t+1} - \Phi_t &= \sum_i u_i \ln \frac{u_i}{w_i'} - \sum_i u_i \ln \frac{u_i}{w_i} \\
&= \sum_i u_i \ln u_i - \sum_i u_i \ln w_i' - \sum_i u_i \ln u_i + \sum_i u_i \ln w_i \\
&= \sum_i u_i \ln \frac{w_i}{w_i'} \\
&= \sum_i u_i \ln \frac{Z}{e^{\eta y x_i}} \\
&= \sum_i u_i \ln Z - \sum_i u_i \eta y x_i \\
&= \ln Z - \eta \underbrace{y(\mathbf{u} \cdot \mathbf{x}_t)}_{\geq \delta} \\
&\leq \ln Z - \eta\delta
\end{aligned}$$

Now we are to bound $Z = \sum_i w_i e^{\eta y x_i}$ by using our favorite bound on the exponential function (see Figure 1).

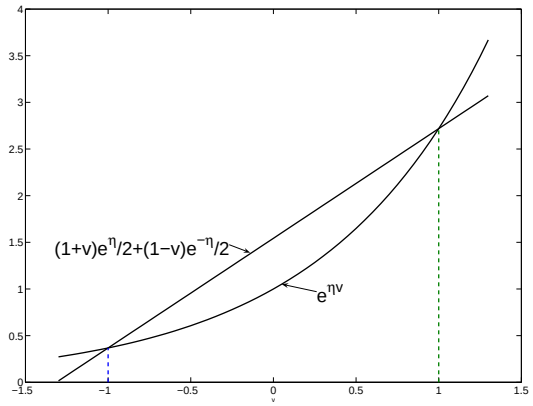

Figure 1: Upperbound an exponential on the range [-1,1] by a linear.

Then we obtain

$$\begin{aligned}
Z &\leq \sum_i w_i \left(\frac{1 + y x_i}{2} e^{\eta} + \frac{1 - y x_i}{2} e^{-\eta} \right) \\
&= \frac{e^{\eta} + e^{-\eta}}{2} \underbrace{\left(\sum_i w_i \right)}_{=1} + \underbrace{\frac{e^{\eta} - e^{-\eta}}{2}}_{\geq 0} \underbrace{\sum_i y w_i x_i}_{= y(\mathbf{w} \cdot \mathbf{x}) \leq 0,} \\
&\leq \frac{e^{\eta} + e^{-\eta}}{2} \qquad (3)
\end{aligned}$$

$$\text{Therefore, } \Phi_{t+1} - \Phi_t \leq \ln\left(\frac{e^{\eta} + e^{-\eta}}{2}\right) - \eta\delta = -c \tag{4}$$

Considering the fact that $\Phi_{T+1} \geq 0$, we get the bound (1). □

More comments: The bound (1) could be problematic in the sense that the denominator is negative when η is large enough. Also we could improve the bound by replacing $\ln N$ with Φ_1.

2.2 Balanced Winnow Algorithm

In previous analysis, we assume that all the components of $\mathbf{u}$ are non-negative. But what if for some i, $u_i < 0$? One "quick and dirty" approach is to double N. See the following example:

$$\begin{aligned} \mathbf{x}_t = (1, -.7, .32) &\rightarrow \mathbf{x}'_t = (1, -.7, .32 \mid -1, .7, -.32) \\ \mathbf{u} = (1, .2, -.2) &\rightarrow \mathbf{u}' = (1, .2, 0 \mid 0, 0, .2) \end{aligned}$$

Then $\mathbf{u}'_i \geq 0$ for each i and $\mathbf{u} \cdot \mathbf{x}_t = \mathbf{u}' \cdot \mathbf{x}'_t$. Also note that$\|\mathbf{x}_t\|_\infty = \|\mathbf{x}'_t\|_\infty$ and $\|\mathbf{u}\|_1 = \|\mathbf{u}'\|_1$.

The algorithm can be formulated as below:

Balanced Winnow Algorithm:

- Initialize: $w^+_{1,i} = w^-_{1,i} = \frac{1}{2N}$
- Predict: $\hat{y}_t = sign(\mathbf{w}^+_t \cdot \mathbf{x}_t - \mathbf{w}^-_t \cdot \mathbf{x}_t)$
- Update
  if no mistake, $\mathbf{w}^+_{t+1} = \mathbf{w}^+_t$, $\mathbf{w}^-_{t+1} = \mathbf{w}^-_t$;
  else,
  $w^+_{t+1,i} = w^+_{t,i} \frac{e^{\eta y_t x_{t,i}}}{Z_t}$
  $w^-_{t+1,i} = w^-_{t,i} \frac{e^{-\eta y_t x_{t,i}}}{Z_t}$.

2.3 Comparison

To compare Winnow/WMA and Perceptron, we find that there are two major differences. One is in the updating rule. Perceptron updates the weight vector additively while Winnow multiplicatively. The other is that different pairs of norms are used. In Perceptron, we use L_2 norm for both $\mathbf{x}$ and $\mathbf{u}$; whereas in Winnow, we use L_∞ norm and L_1 norm respectively. It is also interesting to note that Perceptron algorithm is analogous to Support Vector Machines while Winnow/WMA is similar in spirit to Boosting algorithms. These differences are summarized in the following table.

Perceptron	Winnow/WMA
additive update	multiplicative update
$\|\mathbf{x}_t\|_2$	$\|\mathbf{x}_t\|_\infty$
$\|\mathbf{u}\|_2$	$\|\mathbf{u}\|_1$
SVM's	Boosting

3 Regression

Now we are going to start something rather different. Previously, we focus on how to minimize the number of mistakes, which is equivalent to $0-1$ loss. Now we are going to generalize the loss function to different forms. In other words, the previous goal was to minimize the probability of making a mistake. Now the question could be how to estimate the probability of a given prediction. Let's first look at an example.

A TV station was trying to hire a meteorologist to predict the weather. There were two applicants. During the interview, applicant A predicted that there was a 70% chance of rain the next day while applicant B predicted 80%. It did rain the next day. How could you decide whom to hire? The difficulty is that the actual probability of rain is not known.

We can form the following model for the above example:

- x: weather conditions
- $y = \begin{cases} 1 & \text{if rain} \\ 0 & \text{else} \end{cases}$
- $(x, y) \sim D$.
- The goal is to estimate

$$p(x) = Pr[y = 1|x] = E[y|x]. \tag{5}$$

In fact, y can take real values, for example, how much rain tomorrow. Therefore, this is a "regression" problem, i.e. to estimate the expected value of a variable conditioned on another variable.

Assume applicant A uses $h_1(x)$ to estimate $p(x)$ and applicant B uses $h_2(x)$. Now the problem is converted to how to choose $h_j(x)$, which is closest to $p(x)$. To measure closeness, we need to define a loss function to penalize the difference between $h_j(x)$ and $p(x)$. More will be discussed at next lecture.