# CS260: Machine Learning Theory
# Lecture 9: Irrelevant Features & the Winnow Algorithm
October 24, 2011

Lecturer: Jennifer Wortman Vaughan

## 1 Learning Majority Functions

Recall that in the previous lecture we proved the following mistake bound for the Perceptron algorithm.

**Theorem 1.** *Suppose there exists a* $\mathbf{u}$ *of unit length and values* $\gamma > 0$ *and* $D > 0$ *such that* $\forall t \;\; y_t(\mathbf{x}_t \cdot \mathbf{u}) \geq \gamma$ *and* $\|\mathbf{x}_t\| \leq D$*. Then, the number of mistakes made by the Perceptron algorithm is no more than* $(D/\gamma)^2$*.*

Let's see an example of how we can apply this bound.

Suppose that every day we would like to make a prediction about a binary event, for example, whether or not it will rain. Each day, we have access to the binary predictions of set of $n$ experts (e.g., weather forecasters), some of which are more accurate than others. Suppose that there exists some subset of $k$ experts such that the majority opinion of these $k$ experts is always correct, but we don't know which experts are part of that subset. Over time, we would like to learn which experts to follow.

We can represent this problem as an online classification problem. At each round $t$, let $\mathbf{x}_t \in \{-1, +1\}^n$ denote the vector of predictions of the $n$ experts. Let $\mathbf{w}$ be a vector of length $n$ where $w_i = 1$ if expert $i$ is in the special subset and $w_i = 0$ otherwise. The correct prediction for us to make at time $t$ is 1 if $\mathbf{w} \cdot \mathbf{x}_t \geq 0$ and 0 otherwise.

We could run the Perceptron algorithm to learn the subset of special experts. Assume that $k$ is odd so that the majority is always well-defined. Let's examine how to apply the mistake bound above in this case.

First, we need a perfect target function of unit length. To get this, we can simply scale the vector $\mathbf{w}$, and let $\mathbf{u} = \mathbf{w}/\sqrt{k}$. Next, we must figure out the margin $\gamma$, which is the smallest possible value of $y_t(\mathbf{x}_t \cdot \mathbf{u})$. Since each component of $\mathbf{x}_t$ is either $+1$ or $-1$, and $k$ is odd, the dot product must be a multiple of $1/\sqrt{k}$. Thus, $\gamma \geq 1/\sqrt{k}$. Clearly, $D = \sqrt{n}$.

Applying Theorem 1, we are able to guarantee that the Perceptron makes a number of mistakes no more than $nk$. Note that we do not need to know $k$ to run the algorithm.

(Exercise: Verify that we would get the same mistake bound if we scaled the predictions of the experts.)

This example brings out the fact that the Perceptron mistake bound can sometimes have a "hidden" dependence on the dimension of the problem. In this case, the bound also depends on $k$, which can be viewed as the number of "relevant" features of the problem.

## 2 Winnow

The mistake bound that we derived above has a linear dependence on the total number of experts or features. We will now introduce the Winnow algorithm, which was designed to have better performance in cases in

which the total number of features is large but the number of relevant features is much smaller. Winnow looks similar to the Perceptron algorithm, but uses multiplicative weight updates instead of additive.

Like the Perceptron, Winnow maintains a current weight vector $\mathbf{w}_t$ at each round $t$. We will use $w_{i,t}$ to denote the weight on feature $i$ at round $t$, and $x_{i,t}$ to denote the $i$th component of $\mathbf{x}_t$. We will go back to having $y_t \in \{0, 1\}$ instead of $\{-1, +1\}$, and will assume that each $\mathbf{x}_t$ is a binary string in $\{0, 1\}^n$.

**The Winnow Algorithm** (with parameter $\beta$)

1. Initialize all the weights to one: $w_{1,1} = w_{2,1} = \ldots = w_{n,1} = 1$
2. For each example $\mathbf{x}_t$,
   - If $\mathbf{w}_t \cdot \mathbf{x}_t \geq n$, then output 1, else output 0
   - If the algorithm makes a mistake, then
     - If $y_t = 1$, then $\forall i$ such that $x_{i,t} = 1,\ w_{i,t+1} \leftarrow w_{i,t}(1 + \beta)$
     - If $y_t = 0$, then $\forall i$ such that $x_{i,t} = 1,\ w_{i,t+1} \leftarrow \frac{w_{i,t}}{1+\beta}$
     - In both cases ($y_t = 1$ or 0), $\forall\ x_{i,t} = 0,\ w_{i,t+1} \leftarrow w_{i,t}$

     Else $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$

As we can see, if we make a mistake on a positive (or negative) example at time $t$, we increase (or decrease) the weights of all features for which $x_{i,t} = 1$. These are precisely the features that contribute to $\mathbf{w}_t \cdot \mathbf{x}_t$. Because updates are multiplicative, the weights grow faster than in the Perceptron algorithm, and it turns out that Winnow makes fewer mistakes than the Perceptron algorithm when the number of relevant feature is much less than the total number of features.

## 2.1 Learning Majority Functions with Winnow

It can be shown that Winnow can learn majority functions with a mistake bound of $2k^2 \log n$. Clearly, we can see that the Winnow algorithm gives a better mistake bound than the Perceptron for this class when the number of relevant features is much smaller than the number of total features.

The proof of this result is a little complicated, so we will instead show a mistake bound for a slightly more simple case: learning disjunctions with Winnow. The intuitions are similar.

## 2.2 Learning Disjunctions with Winnow

We consider an example of learning monotone disjunctions using linear thresholds with Winnow. We show the algorithm will make at most $O(k \log n)$ mistakes, where $n$ is the total number of variables (i.e., the dimension of $\mathbf{x}_t$) and $k$ is the number of variables that appear in the disjunction.

Recall that a monotone disjunction is a disjunction in which no literals appear negated. It is easy to extend this result to the class of all disjunctions. The $n$ in the mistake bound will increase to $2n$. (Exercise: Figure out how to do this.)

**Theorem 2.** *Suppose there exists a monotone disjunction $f$ of $k$ variables such that for all $t$, $f(\mathbf{x}_t) = y_t$. Then the Winnow algorithm with $\beta = 1$ makes at most $2 + 3k(1 + \log n)$ mistakes.*

**Proof:** We will first bound the number of mistakes that are made on positive examples, and then separately bound the number of mistakes made on negative examples. Adding these bounds will yield the result.

We will refer to the weights of variables that appear in the disjunction as "relevant weights."

**Step 1: Bound the number of mistakes on positive examples ($y_t = 1$)**

We start by making a few observations:

1. When we make a mistake on a positive example, at least one relevant weight is doubled. If no relevant weight is doubled, this would imply that all relevant features were 0, which would in turn imply that $y_t = 0$, a contradiction.

2. The relevant weights never decrease. Suppose that a relevant weight did decrease. This would imply that some relevant feature was 1 when $y_t = 0$, also a contradiction.

3. Once a relevant feature $i$ has a weight $w_{i,t} \geq n$, it can no longer be updated. If $w_{i,t} \geq n$ and $x_{i,t} = 1$, then $\mathbf{w}_t \cdot \mathbf{x}_t \geq n$ and there will be no mistake. If $w_{i,t} \geq n$ and $x_{i,t} = 0$, the weight will not be updated.

Now, consider any relevant feature $i$. Let $\ell$ be the number of times the weight of feature $i$ has been doubled. Since weights are initialized to 1, the weight of $i$ is then $2^\ell$. Since weights are never updated once they are larger than $n$, we can conclude $2^{\ell-1} < n$, i.e., $\ell < \log n + 1$. This means the weight of each relevant feature can not be doubled for more than $\log n + 1$ times.

Since at least one relevant weight will be doubled for each mistake on positive examples, and each weight cannot be doubled more than $\log n + 1$ times, the number of mistakes on positive examples must be less than $k(\log n + 1)$.

**Step 2: Bound the number of mistakes on negative examples ($y_t = 0$)**

For this part of the proof, we will use the sum of the weights, $W_t = \sum_{i=1}^{n} w_{i,t}$, as a potential function and observe how it changes over time.

Let's again state some observations:

1. Initially we have $W_1 = n$.

2. For all $t$, $W_t > 0$.

3. Every time we make a mistake and $y_t = 1$, the sum of weights is increased by at most $n$. This is because if we make mistakes on positive examples, we will double the weight of $w_{i,t}$ if $x_{i,t} = 1$. Therefore, the sum of weight is increased by $\sum_{i:x_{i,t}=1} w_{i,t} = \mathbf{w}_t \cdot \mathbf{x}_t$. Since we made a mistake and labeled the point as negative, we know that $\mathbf{w}_t \cdot \mathbf{x}_t < n$.

4. Every time we make a mistake and $y_t = 0$, the sum of weights is decreased by at least $n/2$. By a similar argument to the one above, in this case the sum decreases by $(1/2)\mathbf{w}_t \cdot \mathbf{x}_t$, and since we labeled the point positive, $\mathbf{w}_t \cdot \mathbf{x}_t \geq n$.

Let $P_t$ be the number of mistakes on positive examples up to time $t$, and $Q_t$ be the number of mistakes on negative examples up to time $t$. We know that $W_1 = n$. By observations 3 and 4, and the fact that $W_t$ is always positive, for any $t$,

$$0 < W_t \leq n + P_t n - Q_t \frac{n}{2} ,$$

which implies that $Q_t < 2P_t + 2$. Therefore, the number of mistakes we make on negative examples will be less than $2k(\log n + 1) + 2$.

Combining the mistake bounds on positive examples and negative examples, we can show that a mistake bound for learning monotone disjunction using Winnow algorithm is $2 + 3k(\log n + 1)$. □