Machine Learning Coms-4771

Machine Learning Theory
The Winnow Algorithm

Lecture 7

Based on Avrim Blum’s notes (see the link at the web page)
Recap

**SPAM Example:** Each email = a boolean vector indicating which phrases appear and which don’t (in some predetermined set of $n$ phrases). Email $x = (x_1, \ldots, x_n) \in \{0, 1\}^n$.

<table>
<thead>
<tr>
<th>100% free</th>
<th>earn $</th>
<th>double your income</th>
<th>weight loss</th>
<th>\ldots</th>
<th>requested</th>
<th>$f(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$x_4$</td>
<td>$x_5$</td>
<td>\ldots</td>
<td>$x_n$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
</tr>
</tbody>
</table>

**Target function/concept:** A monotone disjunction $f(x) = a$ boolean function of the form $\bigvee_{i \in S} x_i$ for some subset $S \subseteq \{1, \ldots, n\}$. (SPAM if at least one of the phrases in $S$ is present).

**Mistake Bound Model:** View learning as a sequence of trials

- The learner gets an unlabeled example $x$, 
- predicts its classification, 
- learns whether or not it made a mistake.

**Goal:** minimize the number of mistakes

**Mistake Bound Definition:** Algorithm $A$ learns a class of functions $C$ with mistake bound $M$ if $A$ makes at most $M$ mistakes on any sequence of examples consistent with some $f \in C$. 


### Simple algorithm for learning a disjunction

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>our prediction of $f(x)$</th>
<th>$f(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 ($x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5 \lor x_6$)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1 ($x_2 \lor x_3 \lor x_4 \lor x_5 \lor x_6$)</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1 ($x_2 \lor x_3 \lor x_4 \lor x_5 \lor x_6$)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1 ($x_2 \lor x_3 \lor x_4 \lor x_5$)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0 ($x_2 \lor x_3$)</td>
<td>0</td>
</tr>
</tbody>
</table>

(mistakes in red; the target is $x_2 \lor x_3$)

- Algorithm: list all features and cross off bad ones on negative examples.
- Makes at most $n$ mistakes.
- Problem: $n$ can be very large! What if the target function is an OR on a small subset of $r$ relevant features?
- Today: Winnow algorithm which gives us a mistake bound of $O(r \log n)$. 
The Winnow Algorithm (for OR functions)

- Initialize the weights \( w_1 = w_2 = \ldots = w_n = 1 \) on the \( n \) variables.
- Given an example \( x = (x_1, \ldots, x_n) \), output 1 if
  \[
  \sum_{i=1}^{n} w_i x_i \geq n,
  \]
  otherwise output 0.
- If the algorithm makes a mistake:
  - (on positive) If it predicts 0 when \( f(x) = 1 \), then for each \( x_i \) equal to 1, double the value of \( w_i \).
  - (on negative) If it predicts 1 when \( f(x) = 0 \), then for each \( x_i \) equal to 1, cut the value of \( w_i \) in half.
## Winnow in Action

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>prediction of $f(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1 = 1$</td>
<td>$w_2 = 1$</td>
<td>$w_3 = 1$</td>
<td>$w_4 = 1$</td>
<td>$w_5 = 1$</td>
<td>$w_6 = 1$</td>
<td>$0$ ($\sum_i x_i w_i = 1 \geq 6?$)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 ($\sum_i x_i w_i = 4 \geq 6?$)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>double $w_i$ for $x_i = 1$</td>
</tr>
<tr>
<td>$w_1 = 1$</td>
<td>$w_2 = 2$</td>
<td>$w_3 = 1$</td>
<td>$w_4 = 2$</td>
<td>$w_5 = 2$</td>
<td>$w_6 = 2$</td>
<td>0 ($2 \geq 6?$)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1 ($6 \geq 6?$)</td>
</tr>
<tr>
<td>$w_1 = 1$</td>
<td>$w_2 = 2$</td>
<td>$w_3 = 1$</td>
<td>$w_4 = 1$</td>
<td>$w_5 = 1$</td>
<td>$w_6 = 1$</td>
<td>halve $w_i$ for $x_i = 1$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0 ($2 \geq 6?$)</td>
</tr>
</tbody>
</table>

(mistakes in red; the target $f(x) = x_2 \lor x_3$, $n = 6$, $r = 2$)

Algorithm repeated:

- On $x$, predict $1(\sum_i w_i x_i \geq n)$.

- (mistake on positive) If it predicts 0 when $f(x) = 1$, then for each $x_i$ equal to 1, double the value of $w_i$.

- (mistake on negative) If it predicts 1 when $f(x) = 0$, then for each $x_i$ equal to 1, cut the value of $w_i$ in half.
**Theorem** The Winnow learns the class of disjunctions with mistake bound of $2 + 3r \lceil \log n \rceil$ when the target concept $f$ is an OR of $r$ variables.

**Proof**

- **(mistakes on positive examples)** Any mistake on a positive doubles the weight of at least one of the variables in $f$. And a mistake on a negative cannot halve any of the relevant weights. Since we can’t make a mistake on a positive when at least one of the weights is $\geq n$, we can make at most $r \lceil \log n \rceil$ mistakes on positive examples.

- **(mistakes on negative examples)** Initially $W = \sum_i w_i = n$. Each mistake on a positive increases $W$ by at most $n$ (since we had $W \leq n$ and predicted 0 instead of 1). Each mistake on a negative, decreases $W$ by at least $n/2$. Letting $m_n$ and $m_p$ be the number of mistakes on negatives and positives respectively,

  $$n + n \cdot m_p - \frac{n}{2} m_n > 0,$$

  since $W$ always remains positive. Simplifying, $m_n < 2m_p + 2$.

- **Total number of mistakes** $3r \lceil \log n \rceil + 2$. 


What if the examples are not completely consistent with a disjunction?

- A positive example satisfying none of relevant variables can cause $W$ to increase by at most $n$ (resulting in at most 2 additional mistakes on negatives to bring it back down; indeed, each time we predict 1 on a 0, we decrease the irrelevant weight in $W$ by at least $n/2$).

- A negative example satisfying $t$ relevant variables can cause $t$ relevant weights to be halved (resulting in at most $t$ more mistakes on positives to fix, in turn causing up to $2t$ mistakes on negatives).

- Mistake bound goes up by at most $O(\#\text{attribute errors})$. 
Winnow is more general: It can learn the class of linear threshold functions \( f(x) = 1 \) if \( \sum_i a_i x_i \geq b \) for non-negative integers \( a_1, \ldots, a_n, b \).

An \( r \)-OR corresponds to the case when \( b = 1 \) and \( a_i = 1 \) for the \( r \) relevant variables and 0 for others.

Encodes other important functions as well. Read Littlestone's paper linked at the web page.
Think of $N$ experts giving advice to you. (Expert = someone with an opinion, not necessarily someone who knows anything.) There doesn’t have to be a perfect expert.

Want to do nearly as well as the best expert in hindsight.

Can view each expert as a different $f \in C$.

Example: We want to predict the stock market.

<table>
<thead>
<tr>
<th>Expert 1</th>
<th>Expert 2</th>
<th>Expert 3</th>
<th>...</th>
<th>Expert $N$</th>
<th>truth</th>
</tr>
</thead>
<tbody>
<tr>
<td>down</td>
<td>up</td>
<td>up</td>
<td>...</td>
<td>down</td>
<td>up</td>
</tr>
<tr>
<td>up</td>
<td>up</td>
<td>up</td>
<td>...</td>
<td>down</td>
<td>down</td>
</tr>
<tr>
<td>down</td>
<td>down</td>
<td>up</td>
<td>...</td>
<td>down</td>
<td>...</td>
</tr>
</tbody>
</table>

If one expert is perfect, can get at most $\log N$ mistakes with halving algorithm. What if none is perfect? Can we do nearly as well as the best one in hindsight?
Simple Strategy: Iterated Halving

- Run halving, but restart every time we’ve crossed off all experts.
- Makes at most \((\log N)(m + 1)\) mistakes, where \(m\) is the number of mistakes made by the best expert in hindsight.
- Seems wasteful. We keep forgetting everything we’ve learned. Can we do better?
Weighted Majority Algorithm

Making a mistake shouldn't disqualify an expert. Instead of crossing off, just lower the expert’s weight.

Algorithm:

- Start with all experts having weight 1: $w_1 = w_2 = \ldots = w_N = 1$
- Predict based on weighted majority vote: Output 1 if
  \[
  \sum_{i : x_i = 1} w_i \geq \sum_{i : x_i = 0} w_i,
  \]
  otherwise output 0.
- Penalize mistakes by cutting weight in half. If expert $i$ made a mistake, set $w_i \leftarrow w_i/2$; otherwise, keep the weight unchanged.
Weighted Majority Algorithm: Analysis

**Theorem:** The number of mistakes $M$ made by the Weighted Majority is never more than $2.41(m + \log N)$, where $m$ is the number of mistakes made by the best expert so far.

**Proof:** $W = \sum_i w_i =$ total weight, initially $W = N$. After each mistake, at least half of the total weight of experts predicts incorrectly, so $W$ goes down by at least a factor of $1/4$. After the algorithm makes $M$ mistakes, we have

$$W \leq N(3/4)^M.$$ 

If the best expert has made $m$ mistakes, its weight is $1/2^m$ and so

$$W \geq 1/2^m.$$ 

Combining gives $1/2^m \leq N(3/4)^M$. Solving for $M$:

$$M \leq \frac{1}{\log(4/3)}(m + \log N) \leq 2.41(m + \log N).$$
2.41(m + log N) is not so good if the best expert makes a mistake 20% of the time. Can we do better? Yes.

Instead of taking majority vote, use weights as probabilities. So if 70% of the weight predicts “yes”, and 30% predicts “no”, pick 70:30. Intuition: smooth out the worst case.