On-line Algorithms

Cancelled: office hours Friday May 13
Problem 1 on homework #6 clarified
On-line Algorithms

• A Santa Cruz specialty
• For more info, see Avrim Blum’s “On-line algorithms in Machine Learning”
• Rich and interesting Theory
• Learn as you go -- lifelong learning
• Examples test and then train
On-line model:

• Learning is a sequence of trials
• On each trial:
  – Learner gets a (new) instance $x$
  – Learner predicts $y'$
  – Learner gets label $y$ and earns loss $L(y, y')$
• Common losses: (expected) number of mistakes, cumulative square loss, log loss: $\log(1/p_y)$
• Usually want to minimize worst case loss (relative to a comparator) -- $(x, y)$ adversarial
• Also called prediction of individual sequences
On-line algorithm successes

• Calendar completion (Blum 1997)
• Disk Spin-down (Helmbold et. al. 2000)
• Adaptively choosing caching strategies (Gramacy et. al 2002)
• Really massive data sets or streams
**boolean disjunctions (Rivest)**

- Instance $\mathbf{x}$ has $n$ boolean attributes $x_1, \ldots, x_n$
- Learn disjunction, like $x_2 \lor \neg x_5 \lor x_6$ (no noise)
- Algorithm: Keep set of all possible attributes
  - Init: $S =$ all $2n$ literals (attributes and negations)
  - Predict: true iff any literals in $S$ true
  - Update: if predict true but $\mathbf{x}$ labeled false:
    - Remove all literals satisfied by $\mathbf{x}$ from $S$
- Claim: no false positive predictions:
  - $S$ is a superset of target disjunction
- At most $n+1$ mistakes: $|S| \leq 2n, n, n-1, \ldots, 0$
On-line concept learning

• Given a finite concept class $C$ (like intervals of $\{0,1,2, \ldots, k\}$)
• Goal: do nearly as well as best interval
• Assume some interval perfect
• Halving algorithm:
  – predict with majority of the version space
  – Each mistake halves version space
  – Number of mistakes bounded by $\lg(|C|)$

What is worst case?
Randomized Halving Algorithm (Gibb’s Algorithm)

• This algorithm predicts randomly based on how the version space is split
• Number of mistakes depends on outcome of randomization
• Expected number of mistakes at most:
  \[ \ln(|C|) \]
Gibbs Analysis

• Let $v = \text{size of version space}$
• Consider potential $= \lg(v)$, # of halvings to get to 1
• Initially potential $= \lg(|C|)$, can drop to $\lg(1)=0$
• On arbitrary trial, let, Let $r$ be fraction of version space that is correct
  – Probability of mistake is $(1-r)$
  – New potential $= \lg(rv) = \lg(v) - \lg(1/r)$
  – Expected # mistakes per unit drop in potential is:
    $$\frac{(1-r)}{\lg(1/r)} \leq \ln(2)$$ (approaches $\ln(2)$ as $r \to 1$)
• Expected total # mistakes $\leq \ln(2) \lg(|C|) = \ln(|C|)$
\[ f := \frac{(1-r) \ln(2)}{\ln\left(\frac{1}{r}\right)} \]
A Prediction Game

Adversary chooses \((r, 1-r)\) split of \(v\)
Alg picks prob \(p\) of predicting with \(rv\)-side
Adversary chooses outcome (correct side)

<table>
<thead>
<tr>
<th>Outcome</th>
<th>rv correct</th>
<th>p</th>
<th>(1-r)v correct</th>
<th>1-p</th>
</tr>
</thead>
<tbody>
<tr>
<td>mistake probability:</td>
<td>1-p</td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Progress:</td>
<td>(\lg(1/r))</td>
<td>(\lg(1/(1-r)))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expected Mistakes/progress:</td>
<td>((1-p) / \lg(1/r))</td>
<td>(p / \lg(1/(1-r)))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Prediction Game analysis

Adversary wants to maximize & algorithm wants to minimize mistakes/progress

\[ E[\text{mistake/prog}] \leq \max[ (1-p) / \lg(1/r); \ p / \lg(1/(1-r)) ] \]

Alg. Minimizes by setting equal & solving for \( p \):

\[ p = \frac{\lg(1/(1-r))}{(\lg(1/r) + \lg(1/(1-r)))} \]

Plugging back in:

\[ E[\text{mistake/prog}] \leq 1/\lg(1/r(1-r)) \leq 1/\lg(4) = 1/2 \]

Expected total number of mistakes \( \leq \lg(|C|)/2 \)
So far:

- On-line learning of *noise-free* concepts
- Several similar algorithms with and without randomization
- Bounds like $\lg(|C|)$ or $\frac{1}{2} \lg(|C|)$ (not surprising)
- Move to more interesting situations
  - Attribute efficient learning of disjunctions
Winnow algorithm (Littlestone 88)

- Simple algorithm for learning linearly separable functions over \{0,1\}-valued attributes
- Robust against small amounts of noise
- Keeps weight vector \(w\), and bounds depend on “gap” like perceptron
- Uses multiplicative rather than additive updates -- has promotion/demotion steps
- Very good with many irrelevant features – efficiently finds relevant features
Some Winnow Applications

• Patent classification (Koster et.al.)
• System call prediction with suffix trees, (Karampatziakis&Kozen, 2009)
• Text Chunking (part-of-speech labeling) (Zhang et.al. 2002)
• Many other uses/extensions
Winnow1: (disjunctions, no noise)

- Pick update factor $a > 1$, and threshold
- Init. weights $w_i = 1$ for each variable/literal
- Predict 1 (true) if $w \cdot x > \text{threshold}$
- False pos. eliminate: set $w_i = 0$ if $x_i = 1$
- False neg. promote: set $w_i = aw_i$ if $x_i = 1$

Mistake Bound (for disjunction of $k$ literals):

$$ak(1 + \lg a \text{ threshold}) + n/\text{threshold}$$

Can set $a = 2$, threshold $= n/2$, getting $2k(\lg n) + 2$

- Halving alg bound: $\lg(n \text{ choose } k) \approx k \lg n - k \lg k$

Halving alg knows $k$, Easy implementation?
Winnow 1 analysis

\(n = \# \text{ of literals}, \text{ threshold}=n/2, \ a=2\)

- Each of the \(k\) relevant literals promoted at most \(\lg(n/2)+1 = \lg(n)\) times
- At most \(k \lg(n)\) promotions
- At most \(k \lg(n)\) false positive mistakes
- Each false positive removes \(\geq n/2\) weight
- Each false negative adds \(\leq n/2\) weight
- Number false pos. \(\geq 2 + \# \text{ false neg.}\)
- At most \(2 + 2k \lg(n)\) mistakes
- \(\text{Doesn’t need to know } k\) compare with \(n+1\) bound
Winnow2 alg:  
\textit{r-of-\textit{k} threshold funct’s, (credit assignment problem)}

- Pick update factor $a>1$ and threshold
- Init. weights $w_i=1$ for each variable/literal
- Predict 1 (true) if $w \cdot x > \text{threshold}$
- False pos. demote: set $w_i = w_i / a$ if $x_i=1$
- False neg. promote: set $w_i = aw_i$ if $x_i=1$

Bound for r-of-k threshold functions*:

\[ 8r^2 + 5k + 14kr \ln n \]

Can tolerate some noise

*: by setting $a = 1 + 1/(2r)$ ; threshold = $n$
Expert Setting (LW 94, CFHHSW 97)

- Learner competes against a class of other predictors (the experts) could be concepts
- No expert perfect, but want to do almost as well as best expert in class
- Learner gets the experts’ predictions, not instances
- Worst case setting - experts can conspire to mislead algorithm
Example: weather prediction

<table>
<thead>
<tr>
<th>Day</th>
<th>KGO</th>
<th>KCBS</th>
<th>KNBR</th>
<th>Mercury</th>
<th>Chronical</th>
<th>Y.weather</th>
<th>Pred y’</th>
<th>Result y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rain</td>
<td>Sun</td>
<td>Rain</td>
<td>Rain</td>
<td>Rain</td>
<td>Sun</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>
Example: weather prediction

<table>
<thead>
<tr>
<th>Day</th>
<th>KGO</th>
<th>KCBS</th>
<th>KNBR</th>
<th>Mercury</th>
<th>Chronical</th>
<th>Y.weather</th>
<th>Pred y'</th>
<th>Result y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rain</td>
<td>Sun</td>
<td>Rain</td>
<td>Rain</td>
<td>Rain</td>
<td>Sun</td>
<td>Rain</td>
<td>sun</td>
</tr>
</tbody>
</table>
Example: weather prediction

<table>
<thead>
<tr>
<th>Day</th>
<th>KGO</th>
<th>KCBS</th>
<th>KNBR</th>
<th>Mercury</th>
<th>Chronical</th>
<th>Y.weather</th>
<th>Pred y’</th>
<th>Result y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rain</td>
<td>Sun</td>
<td>Rain</td>
<td>Rain</td>
<td>Rain</td>
<td>Sun</td>
<td>Rain</td>
<td>sun</td>
</tr>
<tr>
<td>2</td>
<td>Rain</td>
<td>Sun</td>
<td>Sun</td>
<td>Sun</td>
<td>Rain</td>
<td>Rain</td>
<td>Sun</td>
<td>rain</td>
</tr>
<tr>
<td>3</td>
<td>Rain</td>
<td>Rain</td>
<td>Rain</td>
<td>Rain</td>
<td>Rain</td>
<td>Sun</td>
<td>Sun</td>
<td></td>
</tr>
</tbody>
</table>
Expert Setting (cont)

- Algorithms must quickly find good expert, but must also hedge bets
- Competitive bounds in terms of best expert’s loss - if all expert’s are bad, then the algorithm will be too
- Worst case bounds often have form
  \[ L_{\text{alg}} < L_{\text{best}} + O(\lg(N) + (\lg(N) L_{\text{best}})^{1/2}) \]

(here \(N\)=# of experts; \(L_{\text{alg}}\)=Loss of algorithm, \(L_{\text{best}}\)=loss of best expert)
Weighted Majority alg:

• Each of $n$ Experts $E_i$ predict 0 or 1
• Weight $w_i$ of $E_i$ starts at 1,
• Each trial:
  – predict with weighted majority of the $E_i$’s
  – Slash weights of wrong $E_i$’s by factor $b<1$
  – (can rescale $w$’s)
Weighted Majority analysis:

- Total weight $W = \sum w_i$ sum over $n$ experts
- On master mistake, new $W \leq (\text{old } W) \frac{1+b}{2}$
- If $m$ master mistakes, $W \leq n \left[\frac{(1+b)/2}{2}\right]^m$
- If some $E_i$ makes $k$ mistakes, $w_i = b^k < W$
- So $b^k < n \left[\frac{(1+b)/2}{2}\right]^m$, solve for $m$ …
Weighted Majority analysis:

- Total weight $W = \sum w_i$ sum over $n$ experts
- On master mistake, new $W \leq (\text{old } W) \frac{1+b}{2}$
- If $m$ master mistakes, $W \leq n \left(\frac{1+b}{2}\right)^m$
- If some $E_i$ makes $k$ mistakes, $w_i = b^k < W$
- So $b^k < n \left(\frac{1+b}{2}\right)^m$, solve for $m$ …

$$m < \frac{\lg n + k \lg \left(\frac{1}{b}\right)}{2 \lg \frac{2}{\frac{1+b}{1+b}}}$$
<table>
<thead>
<tr>
<th>$b$</th>
<th>bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/4</td>
<td>$1.4 \lg n + 2.95 k$</td>
</tr>
<tr>
<td>1/2</td>
<td>$2.4 \lg n + 2.4 k$</td>
</tr>
<tr>
<td>7/8</td>
<td>$10.7 \lg n + 2.07 k$</td>
</tr>
</tbody>
</table>

Bounds logarithmic in $n = \#$ experts

Better bounds with randomized predictions
Regret

• How much better would it be to have done something different?
• Regret on a prediction is extra loss
• Regret on a sequence is the extra loss of a comparator (say the best expert)

on to Hedge (Yoav’s slides)
Extensions:

- **Expert alg**: CFHHSW expert pred. in [0,1], loss/progress randomization, doubling trick(b)
- **Exponentiated Gradient algorithm** learns linear combinations of experts (WK ‘97)
- **One armed bandit** problems (Auer et al): partial feedback
Shifting experts (WH98, BW02)

• Competes against shifting sequences of experts - for example: broker A for boom times, broker B for bust times
• Consider weights normalized to sum to 1
• Problem: if new good expert’s wt ≈ 0, many mistakes for it to “catch up”
• Solution: “share” some of lost weight to all experts before renormalizing
Disk Spin-down

- Spin-down hard drive to save power, but spinning it up costs power
- If drive idle for a time-out duration then spin it down
- Want to learn good time-out durations
- Each “expert” is a fixed time-out duration
- Adaptive expert algorithm uses less energy than best time-out in hindsight (HLSS ’00)
Caching

• Many page replacement policies (LRU, LFU, etc.) which is best depends on workload
• Use all policies as experts
  – Must compute actions and losses by keeping meta data for each policy
  – Need to update cache when switching policies
• Switching policies to fit current workload gives good results (GWBA ‘02)
On-Line Summary

- Model: Competitive On-line rather than batch; best shifting or linear combination of features/experts
- Data: whatever experts need
  - experts can be boolean or numeric
- Interpretable? Yes
- Missing values? (sleeping experts)
- Noise/outliers? Good -- depending on $b, \eta$
- Irrelevant features/experts? Pretty good
- Comp. efficiency? Good