On-line Algorithms

On-line Algorithms

• A Santa Cruz specialty
• For more info, see Avrim Blum’s “On-line algorithms in Machine Learning” and Manfred Warmuth’s web page,
• Rich and interesting Theory
• Learn as you go -- lifelong learning
• No training examples, all testing

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)
• 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)
• Classifying patents (Koster et.al. 2002)

boolean disjunctions (Rivest)

• Each \( x \) has \( n \) boolean attributes \( x_1, \ldots, x_n \)
• Learn e.g. \( x_2 \lor \neg x_5 \lor x_6 \) (no noise)
• Algorithm:
  – Init: \( S = \) all \( 2n \) literals (attributes and negations)
  – Predict: true iff any literals in \( S \) true
  – Update: if pred true but \( x \) labeled false:
    • Remove all literals satisfied by \( x \) from \( S \)
• Claim: no false positive predictions:
  \( S \) is a superset of target disjunction
• At most \( n+1 \) mistakes: \(|S| = 2n, n, n-1, \ldots, 0\)

Simple on-line task

• Given a finite concept class \( C \) (like intervals of \( \{0,1,2, \ldots, k\} \))
• Predict 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 \( \lg(|C|) \)
Randomized Halving Algorithm (Gibb’s Algorithm)

- This algorithm predicts randomly based on how the version space is predicted.
- Number of mistakes depends on outcome of randomization.
- Expected number of mistakes at most: \( \ln(|C|) \)
- Also bounds Absolute loss.
- Analysis: …

Gibbs Analysis

- Let \( v \) be the size of the version space, \( n = |C| \) consider potential = \( \lg(v) \), initially \( \lg(n) \), drops to \( \lg(1) = 0 \)
- Consider 1 trial, let \( rv \) of version space be correct, \( r \) is in \( (0,1] \)
  - Probability of mistake is \( (1-r) \)
  - New potential = \( \lg(rv) = \lg(v) - \lg(1/r) \)
  - Expected mistakes per unit drop in potential is: \( (1-r)\lg(1/r) \) maximum of \( \ln(2) \) as \( r \to 1 \)
- Expected total loss \( \leq \ln(2) \lg(n) = \ln(n) \)

Better Randomized Prediction

Adversary chooses \((r, 1-r)\) split of \( v \)
Alg chooses \( p \) of predicting with \( r \)-side
Adversary chooses outcome (correct side)
Expected Loss is \( 1-p \) or \( p \);
Progress is \( \lg(1/r) \) or \( \lg(1/(1-r)) \)
\( E. \) Loss/progress \( \leq \max( (1-p)/\lg(1/r); p/\lg(1/(1-r)) \) ]
Set eq & solve: \( p = \lg(1/(1-r)) / (\lg(1/r) + \lg(1/(1-r)) \)
\( E. \) loss/progress \( \leq 1/\lg(1/(1-r)) \leq 1/\lg(4) = 1/2 \)

Winnow algorithm (Littlestone 88)

- Simple algorithm for learning linearly separable functions and \( \{0,1\} \)-valued attributes
- Robust against small amounts of noise
- Keeps weight vector \( w \) like perceptron, bounds depend on "gap"
- Uses multiplicative rather than additive updates -- has promotion/demotion steps
- Very good with many irrelevant features
- Used with some success for text classification

Winnow1 alg:

- Pick update factor \( a > 1 \), and threshold
- Init. weights \( w_i = 1 \) for each variable/literal
- Predict 1 (true) if \( w \cdot x > \) threshold
- False pos. eliminate: set \( w_i = 0 \) if \( x_i = 1 \)
- False neg. promote: set \( w_i = aw \) if \( x_i = 1 \)
- Mistake Bound (k-disjunctions):
  \( ak(1+\lg t) + n/\text{threshold} \)
  Can set \( a = 2, \) threshold\( = n/2, \) getting \( 2k(\lg n) + 2 \)
  - Halving alg bound: \( \lg(n) \) choose \( k \) = \( k \lg n - k \lg k \)
  - Easy implementation?
Winnow\textsuperscript{2} alg: r-of-k threshold funct’s, (credit assignment)

- Pick update factor $a > 1$ and threshold
- Init. weights $w_i = 1$ for each variable/literal
- Predict 1 (true) if $w \cdot x >$ 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\textsuperscript{*}:
$$8r^2 + 5k + 14kr \ln n$$
Can tolerate some noise

\textsuperscript{*}: by setting $a = 1 + \frac{1}{2r}$; 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>Chronical</th>
<th>Mercurey</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>Chronical</th>
<th>Mercurey</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>Sun</td>
<td>Rain</td>
<td>Rain</td>
<td>Sun</td>
<td>Rain</td>
</tr>
<tr>
<td>2</td>
<td>Rain</td>
<td>Sun</td>
<td>Rain</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>Sun</td>
</tr>
</tbody>
</table>

Example: weather prediction

<table>
<thead>
<tr>
<th>Day</th>
<th>KGO</th>
<th>KCBS</th>
<th>KNBR</th>
<th>Chronical</th>
<th>Mercurey</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>Sun</td>
<td>Rain</td>
<td>Rain</td>
<td>Sun</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>Chronical</th>
<th>Mercurey</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>Sun</td>
<td>Rain</td>
<td>Sun</td>
<td>Rain</td>
<td>Sun</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
$$L_{alg} \leq L_{best} + O(\lg(N) + (\lg(N) L_{best})^{1/2})$$
(here N=\# of experts; $L_{alg}$=Loss of algorithm, $L_{best}$=Loss of best expert)
WM alg:

- Each of $n$ Experts $E_i$ predict 0 or 1
- Weight $w_i$ of $E_i$ starts at 1, slashed by $b < 1$
  each time $E_i$ makes mistake (can rescale $w$’s)
- Total weight $W = \sum w_i$
- On master mistake, new $W \leq (\text{old } W)(1+b)/2$
- If $m$ master mistakes, $W \leq n \left[(1+b)/2\right]^m$
- If some $E_i$ makes $k$ mistakes, $w_i = b^k < W$
- So $b^k < n \left[(1+b)/2\right]^m$, solve for $m$ ...

$m < \frac{\lg n + k \lg(1/b)}{\frac{2}{1 + b}}$

<table>
<thead>
<tr>
<th>$b$</th>
<th>Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/4</td>
<td>$1.4 \lg n + 2.95k$</td>
</tr>
<tr>
<td>1/2</td>
<td>$2.4 \lg n + 2.4k$</td>
</tr>
<tr>
<td>7/8</td>
<td>$10.7 \lg n + 2.07k$</td>
</tr>
</tbody>
</table>

Better bounds with randomized predictions

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 -- depends on learning rate $\theta$
- **Irrelevant features/experts?** Pretty good
- **Comp. efficiency?** Good