On-line Algorithms
On-line Algorithms

- A Santa Cruz specialty
- For more info, see Avrim Blum’s “On-line algorithms in Machine Learning”, or Manfred’s Tutorial
- 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
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:
  \[ \ln(|C|) < \lg(|C|) \]
  (even better method gets \( \frac{\lg(|C|)}{2} \)).
Gibbs Analysis

- Let \( v \) = size of version space
- Consider potential = \( \log(v) \), \# of halvings to get to 1
- Initially potential = \( \log(|C|) \), can drop to \( \log(1)=0 \)
- On arbitrary trial, let, Let \( r \) be fraction of version space that is correct
  - Probability of mistake is \( (1-r) \)
  - New potential = \( \log(rv) = \log(v) - \log(1/r) \) \# of halvings
  - Expected \# mistakes per unit drop in potential is:
    \[ (1-r)/\log(1/r) \leq \ln(2) \] (approaches \( \ln(2) \) as \( r \to 1 \))
- Expected total \# mistakes \leq \( \ln(2) \log(|C|) = \ln(|C|) \)
\[ f := (1-r) / \log_2(1/r) \]

\[ f := \frac{(1-r) \ln(2)}{\ln\left(\frac{1}{r}\right)} \]

> limit(f, r=1);

> plot(f, r=0..1);

**What is worst case?**
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
Expert Setting (LW 94, CFHHSW 97)

• Learner competes against a class of other predictors (the experts) concepts or algorithms
• 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, \( W \leq n \left(\frac{1+b}{2}\right)^m \)
- 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 \( m \) …
Weighted Majority analysis:

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

$$m < \frac{\lg n + k \lg(1 / b)}{2/\lg \frac{2}{1 + b}}$$
<table>
<thead>
<tr>
<th>$b$</th>
<th>bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/4</td>
<td>$1.4 \lg n + 2.95 , k$</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 over a comparator (say the best expert)
• Regret is the benefit of hindsight

on to Hedge (Yoav’s slides)
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)
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, uses upper confidence bounds (UCB, Monte-carlo tree search)
Portfolio story

• Constant rebalanced portfolio:
  – Several investments
  – Keep a fixed fraction of your wealth in each investment
  – Exploits fluctuations in investment values
  – What are the right proportions?
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