Computational Learning Theory

What general laws constrain inductive learning?

We seek theory to relate:

- Probability of successful learning
- Number of training examples
- Complexity of hypothesis space
- Accuracy to which target concept is approximated
- Manner in which training examples presented

Slide CS478–1

PAC Learning Setting

Given:
- set of instances $X$
- set of hypotheses $H$
- set of possible target concepts $C$
- training instances generated by a fixed, unknown probability distribution $\mathcal{D}$ over $X$

Learner observes a sequence $D$ of training examples of form

$\langle x, c(x) \rangle$, for some target concept $c \in C$

- instances $x$ are drawn from distribution $\mathcal{D}$
- teacher provides target value $c(x)$ for each

Learner must output a hypothesis $h$ estimating $c$

- $h$ is evaluated by its performance on subsequent instances drawn according to $\mathcal{D}$

Slide CS478–2
True Error of a Hypothesis

Instance space $X$

Definition: The true error (denoted $\text{error}_D(h)$) of hypothesis $h$ with respect to target concept $c$ and distribution $D$ is the probability that $h$ will misclassify an instance drawn at random according to $D$.

$$\text{error}_D(h) = \Pr_{x \in D} [c(x) \neq h(x)]$$

Slide CS478–3

PAC Learning

Consider a class $C$ of possible target concepts defined over a set of instances $X$ of length $n$, and a learner $L$ using hypothesis space $H$.

Definition: $C$ is PAC-learnable by $L$ using $H$ if for all $c \in C$, distributions $D$ over $X$, $\epsilon$ such that $0 < \epsilon < 1/2$, and $\delta$ such that $0 < \delta < 1/2$,

learner $L$ will with probability at least $(1 - \delta)$ output a hypothesis $h \in H$ such that $\text{error}_D(h) \leq \epsilon$, in time that is polynomial in $1/\epsilon$, $1/\delta$, $n$ and $\text{size}(c)$.

Slide CS478–4
Mistake Bounds

So far: how many examples needed to learn?

What about: how many mistakes before convergence?

Let’s consider similar setting to PAC learning:

- Instances drawn at random from $X$ according to distribution $D$
- Learner must classify each instance before receiving correct classification from teacher
- Can we bound the number of mistakes learner makes before converging?

Slide CS478–5

Mistake Bounds: Find-S

Consider Find-S when $H = \text{conjuntion of boolean literals}$

Find-S:

- Initialize $h$ to the most specific hypothesis $l_1 \land \lnot l_1 \land l_2 \land \lnot l_2 \ldots l_n \land \lnot l_n$
- For each positive training instance $x$
  - Remove from $h$ any literal that is not satisfied by $x$
- Output hypothesis $h$.

How many mistakes before converging to correct $h$?

Slide CS478–6
Mistake Bounds: Halving Algorithm

Consider the Halving Algorithm:

- Learn concept using version space Candidate-Elimination algorithm
- Classify new instances by majority vote of version space members

How many mistakes before converging to correct \(h\)?

- ... in worst case?
- ... in best case?

Slide CS478–7

Optimal Mistake Bounds

Let \(M_A(C)\) be the max number of mistakes made by algorithm \(A\) to learn concepts in \(C\). (maximum over all possible \(c \in C\), and all possible training sequences)

\[
M_A(C) \equiv \max_{c \in C} M_A(c)
\]

Definition: Let \(C\) be an arbitrary non-empty concept class. The optimal mistake bound for \(C\), denoted \(Opt(C)\), is the minimum over all possible learning algorithms \(A\) of \(M_A(C)\).

\[
Opt(C) \equiv \min_{A \in \text{learning algorithms}} M_A(C)
\]

Slide CS478–8
**Weighted Majority Algorithm**

- generalization of the HALVING algorithm
- makes predictions by taking a weighted vote among a pool of prediction algorithms
- learns by altering the weight associated with each prediction algorithm
- accommodates inconsistent training data
- can bound the number of mistakes made

*Slide CS478–9*

Tom's slide goes here. Table 7.1.

*Slide CS478–10*
Relative Mistake Bound for Weighted Majority

Let $D$ be any sequence of training examples.
Let $A$ be any set of $n$ prediction algorithms.
Let $k$ be the minimum number of mistakes made by any algorithm in $A$ for the training sequence $D$.
Then the number of mistakes of $D$ made by the Weighted-Majority algorithm using $\beta = 1/2$ is at most

$$2.4(k + \log_2 n)$$

Empirical Support for Multiplicative Update Algorithms

Calendar scheduling

*Given*: Description of an event to be scheduled
*Predict*: Event’s location, duration, start time, day of week.

Features:
- type of event
- name of the seminar
- position of attendees
- are attendees in the user’s group
- names of the attendees in alphabetical order
Example

(req-event-type meeting) (req-seminar-type nil)
(sponsor-attendees no-value) (department-attendees cs)
(position-attendees faculty) (group-attendees? no)
(req-course-name nil) (department-speakers no-value)
(group-name no-value) (lunchtime? no)
(single-person? yes) (number-of-person 1)
(req-location dh4301c)

1685 examples

Features of the Learning Task

- “Target concept” changes with time.
- Set of possible values for each feature may not be known.

Baseline system: Calendar ApPrentice System

- decision-tree based learning method
- acquires rules sorted by observed performance
- system is run each night using the most recent 180 examples
- merges the new rules into the existing rule set
**Weighted Majority Implementation**

Assumption: Some small set of features will be enough to construct a good predictor.

1. For each pair of features, create one “expert” (prediction algorithm) that examines only those two features and makes predictions based on their values.

2. Weight update has $\beta = 1/2$

3. Each expert performs a simple table lookup.
   - Given a pair of values for its two features, look at the last $k$ times that the pair of values occurred and predict the outcome that occurred most often out of those $k$. ($k = 5$)
   - If the pair of values has never occurred before, predict the most common class value seen so far.

---

**Weighted Majority Extension**

*Speedup strategy:*

- Discard experts if their weights drop too low.
- Allows algorithm to speed up as it learns more.
- Danger if too aggressive in discarding experts.
- Found that for a wide range of thresholds, one can achieve both a significant speedup and negligible loss in performance.
Winnow

Combines opinions of “specialists” that can abstain on any example.

- Create one specialist for each pair of feature—value conditions seen so far.
- Specialist wakes up to make a prediction if both conditions are true.
- Predicts the most popular outcome out of the last $k = 5$ times it had a chance to predict.
- Global prediction is based on a weighted majority vote over all predicting specialists.

Slide CS478–17

- When specialist $i$ first appears, $w_i \leftarrow 1$ and abstains for this example.
- Weight update strategy:
  - If global prediction incorrect,
    * $w_i = 1/2 \cdot w_i$ for $a_i$ that predict incorrectly
    * $w_i = 3/2 \cdot w_i$ for $a_i$ that predict correctly
  - If global prediction correct,
    * $w_i = 1/2 \cdot w_i$ for $a_i$ that predict incorrectly

Slide CS478–18
### Experimental Results

<table>
<thead>
<tr>
<th>Task</th>
<th>CAP</th>
<th>Winnow</th>
<th>Winnow-big</th>
<th>WM</th>
<th>WM-big</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>0.64</td>
<td>0.75</td>
<td>0.76</td>
<td>0.70</td>
<td>0.74</td>
</tr>
<tr>
<td>duration</td>
<td>0.63</td>
<td>0.71</td>
<td>0.74</td>
<td>0.64</td>
<td>0.73</td>
</tr>
<tr>
<td>start-time</td>
<td>0.34</td>
<td>0.51</td>
<td>0.53</td>
<td>0.39</td>
<td>0.50</td>
</tr>
<tr>
<td>day-of-week</td>
<td>0.50</td>
<td>0.57</td>
<td>0.57</td>
<td>0.56</td>
<td>0.56</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>0.53</td>
<td>0.63</td>
<td>0.65</td>
<td>0.57</td>
<td>0.63</td>
</tr>
</tbody>
</table>

**Slide CS478–19**

### Comments

For the Weighted Majority algorithm, the weights answer the question: “if you were only allowed to look at two features, which two do you choose?”

When predicting location,

- best feature: number of people
- best pair: number of people + seminar type

**Slide CS478–20**
Winnow assigns weights to each possible rule of length 2, indicating the extent to which that rule should be trusted:

- If there is a single attendee and he/she is from the ECE department, then 30 minutes.
- If there is more than one attendee and they are research programmers, the 60 minutes.
- If the attendees are faculty members and not from CMU, then 60 minutes.

**Slide CS478–21**

**Bagging Classifiers**

Bagging = Bootstrap aggregating

- A learning(data) set $L$ consists of data 
  $\{(y_n, x_n) : n = 1, \ldots, N\}$. Each $x_n$ is a feature vector; $y_n$ is a class.

- Assume that we have some learning algorithm that can use $L$ to form a classifier $\varphi(x, L)$ that predicts $y$ given $x$.

- Given a sequence of data sets $\{L_k\}$ each with $N$ independent observations drawn from the same distribution as $L$, we can form a sequence of predictors $\{\varphi(x, L_k)\}$.

**Slide CS478–22**
Goal in **bagging** is to use the \( \{L_k\} \) to get a better predictor than the single data set predictor \( \varphi(x, L) \).

One obvious procedure is to replace \( \varphi(x, L) \) by:

- **discrete** \( y \): the majority vote of the \( k \) \( \varphi \)'s
- **numeric** \( y \): the average prediction of the \( k \) \( \varphi \)'s

---

**Bagging Approximates Multiple Data Sets**

Take repeated bootstrap samples \( \{L^{(B)}\} \) from \( L \) and form \( \{\varphi(x, L^{(B)})\} \).

*Bootstrap sampling:* Given set \( L \) containing \( N \) training examples, create \( L^i \) by drawing \( N \) examples at random with replacement from \( L \).

*Hypothesis:* aggregating over bootstrap samples yields higher accuracy than a single classifier.

**Bagging:**

- Create \( k \) bootstrap samples \( L^1 \ldots L^k \).
- Train distinct classifier on each \( L^i \).
- Classify new instance by majority vote / average.
Experimental Method

Given sample $S$ of labeled data, do 100 times and report average

1. Split $S$ randomly into test set $T$ (10%) and training set $D$ (90%).

2. Learn decision tree from $D$
   - $e_S$ ← error of tree on $T$

3. Repeat 50 times: Create bootstrap set $D^i$, construct decision tree using $D$.
   - $e_B$ ← error of majority vote using trees to classify $T$

---

Results

<table>
<thead>
<tr>
<th>Data Set</th>
<th>$e_S$</th>
<th>$e_B$</th>
<th>Decrease</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waveform</td>
<td>29.1</td>
<td>19.3</td>
<td>34%</td>
</tr>
<tr>
<td>Heart</td>
<td>4.9</td>
<td>2.8</td>
<td>43%</td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>5.9</td>
<td>3.7</td>
<td>37%</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>11.2</td>
<td>7.9</td>
<td>29%</td>
</tr>
<tr>
<td>Diabetes</td>
<td>25.3</td>
<td>23.9</td>
<td>6%</td>
</tr>
<tr>
<td>Glass</td>
<td>30.4</td>
<td>23.6</td>
<td>22%</td>
</tr>
<tr>
<td>Soybean</td>
<td>8.6</td>
<td>6.8</td>
<td>21%</td>
</tr>
</tbody>
</table>

---

Slide CS478–25

---

Slide CS478–26
**How many bootstrap samples are enough**

<table>
<thead>
<tr>
<th>Number bootstrap samples</th>
<th>Misclassification Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29.1</td>
</tr>
<tr>
<td>10</td>
<td>21.8</td>
</tr>
<tr>
<td>25</td>
<td>19.4</td>
</tr>
<tr>
<td>50</td>
<td>19.3</td>
</tr>
<tr>
<td>100</td>
<td>19.3</td>
</tr>
</tbody>
</table>

Slide CS478–27

**When Will Bagging Improve Accuracy?**

Depends on the stability of the base-level classifiers.

A learner is *unstable* if a small change to the training set causes a large change in the output hypothesis.

- If small changes in $L$ cause small changes in $\varphi$ then $\varphi \approx \varphi_B$.
- If small changes in $L$ cause large changes in $\varphi$ then there will be an improvement in performance.

Slide CS478–28
**Conclusion of Experiments**

- Bagging helps unstable procedures.
- Bagging hurts the performance of stable procedures.
- Neural nets, decision/regression trees, linear regression are unstable.
- k-nn is stable.

---

**Bagging Nearest Neighbor Classifiers**

No difference between $e_S$ and $e_B$

Reason:

Probability than a particular instance will be in any one Bootstrap replicate is .632

An instance $x$ will have a different label predicted for it by the aggregate method only if $x$’s nearest neighbor is missing from at least half of bootstrap learning sets

The probability of this happening is $P(\text{number of heads in N tosses is less than N/2})$ when the probability of a head is .632

Clearly as N grows this gets small.