# Notes on Machine Learning for 16.410 and 16.413

(Notes adapted from Tom Mitchell and Andrew Moore.)

## Learning = improving with experience

- Improve over task T (e.g, Classification, control tasks)
- with respect to performance measure P (e.g., accuracy, speed, etc.)
- based on experience E (direct, indirect, teacher-provided, from exploration).

Notation:

- Instances $x_1, x_2, \ldots$
- Target concept $C$ labels instance $x$ with label $c(x)$
- Labelled data $D$ is a set of pairs $\langle x, c(x)\rangle$
- Hypothesis $h$ is a possible target concept that also labels (correctly or incorrectly) each instance $x$ with a label $h(x)$

## The inductive learning hypothesis

Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

## Hypotheses and Version spaces

A hypothesis $h$ is **consistent** with a set of training examples $D$ of target concept $c$ if and only if $h(x) = c(x)$ for each training example $\langle x, c(x)\rangle$ in $D$.

$$Consistent(h, D) \equiv (\forall \langle x, c(x)\rangle \in D) h(x) = c(x)$$

The **version space**, $VS_{H,D}$, with respect to hypothesis space $H$ and training examples $D$, is the subset of hypotheses from $H$ consistent with all training examples in $D$.

$$VS_{H,D} \equiv \{h \in H | Consistent(h, D)\}$$

#### List-Then-Eliminate Algorithm

1. $VersionSpace \leftarrow$ a list containing every hypothesis in $H$
2. For each training example, $\langle x, c(x)\rangle$
   remove from $VersionSpace$ any hypothesis $h$ for which $h(x) \neq c(x)$
3. Output the list of hypotheses in $VersionSpace$

**What's wrong with this algorithm?**

## Decision Trees

- Each internal node tests an attribute
- Each branch corresponds to attribute value
- Each leaf node assigns a classification
- Instances describable by attribute-value pairs
- Target function is discrete valued
- Disjunctive hypothesis may be required
- Possibly noisy training data

## Top-Down Induction of Decision Trees

Main loop:

1. $A \leftarrow$ the ”best” decision attribute for next node $n$
2. Assign $A$ as decision attribute for node $n$
3. For each value of $A$, create new descendant of node $n$
4. Assign training examples to leaf nodes
5. If training examples are perfectly classified, then stop, else iterate over new leaf nodes

## ID3 (Quinlan 1986)

“Best” decision attribute maximizes information gain

- $S$ is a sample of data taken from $D$
- $Entropy(S)$ = expected number of bits needed to encode the label $c(x)$ of randomly drawn members of $s$ (under the optimal, shortest-length code)

Information theory (Shannon 1951): optimal length code assigns $-log_2 p$ bits to message having probability $p$. Expected number of bits to encode $c(x)$ of random member $x$ of $S$:

$$\begin{aligned} H(x) &\equiv E\,[\text{number of bits}] \\ &= E\,[-\log_2 p(c(x))] \\ &= -\sum_{c_i} p(c_i(x)) \log_2 p(c_i(x)) \end{aligned}$$

Information theory (Shannon 1951): information gain is the expected reduction in entropy that results from partitioning data (e.g., using attribute $A$).

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

- Hypothesis space includes all possible target concepts
- Outputs a single hypothesis
- Statisically-based search choices
  Robust to noisy data...
- Inductive bias: approx ”prefer shortest tree”

## Overfitting

Consider error of hypothesis $h$ over

- training data: $error_{train}(h)$
- entire distribution $D$ of data: $error_D(h)$

  Hypothesis $h \in H$ overfits training data if there is an alternative hypothesis $h' \in H$ such that

  $$error_{train}(h) < error_{train}(h')$$

  and

  $$error_D(h) > error_D(h')$$

How can we avoid overfitting?

- stop growing when data split not statistically significant
- grow full tree, then post-prune

How to select ”best” tree:

- Measure performance over training data
- Measure performance over separate validation data set
- MDL: minimize $size(tree) + size(misclassifications(tree))$

Reduced-Error Pruning

- Split data into $training$ and $validation$ set
- Do until further pruning is harmful:

  1. Evaluate impact on $validation$ set of pruning each possible node (plus those below it)
  2. Greedily remove the one that most improves $validation$ set accuracy

- produces smallest version of most accurate subtree
- What if data is limited?

## Rule Post-Pruning

1. Convert tree to equivalent set of rules
2. Prune each rule independently of others
3. Sort final rules into desired sequence for use

Perhaps most frequently used method (e.g., C4.5)

## Continuous Valued Attributes

Create a discrete attribute to test continuous

- $Temperature = 82.5$
- $(Temperature > 72.3) = t, f$

## Attributes with Many Values Problem

- If attribute has many values, $Gain$ will select it
- Imagine using $Date = Jun$ 3 1996 as attribute
- One approach: use $GainRatio$ instead

$$
\begin{aligned}
GainRatio(S,A) &\equiv \frac{Gain(S,A)}{EntropyOfSplit(S,A)} \\
EntropyOfSplit(S,A) &\equiv - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}
\end{aligned}
$$

## Unknown Attribute Values

What if some examples missing values of A? Use training example anyway, sort through tree

- If node $n$ tests $A$, assign most common value of $A$ among other examples sorted to node $n$
- assign most common value of $A$ among other examples with same target value
- assign probability $p_i$ to each possible value $v_i$ of $A$
- assign fraction $p_i$ of example to each descendant in tree
- Classify new examples in same fashion

## Some Issues in Machine Learning

- What algorithms can approximate functions well (and when)?
- How does number of training examples influence accuracy?
- How does complexity of hypothesis representation impact it?
- How does noisy data influence accuracy?
- What are the theoretical limits of learnability?
- How can prior knowledge of learner help?
- What clues can we get from biological learning systems?
- How can systems alter their own representations?

## Designing a Learning Algorithm