Learning

*Learning* is improvement of performance (time, accuracy).

*Inductive inference* is improving accuracy by generalizing from experience. An *example* is a single, specific experience.

In *supervised learning*, each example is an input/output pair. *Classification* is when the set of outputs is finite. *Concept learning* is when there are two possible outputs.

In *unsupervised learning*, examples do not always have outputs.

In *reinforcement learning*, an agent performs a series of actions, receiving intermittent feedback.

In *batch* learning, the learner receives all the examples at the same time. In *online* learning, the learner receives the examples one at a time.
<table>
<thead>
<tr>
<th>No.</th>
<th>Attributes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Outlook</td>
<td>Temp</td>
</tr>
<tr>
<td>1</td>
<td>sunny</td>
<td>hot</td>
</tr>
<tr>
<td>2</td>
<td>sunny</td>
<td>hot</td>
</tr>
<tr>
<td>3</td>
<td>overcast</td>
<td>hot</td>
</tr>
<tr>
<td>4</td>
<td>rain</td>
<td>mild</td>
</tr>
<tr>
<td>5</td>
<td>rain</td>
<td>cool</td>
</tr>
<tr>
<td>6</td>
<td>rain</td>
<td>cool</td>
</tr>
<tr>
<td>7</td>
<td>overcast</td>
<td>cool</td>
</tr>
<tr>
<td>8</td>
<td>sunny</td>
<td>mild</td>
</tr>
<tr>
<td>9</td>
<td>sunny</td>
<td>cool</td>
</tr>
<tr>
<td>10</td>
<td>rain</td>
<td>mild</td>
</tr>
<tr>
<td>11</td>
<td>sunny</td>
<td>mild</td>
</tr>
<tr>
<td>12</td>
<td>overcast</td>
<td>mild</td>
</tr>
<tr>
<td>13</td>
<td>overcast</td>
<td>hot</td>
</tr>
<tr>
<td>14</td>
<td>rain</td>
<td>mild</td>
</tr>
</tbody>
</table>

Inductive Learning

The learner learns a *hypothesis* $h$ from a set of *training* examples, and evaluates $h$ empirically on a set of *test* examples or theoretically on the *probability distribution* of the examples.

**Inductive bias** refers to the hypotheses that the learner prefers. One kind of inductive bias is to restrict the *hypothesis space*. 
Perfect learning cannot be guaranteed from a finite set of training examples: the training examples might not cover all the possibilities and might not be representative.

The goal of $PAC$ learning ($PAC = \text{“probably approximately correct”}$) is to find a hypothesis that is unlikely ($\delta$ or less) to have high error ($\epsilon$ or more).

---

**Decision Trees**

- Decision trees are a representation for classification.
  - The root is labeled by an attribute.
  - Edges are labeled by attribute values.
  - Edges go to decision trees or leaves.
  - Each leaf is labeled by a class.

- Growth Phase: The tree is constructed top-down.
  - Find the “best” attribute.
  - Partition examples based on the attribute’s values.

- Pruning Phase: The tree is pruned bottom-up.
  - For each node, keep subtree or change to leaf.
Example of a Decision Tree

Algorithm for Growing Decision Trees

\texttt{Grow\_DT}(\textit{examples})

1. \(N \leftarrow \) a new node
2. \(N.\text{class} \leftarrow \) most common class in \textit{examples}
3. \textbf{if} \textit{examples} have identical class or values
4. \textbf{then return} \(N\)
5. \(N.\text{test} \leftarrow \) best attribute (or test)
6. \textbf{for} each value \(v_j\) of \(N.\text{test}\)
7. \(\textit{examples}_j \leftarrow \textit{examples}\) with \(N.\text{test} = v_j\)
8. \textbf{if} \textit{examples}_j is empty
9. \textbf{then} \(N.\text{branch}_j \leftarrow N.\text{class}\)
10. \textbf{else} \(N.\text{branch}_j \leftarrow \texttt{Grow\_DT}(\textit{examples}_j)\)
11. \textbf{return} \(N\)
Measuring Attributes: Information Gain

- $p$ positive examples and $n$ negative examples
- The information context is:

\[I(p, n) = -\frac{p}{p + n} \log_2 \frac{p}{p + n} - \frac{n}{p + n} \log_2 \frac{n}{p + n}\]

- Attribute $A$ has $v$ values, $p_j$ positive examples and $n_j$ negative examples when $A = v_j$
- The **Remainder** of $A$ is:

\[\text{Remainder}(A) = \sum_{j=1}^{v} \frac{p_j + n_j}{p + n} I(p_j, n_j)\]

- The information gain of $A$ is:

\[\text{Gain}(A) = I(p, n) - \text{Remainder}(A)\]
Example of Attribute Selection

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Sunny</th>
<th>Overcast</th>
<th>Rain</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 pos</td>
<td>4 pos</td>
<td>3 pos</td>
<td></td>
</tr>
<tr>
<td>3 neg</td>
<td>0 neg</td>
<td>2 neg</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{Gain(Outlook)} \approx 0.246 \]

<table>
<thead>
<tr>
<th>Temp</th>
<th>Cool</th>
<th>Mild</th>
<th>Hot</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 pos</td>
<td>4 pos</td>
<td>2 pos</td>
<td></td>
</tr>
<tr>
<td>1 neg</td>
<td>2 neg</td>
<td>2 neg</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{Gain(Temp)} \approx 0.029 \]
Humidity

<table>
<thead>
<tr>
<th></th>
<th>9 pos, 5 neg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>6 pos, 1 neg</td>
</tr>
<tr>
<td>High</td>
<td>3 pos, 4 neg</td>
</tr>
</tbody>
</table>

Gain(Humidity) ≈ 0.152

Wind

<table>
<thead>
<tr>
<th></th>
<th>9 pos, 5 neg</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>3 pos, 3 neg</td>
</tr>
<tr>
<td>False</td>
<td>6 pos, 2 neg</td>
</tr>
</tbody>
</table>

Gain(Wind) ≈ 0.048

Outlook has the highest gain.
Overcast branch is pure.
Need to construct DTs for Sunny and Rain branches.

---

Alternative Attributes Measures

- Maximize Information Gain Ratio
  \[
  GainRatio(A) = \frac{Gain(A)}{I(p_1 + n_1, \ldots, p_v + n_v)}
  \]

- Minimize Gini Index
  \[
  Gini(p, n) = 1 - \left( \frac{p}{p + n} \right)^2 - \left( \frac{n}{p + n} \right)^2
  \]

  \[
  GiniIndex(A) = \sum_{j=1}^{v} \frac{p_j + n_j}{p + n} Gini(p_j, n_j)
  \]

- “Maximize” Chi-Squared Statistic
  \[
  \chi^2 = \sum_{j=1}^{v} \frac{(p_j - p s_j)^2}{p s_j} + \frac{(n_j - n s_j)^2}{n s_j}
  \]
  where \( s_j = \frac{(p_j + n_j)}{(p + n)} \)
Special Cases

- Attribute $A$ is numeric.
  - Solution: Find best $A \leq v$ test. Requires sorting.
  - Solution: Discretization. Partition $A$ into ranges.

- Attribute $A$ has missing values.
  - Solution: Pretend missing is just another value.
  - Solution: Ignore missing values in attr. selection. Split examples with missing values across branches.

- Attribute $A$ has many discrete values.
  - Solution: Find best $A = v$ test. Forms binary tree.
  - Solution: Partition attribute values into subsets.

Pruning Decision Trees

- Why are there errors?
  - Examples might have noise and/or outliers.
  - DT approximates decision boundary.

- Results in overfitting at lower levels of DT (fitting to noise, outliers, or statistical fluctuations)

- Pruning
  - Prepruning: Avoid creation of subtrees based on number of examples or attribute relevance.
  - Postpruning: Create overfitting DT and substitute subtrees with leaves if estimated error is reduced.
Estimating Error

- Use a “validation” set of examples. (training set, validation set, test set should be disjoint)
- Minimum Description Length principle (minimize size of tree and minimize size of errors)
- Add some error to each leaf (C4.5).
  - Suppose a leaf has $e$ errors on $n$ examples.
  - Find 75% confidence interval using binomial dist.
  - Estimate true error as upper limit of interval.

Algorithm for Pruning Decision Trees

**Prune_DT**($N$: node, examples)
1. $\text{leaferr} \leftarrow$ number of examples $\neq N$.class
2. revise $\text{leaferr}$ upward if examples were training set
3. if $N$ is a leaf then return $\text{leaferr}$
4. $\text{treeerr} \leftarrow 0$
5. for each value $v_j$ of $N$.test
6. $\text{examples}_j \leftarrow$ examples with $N$.test $= v_j$
7. $\text{suberr} \leftarrow \text{Prune_DT}(N.\text{branch}_j, \text{examples}_j)$
8. $\text{treeerr} \leftarrow \text{treeerr} + \text{suberr}$
9. if $\text{leaferr} < \text{treeerr}$
10. then make $N$ a leaf; return $\text{leaferr}$
11. else return $\text{treeerr}$
Ensemble Learning

There are many algorithms for learning a single hypothesis. *Ensemble learning* will learn and combine a collection of hypotheses by running the algorithm on different training sets.

*Bagging* (briefly mentioned in the book) runs a learning algorithm on repeated subsamples of the training set. If there are $n$ examples, then a subsample of $n$ examples is generated by sampling with replacement. On a test example, each hypothesis votes for a class.

---

Boosting

In *boosting*, each hypotheses is given a weight. When the hypothesis votes for a class, its weight is added for that class.

The hypotheses are learned in sequence. After each hypothesis is learned, its weight is based on its error rate, and the weights of the training examples (initially all equal) are also modified.
**AdaBoost** *(examples, algorithm, iterations)*
1. $n \leftarrow$ number of examples
2. initialize weights $w[1 \ldots n]$ to 1.0
3. **for** $i$ **from** 1 **to** iterations
4. $h[i] \leftarrow$ *algorithm*(examples)
5. $error \leftarrow$ (sum of exs. misclassified by $h[i]$) / $n$
6. **for** $j$ **from** 1 **to** $n$
7. **if** $h[i]$ is correct on example $j$
8. **then** $w[j] \leftarrow w[j] \times error/(1 - error)$
9. normalize $w[1 \ldots n]$ so it sums to $n$
10. weight of $h[i] \leftarrow \log((1 - error)/error)$
11. **return** $h[1 \ldots iterations]$ and their weights

Using the 14 examples as a training set:

The hypothesis windy = false $\leftrightarrow$ class = pos is correct for 9 of the 14 examples.
The weights of the correctly classified examples are changed from 1 to 5/9, then all examples are multiplied by 14/10 so they sum up to 14 again.
This hypothesis has a weight of log(9/5).
Now the hypothesis outlook = overcast ↔ class = pos is correct for 9.49 of the examples. The weights of the correctly classified examples are multiplied times 0.475, then all examples are multiplied by 1.55 so they sum up to 14 again. This hypothesis has a weight of log(9.49/4.51).

Note that after weight updating, the sum of the correctly classified examples equals the sum of the incorrectly classified examples. The next hypothesis must be different from the previous one to have error less than 1/2.

Example of a Decision List

1. **Outlook**: Overcast
   - T → pos
   - F

2. **Outlook**: Sunny
   - T → neg
   - F

3. **Outlook**: Sunny & Humidity=High
   - T → pos
   - F

4. **Windy**: True
   - T → neg
   - F → pos