1. Bayesian “Learning”

2. Decision Trees

(a) Strictly speaking, DT is not an ML algorithm. It’s a decision tool, for which we have algorithms
(b) Type is (typically) supervised, set based, classifier that uses both data and parameters.
(c) Training input: state vector, plus label.
(d) Testing input: state vector
(e) Testing output: training label
(f) Start with how the tree works in testing:
   i. A bunch of questions are arranged in a tree such as:
ii. When input comes in, start at root

iii. Ask each question, draw answer from input, move to next question based on answer
iv. Each path is logical statement, where each question is a predicate in a boolean formula:

\[ \text{Seller(Reliablie)} \land \text{Cheaper(close)} \land \text{Ram(sufficient)} \land \text{Waranty(yes)} \land \text{Type(laptop)} \land \text{Weight(light)} \rightarrow \text{Buy(yes)} \]

\[ \text{Seller(Reliablie)} \land \text{Cheaper(no)} \rightarrow \text{Buy(no)} \]

v. These trees are expressive (can represent any boolean statement), compact (typically smaller than truth table), and intuitive (people use them all the time).

(g) So how do we come up with the tree?

i. Start with data (duh), labels (yes or no, 0 or 1) and the set of possible predicates.

ii. Each example will have all its values filled as in this table:

<table>
<thead>
<tr>
<th>unique ID</th>
<th>1</th>
<th>2</th>
<th>3 ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>reliability</td>
<td>.8</td>
<td>.2</td>
<td>.9 ...</td>
</tr>
<tr>
<td>price</td>
<td>1000</td>
<td>200</td>
<td>800...</td>
</tr>
<tr>
<td>color</td>
<td>red</td>
<td>blue</td>
<td>green...</td>
</tr>
<tr>
<td>Speed</td>
<td>1GHz</td>
<td>100GHz</td>
<td>2GHz...</td>
</tr>
<tr>
<td>RAM</td>
<td>1G</td>
<td>5G</td>
<td>1G...</td>
</tr>
<tr>
<td>Waranty</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Desktop</td>
<td>yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Light</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Monitor</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>turbo button</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Buy</td>
<td>No</td>
<td>No</td>
<td>Yes...</td>
</tr>
</tbody>
</table>

(h) Trivial method: Build a path for each data item. No generalization to new data. Too large a tree.

(i) Concept: we want the minimal tree that still characterizes the data.

(j) This is done by putting the “best” predicate at the root. A perfect predicate divides the data set into just yes and just no.

(k) Most predicates are not perfect. But one idea is that better predicates are ones that best divide the data, that divide the data into groups that are the most pure, either yes or no.

(l) But how do we measure purity of the sets? There is a way, borrowed from information theory, called Entropy. This is measured as: \[ E(S) = \sum_{i=1}^{c} -p_i \lg p_i \], where there are \( c \) outputs of the tree, in this case just 2 yes and no, and \( p_i \) is the proportion of the set of the data that are labelled \( i \).

(m) Example:

<table>
<thead>
<tr>
<th>Day</th>
<th>Outlook</th>
<th>Temp</th>
<th>Humidity</th>
<th>Wind</th>
<th>HoldParade</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D2</td>
<td>Sunny</td>
<td>Hot</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D3</td>
<td>Overcast</td>
<td>Hot</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D4</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D5</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D6</td>
<td>Rain</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>No</td>
</tr>
<tr>
<td>D7</td>
<td>Overcast</td>
<td>Cool</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D8</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>No</td>
</tr>
<tr>
<td>D9</td>
<td>Sunny</td>
<td>Mild</td>
<td>High</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D10</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D11</td>
<td>Rain</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D12</td>
<td>Sunny</td>
<td>Mild</td>
<td>Normal</td>
<td>Strong</td>
<td>Yes</td>
</tr>
<tr>
<td>D13</td>
<td>Overcast</td>
<td>Hot</td>
<td>Normal</td>
<td>Weak</td>
<td>Yes</td>
</tr>
<tr>
<td>D14</td>
<td>Rain</td>
<td>Mild</td>
<td>High</td>
<td>Strong</td>
<td>No</td>
</tr>
</tbody>
</table>

(table due to Mitchell, 1997).
The entropy of this set, with \( \text{HoldParade} \) as the label, is:

\[
E(S) = \sum_{i=1}^{c} -p_i \log p_i
\]

\[
= -p_{\text{yes}} \log p_{\text{yes}} - p_{\text{no}} \log p_{\text{no}}
\]

\[
= -\frac{9}{14} \log \frac{9}{14} - \frac{5}{14} \log \frac{5}{14}
\]

\[
= 0.94
\]

(n) If the set is evenly divided, we get 1. If it’s pure, we get 0.

(o) Once we can measure the purity of a set, we can measure the gain of purity of splitting the set based on a predicate:

\[
G(S, P) = E(S) - \sum_{v \in V(P)} \frac{|S_v|}{|S|} E(S_v)
\]

This says, if we have a set \( S \) and we split it with \( P \), and \( P \) can take one of the values in the set \( V(P) \), then the information gain of splitting the set using predicate \( P \) is the entropy of \( S \), minus the weighted sum of the sets we split \( S \) into.

The gain of splitting the set on predicate \( \text{Humidity} \) is:

\[
G(S, \text{H}) = E(S) - \sum_{v \in V(\text{H})} \frac{|S_v|}{|S|} E(S_v)
\]

\[
= 0.94 - \frac{|S_{\text{high}}|}{|S|} E(S_{\text{high}}) + \frac{|S_{\text{normal}}|}{|S|} E(S_{\text{normal}})
\]

\[
E(S_{\text{high}}) = -p_{\text{yes}} \log p_{\text{yes}} - p_{\text{no}} \log p_{\text{no}}
\]

\[
= -\frac{3}{7} \log \frac{3}{7} - \frac{4}{7} \log \frac{4}{7}
\]

\[
= 0.98
\]

\[
E(S_{\text{normal}}) = -p_{\text{yes}} \log p_{\text{yes}} - p_{\text{no}} \log p_{\text{no}}
\]

\[
= -\frac{6}{7} \log \frac{6}{7} - \frac{1}{7} \log \frac{1}{7}
\]

\[
= 0.59
\]

\[
G(S, \text{H}) = 0.94 - \frac{3}{7} 0.98 + \frac{4}{7} 0.59
\]

\[
\approx 0.155
\]

(p) We select the predicate that gives us the highest information gain.

(q) And repeat for each of the split sets. If a set is ever pure, we stop on that branch.

(r) Some problems:

i. Very specific predicates, such as “date” split perfectly, but have no generalization power.

ii. Growing perfect trees is bad when there’s noise in the data.

iii. Continuous valued predicates require extra work.

iv. What do you do if you are missing a value in a datum?