Machine Learning (CSE 446):
Decision Trees

Sham M Kakade
© 2018

University of Washington
skakade@cs.washington.edu
Announcements

- First assignment posted. Due Thurs, Jan 18th. Remember the late policy (see the website).
- TA office hours posted.
  (Please check website before you go, just in case of changes.)
- Today: Decision Trees, the supervised learning
Let $\phi$ be (one such) function that maps from inputs $x$ to values. There could be many such functions, sometimes we write $\Phi(x)$ for the feature “vector” (it’s really a “tuple”).

- If $\phi$ maps to $\{0, 1\}$, we call it a “binary feature (function).”
- If $\phi$ maps to $\mathbb{R}$, we call it a “real-valued feature (function).”
- $\phi$ could map to categorical values.
- Ordinal values, integers, ...

Often, there isn’t much of a difference between $x$ and the tuple of features.
Features

Data derived from https://archive.ics.uci.edu/ml/datasets/Auto+MPG

mpg; cylinders; displacement; horsepower; weight; acceleration; year; origin

<table>
<thead>
<tr>
<th>mpg</th>
<th>cylinders</th>
<th>displacement</th>
<th>horsepower</th>
<th>weight</th>
<th>acceleration</th>
<th>year</th>
<th>origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.0</td>
<td>8</td>
<td>307.0</td>
<td>130.0</td>
<td>3504.</td>
<td>12.0</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>15.0</td>
<td>8</td>
<td>350.0</td>
<td>165.0</td>
<td>3693.</td>
<td>11.5</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>18.0</td>
<td>8</td>
<td>318.0</td>
<td>150.0</td>
<td>3436.</td>
<td>11.0</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>16.0</td>
<td>8</td>
<td>304.0</td>
<td>150.0</td>
<td>3433.</td>
<td>12.0</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>17.0</td>
<td>8</td>
<td>302.0</td>
<td>140.0</td>
<td>3449.</td>
<td>10.5</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>15.0</td>
<td>8</td>
<td>429.0</td>
<td>198.0</td>
<td>4341.</td>
<td>10.0</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>14.0</td>
<td>8</td>
<td>454.0</td>
<td>220.0</td>
<td>4354.</td>
<td>9.0</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>14.0</td>
<td>8</td>
<td>440.0</td>
<td>215.0</td>
<td>4312.</td>
<td>8.5</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>14.0</td>
<td>8</td>
<td>455.0</td>
<td>225.0</td>
<td>4425.</td>
<td>10.0</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>15.0</td>
<td>8</td>
<td>390.0</td>
<td>190.0</td>
<td>3850.</td>
<td>8.5</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>15.0</td>
<td>8</td>
<td>383.0</td>
<td>170.0</td>
<td>3563.</td>
<td>10.0</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>14.0</td>
<td>8</td>
<td>340.0</td>
<td>160.0</td>
<td>3609.</td>
<td>8.0</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>15.0</td>
<td>8</td>
<td>400.0</td>
<td>150.0</td>
<td>3761.</td>
<td>9.5</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>14.0</td>
<td>8</td>
<td>455.0</td>
<td>225.0</td>
<td>3086.</td>
<td>10.0</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>24.0</td>
<td>4</td>
<td>113.0</td>
<td>95.00</td>
<td>2372.</td>
<td>15.0</td>
<td>70</td>
<td>3</td>
</tr>
<tr>
<td>22.0</td>
<td>6</td>
<td>198.0</td>
<td>95.00</td>
<td>2833.</td>
<td>15.5</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>18.0</td>
<td>6</td>
<td>199.0</td>
<td>97.00</td>
<td>2774.</td>
<td>15.5</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>21.0</td>
<td>6</td>
<td>200.0</td>
<td>85.00</td>
<td>2587.</td>
<td>16.0</td>
<td>70</td>
<td>1</td>
</tr>
<tr>
<td>27.0</td>
<td>4</td>
<td>97.00</td>
<td>88.00</td>
<td>2130.</td>
<td>14.5</td>
<td>70</td>
<td>3</td>
</tr>
<tr>
<td>26.0</td>
<td>4</td>
<td>97.00</td>
<td>46.00</td>
<td>1835.</td>
<td>20.5</td>
<td>70</td>
<td>2</td>
</tr>
<tr>
<td>25.0</td>
<td>4</td>
<td>110.0</td>
<td>87.00</td>
<td>2672.</td>
<td>17.5</td>
<td>70</td>
<td>2</td>
</tr>
<tr>
<td>24.0</td>
<td>4</td>
<td>107.0</td>
<td>90.00</td>
<td>2430.</td>
<td>14.5</td>
<td>70</td>
<td>2</td>
</tr>
</tbody>
</table>

Input: a row in this table. a feature mapping corresponds to a column.

Goal: predict whether mpg is \(< 23\) ("bad" = 0) or above ("good" = 1) given other attributes (other columns).

201 "good" and 197 "bad"; guessing the most frequent class (good) will get 50.5% accuracy.
Let's build a classifier!

- Let's just try to build a classifier.
  (This is our first learning algorithm)
- For now, let’s ignore the “test” set and trying to “generalize”
- Let’s start with just looking at a simple classifier.
  What is a simple classification rule?
Contingency Table

<table>
<thead>
<tr>
<th>values of $y$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

values of feature $\phi$

$v_1$  $v_2$  $\cdots$  $v_K$

|   |   |   |
### Decision Stump Example

<table>
<thead>
<tr>
<th>$y$</th>
<th>maker</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>america</td>
<td>174</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>europe</td>
<td>75</td>
<td>56</td>
</tr>
</tbody>
</table>

↓ ↓ ↓

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
</table>

$\frac{7}{18}$
Decision Stump Example

\[
\begin{array}{c|ccc}
 y & \text{america} & \text{europe} & \text{asia} \\
0 & 174 & 14 & 9 \\
1 & 75 & 56 & 70 \\
\end{array}
\]

root

\[
\begin{array}{c|ccc}
0 & \text{america} & \text{europe} & \text{asia} \\
\end{array}
\]

\[
\begin{array}{c|ccc}
197:201 & \text{america} & \text{europe} & \text{asia} \\
\end{array}
\]

\[
\begin{array}{c|ccc}
\text{america} & 174:75 & \text{europe} & 14:56 & \text{asia} & 9:70 \\
\end{array}
\]

\[
\begin{array}{c|ccc}
\text{maker?} & 0 & 1 & 1 \\
\end{array}
\]
Decision Stump Example

<table>
<thead>
<tr>
<th>( y )</th>
<th>maker</th>
<th>america</th>
<th>europe</th>
<th>asia</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>174</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>75</td>
<td>56</td>
<td>70</td>
</tr>
</tbody>
</table>

\[ \text{Errors: } 75 + 14 + 9 = 98 \quad \text{(about 25\%)} \]
Decision Stump Example
Decision Stump Example

Errors: $1 + 20 + 1 + 11 + 3 = 36$ (about 9%)
Key Idea: Recursion

A single feature **partitions** the data.

For each partition, we could choose another feature and partition further.

Applying this recursively, we can construct a **decision tree**.
Decision Tree Example

Error reduction compared to the cylinders stump?
Decision Tree Example

Error reduction compared to the cylinders stump?
Decision Tree Example

Error reduction compared to the cylinders stump?
Decision Tree Example

Error reduction compared to the cylinders stump?
Decision Tree: Making a Prediction

root

\[ n: p \]

\[ \phi_1? \]

\[ \begin{array}{c}
0 \\
\text{n}_0: p_0 \\
\end{array} \]

\[ \begin{array}{c}
1 \\
\text{n}_1: p_1 \\
\end{array} \]

\[ \phi_2? \]

\[ \begin{array}{c}
0 \\
\text{n}_{10}: p_{10} \\
\end{array} \]

\[ \begin{array}{c}
1 \\
\text{n}_{11}: p_{11} \\
\end{array} \]

\[ \phi_3? \]

\[ \begin{array}{c}
0 \\
\text{n}_{100}: p_{100} \\
\end{array} \]

\[ \begin{array}{c}
1 \\
\text{n}_{101}: p_{101} \\
\end{array} \]

\[ \phi_4? \]

\[ \begin{array}{c}
0 \\
\text{n}_{110}: p_{110} \\
\end{array} \]

\[ \begin{array}{c}
1 \\
\text{n}_{111}: p_{111} \\
\end{array} \]
Data: decision tree $t$, input example $x$
Result: predicted class

if $t$ has the form $\text{LEAF}(y)$ then
  return $y$;
else
  # $t.\phi$ is the feature associated with $t$;
  # $t.$child($v$) is the subtree for value $v$;
  return $\text{DTreeTest}(t.$child($t.\phi(x)), x))$;
end

Algorithm 1: $\text{DTreeTest}$
Decision Tree: Making a Prediction

Equivalent boolean formulas:

\[(\phi_1 = 0) \Rightarrow [n_0 < p_0]\]

\[(\phi_1 = 1) \land (\phi_2 = 0) \land (\phi_3 = 0) \Rightarrow [n_{100} < p_{100}]\]

\[(\phi_1 = 1) \land (\phi_2 = 0) \land (\phi_3 = 1) \Rightarrow [n_{101} < p_{101}]\]

\[(\phi_1 = 1) \land (\phi_2 = 1) \land (\phi_4 = 0) \Rightarrow [n_{110} < p_{110}]\]

\[(\phi_1 = 1) \land (\phi_2 = 1) \land (\phi_4 = 1) \Rightarrow [n_{111} < p_{111}]\]
Assume we have $D$ binary features.
Each feature could be set to 0, or set to 1, or excluded (wildcard/don’t care).
$3^D$ formulas.
Building a Decision Tree
Building a Decision Tree

We chose feature $\phi_1$. Note that $n = n_0 + n_1$ and $p = p_0 + p_1$. 
We chose not to split the left partition. Why not?
Building a Decision Tree

root
n:p

ϕ₁?

0
n₀:p₀

1
n₁:p₁

ϕ₂?

0
n₁₀:p₁₀

1
n₁₁:p₁₁
Building a Decision Tree

```
root
\n\nϕ₁?
\n0
n₀:p₀
1
n₁:p₁
ϕ₂?
\n0
n₁₀:p₁₀
1
n₁₁:p₁₁
ϕ₃?
\n0
n₁₀₀:p₁₀₀
1
n₁₀₁:p₁₀₁
```
Building a Decision Tree
Greedily Building a Decision Tree (Binary Features)

**Data:** data $D$, feature set $\Phi$

**Result:** decision tree

if all examples in $D$ have the same label $y$, or $\Phi$ is empty and $y$ is the best guess then

return Leaf($y$);

else

for each feature $\phi$ in $\Phi$ do

partition $D$ into $D_0$ and $D_1$ based on $\phi$-values;

let mistakes($\phi$) = (non-majority answers in $D_0$) + (non-majority answers in $D_1$);

end

let $\phi^*$ be the feature with the smallest number of mistakes;

return Node($\phi^*$, $\{0 \rightarrow \text{DTreeTrain}(D_0, \Phi \setminus \{\phi^*\}), 1 \rightarrow \text{DTreeTrain}(D_1, \Phi \setminus \{\phi^*\})\}$);

end

**Algorithm 2:** DTreeTrain
What could go wrong?

- Suppose we split on a variable with many values? (e.g. a continuous one like “displacement”)
- Suppose we built out our tree to be very deep and wide?
Danger: Overfitting

- Error rate (lower is better)
- Depth of the decision tree
- Training data
- Unseen data
- Overfitting

Graph showing the relationship between error rate and depth of the decision tree for training and unseen data.
Detecting Overfitting

If you use all of your data to train, you won’t be able to draw the red curve on the preceding slide!
Detecting Overfitting

If you use all of your data to train, you won’t be able to draw the red curve on the preceding slide!

Solution: hold some out. This data is called **development data**. More terms:
- Decision tree max depth is an example of a **hyperparameter**
- “I used my development data to **tune** the max-depth hyperparameter.”
Detecting Overfitting

If you use all of your data to train, you won’t be able to draw the red curve on the preceding slide!

Solution: hold some out. This data is called **development data**. More terms:

- Decision tree max depth is an example of a **hyperparameter**
- “I used my development data to **tune** the max-depth hyperparameter.”

Better yet, hold out two subsets, one for tuning and one for a true, honest-to-science **test**.

Splitting your data into training/development/test requires careful thinking. Starting point: randomly shuffle examples with an 80%/10%/10% split.
The “i.i.d.” Supervised Learning Setup

- Let $\ell$ be a loss function; $\ell(y, \hat{y})$ is what we lose by outputting $\hat{y}$ when $y$ is the correct output. For classification:

$$\ell(y, \hat{y}) = [y \neq \hat{y}]$$

- Let $D(x, y)$ define the true probability of input/output pair $(x, y)$, in “nature.” We never “know” this distribution.

- The training data $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ are assumed to be identical, independently, distributed (i.i.d.) samples from $D$.

- The test data are also assumed to be i.i.d. samples from $D$.

- The space of classifiers we’re considering is $\mathcal{F}$; $f$ is a classifier from $\mathcal{F}$, chosen by our learning algorithm.