Goals for the last lecture

you should understand the following concepts

• the decision tree representation
• the standard top-down approach to learning a tree
• Occam’s razor
• entropy and information gain
Goals for this lecture

you should understand the following concepts

• test sets and unbiased estimates of accuracy
• overfitting
• early stopping and pruning
• validation sets
• regression trees
• probability estimation trees
Stopping criteria

We should form a leaf when

- all of the given subset of instances are of the same class
- we’ve exhausted all of the candidate splits

Is there a reason to stop earlier, or to prune back the tree?
How to assess the accuracy of a tree?

• can we just calculate the fraction of training instances that are correctly classified?

• consider a problem domain in which instances are assigned labels at random with $P(Y = t) = 0.5$
  
  • how accurate would a learned decision tree be on previously unseen instances?

  • how accurate would it be on its training set?
How can we assess the accuracy of a tree?

- to get an unbiased estimate of a learned model’s accuracy, we must use a set of instances that are held-aside during learning
- this is called a test set
Overfitting
Overfitting

• consider error of model \( h \) over
  • training data: \( \text{error}_D(h) \)
  • entire distribution of data: \( \text{error}_P(h) \)

• model \( h \in H \) overfits the training data if it has a low error on the training data but a high error on the entire distribution.
Example 1: overfitting with noisy data

suppose

• the target concept is $Y = X_1 \land X_2$
• there is noise in some feature values
• we’re given the following training set

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
<th>$X_5$</th>
<th>...</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>...</td>
<td>t</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>f</td>
<td>f</td>
<td>t</td>
<td>...</td>
<td>t</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>...</td>
<td>t</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>f</td>
<td>t</td>
<td>f</td>
<td>...</td>
<td>f</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>f</td>
<td>t</td>
<td>f</td>
<td>...</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>t</td>
<td>f</td>
<td>t</td>
<td>...</td>
<td>f</td>
</tr>
</tbody>
</table>

noisy value
Example 1: overfitting with noisy data

correct tree

```
X_1
  \_ T \_ F
  \_ X_2 \_ f
  \_ t \_ f
```

tree that fits noisy training data

```
X_1
  \_ T \_ F
  \_ X_2 \_ f
  \_ t \_ f
  \_ X_3 \_ f
  \_ t \_ f
```
Example 2: overfitting with noise-free data

suppose

• the target concept is $Y = X_1 \land X_2$
• $P(X_3 = t) = 0.5$ for both classes
• $P(Y = t) = 0.67$
• we’re given the following training set

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
<th>$X_5$</th>
<th>...</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
<td>f</td>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>f</td>
<td>t</td>
<td>f</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>f</td>
<td>f</td>
<td>t</td>
<td>t</td>
<td>f</td>
</tr>
</tbody>
</table>
Example 2: overfitting with noise-free data

- because the training set is a limited sample, there might be (combinations of) features that are correlated with the target concept by chance
Example 3: regression using polynomial

\[ t = \sin(2\pi x) + \epsilon \]

Figure from *Machine Learning and Pattern Recognition*, Bishop
Example 3: regression using polynomial

\[ t = \sin(2\pi x) + \epsilon \]

Regression using polynomial of degree \( M \)
Example 3: regression using polynomial

\[ t = \sin(2\pi x) + \epsilon \]
Example 3: regression using polynomial

\[ t = \sin(2\pi x) + \epsilon \]
Example 3: regression using polynomial

\[ t = \sin(2\pi x) + \epsilon \]
Example 3: regression using polynomial
General phenomenon

Figure from *Deep Learning*, Goodfellow, Bengio and Courville
Prevent overfitting

• cause: training error and expected error are different
  1. there may be noise in the training data
  2. training data is of limited size, resulting in difference from the true distribution
  3. larger the hypothesis class, easier to find a hypothesis that fits the difference between the training data and the true distribution

• prevent overfitting:
  1. cleaner training data help!
  2. more training data help!
  3. throwing away unnecessary hypotheses helps! (Occam’s Razor)
Avoiding overfitting in DT learning

two general strategies to avoid overfitting

1. *early stopping*: stop if further splitting not justified by a statistical test
   • Quinlan’s original approach in ID3

2. *post-pruning*: grow a large tree, then prune back some nodes
   • more robust to myopia of greedy tree learning
Pruning in C4.5

1. split given data into training and validation (tuning) sets
2. grow a complete tree
3. do until further pruning is harmful
   • evaluate impact on tuning-set accuracy of pruning each node
   • greedily remove the one that most improves tuning-set accuracy
Validation sets

• a validation set (a.k.a. tuning set) is a subset of the training set that is held aside
  • not used for primary training process (e.g. tree growing)
  • but used to select among models (e.g. trees pruned to varying degrees)
Probability estimation trees

- in a PE tree, leaves estimate the probability of each class
- could simply use training instances at a leaf to estimate probabilities, but …
- *smoothing* is used to make estimates less extreme (we’ll revisit this topic when we cover Bayes nets)

\[
\begin{align*}
X_5 > 10 \\
X_3 \\
\text{D: [3+, 3-]} & \quad \text{D: [0+, 8-]} \\
P(Y=\text{pos}) = 0.5 & \quad P(Y=\text{pos}) = 0.8 \\
P(Y=\text{neg}) = 0.5 & \quad P(Y=\text{neg}) = 0.2 \\
P(Y=\text{pos}) = 0.5 & \quad P(Y=\text{pos}) = 0.1 \\
P(Y=\text{neg}) = 0.5 & \quad P(Y=\text{neg}) = 0.9
\end{align*}
\]
Regression trees

- In a regression tree, leaves have functions that predict numeric values instead of class labels.
- The form of these functions depends on the method:
  - CART uses constants.
  - Some methods use linear functions.

\[
\begin{align*}
X_5 & > 10 \\
X_3 & \\
X_2 & > 2.1 \\
Y & = 5 \\
Y & = 3.5 \\
Y & = 24 \\
Y & = 3.2 \\
Y & = 2X_4 + 5 \\
X_2 & > 2.1 \\
Y & = 1 \\
Y & = 3X_4 + X_6 \\
Y & = 3.2
\end{align*}
\]
Regression trees in CART

- CART does least squares regression which tries to minimize

\[
\sum_{i=1}^{|D|} (y^{(i)} - \hat{y}^{(i)})^2
\]

- at each internal node, CART chooses the split that most reduces this quantity.
\textit{m-of-n} splits

- a few DT algorithms have used \textit{m-of-n} splits [Murphy & Pazzani ‘91]
- each split is constructed using a heuristic search process
- this can result in smaller, easier to comprehend trees

\begin{itemize}
  \item\hspace{1cm} 5 of \{ROI[13] > 1.07, vs_Yen_future < 0.39, interest_rate[1] = false, vs_Dem_future[2] > 0.17, vs_Dem_future[3] < 0.17, vs_SFr_future < 0.39, Dem_USD_ex[8] < -1.02, Dem_future[5] > -1.38, Dem_USD_ex[3] < -0.10, vs_SFr_future < 0.16\} test is satisfied if 5 of 10 conditions are true
\end{itemize}

tree for exchange rate prediction
[Craven & Shavlik, 1997]
Searching for $m$-of-$n$ splits

$m$-of-$n$ splits are found via a hill-climbing search

- initial state: best 1-of-1 (ordinary) binary split
- evaluation function: information gain
- operators:

\[
\begin{align*}
\text{$m$-of-$n$} & \quad \Rightarrow \quad \text{$m$-of-($n+1$)} \\
1 \text{ of } \{ X_1=\text{t}, X_3=\text{f} \} & \quad \Rightarrow \quad 1 \text{ of } \{ X_1=\text{t}, X_3=\text{f}, X_7=\text{t} \} \\
\text{$m$-of-$n$} & \quad \Rightarrow \quad \text{($m+1$)-of-($n+1$)} \\
1 \text{ of } \{ X_1=\text{t}, X_3=\text{f} \} & \quad \Rightarrow \quad 2 \text{ of } \{ X_1=\text{t}, X_3=\text{f}, X_7=\text{t} \}
\end{align*}
\]
Lookahead

- most DT learning methods use a hill-climbing search
- a limitation of this approach is myopia: an important feature may not appear to be informative until used in conjunction with other features
- can potentially alleviate this limitation by using a lookahead search [Norton ‘89; Murphy & Salzberg ‘95]
- empirically, often doesn’t improve accuracy or tree size
Choosing best split in ordinary DT learning

OrdinaryFindBestSplit(set of training instances \( D \), set of candidate splits \( C \))

\[
\text{maxgain} = -\infty
\]

for each split \( S \) in \( C \)

\[
\text{gain} = \text{InfoGain}(D, S)
\]

if \( \text{gain} > \text{maxgain} \)

\[
\text{maxgain} = \text{gain}
\]

\[
S_{\text{best}} = S
\]

return \( S_{\text{best}} \)
Choosing best split with lookahead (part 1)

LookaheadFindBestSplit(set of training instances $D$, set of candidate splits $C$)

$$\text{maxgain} = -\infty$$

for each split $S$ in $C$

$$\text{gain} = \text{EvaluateSplit}(D, C, S)$$

if $\text{gain} > \text{maxgain}$

$$\text{maxgain} = \text{gain}$$

$$S_{\text{best}} = S$$

return $S_{\text{best}}$
Choosing best split with lookahead (part 2)

\[
\text{EvaluateSplit}(D, C, S)
\]

\[
\begin{align*}
\text{if a split on } S & \text{ separates instances by class (i.e. } H_D(Y \mid S) = 0 \text{)} \\
& \text{// no need to split further} \\
& \text{return } H_D(Y) - H_D(Y \mid S)
\end{align*}
\]

\[
\text{else} \\
\text{for each outcome } k \text{ of } S \\
& \text{// see what the splits at the next level would be} \\
& D_k = \text{subset of instances that have outcome } k \\
& S_k = \text{OrdinaryFindBestSplit}(D_k, C - S) \\
& \text{// return information gain that would result from this 2-level subtree} \\
& \text{return } \\
& H_D(Y) - \left( \sum_k \frac{|D_k|}{|D|} H_{D_k}(Y \mid S = k, S_k) \right)
\]
Calculating information gain with lookahead

Suppose that when considering Humidity as a split, we find that Wind and Temperature are the best features to split on at the next level.

We can assess value of choosing Humidity as our split by

\[
H_D(Y) - \left( \frac{14}{23} H_D(Y \mid \text{Humidity} = \text{high, Wind}) + \frac{9}{23} H_D(Y \mid \text{Humidity} = \text{low, Temperature}) \right)
\]
Calculating information gain with lookahead

Using the tree from the previous slide:

\[
\frac{14}{23} H_D(Y \mid \text{Humidity} = \text{high}, \text{Wind}) + \frac{9}{23} H_D(Y \mid \text{Humidity} = \text{low}, \text{Temperature})
\]

\[
= \frac{5}{23} H_D(Y \mid \text{Humidity} = \text{high}, \text{Wind} = \text{strong}) + \frac{9}{23} H_D(Y \mid \text{Humidity} = \text{high}, \text{Wind} = \text{weak}) + \frac{4}{23} H_D(Y \mid \text{Humidity} = \text{low}, \text{Temperature} = \text{high}) + \frac{5}{23} H_D(Y \mid \text{Humidity} = \text{low}, \text{Temperature} = \text{low})
\]

\[
H_D(Y \mid \text{Humidity} = \text{high}, \text{Wind} = \text{strong}) = - \frac{2}{5} \log \left( \frac{2}{5} \right) - \frac{3}{5} \log \left( \frac{3}{5} \right)
\]

\[\vdots\]
Comments on decision tree learning

- widely used approach
- many variations
- provides humanly comprehensible models when trees not too big
- insensitive to monotone transformations of numeric features
- standard methods learn axis-parallel hypotheses*
- standard methods not suited to on-line setting*
- usually not among most accurate learning methods

* although variants exist that are exceptions to this
THANK YOU

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, and Pedro Domingos.