ELEN 4903: Machine Learning
Lecture 12, 3/1/2016

Prof. John Paisley

Columbia University
DECISION TREES
A decision tree maps input $x \in \mathbb{R}^d$ to output $y$ using binary decision rules:

- Each node in the tree has a splitting rule.
- Each leaf node is associated with an output value (outputs can repeat).

Each splitting rule is of the form

$$h(x) = 1\{x(j) > t\}$$

for some dimension $j$ of $x$ and $t \in \mathbb{R}$.

Using these transition rules, a path to a leaf node gives the prediction.

(One-level tree = decision stump)
**Motivation**: Partition the space so that data in a region have same prediction

- **Left**: Difficult to define a “rule”.
- **Right**: Easy to define a recursive splitting rule.
Regression Trees

If we think in terms of trees, we can define a simple rule for partitioning the space. The left and right figures represent the same regression function.
Adding an output dimension to the figure (right), we can see how regression trees can learn a step function approximation to the data.
Classifying irises using sepal and petal measurements:

- $x \in \mathbb{R}^2$, $y \in \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width
Classifying irises using sepal and petal measurements:

- $x \in \mathbb{R}^2$, $y \in \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width
Classifying irises using sepal and petal measurements:

- $x \in \mathbb{R}^2$, $y \in \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width

Graph:

$\ x_1 > 1.7$
Classifying irises using sepal and petal measurements:

- \( x \in \mathbb{R}^2 \), \( y \in \{1, 2, 3\} \)
- \( x_1 = \text{ratio of sepal length to width} \)
- \( x_2 = \text{ratio of petal length to width} \)

Classifying irises using sepal and petal measurements:

- \( x_1 > 1.7 \)
- \( \hat{y} = 1 \)
- \( \hat{y} = 3 \)
Classification Trees (Example)

Classifying irises using sepal and petal measurements:

- $x \in \mathbb{R}^2$, $y \in \{1, 2, 3\}$
- $x_1 = \text{ratio of sepal length to width}$
- $x_2 = \text{ratio of petal length to width}$

Diagram:
- $x_1 > 1.7$
- $\hat{y} = 1$
- $x_2 > 2.8$
Classifying irises using sepal and petal measurements:

- $x \in \mathbb{R}^2$, $y \in \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width

$\hat{y} = 1$
$x_1 > 1.7$
$x_2 > 2.8$

$\hat{y} = 2$
$\hat{y} = 3$
The basic method for learning trees is with a top-down greedy algorithm.

- Start with a single leaf node containing all data
- Loop through the following steps:
  - Pick a leaf to split that reduces uncertainty the most.
  - Figure out the $\leq$ decision rule on one of the dimensions.
  - Stopping rule discussed later.

Label/response of the leaf is majority-vote/average of data assigned to it.
**Growing a Regression Tree**

How do we grow a regression tree?

- For $M$ regions of the space, $R_1, \ldots, R_M$, the prediction function is
  \[
  f(x) = \sum_{m=1}^{M} c_m \mathbb{1}\{x \in R_m\}.
  \]

So for a fixed $M$, we need $R_m$ and $c_m$.

**Goal:** Try to minimize $\sum_i (y_i - f(x_i))^2$.

1. Find $c_m$ given $R_m$: Simply the average the $y_i$ of all $x_i \in R_m$.
2. How do we find regions? Consider splitting region $R$ at value $s$ of dim $j$:
   - Define $R^-(j, s) = \{x_i \in R | x_i(j) \leq s\}$ and $R^+(j, s) = \{x_i \in R | x_i(j) > s\}$
   - Pick the $j$ and $s$ using simple search over $j$ and calculating the best $s$:
     \[
     \min_{j, s} \left( \min_{c_1} \sum_{x_i \in R^- (j, s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R^+ (j, s)} (y_i - c_2)^2 \right)
     \]
GROWING A CLASSIFICATION TREE

For regression: Squared error is a natural way to define the splitting rule.

For classification: Need some measure of how badly a region classifies data and how much it can improve if it’s split.

K-class problem: For all $x \in R_m$, let $p_k$ be empirical fraction labeled $k$.

Measures of quality of $R_m$ include

1. Classification error: $1 - \max_k p_k$
2. Gini index: $1 - \sum_k p_k^2$
3. Entropy: $-\sum_k p_k \ln p_k$

- These are all maximized when $p_k$ is uniform on the $K$ classes in $R_m$.
- These are minimized when $p_k = 1$ for some $k$ ($R_m$ only contains one class)
Growing a Classification Tree

Search $R_1$ and $R_2$ for splitting options.

1. $R_1$: $y = 1$ leaf classifies perfectly
2. $R_2$: $y = 3$ leaf has Gini index

$$u(R_2) = 1 - \left( \frac{1}{101} \right)^2 - \left( \frac{50}{101} \right)^2 - \left( \frac{50}{101} \right)^2$$

$$= 0.5098$$

Gini improvement from split $R_m$ to $R_m^- \ & R_m^+$:

$$u(R_m) - \left( p_{R_m^-} \cdot u(R_m^-) + p_{R_m^+} \cdot u(R_m^+) \right)$$

$p_{R_m^+}$: Fraction of data in $R_m$ split into $R_m^+$.

$u(R_m^+)$: New quality measure in region $R_m^+$. 
GROWING A CLASSIFICATION TREE

Search $R_1$ and $R_2$ for splitting options.

1. $R_1$: $y = 1$ leaf classifies perfectly
2. $R_2$: $y = 3$ leaf has Gini index

$$u(R_2) = 1 - \left( \frac{1}{101} \right)^2 - \left( \frac{50}{101} \right)^2 - \left( \frac{50}{101} \right)^2$$

$$= 0.5098$$

Check split $R_2$ with $\mathbb{1}\{x_1 > t\}$
GROWING A CLASSIFICATION TREE

Search $R_1$ and $R_2$ for splitting options.

1. $R_1$: $y = 1$ leaf classifies perfectly
2. $R_2$: $y = 3$ leaf has Gini index

$$u(R_2) = 1 - \left( \frac{1}{101} \right)^2 - \left( \frac{50}{101} \right)^2 - \left( \frac{50}{101} \right)^2$$

$$= 0.5098$$

Check split $R_2$ with $\mathbb{1}\{x_2 > t\}$
Growing a Classification Tree

Search $R_1$ and $R_2$ for splitting options.

1. $R_1$: $y = 1$ leaf classifies perfectly
2. $R_2$: $y = 3$ leaf has Gini index

$$u(R_2) = 1 - \left( \frac{1}{101} \right)^2 - \left( \frac{50}{101} \right)^2 - \left( \frac{50}{101} \right)^2$$

$$= 0.5098$$

Check split $R_2$ with $\mathbb{1}\{x_2 > t\}$
**Pruning a Tree**

**Q:** When should we stop growing a tree?

**A:** Uncertainty reduction is not best way.

**Example:** Any split of $x_1$ or $x_2$ at right will show zero reduction in uncertainty.

However, we can learn a perfect tree on this data by partitioning in quadrants.

*Pruning* is the method most often used: Grow the tree to a very large size. Then use an algorithm to trim it back.

(We won’t cover the algorithm, but it’s non-trivial.)
**Example: Spam Filtering**

**Data**

- 4601 email messages, classes = \{email, spam\}, 39.4% are spam

<table>
<thead>
<tr>
<th></th>
<th>george</th>
<th>you</th>
<th>your</th>
<th>hp</th>
<th>free</th>
<th>hpl</th>
<th>!</th>
<th>our</th>
<th>re</th>
<th>edu</th>
<th>remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>spam</td>
<td>0.00</td>
<td>2.26</td>
<td>1.38</td>
<td>0.02</td>
<td>0.52</td>
<td>0.01</td>
<td>0.51</td>
<td>0.13</td>
<td>0.01</td>
<td>0.28</td>
<td></td>
</tr>
<tr>
<td>email</td>
<td>1.27</td>
<td>1.27</td>
<td>0.44</td>
<td>0.90</td>
<td>0.07</td>
<td>0.43</td>
<td>0.11</td>
<td>0.18</td>
<td>0.42</td>
<td>0.29</td>
<td>0.01</td>
</tr>
</tbody>
</table>

**Tree classifier with 17 leaves**

<table>
<thead>
<tr>
<th></th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>email spam</td>
</tr>
<tr>
<td>email</td>
<td>57.3% 4.0%</td>
</tr>
<tr>
<td>spam</td>
<td>5.3% 33.4%</td>
</tr>
</tbody>
</table>
FIGURE 9.5. The pruned tree for the spam example. The split variables are shown in blue on the branches, and the classification is shown in every node. The numbers under the terminal nodes indicate misclassification rates on the test data.
Training error goes to zero as size of tree increases.

Testing error decreases, but then increases because of overfitting.
The Bootstrap
THE BOOTSTRAP: A RESAMPLING TECHNIQUE

We briefly present a technique called the bootstrap. This statistical technique is used as the basis for ensemble classifiers.

Bootstrap

Bootstrap (i.e., resampling) is a technique for improving estimators.

Resampling = Sampling from the empirical distribution of the data

Application to ensemble methods

- We will use resampling to generate many “mediocre” classifiers.
- We then discuss how “bagging” these classifiers improves performance.
- First, we cover the bootstrap in a simpler context.
**BOOTSTRAP: BASIC ALGORITHM**

**Input**

- A sample of data $x_1, \ldots, x_n$.
- An estimate $\hat{S}$ rule of a statistic $S$
  
  For example, $\hat{S} = \text{med}(x_{1:n})$ estimates true median $S$ (infinite data case)

**Bootstrap algorithm**

1. **Generate bootstrap samples $B_1, \ldots, B_B$.**
   - Create $B_b$ by picking an $x \in \{x_1, \ldots, x_n\}$ at random $n$ times.
   - A particular $x_i$ can appear in $B_b$ many times (it’s simply duplicated).

2. **Evaluate the estimator on each $B_b$ by pretending it’s the data set:**

   $$\hat{S}_b := \hat{S}(B_b)$$

3. **Estimate the mean and variance of $S$:**

   $$\mu_B = \frac{1}{B} \sum_{b=1}^{B} \hat{S}_b, \quad \sigma^2_B = \frac{1}{B} \sum_{b=1}^{B} (\hat{S}_b - \mu_B)^2$$
Example: Variance Estimation of the Median

- The median of \( x_1, \ldots, x_n \) (for \( x \in \mathbb{R} \)) is found by simply sorting them and taking the middle one (or average of the two middle ones).

- How confident can we be in our estimate \( \text{median}(x_1, \ldots, x_n) \)?
  - Find it’s variance.
  - But how? Answer: By bootstrapping the data.

1. Generate bootstrap data sets \( \mathcal{B}_1, \ldots, \mathcal{B}_B \).

2. Calculate (notice \( \hat{S}_{\text{mean}} \) is the mean of the median):

\[
\hat{S}_{\text{mean}} = \frac{1}{B} \sum_{b=1}^{B} \text{median}(\mathcal{B}_b), \quad \hat{S}_{\text{var}} = \frac{1}{B} \sum_{b=1}^{B} \left( \text{median}(\mathcal{B}_b) - \hat{S}_{\text{mean}} \right)^2
\]

- The procedure is remarkably simple, but has lots of theory motivating it.
Bagging and Random Forests
Bagging uses the bootstrap for regression or classification:

\[
\text{Bagging} = \text{Bootstrap aggregation}
\]

**Algorithm**

For \( b = 1, \ldots, B \):

1. Draw a bootstrap sample \( \mathcal{B}_b \) of size \( n \) from training data.
2. Train a classifier or regression model \( f_b \) on \( \mathcal{B}_b \).

▶ For a new point \( x_0 \), compute:

\[
f_{\text{avg}}(x_0) = \frac{1}{B} \sum_{b=1}^{B} f_b(x_0)
\]

▶ For regression, \( f_{\text{avg}}(x_0) \) is the prediction.

▶ For classification, view \( f_{\text{avg}}(x_0) \) as an average over \( B \) votes. Pick the majority.
Example: Bagging Trees

- Binary classification, $x \in \mathbb{R}^5$.
- Note the variation among bootstrapped trees.
- Take-home message:
  With bagging, each tree doesn’t have to be great, just “ok”.
- Bagging often improves results when the function is non-linear.
Random Forests

Drawbacks of Bagging

- Bagging works on trees because of the bias-variance tradeoff ($\uparrow$ bias, $\downarrow$ variance).
- However, the bagged trees are correlated.
- In general, when bootstrap samples are correlated, the benefit of bagging decreases.

Random Forests

Modification of bagging with trees designed to further reduce correlation.

- A very simple modification.
- Still learn a tree on each bootstrap set, $B_b$.
- To split a region, only consider random subset of dimensions of $x \in \mathbb{R}^d$. 
Random Forests: Algorithm

Training

Input parameter: \( m \) (positive integer with \( m < d \), often \( m \approx \sqrt{d} \))

For \( b = 1, \ldots, B \):

1. Draw a bootstrap sample \( B_b \) of size \( n \) from training data.
2. Train a tree classifier on \( B_b \), where each split is computed as follows:
   - Randomly select \( m \) dimensions of \( x \in \mathbb{R}^d \), newly sampled for each split.
   - Make the best split restricted to this subset of dimensions.

- Bagging for trees: Bag trees learned using the original algorithm.
- Random forests: Bag trees learned using algorithm on this slide.
Random Forests

Example problem

- Random forest classification.
- Forest size: A few hundred trees.
- Notice that there is a bias to align decision boundary with the axis.