A Decision Stump

mpg values: bad  good

root

22  18
p-chance = 0.001

cylinders = 3
0  0
Predict bad

cylinders = 4
4  17
Predict good

cylinders = 5
1  0
Predict bad

cylinders = 6
8  0
Predict bad

cylinders = 8
9  1
Predict bad

17 > 4
Basic Decision Tree Building Summarized

**BuildTree**(DataSet, Output)
- If all output values are the same in DataSet, return a leaf node that says "predict this unique output"
- If all input values are the same, return a leaf node that says "predict the majority output"
- Else find attribute X with highest Info Gain
- Suppose X has $n_X$ distinct values (i.e. X has arity $n_X$):
  - Create and return a non-leaf node with $n_X$ children.
  - The $i$th child should be built by calling
    ```
    BuildTree(DS_i, Output)
    ```
    Where $DS_i$ built consists of all those records in DataSet for which $X = i$th distinct value of $X$. 
The test set error is much worse than the training set error... why?
Decision trees & Learning Bias

- Many strategies for picking simpler trees:
  - Fixed depth
  - Fixed number of leaves
  - Or something smarter...

Decision trees will overfit

- Standard decision trees are have no learning biased
  - Training set error is always zero!
    - (If there is no label noise)
  - Lots of variance
  - Will definitely overfit!!!
  - Must bias towards simpler trees

©Carlos Guestrin 2005-2007
Consider this split

A chi-square test

- Suppose that mpg was completely uncorrelated with maker.
- What is the chance we’d have seen data of at least this apparent level of association anyway?
A chi-square test

Suppose that mpg was completely uncorrelated with maker.
What is the chance we'd have seen data of at least this apparent level of association anyway?
By using a particular kind of chi-square test, the answer is 7.2%.

(Such simple hypothesis tests are very easy to compute, unfortunately, not enough time to cover in the lecture, but in your homework, you'll have fun! :))

Using Chi-squared to avoid overfitting

Build the full decision tree as before
But when you can grow it no more, start to prune:
- Beginning at the bottom of the tree, delete splits in which $p_{\text{chance}} > \text{MaxPchance}$
- Continue working you way up until there are no more prunable nodes

$\text{MaxPchance}$ is a magic parameter you must specify to the decision tree, indicating your willingness to risk fitting noise.
Pruning example

- With MaxPchance = 0.1, you will see the following MPG decision tree:

Note the improved test set accuracy compared with the unpruned tree.

MaxPchance

- Technical note MaxPchance is a regularization parameter that helps us bias towards simpler models.

We’ll learn to choose the value of these magic parameters soon!
Real-Valued inputs

What should we do if some of the inputs are real-valued?

- Infinite number of possible split values!!!
- Finite dataset, only finite number of relevant splits!

Idea One: Branch on each possible real value

“One branch for each numeric value” idea:

Hopeless: with such high branching factor will shatter the dataset and overfit
Threshold splits

- Binary tree, split on attribute X
  - One branch: $X < t$
  - Other branch: $X \geq t$

Choosing threshold split

- Binary tree, split on attribute X
  - One branch: $X < t$
  - Other branch: $X \geq t$

- Search through possible values of $t$
  - Seems hard!!!

- But only finite number of $t$’s are important
  - Sort data according to X into \{x_1, \ldots, x_m\}
  - Consider split points of the form $x_i + (x_{i+1} - x_i)/2$
A better idea: thresholded splits

- Suppose X is real valued
- Define $IG(Y|X:t)$ as $H(Y) - H(Y|X:t)$
- Define $H(Y|X:t) = H(Y|X < t) P(X < t) + H(Y|X >= t) P(X >= t)$
  - $IG(Y|X:t)$ is the information gain for predicting Y if all you know is whether X is greater than or less than t
- Then define $IG^*(Y|X) = \max_t IG(Y|X:t)$
- For each real-valued attribute, use $IG^*(Y|X)$ for assessing its suitability as a split
  - then pick best $x_i = \arg \max IG^*(Y|X_i)$
- Note, may split on an attribute multiple times, with different thresholds

Example with MPG
Example tree using reals

What you need to know about decision trees

- Decision trees are one of the most popular data mining tools
  - Easy to understand
  - Easy to implement
  - Easy to use
  - Computationally cheap (to solve heuristically)
- Information gain to select attributes (ID3, C4.5, ...)
- Presented for classification, can be used for regression and density estimation too
- Decision trees will overfit!!!
  - Zero bias classifier → Lots of variance
  - Must use tricks to find "simple trees", e.g.,
    - Fixed depth/Early stopping
    - Pruning
    - Hypothesis testing
Acknowledgements

- Some of the material in the decision trees presentation is courtesy of Andrew Moore, from his excellent collection of ML tutorials:
  - [http://www.cs.cmu.edu/~awm/tutorials](http://www.cs.cmu.edu/~awm/tutorials)

Announcements

- Homework 1 due Wednesday beginning of class
  - started early, started early, started early, started early, started early, started early

- Exam dates set:
  - Midterm: Thursday, Oct. 25th, 5-6:30pm, MM A14
  - Final: Tuesday, Dec. 11, 05:30PM-08:30PM
Fighting the bias-variance tradeoff

- **Simple (a.k.a. weak) learners are good**
  - e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees)
  - Low variance, don’t usually overfit

- **Simple (a.k.a. weak) learners are bad**
  - High bias, can’t solve hard learning problems

- Can we make weak learners always good???
  - No!!
  - But often yes…

Voting (Ensemble Methods)

- Instead of learning a single (weak) classifier, learn many weak classifiers that are good at different parts of the input space

- Output class: (Weighted) vote of each classifier
  - Classifiers that are most “sure” will vote with more conviction
  - Classifiers will be most “sure” about a particular part of the space
  - On average, do better than single classifier!

\[
H(x) = \text{Sign} \left\{ \sum_{t=1}^{T} \text{weight} \cdot h_t(x) \right\} \\
Y \in \{-1, +1\}
\]

- But how do you ???
  - force classifiers to learn about different parts of the input space?
  - weigh the votes of different classifiers?
Boosting [Schapire, 1989]

- Idea: given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote.

- On each iteration $t$:
  - weight each training example by how incorrectly it was classified
  - Learn a hypothesis $h_t$
  - A strength for this hypothesis $\alpha_t$

- Final classifier:

$$\hat{h}(x) = \text{Sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

- Practically useful
- Theoretically interesting

Learning from weighted data

- Sometimes not all data points are equal
  - Some data points are more equal than others

- Consider a weighted dataset
  - $D(i)$ – weight of $i$th training example $(x^i, y^i)$
  - Interpretations:
    - $i$th training example counts as $D(i)$ examples
    - If I were to "resample" data, I would get more samples of "heavier" data points

- Now, in all calculations, whenever used, $i$th training example counts as $D(i)$ "examples"
  - e.g., MLE for Naïve Bayes, redefine $\text{Count}(Y=y)$ to be weighted count

$$\begin{align*}
\hat{\theta}_\text{prior} & \rightarrow \frac{\text{Count}(Y=y)}{m} \\
\text{weighted:} \quad \hat{\theta}_\text{weighted} & = \frac{\sum_{i=1}^{m} D(i) \cdot \text{Indicator}(y^i=y)}{\sum_{i=1}^{m} D(i)}
\end{align*}$$
Given: \((x_1, y_1), \ldots, (x_m, y_m)\) where \(x_i \in X, y_i \in Y = \{-1, +1\}\)
Initialize \(D_1(i) = 1/m\).
For \(t = 1, \ldots, T\):

- Train base learner using distribution \(D_t\).
- Get base classifier \(h_t : X \rightarrow \mathbb{R}\).
- Choose \(\alpha_t \in \mathbb{R}\).
- Update:
  \[
  D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}
  \]
  where \(Z_t\) is a normalization factor
  \[
  Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))
  \]
Output the final classifier:
  \[
  H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right).
  \]

Figure 1: The boosting algorithm AdaBoost.

Given: \((x_1, y_1), \ldots, (x_m, y_m)\) where \(x_i \in X, y_i \in Y = \{-1, +1\}\)
Initialize \(D_1(i) = 1/m\).
For \(t = 1, \ldots, T\):

- Train base learner using distribution \(D_t\).
- Get base classifier \(h_t : X \rightarrow \mathbb{R}\).
- Choose \(\alpha_t \in \mathbb{R}\).
- Update:
  \[
  D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}
  \]
  \[
  Z_t = \frac{1}{\sum_{i=1}^{m} D_t(i) \sum_{i=1}^{m} D_t(i) \delta(h_t(x_i) \neq y_i)}
  \]

\(\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)\)

\(\epsilon_t = \frac{1}{m} \sum_{i=1}^{m} D_t(i) \delta(h_t(x_i) \neq y_i)\)

\(\epsilon_t \rightarrow 0 \Rightarrow \alpha_t \rightarrow +\infty\)
\(\epsilon_t \rightarrow 1 \Rightarrow \alpha_t \rightarrow -\infty\)

as \(\epsilon_t \rightarrow 0\), \(\alpha_t \rightarrow +\infty\) - really trust opposite
as \(\epsilon_t \rightarrow 1\), \(\alpha_t \rightarrow -\infty\) - really trust opposite

if \(\epsilon_t = 0.5\), \(\alpha_t = 0\), random classifier wins
bad \(\Rightarrow 2^{\alpha_t}\)
What $\alpha_t$ to choose for hypothesis $h_t$?

[Schapire, 1989]

Training error of final classifier is bounded by:

$$\frac{1}{m} \sum_{i=1}^{m} \delta(H(x_i) \neq y_i) \leq \frac{1}{m} \sum_{i=1}^{m} \exp(-y_i f(x_i))$$

Where

$$f(x) = \sum_{t} \alpha_t h_t(x); H(x) = \text{sign}(f(x))$$

and

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

(some conditions apply)
Training error of final classifier is bounded by:

\[
\frac{1}{m} \sum_{i=1}^{m} \delta(H(x_i) \neq y_i) \leq \frac{1}{m} \sum_{i} \exp(-y_if(x_i)) = \prod_{t} Z_t
\]

Where \( f(x) = \sum_{t} \alpha_t h_t(x); H(x) = \text{sign}(f(x)) \)

If we minimize \( \prod_{t} Z_t \), we minimize our training error

We can tighten this bound greedily, by choosing \( \alpha_t \) and \( h_t \) on each iteration to minimize \( Z_t \):

\[
Z_t = \sum_{i=1}^{m} D_t(i) \exp\left(-\alpha_t y_i h_t(x_i)\right)
\]

What \( \alpha_t \) to choose for hypothesis \( h_t \)?

We can minimize this bound by choosing \( \alpha_t \) on each iteration to minimize \( Z_t \):

\[
Z_t = \sum_{i=1}^{m} D_t(i) \exp\left(-\alpha_t y_i h_t(x_i)\right)
\]

For boolean target function, this is accomplished by [Freund & Schapire '97]:

\[
\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)
\]

You'll prove this in your homework! 😊
Strong, weak classifiers

- If each classifier is (at least slightly) better than random
  - $\varepsilon_t < 0.5$
- AdaBoost will achieve zero training error (exponentially fast):
  \[
  \frac{1}{m} \sum_{i=1}^{m} \delta(H(x_i) \neq y_i) \leq \prod_{t} Z_t \leq \exp \left( -2 \sum_{t=1}^{T} (1/2 - \varepsilon_t)^2 \right) \leq \varepsilon^T \]
- Is it hard to achieve better than random training error

Boosting results – Digit recognition

- Robust to overfitting
- Test set error decreases even after training error is zero

[Schapire, 1989]