Classification/Regression Trees and Random Forests

Fabio G. Cozman - fgcozman@usp.br

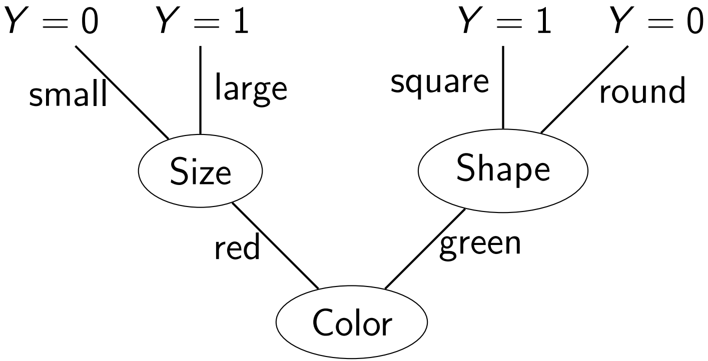
October 29, 2019

Classification tree

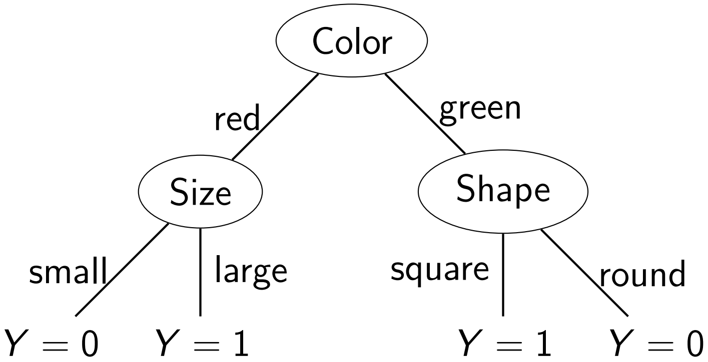
- Consider binary class variable Y and features $X_1, \ldots, X_n$.
- Decide $\hat{Y}$ after a series of splits.
  - This process can be drawn as a tree.

Classification tree

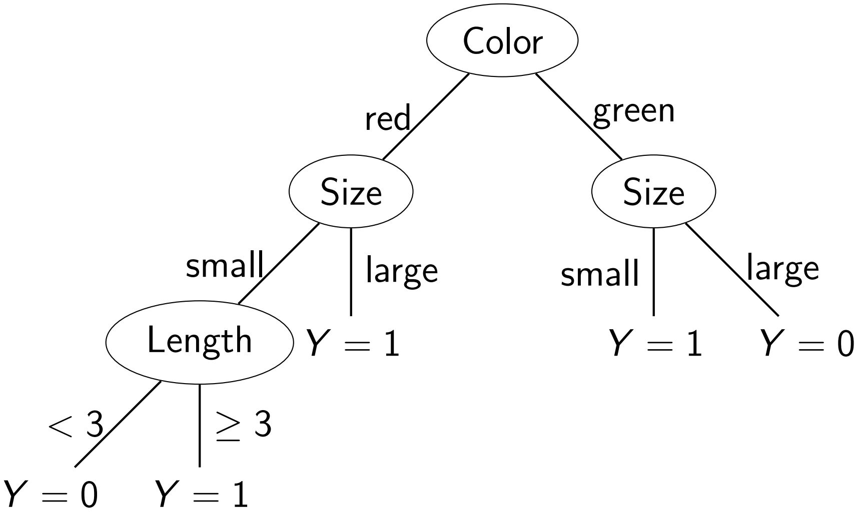
- Consider binary class variable Y and features $X_1, \dots, X_n$.
- Decide $\hat{Y}$ after a series of splits.
  - This process can be drawn as a tree.



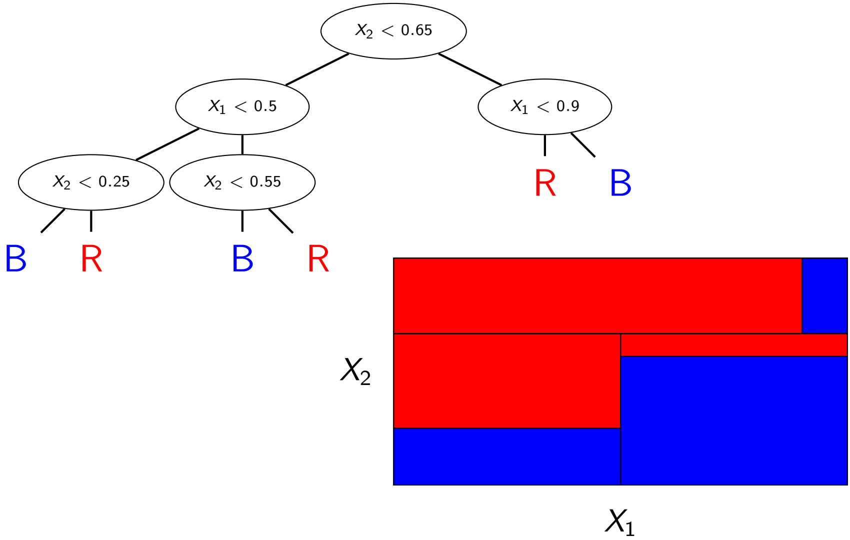
Usual drawing: upside down



Possibly continuous features



Splits are parallel to axes



Some advantages

1. Trees are easy to draw and to understand.
   - Trees can be “translated” to rules.
2. Trees are more flexible in capturing boundaries between classes.
3. Trees can handle continuous and categorical features.

Growing a tree

- Finding "best" tree is computationally hard.

Growing a tree

- Finding “best” tree is computationally hard.
- Basic and popular idea:
    - Select the “best” split; then split and repeat.
    - Keep doing this until some stopping criterion is met.

The C4.5 algorithm

- Algorithm developed by Quinlan.
- Successor to (also popular) ID3.
- Based on entropy and related ideas.

Possible splits

- On values of a categorical feature.
- On thresholds that separate the observations with respect to a continuous feature.

Entropy and information gain

- Entropy of set of observations D:

$$I(D) = -\sum_{k=1}^{K} \frac{|D_k|}{|D|} \log_2 \frac{|D_k|}{|D|}.$$

where D_k is the set of observations such that $\{Y = k\}$.

Entropy and information gain

- Entropy of set of observations D:

$$I(D) = -\sum_{k=1}^{K} \frac{|D_k|}{|D|} \log_2 \frac{|D_k|}{|D|}.$$

  where D_k is the set of observations such that $\{Y = k\}$.

- Information gain of a partition of D into datasets $D^1, \ldots, D^m$:

$$I(D) - \sum_{j=1}^{m} \frac{|D^j|}{|D|} I(D^j).$$

Gain ratio

- Information gain is used in ID3.
- Gain ratio of a partition of D into datasets $D^1, \ldots, D^m$:

$$\frac{I(D) - \sum_{j=1}^{m} \frac{|D^j|}{|D|} I(D^j)}{-\sum_{j=1}^{m} \frac{|D^j|}{|D|} \log_2 \frac{|D^j|}{|D|}}.$$

Algorithm

1. Input: a training dataset.

Algorithm

1. Input: a training dataset.
2. If dataset is empty, stop
   - (actually, stop if size is "small");

   if all labels in dataset are identical, return a leaf with this label.

Algorithm

1. Input: a training dataset.
2. If dataset is empty, stop
   - (actually, stop if size is "small");

   if all labels in dataset are identical, return a leaf with this label.
3. For each possible split, compute the gain ratio of the resulting partition of training dataset.
4. Greedy step: Select split with largest gain ratio.

Algorithm

1. Input: a training dataset.
2. If dataset is empty, stop
   - (actually, stop if size is "small");

   if all labels in dataset are identical, return a leaf with this label.
3. For each possible split, compute the gain ratio of the resulting partition of training dataset.
4. Greedy step: Select split with largest gain ratio.
5. The split corresponds to a node; each element of the partition of the training dataset is an edge to a possible node.
   - Recursive step: Call the algorithm with each such element.

Example: training dataset

	X_1	X_2	$\dots$	X_n	Y
d_1	a				A
d_2	a				A
d_3	c				C
d_4	b				A
d_5	b				A
d_6	c				C
d_7	c				C
d_8	b				B
d_9	b				A
d_{10}	c				A
d_{11}	c				C
d_{12}	a				A
d_{13}	b				B

Example: training dataset

	X_1	X_2	$\ldots$	X_n	Y
d_1	a				A
d_2	a				A
d_3	c				C
d_4	b				A
d_5	b				A
d_6	c				C
d_7	c				C
d_8	b				B
d_9	b				A
d_{10}	c				A
d_{11}	c				C
d_{12}	a				A
d_{13}	b				B

Entropy $I(D) = 1.419556$.

Example: training dataset

	X_1	X_2	$\ldots$	X_n	Y
d_1	a				A
d_2	a				A
d_3	c				C
d_4	b				A
d_5	b				A
d_6	c				C
d_7	c				C
d_8	b				B
d_9	b				A
d_{10}	c				A
d_{11}	c				C
d_{12}	a				A
d_{13}	b				B

Entropy $I(D) = 1.419556$.

Split on X_1:

$I(D^a) = 0$

$I(D^b) = 0.971$

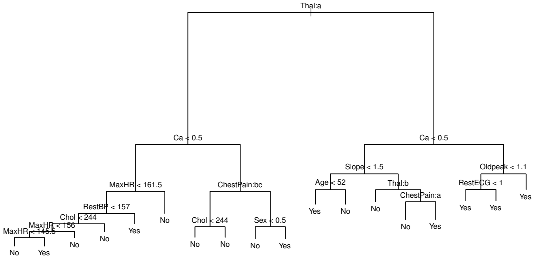
$I(D^c) = 0.722$

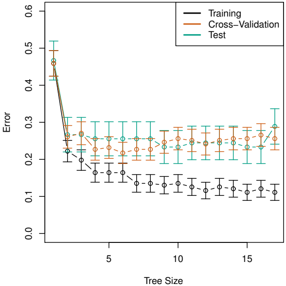
Info.Gain: 0.7685.

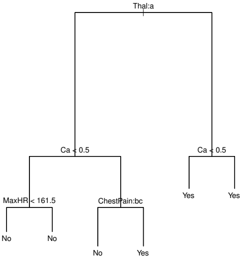
Additional techniques

- C4.5 discounts gains when data are missing.
- C4.5 does some pruning of the classification tree *after* it is built (check whether splits can be removed).
    - In practice there are many pruning techniques.
- There is also a C5.0 (commercial), but it is rarely used.

A classification tree







CART

- Classification And Regression Trees: very similar to C4.5.
- Focus on binary splits (some categorical values may be grouped in the process).
- Splits are evaluated using *Gini index*.

Gini index

- Suppose we have a region containing some observations collected in a dataset D.
- The *Gini index* of this region is

$$G = \sum_{k=1}^{K} \frac{|D_k|}{|D|}\left(1 - \frac{|D_k|}{|D|}\right) = 1 - \sum_{k=1}^{K}\left(\frac{|D_k|}{|D|}\right)^2.$$

Gini index

- Suppose we have a region containing some observations collected in a dataset D.
- The *Gini index* of this region is

$$G = \sum_{k=1}^{K} \frac{|D_k|}{|D|}\left(1 - \frac{|D_k|}{|D|}\right) = 1 - \sum_{k=1}^{K}\left(\frac{|D_k|}{|D|}\right)^2.$$

- The Gini index is a measure of "purity" (small if proportions are closer either to zero or to one).

Evaluating a split

- Consider a node with dataset D (with Gini index g), and a split into two nodes.
  - Suppose the first node has dataset D^1 and Gini index g_1.
  - Suppose the second node has dataset D^2 and Gini index g_2.
- The split is evaluated as

$$g - \frac{|D^1|}{|D|} g_1 - \frac{|D^2|}{|D|} g_2.$$

Many extensions

- Tests may involve combinations of features.
- Construction of tree may be optimal of at least approximately optimal.
- Missing data may be handled with care.

CART: Regression trees

- Suppose Y is a continuous variable.
- Idea is the same: recursively split on features using some criterion.
- New: Each leaf contains a set of observations that can be averaged (or perhaps a linear regression is run locally, etc).

Splitting

- Idea is to minimize RSS:

$$\sum_{d_j \in D} (y_j - \hat{y}_j)^2.$$

- To do so, we must find feature X_i and threshold η that minimizes

$$\sum_{d_j \in D, X_i < \eta} (y_j - \hat{y}_j)^2 + \sum_{d_j \in D, X_i \geq \eta} (y_j - \hat{y}_j)^2.$$

Splitting

- Idea is to minimize RSS:

$$\sum_{d_j \in D} (y_j - \hat{y}_j)^2.$$

- To do so, we must find feature X_i and threshold η that minimizes

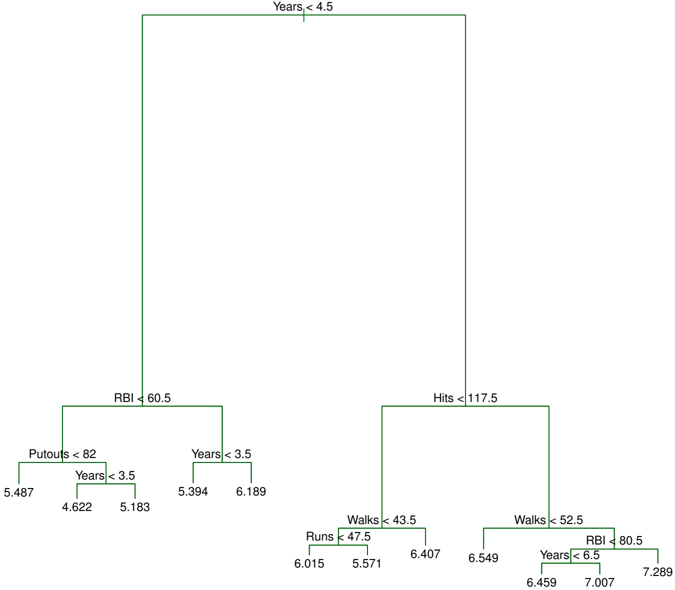
$$\sum_{d_j \in D, X_i < \eta} (y_j - \hat{y}_j)^2 + \sum_{d_j \in D, X_i \geq \eta} (y_j - \hat{y}_j)^2.$$

- Fact: for each feature it is "easy" to find the corresponding η.

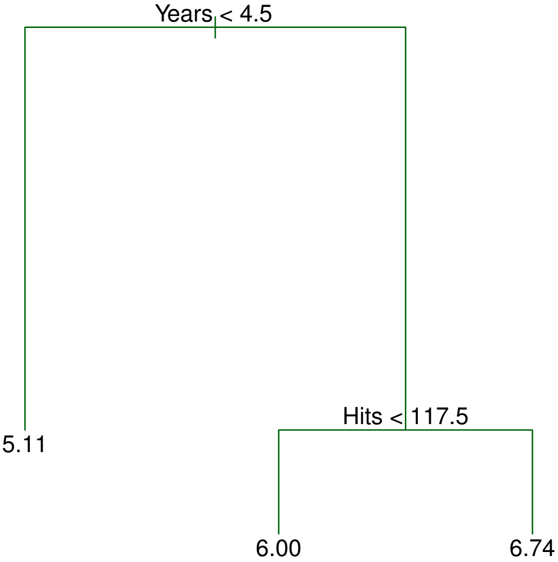
Pruning

- Pruning cuts the depth of the tree.
  - A leaf may then contain a set of observations.
  - Usually classification is obtained by voting.
- Goal is to prevent overfitting:
  1. Consider all possible node removals.
  2. For each one, compute $\sum_{d_j \in D} (y_j - \hat{y}_j)^2 + \alpha |T|$, where $|T|$ is the number of leaves of tree and α is a parameter (get α by cross-validation).
  3. Select node removals to minimize this latter quantity.
  4. Select α and run the whole process with this α.

A regression tree



A regression tree, after pruning



Some disadvantages of trees

Recall:

1. Trees are easy to understand.
2. Trees can be "translated" to rules.
3. Trees are more flexible than logistic (basically linear) regression.
4. Trees can handle continuous and categorical features.

But:

1. Trees are not very accurate.
2. Trees are not robust; very sensitive to training dataset (variance is quite high).

Bagging trees and random forests

- A single tree is nice, but not very accurate and very sensitive.
- How about learning a set of trees and averaging the results?
- This idea leads to *bagging trees* and *random forests*.

Bagging trees

- If you have several estimates of a quantity, averaging them reduces variance.
- Idea: to produce several estimates, resample training dataset (inspired by a technique called *bootstrap*; hence name "bootstrap aggregation").

Bagged regression tree

- Sample with replacement M datasets.
- For each one, produce a regression tree.
- To produce $\hat{y}$, start by generating the M values $\hat{y}_j$ (one per tree), then

$$\hat{y} = (1/M) \sum_j \hat{y}_j.$$

Bagged classification tree

- Sample with replacement M datasets.
- For each one, produce a classification tree.
- To produce $\hat{y}$, start by generating the M values $\hat{y}_j$ (one per tree), then take a vote.

Important

Bagging is a general idea that can be used with all classifiers and regressors!

Digression: MSE without cross-validation!

- When we have many datasets, each observation is used in some resampled datasets, and ignored in others.
- For each observation, produce an average over all trees that were learned *without* the observation.
- Then compute the error for this observation.
- For large M, the sum of squared errors is a good estimate of the MSE (!!).

Random forests

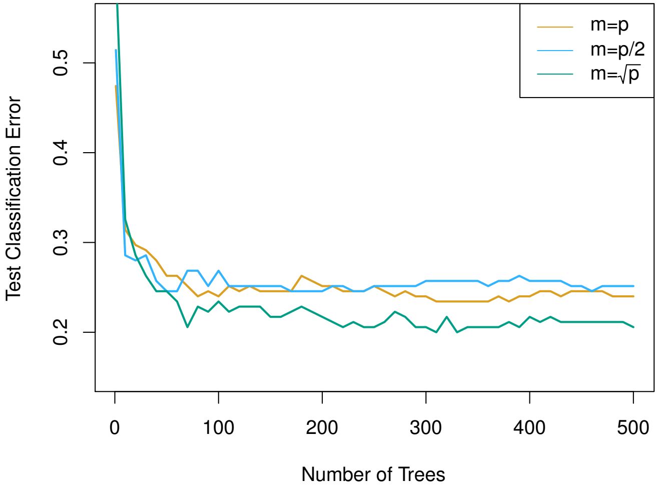
- Suppose we produce a bagged tree with M trees.
- But suppose that, when we grow each tree,
  - for *each* split,
  - we *randomly* select a subset of m features.
- The result is a *random forest*.

Intuition

- By randomly affecting splits, each tree becomes uncorrelated from the others.
- Remember: they are all built with the same data.
- By reducing correlation, variance is reduced.

- This seems silly and bizarre, but it works quite well.

Gene expression: 15 labels, 500 features



Single tree: error rate about 45%.

A note

Some of the figures in this presentation are taken from *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.