# Classification and Regression Trees

David S. Rosenberg
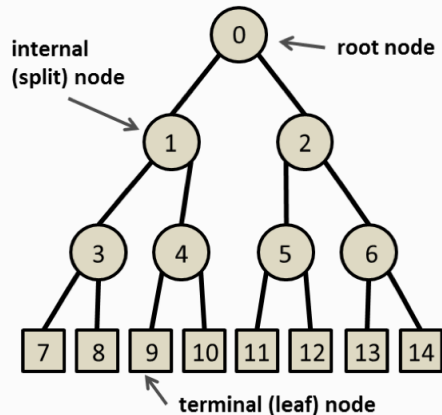
Bloomberg ML EDU

November 21, 2017

# Trees

# Tree Terminology

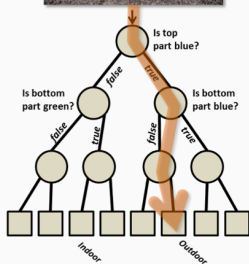**A general tree structure**



internal (split) node

root node

terminal (leaf) node

From Criminisi et al. MSR-TR-2011-114, 28 October 2011.

# A Binary Decision Tree

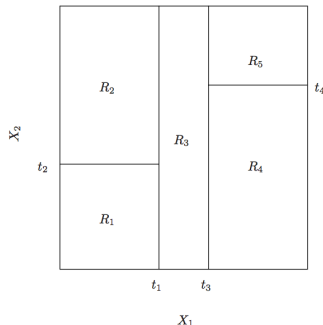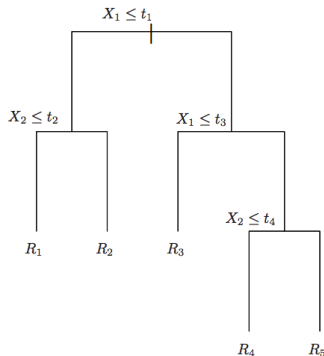**binary tree**: each node has either 2 children or 0 children



From Criminisi et al. MSR-TR-2011-114, 28 October 2011.

# Binary Decision Tree on $\mathbf{R}^2$

- Consider a binary tree on $\{(X_1, X_2) \mid X_1, X_2 \in \mathbf{R}\}$



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.

# Types of Decision Trees

- We'll only consider
  - **binary trees** (vs multiway trees where nodes can have more than 2 children)
  - decisions at each node involve only a single feature (i.e. input coordinate)
  - for continuous variables, splits always of the form

$$x_i \leqslant t$$

  - for discrete variables, partitions values into two groups
- Other types of splitting rules
  - **oblique decision trees** or **binary space partition trees** (BSP trees) have a linear split at each node
  - **sphere trees** – space is partitioned by a sphere of a certain radius around a fixed point

# Regression Trees

# Binary Regression Tree on $\mathbf{R}^2$

- Consider a binary tree on $\{(X_1, X_2) \mid X_1, X_2 \in \mathbf{R}\}$
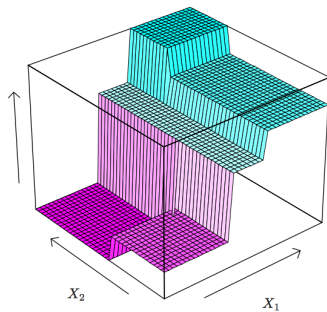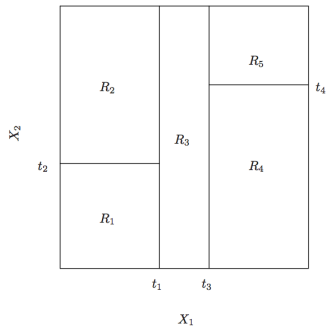


From *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

# Fitting a Regression Tree

- The decision tree gives the partition of $\mathcal{X}$ into regions:

$$\{R_1, \ldots, R_M\}.$$

- Recall that a partition is a **disjoint union**, that is:

$$\mathcal{X} = R_1 \cup R_2 \cup \cdots \cup R_M$$

  and

$$R_i \cap R_j = \emptyset \quad \forall i \neq j$$

# Fitting a Regression Tree

- Given the partition $\{R_1, \ldots, R_M\}$, final prediction is

$$f(x) = \sum_{m=1}^{M} c_m 1(x \in R_m)$$

- How to choose $c_1, \ldots, c_M$?
- For loss function $\ell(\hat{y}, y) = (\hat{y} - y)^2$, best is

$$\hat{c}_m = \mathsf{ave}(y_i \mid x_i \in R_m).$$

# Trees and Overfitting

- If we do enough splitting, every unique $x$ value will be in its own partition.
- This very likely overfits.

- As usual, we need to control the complexity of our hypothesis space.

- CART (Breiman et al. 1984) uses **number of terminal nodes**.
- Tree depth is also common.

# Complexity of a Tree

- Let $|T| = M$ denote the number of terminal nodes in $T$.
- We will use $|T|$ to measure the complexity of a tree.
- For any given complexity,
  - we want the tree minimizing square error on training set.
- Finding the optimal binary tree of a given complexity is computationally intractable.
- We proceed with a **greedy algorithm**
  - Means build the tree one node at a time, without any planning ahead.

# Root Node, Continuous Variables

- Let $x = (x_1, \ldots, x_d) \in \mathbf{R}^d$. ($d$ features)
- **Splitting variable** $j \in \{1, \ldots, d\}$.
- **Split point** $s \in \mathbf{R}$.
- Partition based on $j$ and $s$:

$$R_1(j,s) = \{x \mid x_j \leqslant s\}$$
$$R_2(j,s) = \{x \mid x_j > s\}$$

# Root Node, Continuous Variables

- For each splitting variable $j$ and split point $s$,

$$\hat{c}_1(j,s) = \text{ave}(y_i \mid x_i \in R_1(j,s))$$
$$\hat{c}_2(j,s) = \text{ave}(y_i \mid x_i \in R_2(j,s))$$

- Find $j,s$ minimizing loss

$$L(j,s) = \sum_{i:x_i \in R_1(j,s)} (y_i - \hat{c}_1(j,s))^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{c}_2(j,s))^2$$

- How?

- Consider splitting on the $j$'th feature $x_j$.
- If $x_{j(1)}, \ldots, x_{j(n)}$ are the sorted values of the $j$'th feature,
  - we only need to check split points between adjacent values
  - traditionally take split points halfway between adjacent values:

$$s_j \in \left\{ \frac{1}{2} \left( x_{j(r)} + x_{j(r+1)} \right) \mid r = 1, \ldots, n-1 \right\}.$$

- So only need to check performance of $n-1$ splits.

1. We have determined $R_1$ and $R_2$
2. Find best split for points in $R_1$
3. Find best split for points in $R_2$
4. Continue...

- When do we stop?

# Complexity Control Strategy

- If the tree is too big, we may overfit.
- If too small, we may miss patterns in the data (underfit).
- Can limit max depth of tree.
- Can require all leaf nodes contain a minimum number of points.
- Can require a node have at least a certain number of data points to split.
- Can do **backward pruning** (the approach of **CART** (Breiman et al 1984):
    1. Build a really big tree (e.g. until all regions have $\leqslant 5$ points).
    2. **"Prune"** the tree back greedily all the way to the root, assessing performance on validation.

# Classification Trees

# Classification Trees

- Consider classification case: $\mathcal{Y} = \{1, 2, \ldots, K\}$.
- We need to modify
  - criteria for splitting nodes

# Classification Trees

- Let node $m$ represent region $R_m$, with $N_m$ observations
- Denote proportion of observations in $R_m$ with class $k$ by

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{\{i : x_i \in R_m\}} 1(y_i = k).$$

- **Predicted classification** for node $m$ is

$$k(m) = \underset{k}{\arg\max}\, \hat{p}_{mk}.$$

- **Predicted class probability distribution** is $(\hat{p}_{m1}, \ldots, \hat{p}_{mK})$.

# Misclassification Error

- Consider node $m$ representing region $R_m$, with $N_m$ observations
- Suppose we predict

$$k(m) = \arg\max_k \hat{p}_{mk}$$

  as the class for all inputs in region $R_m$.
- What is the misclassification rate on the training data?
- It's just

$$1 - \hat{p}_{mk(m)}.$$

# What loss function to use for node splitting?

- Natural loss function for classification is 0/1 loss.
- Is this tractable for finding the best split? Yes!
- Should we use it? Maybe not!
- If we're only splitting once, then make sense to split using ultimate loss function (say 0/1).
- But we can split nodes repeatedly – don't have to get it right all at once.

# Splitting Example

- Two class problem: 4 observations in each class.
- Split 1: (3,1) and (1,3) [each region has 3 of one class and 1 of other]

- Split 2: (2,4) and (2,0) [one region has 2 of one class and 4 of other, other region pure]

- Misclassification rate for the two splits are same. (2).

- In split 1, we'll want to split each node again, and
    - we'll end up with a leaf node with a single element.node .
- In split 2, we're already done with the node $(2, 0)$.

# Splitting Criteria

- Eventually we want **pure** leaf nodes (i.e. as close to a single class as possible).

- We'll find splitting variables and split point minimizing some **node impurity** measure.

# Two-Class Node Impurity Measures

- Consider binary classification
- Let $p$ be the relative frequency of class 1.
- Here are three node impurity measures as a function of $p$
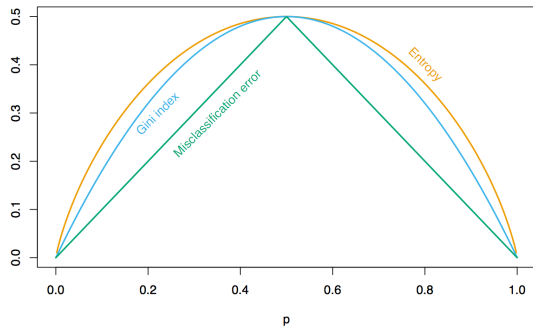


HTF Figure 9.3

# Classification Trees: Node Impurity Measures

- Consider leaf node $m$ representing region $R_m$, with $N_m$ observations
- Three measures $Q_m(T)$ of **node impurity** for leaf node $m$:
  - Misclassification error:
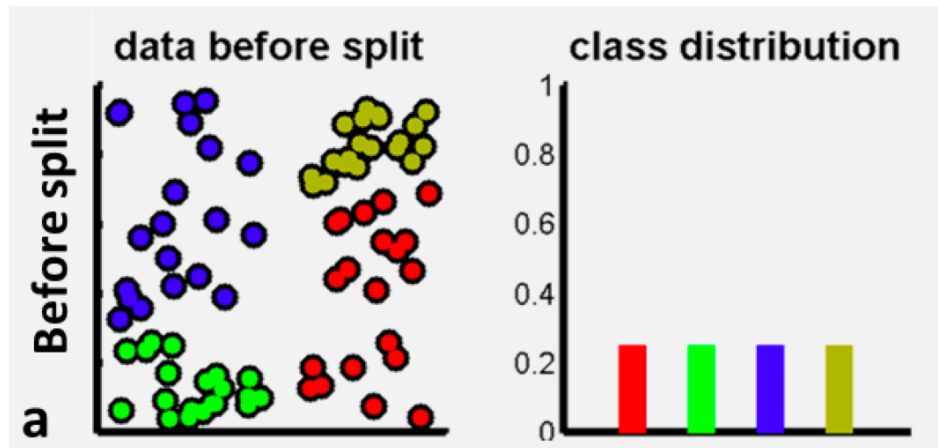    $$1 - \hat{p}_{mk(m)}.$$

  - Gini index:
    $$\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

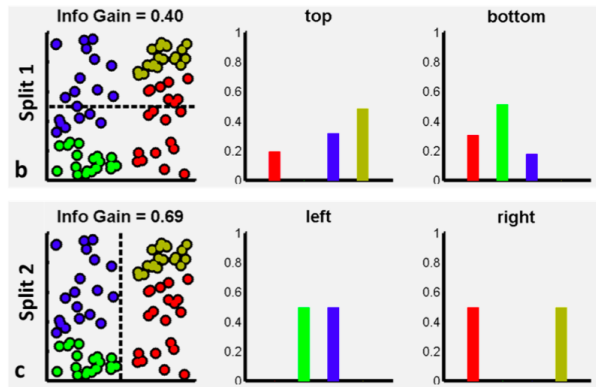  - Entropy or deviance (equivalent to using information gain):
    $$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

# Class Distributions: Pre-split



From Criminisi et al. MSR-TR-2011-114, 28 October 2011.

# Class Distributions: Split Search



(Maximizing information gain is equivalent to minimizing entropy.)

From Criminisi et al. MSR-TR-2011-114, 28 October 2011.

# Splitting nodes: How exactly do we do this?

- Let $R_L$ and $R_R$ be regions corresponding to a potential node split.
- Suppose we have $N_L$ points in $R_L$ and $N_R$ points in $R_R$.
- Let $Q(R_L)$ and $Q(R_R)$ be the node impurity measures.
- Then find split that minimizes the **weighted average of node impurities**:

$$N_L Q(R_L) + N_R Q(R_R)$$

# Classification Trees: Node Impurity Measures

- For building the tree, Gini and Entropy seem to be more effective.
- They push for more pure nodes, not just misclassification rate
- A good split may not change misclassification rate at all!

- Two class problem: 4 observations in each class.
- Split 1: (3,1) and (1,3) [each region has 3 of one class and 1 of other]
- Split 2: (2,4) and (2,0) [one region has 2 of one class and 4 of other, other region pure]
- Misclassification rate for two splits are same.
- **Gini and entropy split prefer Split 2.**

# Trees in General

# Missing Features

- What to do about missing features?
  - Throw out inputs with missing features
  - Impute missing values with feature means
  - If a categorical feature, let "missing" be a new category.

- For trees, we can use **surrogate splits**
  - For every internal node, form a list of surrogate features and split points
  - Goal is to approximate the original split as well as possible
  - Surrogates ordered by how well they approximate the original split.
    - In terms of how many examples are sent in the same direction by each split.

# Categorical Features

- Suppose we have a categorical feature with $q$ possible values (unordered).
- We want to find the best split into 2 groups
- There are $2^{q-1} - 1$ distinct splits.
- Is this tractable? Maybe not in general. But...
- For binary classification $\mathcal{Y} = \{0, 1\}$, there is an efficient algorithm.
    - Assign each category a number, the proportion of class 0.
    - Then find optimal split as though it were a numeric feature.
    - Proved to be equivalent to search over all splits in (Breiman et al. 1984).
- Otherwise, can use approximations..
- Statistical issues?
    - If a category has a very large number of categories, we can overfit.
    - Extreme example: Row Number could lead to perfect classification with a single split.

- Trees have to work much harder to capture linear relations.
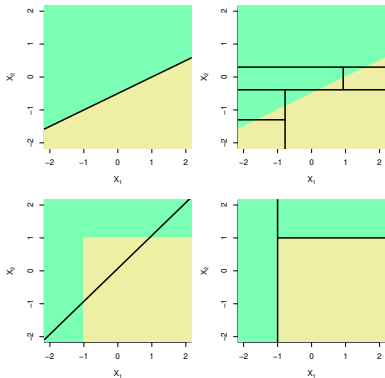


From *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

# Interpretability

- Trees are certainly easy to explain.
- You can show a tree on a slide.
- Small trees seem interpretable.
- For large trees, maybe not so easy.

# Trees for Nonlinear Feature Discovery

- Suppose tree $T$ gives partition $R_1, \ldots, R_m$.
- Predictions are

$$f(x) = \sum_{m=1}^{M} c_m 1(x \in R_m)$$

- Each region $R_m$ can be viewed as giving a feature function $x \mapsto 1(x \in R_m)$.
- Can use these nonlinear features in e.g. lasso regression.

# Comments about Trees

- Trees make no use of **geometry**
  - No inner products or distances
  - called a "nonmetric" method
  - **Feature scale irrelevant**
- Predictions are not continuous
  - not so bad for classification
  - may not be desirable for regression

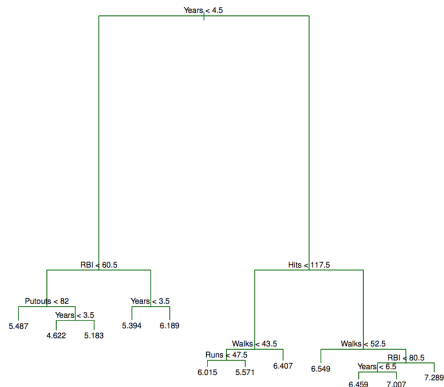# Tree Pruning

# Stopping Conditions for Building the Big Tree

- First step is to build the "big tree".
- Keep splitting nodes until every node either has
  - Zero error OR
  - Node has $C$ or fewer examples (typically $C = 5$ or $C = 1$)

# Pruning the Tree

- Consider an internal node $n$.
- To prune the subtree rooted at $n$
  - eliminate all descendents of $n$
  - $n$ becomes a terminal node

# Tree Pruning

- Full Tree $T_0$

# Tree Pruning

- Subtree $T \subset T_0$



Years < 4.5

Hits < 117.5

5.11

6.00

6.74

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.

# Empirical Risk and Tree Complexity

- Suppose we want to prune a big tree $T_0$.

- Let $\hat{R}(T)$ be the empirical risk of $T$ (i.e. square error on training)

- Clearly, for any subtree $T \subset T_0$, $\hat{R}(T) \geqslant \hat{R}(T_0)$.

- Let $|T|$ be the number of terminal nodes in $T$.
- $|T|$ is our measure of complexity for a tree.

# Cost Complexity (or Weakest Link) Pruning

**Definitions**

The **cost complexity criterion** with parameter $\alpha$ is

$$C_\alpha(T) = \hat{R}(T) + \alpha |T|$$

- Trades off between empirical risk and complexity of tree.

- Cost complexity pruning:
  - For each $\alpha$, find the subtree $T \subset T_0$ minimizing $C_\alpha(T)$ (on training data).
  - Use cross validation to find the right choice of $\alpha$.

# Do we need to search over all subtrees?

- The **cost complexity criterion** with parameter $\alpha$ is

$$C_\alpha(T) = \hat{R}(T) + \alpha|T|$$

- $C_\alpha(T)$ has familiar regularized ERM form, but
- Cannot just differentiate w.r.t. parameters of a tree $T$.

- To minimize $C_\alpha(T)$ over subtrees $T \subset T_0$,
  - seems like we need to evaluate exponentially many[1] subtrees $T \subset T_0$.
  - (In particular, we can include or exclude any subset of internal nodes that are parents of leaf nodes.)

- Amazingly, we only need to try $N_{\text{Int}}$, where $N_{\text{Int}}$ is the number of internal nodes of $T_0$.

---

[1]See On subtrees of trees.

# Cost Complexity Greedy Pruning Algorithm

- Find a proper[2] subtree $T_1 \subset T_0$ that minimizes $\hat{R}(T_1) - \hat{R}(T_0)$.
    - Can get $T_1$ by removing a single pair of leaf nodes, and their internal node parent becomes a leaf node.
    - This $T_1$ will have 1 fewer internal node than $T_0$. (And 1 fewer leaf node.)
- Then find proper subtree $T_2 \subset T_1$ that minimizes minimizes $\hat{R}(T_2) - \hat{R}(T_1)$.
- Repeat until we have removed all interal nodes are left with just a single node (a leaf node).
- If $N_{\mathsf{Int}}$ is the number of internal nodes of $T_0$, then we end up with a nested sequence of trees:

$$\mathcal{T} = \left\{ T_0 \supset T_1 \supset T_2 \supset \cdots \supset T_{|N_{\mathsf{Int}}|} \right\}$$

---

[2] $T_1$ is a proper subtree of $T_0$ if tree $T_1 \subset T_0$ and $T_1 \neq T_0$.

# Greedy Pruning is Sufficient

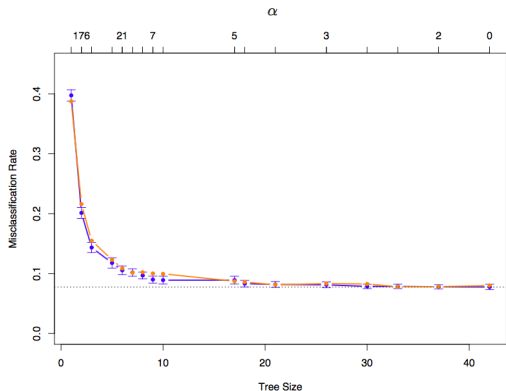- Cost complexity pruning algorithm gives us a set of nested trees:

$$\mathcal{T} = \left\{ T_0 \supset T_1 \supset T_2 \supset \cdots \supset T_{|N_{\text{Int}}|} \right\}$$

- Breiman et al. (1984) proved that this is all you need. That is:

$$\left\{ \underset{T \subset T_0}{\arg\min}\, C_\alpha(T) \mid \alpha \geqslant 0 \right\} \subset \mathcal{T}$$

- Only need to evaluate $N_{\text{Int}}$ trees.

For each $\alpha$, we find optimal tree $T_\alpha$ on training set. Corresponding tree size $|T_\alpha|$ is shown on bottom. Blue curves gives error rate estimates from cross-validation (tree-size in each fold may be different from $|T_\alpha|$). Orange curve is test error.