Prediction Trees

CART, Bagging, and Random Forest

EN5422/EV4238 | Fall 2023

w07_trees.pdf

(Week 7 – 1/2)

Contents

1 Classification and Pattern Recognition

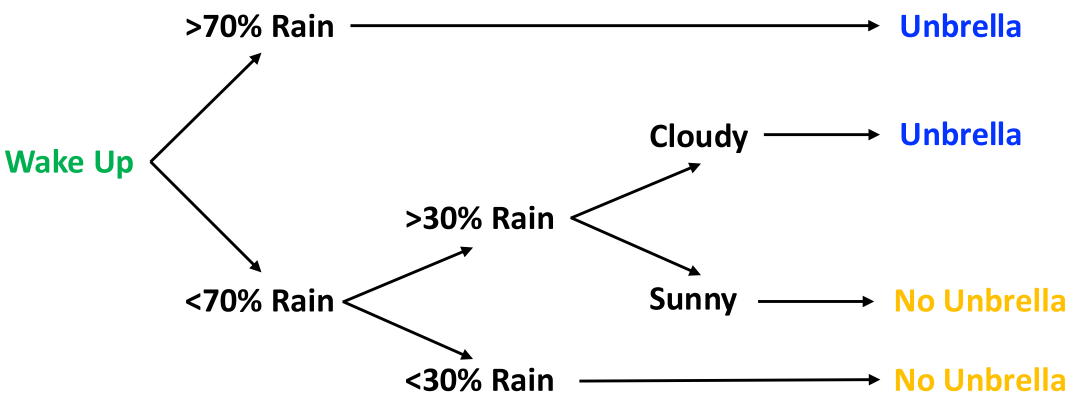
Tree-based methods:

1. Partition the feature space into a set of (hyper) rectangles
2. Fit a simple model (e.g., constant) in each one.

They are conceptually simple yet powerful.

- Main Characteristics:
  - flexibility, intuitive, non-model based
  - natural graphical display, easy to interpret
  - building blocks of Random Forest and (tree-based) Boosting
  - naturally includes feature interactions
  - reduces need for monotonic feature transformations
- Main Implementations:
  - CART (Classification and Regression Trees) by Breiman, Friedman, Olshen, Stone (1984)
  - C4.5 Quinlan (1993)
  - Conditional Inference Trees (scikit-learn Python library)

1.1 Decision Trees



1.2 Building Trees

As usual, we want to find the tree that minimize some loss.

- Classification trees have class probabilities at the leaves (e.g., the probability it will be in heavy rain is 0.9)
    - e.g., Loss = Cross-Entropy (or log loss)

Example: Cross-Entropy (log loss)

Predicted Probabilities: The probabilities that your model predicts for the positive class (e.g., rain).

Actual Labels: These are the true labels for your data (e.g., 0 for no rain, 1 for rain).

Cross-Entropy Loss: The formula for binary cross-entropy loss is:

$$-\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

Where N is the number of samples, y_i is the actual label for the i-th sample, and p_i is the predicted probability for the i-th sample.

```
import numpy as np

# Sample predictions (probabilities of positive class)
predictions = np.array([0.9, 0.7, 0.2, 0.4])

# Actual labels
actual_labels = np.array([1, 1, 0, 0])

# Calculate cross-entropy loss
cross_entropy_loss = -np.mean(actual_labels * np.log(predictions) + (1 -
actual_labels) * np.log(1 - predictions))

print(f"Cross-Entropy Loss: {cross_entropy_loss}")
```

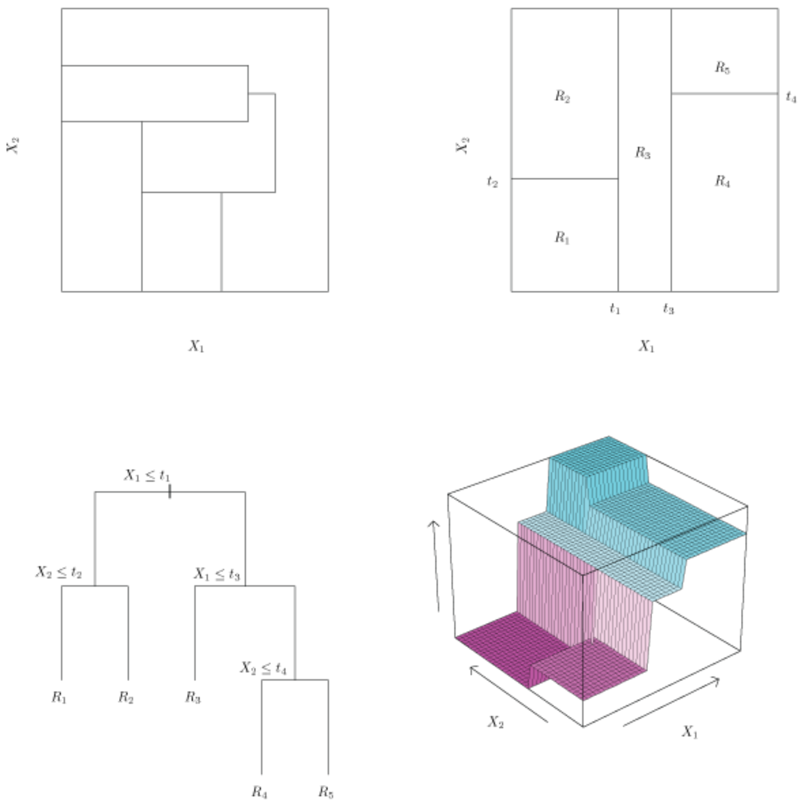
```
Cross-Entropy Loss: 0.2990011586691898
```

- Regression trees have a mean response at the leaves. (e.g., the expected amount of rain is 2 cm).
    - e.g., Loss = Mean squared error.

1.3 Recursive Binary Partition (CART)

Because the number of possible trees is too large to exhaustively search, we usually restrict attention to recursive binary partition trees (CART).

- These are also easy to interpret



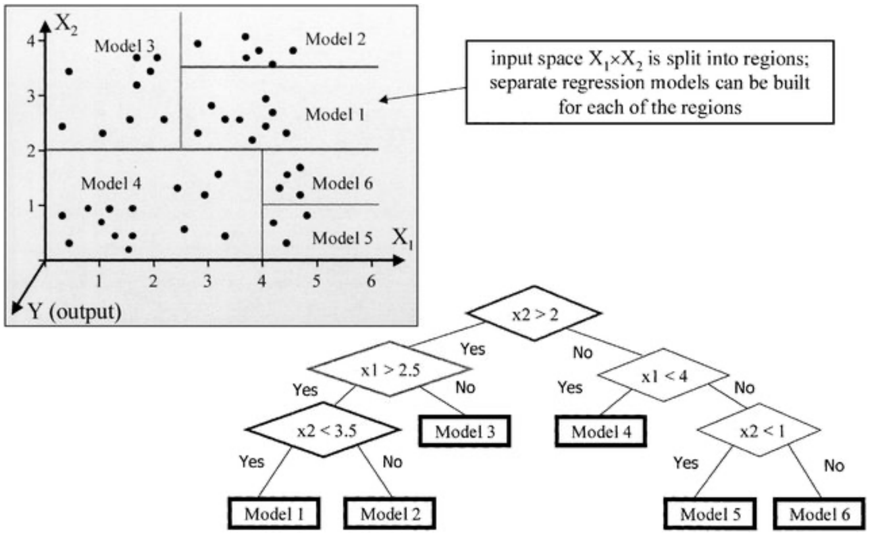
Think of *reverse* of agglomerative hierarchical clustering

- In hierarchical clustering, we started with all observations in clusters of size 1 and then sequentially grouped them together, according to some measure of *homogeneity/similarity/distance/dissimilarity/loss*, until there was one big cluster.
  - The optimal clustering is usually somewhere between the two extremes
- In CART, all observations start in one big group and are split into two subgroups. Each subgroup is then split into two additional subgroups. This is repeated until some stopping criteria is met (e.g., not enough observations in to split further). The terminal subgroup (leaf nodes) are used to make predictions.
  - The splitting is also based on some measure of homogeneity/similarity/loss.
  - Since we are in a supervised setting, the splitting criterion should be based on how well the new groups estimate the outcome variable.

- There is another important difference: in CART only a single feature is used to determine the split into subgroups.

Note: M5 Model Trees

M5 model tree contains a linear regression model that makes predictions based on the input features. This approach allows M5 model trees to capture linear relationships within subsets of the data, making them more flexible and potentially more accurate for certain types of regression problems.



(Solomatine&Xue)

Note: CART vs Constructing All Possible Trees

1. **Evaluating All Splits in CART**:
    - **Local Optimization**: At each step, CART evaluates all possible ways to split the data based on the current set of features. However, this evaluation is local to the current node of the tree. It only looks for the best split at that particular step, not for the entire tree.
    - **Greedy Approach**: CART uses a greedy algorithm, meaning it makes the best decision at each step without considering future implications. It chooses the split that provides the best improvement in purity or reduction in error at the current node, then moves on to the next node.
2. **Constructing All Possible Trees**:
    - **Global Optimization**: Constructing all possible trees would mean considering every possible combination of splits across all levels of the tree. This isn't just choosing the best split at each node; it's about considering every possible shape and structure the tree could take.
    - **Exponential Complexity**: The number of possible trees grows exponentially with the number of features and the depth of the tree. Even for a modest number of features and depth, the total number of possible trees becomes computationally infeasible.
3. **Practical Example**:

- With CART, at each node, you might evaluate, say, 10 different ways to split the data. Once you choose the best split, you move on to the next node and again evaluate 10 splits, and so on. The process is sequential and each decision is based only on the current node.
- In contrast, constructing all possible trees means you would look at every combination of splits across all nodes. If you had 10 possible splits at each node and wanted to build a tree of depth 3, you would have to evaluate 10×10×10=1,000 different trees, and this number increases exponentially with depth.

1.3.1 Model and Model Parameters

Trees model the outcome as a *constant* in each region

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^{M} \hat{c}_m \, \mathbb{1}\left(\mathbf{x} \in \hat{R}_m\right)$$

- The *model parameters* of a tree, T, with M leaf nodes, are:
  - The region (leaf nodes) $R_1, \ldots, R_M$
  - The coefficients/scores for the regions $c_1, \ldots, c_M$
- The coefficients are based on the loss
  - Under Squared Error (regression):

$$\hat{c}_m = \text{Ave}(\{y_i : \mathbf{x}_i \in R_m\}) = \frac{1}{N_m} \sum_{i:\mathbf{x}_i \in R_m} y_i$$

  - Under log-loss (soft classification), the coefficients are probability *vectors* (one element for each class; sums to one).

$$\begin{aligned}\hat{c}_{mk} &= \text{Proportion of class } k \text{ } i\text{n region } R_m \\ &= \frac{1}{N_m} \sum_{i:\mathbf{x}_i \in R_m} \mathbb{1}(y_i = k)\end{aligned}$$

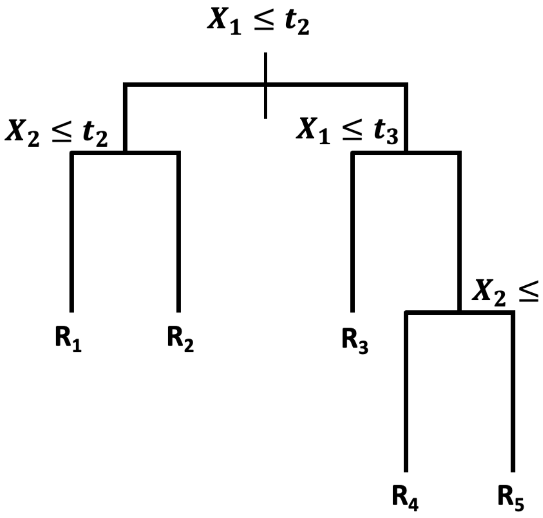
  - Under 0-1 loss (hard classification), the coefficients are one-hot vectors.

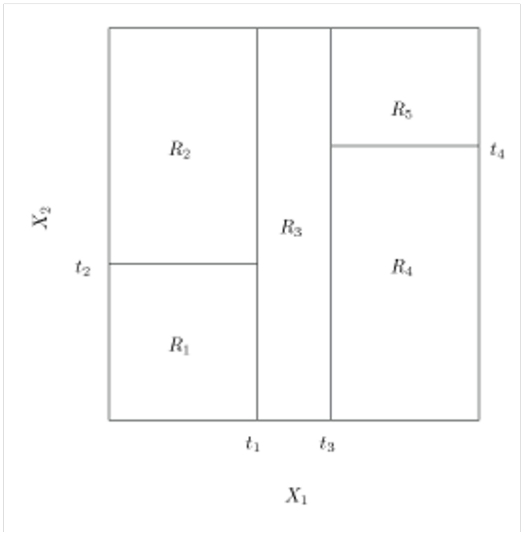
$$\begin{aligned}\hat{c}_{mk} &= \text{One hot for majority class} \\ &= \mathbb{1}(k \text{ is majority class in region } R_m)\end{aligned}$$

  - Other options possible; choose the coefficients to optimize your particular objective function.

Note: check the loss (implicitly) used in growing the tree!

1.3.2 Basis Expansion Interpretation



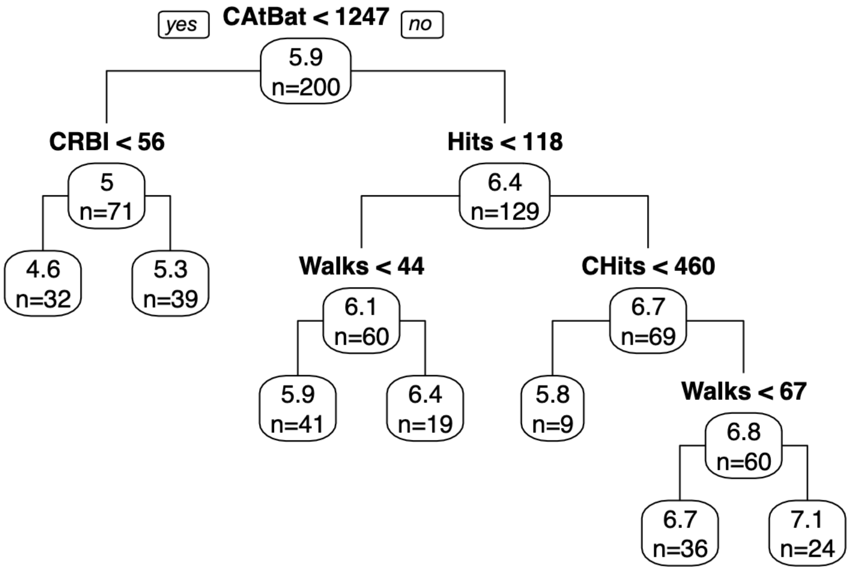


$$f(x) = \sum_{m=1}^{M} \theta_m b_m(x)$$

$$
\begin{aligned}
&R_1: b_1(x_1, x_2) = \mathbb{1}(x_1 \leq t_1)\mathbb{1}(x_2 \leq t_2)\\
&R_2: b_2(x_1, x_2) = \mathbb{1}(x_1 \leq t_1)\mathbb{1}(x_2 > t_2)\\
&R_3: b_3(x_1, x_2) = \mathbb{1}(x_1 > t_1)\mathbb{1}(x_1 \leq t_3)\\
&R_4: b_4(x_1, x_2) = \mathbb{1}(x_1 > t_1)\mathbb{1}(x_1 > t_3)\mathbb{1}(x_2 \leq t_4)\\
&R_5: b_5(x_1, x_2) = \mathbb{1}(x_1 > t_1)\mathbb{1}(x_1 > t_3)\mathbb{1}(x_2 > t_4)
\end{aligned}
$$

1.3.3 Example: Baseball Salaries

The ISLR Python package (corresponding to the ISLR textbook), contains data (Hitters) on Major League Baseball players for the 1986-1987 season.

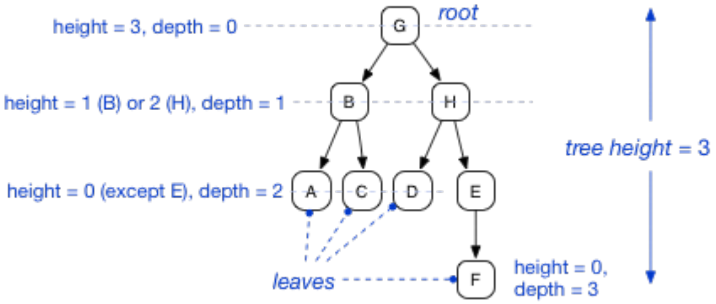


Your Turn #1: Tree Interpretation

1. How many leaves are on the tree?

2. What do the numbers in the boxes mean?

3. How could you evaluate the prediction in a leaf node?



(cs.cornell.edu)

1.4 Growing a Tree

CART uses a greedy algorithm to grow a tree.

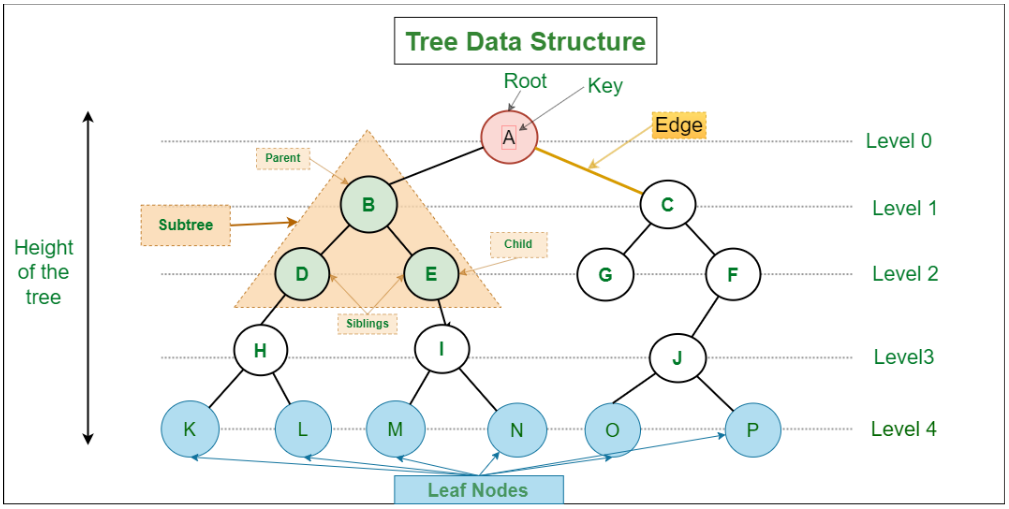
- Split the feature space into two pieces and predict the outcome in each region.
  - Find the predictor *j* (out of 1, 2, …, *p*) and split point *t* (from unique ordered values of X_j or categories) to minimize the loss function.
  - Produce two regions:

$$R_1(j,t) = \{x: x_j \le t\} \text{ and } R_2(j,t) = \{x: x_j > t\} \text{ Numeric/Ordered Feature}$$

Or

$$R_1(j,t) = \{x: x_j \in A_j\} \text{ and } R_2(j,t) = \{x: x_j \notin A_j\} \text{ Nominal/Categorial Feature}$$

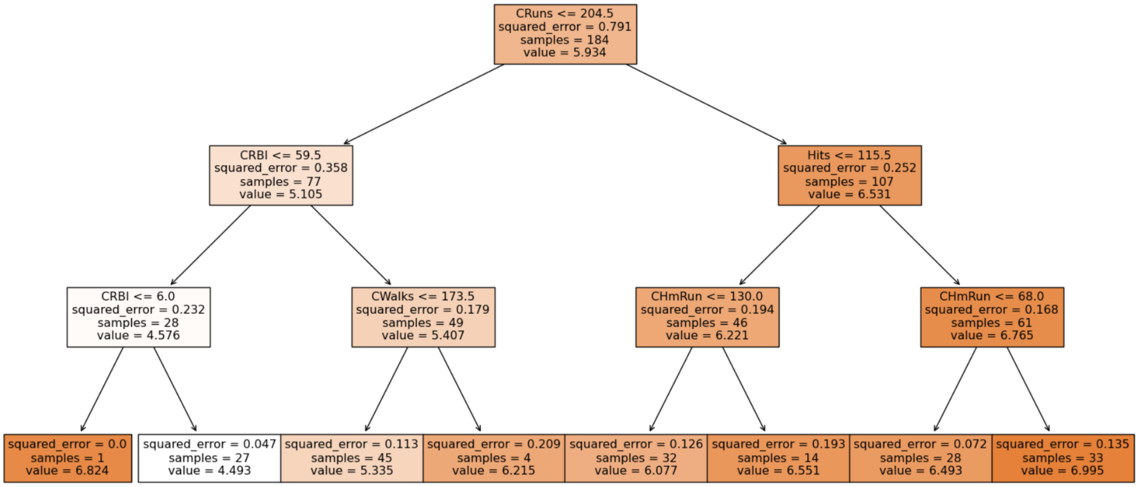
- Repeat this step for each child region



(geeksforgeek.org) (something is wrong with this figure… did you find it?)

- Continue until stopping criteria met, e.g.,
  - Minimum number of observations in region
  - Loss function has minimal improvement
  - Maximum depth (number of interactions)
- The final regions are called leaf nodes.

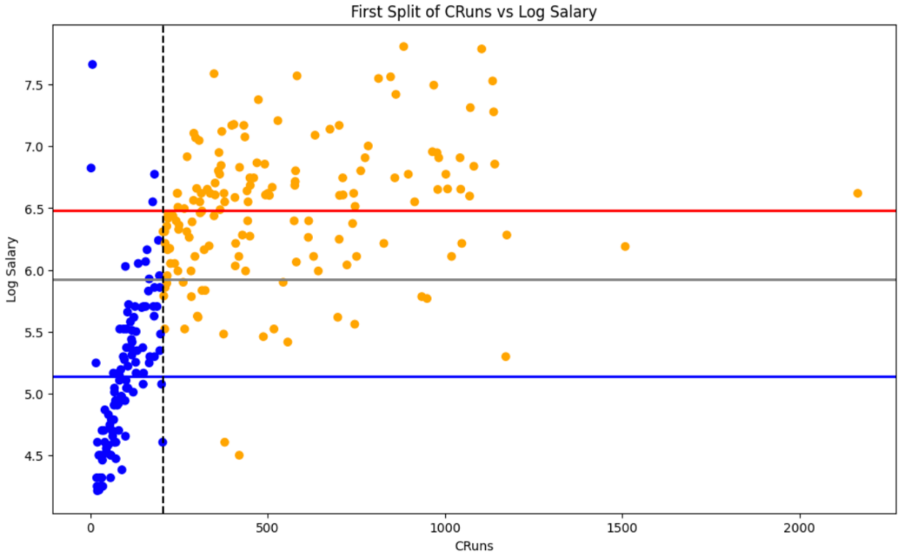
1.5 Splitting Details



1.5.1 Regression Trees and Numeric Features

Notice in the fitted tree for the baseball data that the first split was based on a player's *Career Runs* (CRuns). Specifically, if a player has less than 204.5 runs they go down the left side, otherwise they go down the right side.

Let's examine the first split:



- This is basically a univariate *change point model*
  - The split point (CRuns < 204.5) is the best change point (change in mean) using a Gaussian model
  - An alternative perspective is to see that the reduction in MSE/SSE is maximized by splitting at (CRuns < 204.5) and fitting the data on each side of the split with a constant.

Splitting Details: Squared Error Loss

Notation

- $y \in \mathbb{R}$
- $x = [x_1, \ldots, x_p]^T$
- n observations (in current node/region)

Consider a split on feature *j*:

- Before the split, the quality of the model, based on SSE is:

$$Q_0 = \sum_{i=1}^{n}(y_i - \bar{y})^2 \quad \text{where } \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

- Consider the split at *s* (on feature *j*)

Left Region

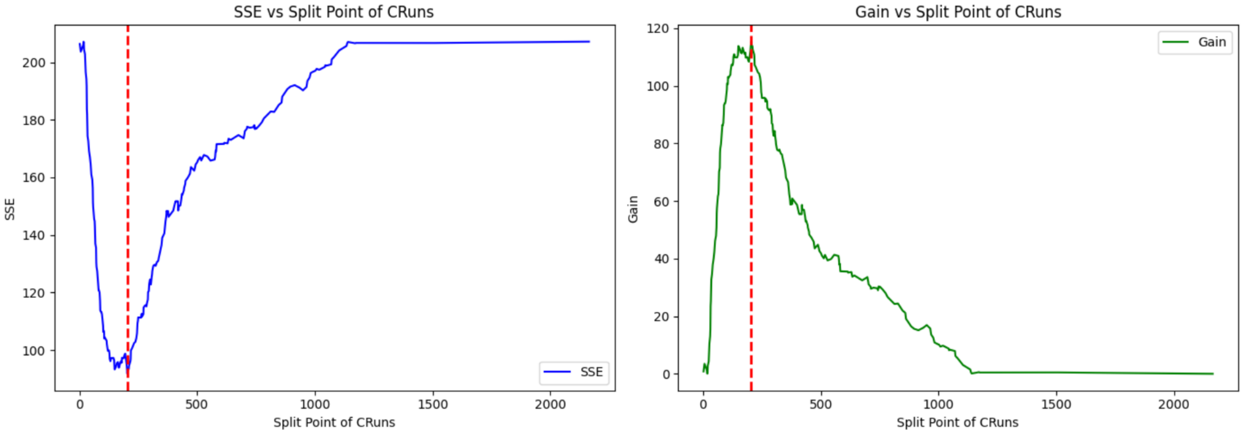
$$R_1(s) = \{x: x_j < s\}$$
$$\bar{y}_1(s) = \frac{1}{n_1}\sum_{\{i:x_i \in R_1(s)\}} y_i$$
$$Q_1(s) = \sum_{\{i:x_i \in R_1(s)\}} (y_i - \bar{y}_1(s))^2$$

Right Region

$$R_2(s) = \{x: x_j \geq s\}$$
$$\bar{y}_2(s) = \frac{1}{n_2}\sum_{\{i:x_i \in R_2(s)\}} y_i$$
$$Q_2(s) = \sum_{\{i:x_i \in R_2(s)\}} (y_i - \bar{y}_2(s))^2$$

- Updated SSE: $Q(s) = Q_1(s) + Q_2(s)$
- Gain(s) = $Q_0 - Q(s)$

We can examine the SSE (or Gain) for all possible split points:

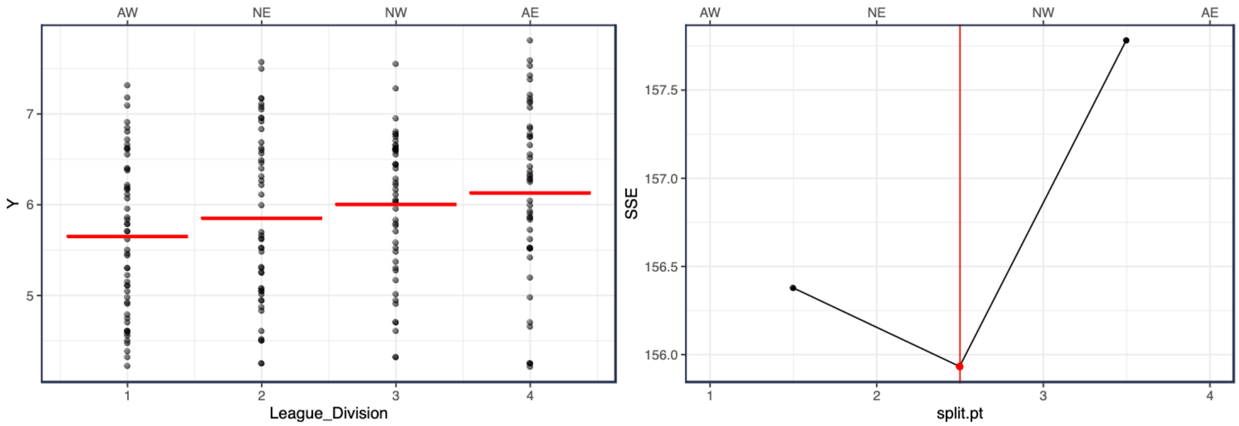


1.5.2 Regression Trees and Categorical (Nominal) Features

A categorical feature (with k levels) can be split into two groups $2^{k-1} - 1$ different way.

- $k = 3$: 3 possible partitions.
- $k = 4$: 7 possible partitions.
- $k = 10$: 511 possible partitions

The CART approach sorts the categories by the mean response (recodes to numeric) and then splits like it's a numeric feature.



Note

- Note: features with many levels will be split too often. Consider the quote from ESL (pg. 310):
  The partitioning algorithm tends to favor categorical predictors with many levels k; the number of partitions grows exponentially in k, and the more choices we have, the more likely we can find a good one for the data at hand. This can lead to severe overfitting if k is large, and such variables should be avoided.

- An alternative is to use one-hot-encoding to split a categorical feature into k new features as done by XGBoost.

- There are other ways to *encode* categorical data so they can be treated like numeric (i.e., ordered data). See CatBoost.

1.5.3 Classification Trees

A classification tree is used when the outcome is categorical $y \in \mathcal{G} = (1,2,\dots,K)$.

- In region R_m, the probability of class k can be estimated:

$$\bar{p}_m(k) = \widehat{Pr}(y = k | x \in R_m) = \frac{1}{n_m} \sum_{\{i:x \in R_m\}} \mathbb{1}(y_i = k) = \frac{n_{m,k}}{n_m}$$

- Each region has K-vector of probability estimates: $\bar{p}_m = [\bar{p}_m(1), \dots, \bar{p}_m(K)]$
    - $\sum_{k=1}^{K} \bar{p}_m(k) = 1$

There are three common measures of *node impurity* in this setting:

1. Misclassification Errors:

$$Q_m = 1 - \max_k \bar{p}_m(k)$$

2. Gini Index:

$$\begin{aligned} Q_m &= \sum_{k=1}^{K} \bar{p}_m(k)(1 - \bar{p}_m(k)) \\ &= 1 - \sum_{k=1}^{K} \bar{p}_m^2(k) \end{aligned}$$

3. Cross-entropy/Deviance:

$$\begin{aligned} Q_m &= -\sum_{k=1}^{K} \bar{p}_m(k) \log \bar{p}_m(k) \\ &= \sum_{k=1}^{K} \bar{p}_m(k) \log \frac{1}{\bar{p}_m(k)} \end{aligned}$$

1.5.4 Splitting Summary

For each iteration, we calculate the Gain/Loss for *all* feature $j = 1, 2, \dots, p$ and *all* possible split and choose the pair that minimizes the loss (or maximizes the gain):

$$(j^*, s^*) = \arg \min_{j,s} \text{Loss}\,(j, s) = \arg \max_{j,s} \text{Gain}\,(j, s)$$

- where Loss (j, s) is the loss of splitting the current node on the j predictor at split point s.

1.6 Stopping and Pruning

- Tree Size
  - A large tree (many leaf nodes with few observation) can overfit
  - A small tree can not capture important structure
  - Tree size is a tuning parameter governing the mode's complexity, and the optimal tree size should be adaptively chosen form the data
- Early Stopping
  - Stop splitting when loss function has insignificant improvement (like forward stepwise)
  - However, a seemingly worthless initial split can lead to a good split further down the tree (short-sighted)
- Pruning
  - Grow a full tree (minimum node size) and prune back (like backwards stepwise)

1.6.1 Stopping and Pruning

- Let N_m be the number of observations in node R_m and $Q_m(T)$ be the loss in region *m* for a given tree *T*. e.g., using sum of squared error loss,

$$Q_m(T) = \sum_{\{i:x_i \in R_m\}} (y_i - \hat{c}_m)^2$$

- *Weakest link pruning*: Successively collapse the internal node that produces the smallest per-node increase in $\sum_{m=1}^{|\mathrm{T}|} Q_m(T)$ until you reach the single node (root) tree. This produces a finite sequence of sub-trees.
- For each sub-tree *T*, define its cost complexity

$$C_\lambda(T) = \frac{1}{n}\sum_{m=1}^{|T|} Q_m(T) + \lambda|T| = \mathrm{Loss}(T) + \lambda\ \mathrm{Penealty(T)}$$

where the *m* covers all terminal nodes in *T* and λ is a penalty on tree size $|T|$.

- Note: The complexity of a tree in this setting is measured by the *number of leaf nodes*, $|T|$

1.6.2 Penalty Tuning

- For each λ, there is a unique smallest sub-tree T_λ that minimizes $C_\lambda(T)$.
- The sequence of sub-trees from weakest link pruning contains every T_λ.
- The tuning parameter, λ can be chosen by: cross-validation, AIC/BIC, Out-of-Bag (OOB), etc.

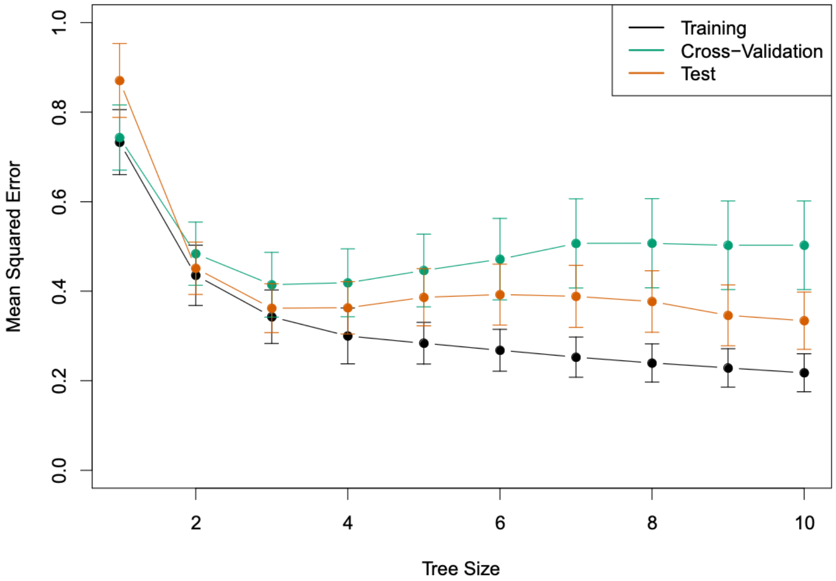


FIGURE 8.5. *Regression tree analysis for the* Hitters *data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.*

1.7 Special Considerations

1.7.1 Missing Predictor Values

- Omit observations with missing values
- For categorical predictors, create an additional level "missing"
  - This can reveal important patterns if *missing* is not at random.
- Surrogate splits
  - At every split, create a list of *surrogate splits* that mimics the original splits
    - For prediction, if an observation has a missing value, use the surrogate splits to determine which child group it should go into
- Alternatively, before splitting, we could impute (replace missing values with an estimate)
  - The surrogate approach is similar to imputation, but is based on the "nearest neighbors"; the observations in the same branch of the tree.

1.7.2 Binary Splitting

- Multiway splits are possible, but could partition the data too quickly (overfit)
- Multiway splits can be achieved from a combination of binary splits
  - i.e., split on X_1 at s_1 and then split again on X_1 at s_2
  - This will/should happen when the true response is not a constant.

1.7.3 Variable/Feature Importance

There are several ways to measure the *importance* of a feature in a tree. Here are only a few:

1. The number of times the feature was used to make a split

$$\mathcal{J}_j(T) = \sum_t \mathbb{1}\,(\text{split } t \text{ uses feature } j)$$

   The sum is over all splits *t* in tree *T*.

2. The total reduction in loss (or increase in gain) for all the splits made by the feature

   - This is a weighted version of number of times feature used to make a split
   - In CART, the features used to make the surrogate splits are also included

The importance of predictor *j* in a single tree *T*:

$$\mathcal{J}_j(T) = \sum_t \text{gain}(t) \cdot \mathbb{1}\,(\text{split } t \text{ uses feature } j)$$

That is, the importance of feature *j* in tree *T* is the total *gain* from all splits involving feature *j*. In the equation, the sum is over all splits *t* in tree *T*.

3. Permutation based importance: re-run the tree, but include a *permuted* version of the feature. Any decrease in performance indicates the feature was important.
   - Note: this is not perfect; consider what happens to features that are highly correlated/associated with other features.

$$\mathcal{J}_j(T) = \text{Loss(Tree with permuted column } j) - \text{Loss(Tree)}$$

We will explore other ways to assess variable importance later in the course.

1.8 Tree Advantages

- Handles categorical and continuous data in consistent manner
- Automatic variable selection (any predictor not split)
- Automatically discover interactions between multiple predictors
  - The tree depth determines the possible number of interactions
- Because the partitions are made from the data, tree gives a *locally adaptive* estimate
- Invariant to monotone transformations (some caveats)
- Can be robust o outliers in the feature space. This is because trees split on sample quantiles rather than raw values.
- Easy to interpret

1.9 Tree Limitations

- Instability (high variance) due to greedy hierarchical structure; one change at top split can change remaining tree.
- Difficulty capturing additive structure

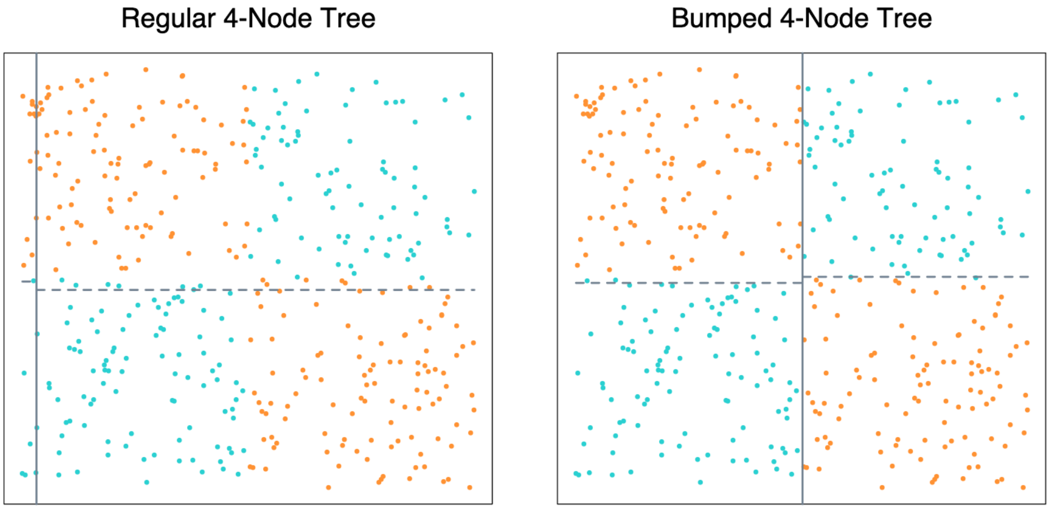


FIGURE 8.13. *Data with two features and two classes (blue and orange), displaying a pure interaction. The left panel shows the partition found by three splits of a standard, greedy, tree-growing algorithm. The vertical grey line near the left edge is the first split, and the broken lines are the two subsequent splits. The algorithm has no idea where to make a good initial split, and makes a poor choice. The right panel shows the near-optimal splits found by bumping the tree-growing algorithm* 20 *times.*