INFO 6105
Data Science Engineering Methods and Tools

Lecture 7
Regression Trees

Ebrahim Nasrabadi
nasrabadi@northeastern.edu

College of Engineering
Northeastern University

Fall 2019

Tree-Based Methods

- Decision trees can be applied to both regression and classification problems.

Tree-Based Methods

- Decision trees can be applied to both regression and classification problems.
- The basic idea is to *stratify* or *segment* the predictor space (e.g., the set of all possible values for the features) into a number of simple regions.

Tree-Based Methods

- Decision trees can be applied to both regression and classification problems.
- The basic idea is to *stratify* or *segment* the predictor space (e.g., the set of all possible values for the features) into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision-tree methods.

Tree-Based Methods

- Decision trees can be applied to both regression and classification problems.
- The basic idea is to *stratify* or *segment* the predictor space (e.g., the set of all possible values for the features) into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision-tree methods.
- We first consider regression trees to predict a quantitative variable.

Hitters Dataset

We study the Hitters data set to describe tree-based models.

Hitters Dataset: A data set including 322 observations of major league players from the 1986 and 1987 seasons on the following 20 variables:

- AtBat Number of times at bat in 1986
- Hits Number of hits in 1986
- HmRun Number of home runs in 1986
- Runs Number of runs in 1986
- RBI Number of runs batted in in 1986
- Walks Number of walks in 1986
- Years Number of years in the major leagues
- AtBat Number of times at bat during his career
- Hits Number of hits during his career
- HmRun Number of home runs during his career
- Runs Number of runs during his career
- RBI Number of runs batted in during his career
- Walks Number of walks during his career
- League A factor with levels A and N indicating player's league at the end of 1986
- Division A factor with levels E and W indicating player's division at the end of 1986
- PutOuts Number of put outs in 1986
- Assists Number of assists in 1986
- Errors Number of errors in 1986
- Salary 1987 annual salary on opening day in thousands of dollars
- NewLeague A factor with levels A and N indicating player's league at the beginning of 1987

Predicting Baseball Players' Salaries

Using Hitters dataset, we want to predict a baseball player's Salary (thousands of dollars) based on

- Years (the number of years that he has played in the major leagues)
- Hits (the number of hits that he made in the previous year).

	Years	Hits	Salary
-Andy Allanson	1	66	NA
-Alan Ashby	14	81	475.000
-Alvin Davis	3	130	480.000
-Andre Dawson	11	141	500.000
-Andres Galarraga	2	87	91.500
-Alfredo Griffin	11	169	750.000

Figure: Hitters Dataset

Preprocessing

Preprocessing

- Remove observations that have missing Salary values

Preprocessing

- Remove observations that have missing Salary values
- Log-transform Salary so that its distribution has more of a typical bell-shape.

Preprocessing

- Remove observations that have missing Salary values
- Log-transform Salary so that its distribution has more of a typical bell-shape.

Preprocessing

- Remove observations that have missing Salary values
- Log-transform Salary so that its distribution has more of a typical bell-shape.

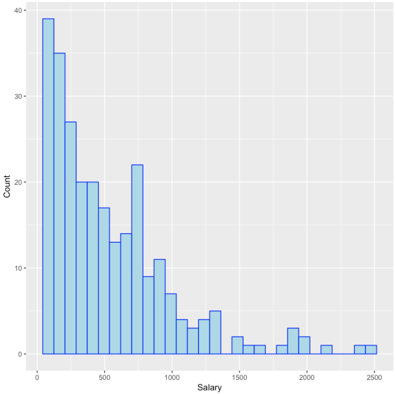


Figure: Salary Histogram

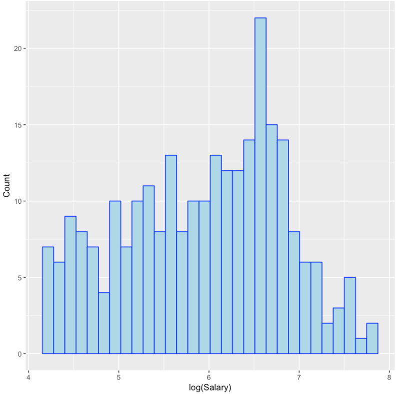
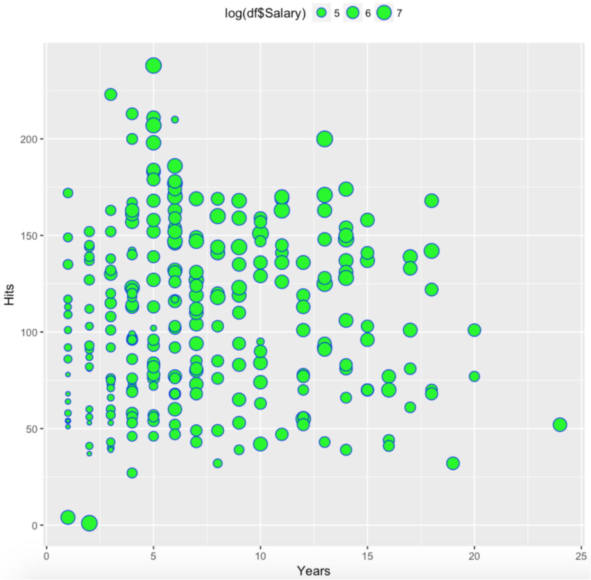
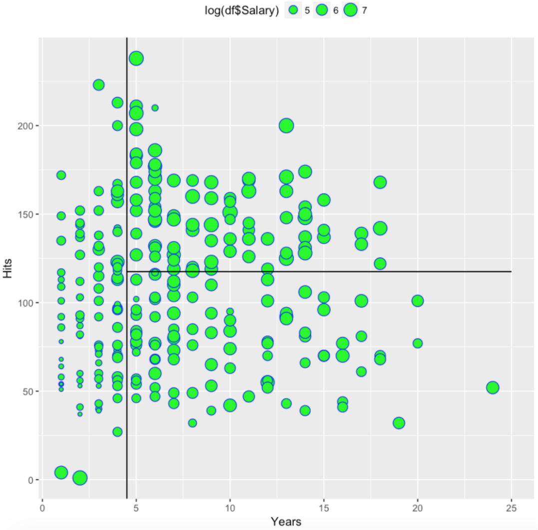


Figure: Log(Salary) Histogram

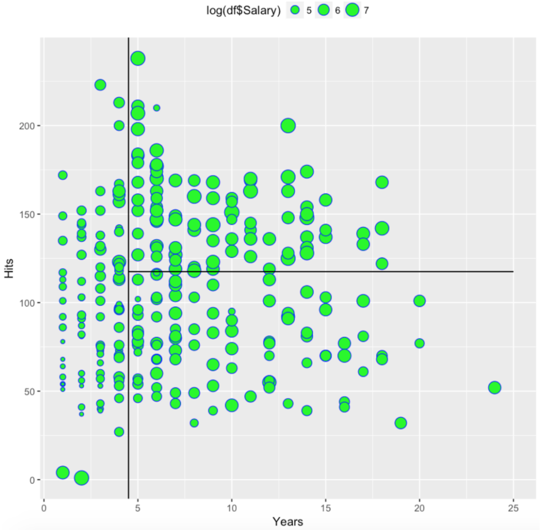
Baseball Players' Salaries

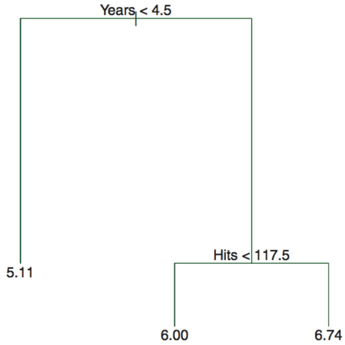


Regression tree for the Hitters dataset



Regression tree for the Hitters dataset





Some obervations

- The first split assigns players having Years < 4.5 to the left branch.

Some obervations

- The first split assigns players having Years < 4.5 to the left branch.
- At a given internal node, the label of the form $X_j < t_k$ indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$.

Some obervations

- The first split assigns players having Years < 4.5 to the left branch.
- At a given internal node, the label of the form $X_j < t_k$ indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$.
- For instance, the split at the top of the tree results in two large branches.

Some obervations

- The first split assigns players having Years < 4.5 to the left branch.
- At a given internal node, the label of the form $X_j < t_k$ indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$.
- For instance, the split at the top of the tree results in two large branches.
  - The left-hand branch corresponds to Years < 4.5, and

Some obervations

- The first split assigns players having Years < 4.5 to the left branch.
- At a given internal node, the label of the form $X_j < t_k$ indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$.
- For instance, the split at the top of the tree results in two large branches.
  - The left-hand branch corresponds to Years < 4.5, and
  - the right-hand branch corresponds to Years ≥ 4.5.

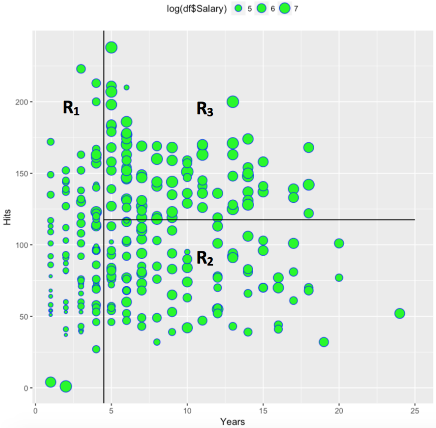
Some obervations

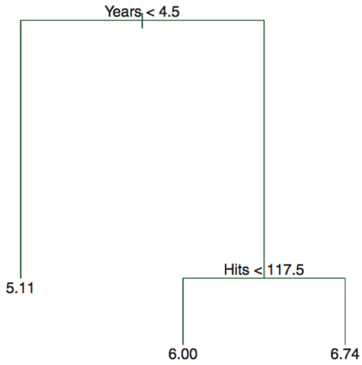
- The first split assigns players having Years < 4.5 to the left branch.
- At a given internal node, the label of the form $X_j < t_k$ indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$.
- For instance, the split at the top of the tree results in two large branches.
  - The left-hand branch corresponds to Years < 4.5, and
  - the right-hand branch corresponds to Years ≥ 4.5.
- The tree has two internal nodes and three terminal nodes, or leaves.

Some obervations

- The first split assigns players having Years < 4.5 to the left branch.
- At a given internal node, the label of the form $X_j < t_k$ indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$.
- For instance, the split at the top of the tree results in two large branches.
  - The left-hand branch corresponds to Years < 4.5, and
  - the right-hand branch corresponds to Years ≥ 4.5.
- The tree has two internal nodes and three terminal nodes, or leaves.
- The number in each leaf is the mean of the response for the observations that fall there.

Regression Tree for the Hitters dataset





The tree segments the players into three regions of predictor space:

$$R_1 = \{X|\text{Years} < 4.5\},$$
$$R_2 = \{X|\text{Years} \geq 4.5, \text{Hits} < 117.5\},$$
$$R_3 = \{X|\text{Years} \geq 4.5, \text{Hits} \geq 117.5\}.$$

Terminology for Trees

- In keeping with the tree analogy, the regions R_1, R_2, and R_3 are known as terminal nodes
- Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as internal nodes
- In the hitters tree, the two internal nodes are indicated by the text Years < 4.5 and Hits < 117.5.
- We refer to the segments of the trees that connect the nodes as branches.

Interpretation of Results

- Years is the most important factor in determining Salary, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of Hits that he made in the previous year seems to play little role in his Salary.
- But among players who have been in the major leagues for five or more years, the number of Hits made in the previous year does affect Salary, and players who made more Hits last year tend to have higher salaries.
- The regression tree is an over-simplification of the true relationship between Hits, Years, and Salary, but it is easy to display, interpret and explain

Making Predictions

- We predict the response for a new observation using the mean of the training observations in the region to which the new observation belongs.

Making Predictions

- We predict the response for a new observation using the mean of the training observations in the region to which the new observation belongs.
- For the players in the data set with
  - Years< 4.5 (i.e., in region R_1), the mean log salary is 5.107, and so we make a prediction of $e^{5.107}$ thousands of dollars, i.e., $165,174, for these players.

Making Predictions

- We predict the response for a new observation using the mean of the training observations in the region to which the new observation belongs.
- For the players in the data set with
  - Years< 4.5 (i.e., in region R_1), the mean log salary is 5.107, and so we make a prediction of $e^{5.107}$ thousands of dollars, i.e., $165,174, for these players.
  - Years≥ 4.5 and Hits < 117.5 (in region R_2), the mean log salary is 6.740, and so we make a prediction of $e^{5.999}$ thousands of dollars, i.e., $402,834, for these players.

Making Predictions

- We predict the response for a new observation using the mean of the training observations in the region to which the new observation belongs.
- For the players in the data set with
  - Years< 4.5 (i.e., in region R_1), the mean log salary is 5.107, and so we make a prediction of $e^{5.107}$ thousands of dollars, i.e., \$165,174, for these players.
  - Years≥ 4.5 and Hits < 117.5 (in region R_2), the mean log salary is 6.740, and so we make a prediction of $e^{5.999}$ thousands of dollars, i.e., \$402,834, for these players.
  - with Years≥ 4.5 and Hits ≥ 117.5 (in region R_3), the mean log salary is 6.740, and so we make a prediction of $e^{6.740}$ thousands of dollars, i.e., \$845,345, for these players.

Regression trees

Suppose that we are given n training observations

$$(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_n, y_n)$$

where $\vec{x}_i \in \mathbb{R}^m$ and $y_i \in \mathbb{R}$.

Regression trees

Suppose that we are given n training observations

$$(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_n, y_n)$$

where $\vec{x}_i \in \mathbb{R}^m$ and $y_i \in \mathbb{R}$.

There are two steps for building a regression tree:

1. Fitting a tree: We divide the predictor space–that is, the set of possible values for predictors into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$:

$$R_1 \cup R_2 \cup \ldots \cup R_J = \text{feature space}$$
$$R_i \cap R_j = \emptyset \qquad \forall i \neq j$$

2. Making Predictions: For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j

Building a regression tree

Question How do we construct the regions $R_1, R_2, \ldots, R_J$?

Building a regression tree

Question How do we construct the regions $R_1, R_2, \ldots, R_J$?

- In theory, the regions could have any shape.

Building a regression tree

Question How do we construct the regions $R_1, R_2, \ldots, R_J$?

- In theory, the regions could have any shape.
- However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.

Building a regression tree

Question How do we construct the regions $R_1, R_2, \ldots, R_J$?

- In theory, the regions could have any shape.
- However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.
- **Goal:** Find regions $R_1, R_2, \ldots, R_J$ that minimize RSS:

$$\text{RSS} = \sum_{j=1}^{J} \sum_{i: x_i \in R_j} (y_i - \hat{y}_{R_j})^2$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

Top-down, greedy approach

Challenge: It is computationally infeasible to consider every possible partition of the feature space into J boxes.

Top-down, greedy approach

Challenge: It is computationally infeasible to consider every possible partition of the feature space into J boxes.

Solution: W take a *top-down*, *greedy* approach (known as *recursive binary splitting*).

- The approach is top-down because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.

Top-down, greedy approach

Challenge: It is computationally infeasible to consider every possible partition of the feature space into J boxes.

Solution: W take a *top-down, greedy* approach (known as *recursive binary splitting*).

- The approach is top-down because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Top-down, greedy approach

Challenge: It is computationally infeasible to consider every possible partition of the feature space into J boxes.

Solution: W take a *top-down, greedy* approach (known as *recursive binary splitting*).

- The approach is top-down because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.
- The algorithm needs to decide on the splitting variables and split points at each iteration.

Algorithm

- Root Node: Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j,s) = \{\vec{x} \in \mathbb{R}^m | x_j < s\}$$
$$R_2(j,s) = \{\vec{x} \in \mathbb{R}^m | x_j \geq s\}$$

- Root Node: Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j,s) = \{\vec{x} \in \mathbb{R}^m | x_j < s\}$$
$$R_2(j,s) = \{\vec{x} \in \mathbb{R}^m | x_j \geq s\}$$

- We seek the splitting variable j and split point s that solve

$$\min_{j,s} \left(\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2 \right)$$

where $\hat{y}_{R_1}$ is the mean response for the training observations in $R_1(j,s)$, and $\hat{y}_{R_2}$ is the mean response for the training observations in $R_2(j,s)$.

How does it work?

Question: How to find j and s and what is the complexity?

How does it work?

Question: How to find j and s and what is the complexity?

- Consider splitting on variable j.

How does it work?

Question: How to find j and s and what is the complexity?

- Consider splitting on variable j.
- If $x_{j(1)}, x_{j(2)}, \ldots, x_{j(n)}$ are the sorted values of the jth feature,

How does it work?

Question: How to find j and s and what is the complexity?

- Consider splitting on variable j.
- If $x_{j(1)}, x_{j(2)}, \ldots, x_{j(n)}$ are the sorted values of the jth feature,
  - we only need to check split points between adjacent values

How does it work?

Question: How to find j and s and what is the complexity?

- Consider splitting on variable j.
- If $x_{j(1)}, x_{j(2)}, \ldots, x_{j(n)}$ are the sorted values of the jth feature,
  - we only need to check split points between adjacent values
  - traditionally take split points between adjacent values:

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

How does it work?

Question: How to find j and s and what is the complexity?

- Consider splitting on variable j.
- If $x_{j(1)}, x_{j(2)}, \ldots, x_{j(n)}$ are the sorted values of the jth feature,
  - we only need to check split points between adjacent values
  - traditionally take split points between adjacent values:

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

  - Only need to check $(n-1)$ split points.

How does it work?

Question: How to find j and s and what is the complexity?

- Consider splitting on variable j.
- If $x_{j(1)}, x_{j(2)}, \ldots, x_{j(n)}$ are the sorted values of the jth feature,
  - we only need to check split points between adjacent values
  - traditionally take split points between adjacent values:

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

  - Only need to check $(n-1)$ split points.
- Finding the values of j and s that minimize RSS can be done quite quickly, especially when the number of features m is not too large.

How does it work?

Question: How to find j and s and what is the complexity?

- Consider splitting on variable j.
- If $x_{j(1)}, x_{j(2)}, \ldots, x_{j(n)}$ are the sorted values of the jth feature,
  - we only need to check split points between adjacent values
  - traditionally take split points between adjacent values:

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

  - Only need to check $(n-1)$ split points.
- Finding the values of j and s that minimize RSS can be done quite quickly, especially when the number of features m is not too large.

How does it work?

Question: How to find j and s and what is the complexity?

- Consider splitting on variable j.
- If $x_{j(1)}, x_{j(2)}, \ldots, x_{j(n)}$ are the sorted values of the jth feature,
  - we only need to check split points between adjacent values
  - traditionally take split points between adjacent values:

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

  - Only need to check $(n-1)$ split points.
- Finding the values of j and s that minimize RSS can be done quite quickly, especially when the number of features m is not too large.

Using the above process, we can determine R_1 and R_2

Algorithm (Cont.)

- Next, we repeat the process, looking for the best predictor and best cut-point in order to split the data further so as to minimize the RSS within each of the resulting regions.

Algorithm (Cont.)

- Next, we repeat the process, looking for the best predictor and best cut-point in order to split the data further so as to minimize the RSS within each of the resulting regions.
- Stopping Criterion: The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Algorithm (Cont.)

- Next, we repeat the process, looking for the best predictor and best cut-point in order to split the data further so as to minimize the RSS within each of the resulting regions.
- Stopping Criterion: The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.
- Making predictions: Once the regions $R_1, \dots, R_J$ have been created, we predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

Complexity Control Strategy

Question: How large should we grow the tree?

Complexity Control Strategy

Question: How large should we grow the tree?

- If the tree is too big, we may overfit the data (e.g., the tree may produce good predictions on the training observations, but poor performance on test data)

Complexity Control Strategy

Question: How large should we grow the tree?

- If the tree is too big, we may overfit the data (e.g., the tree may produce good predictions on the training observations, but poor performance on test data)
- If the tree is too small, we may not capture the important structure and might miss patterns in the data (under-fit the data)

Regression trees in R

```
library(ISLR)
library(tree)

?Hitters

Hitters = subset(Hitters, !(is.na(Hitters$Salary)))

n_row <- nrow(Hitters)
train = sample(1:n_row, n_row*0.8)
df_train = Hitters[train,]
df_test = Hitters[-train,]
```

Regression trees in R

```
> treefit
node), split, n, deviance, yval
      * denotes terminal node

 1) root 210 176.700 5.922
   2) Years < 4.5 74  39.350 5.101
     4) Years < 3.5 50  20.240 4.858
       8) Hits < 114 37  16.380 4.717
        16) Hits < 40.5 5  10.400 5.511 *
        17) Hits > 40.5 32   2.345 4.593 *
       9) Hits > 114 13   1.008 5.260 *
     5) Years > 3.5 24  10.000 5.608
      10) Hits < 106 11   1.685 5.271 *
      11) Hits > 106 13   6.016 5.892 *
   3) Years > 4.5 136  60.350 6.369
     6) Hits < 117.5 67  23.130 5.997
      12) Years < 6.5 22   6.607 5.676 *
      13) Years > 6.5 45  13.150 6.154
        26) Hits < 45.5 7   1.920 5.663 *
        27) Hits > 45.5 38   9.237 6.244 *
     7) Hits > 117.5 69  18.970 6.730 *
```

Note: Here deviance is just mean squared error.

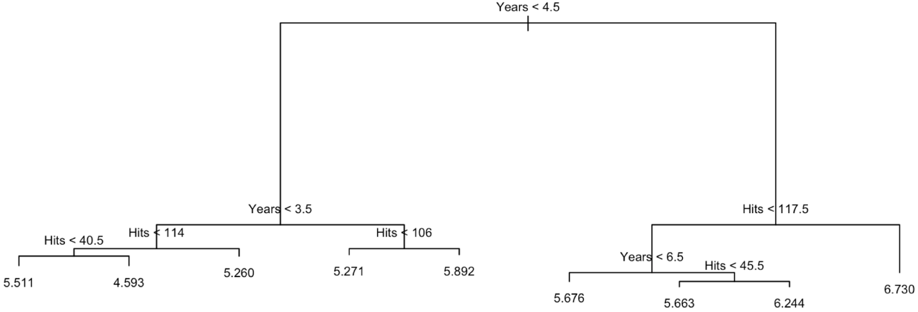
Regression trees in R

```
> summary(treefit)

Regression tree:
tree(formula = log(Salary) ~ Years + Hits, data = df_train)
Number of terminal nodes:  9
Residual mean deviance:  0.2895 = 58.18 / 201
Distribution of residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-2.230000 -0.275200 -0.008827  0.000000  0.346300  2.152000
```

Regression trees in R

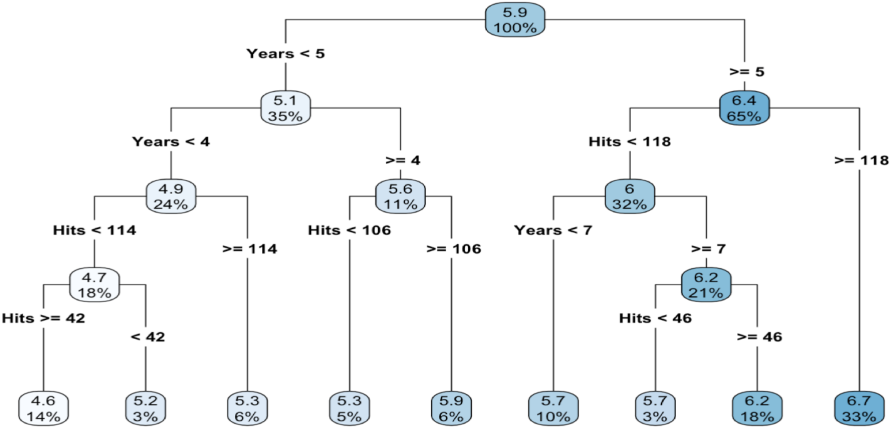
```
> summary(treefit)

Regression tree:
tree(formula = log(Salary) ~ Years + Hits, data = df_train)
Number of terminal nodes:  9
Residual mean deviance:  0.2895 = 58.18 / 201
Distribution of residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-2.230000 -0.275200 -0.008827  0.000000  0.346300  2.152000
```

```
> plot(treefit)
> text(treefit,cex=0.75)
```



Regression trees in R using rpart

```
library(rpart)
library(rpart.plot)
treefit_rpart <- rpart(log(Salary) ~ Years + Hits, data=df_train)
rpart.plot(treefit_rpart, type = 4)
```



Regression trees in R using rpart

```
> y_train_pred = predict(treefit_rpart, data=df_train)
> y_test_pred = predict(treefit_rpart, newdata=df_test)
>
> R2_train <- 1 - (sum((log(df_train$Salary) - y_train_pred )^2)/
+                    sum((log(df_train$Salary) - mean(log(df_train$Salary)))^2))
> R2_test <- 1 - (sum((log(df_test$Salary) - y_test_pred )^2)/
+                   sum((log(df_test$Salary) - mean(log(df_test$Salary)))^2))
> R2_train
[1] 0.6661017
> R2_test
[1] 0.5313479
>
> RMSE_train  = sqrt(mean((log(df_train$Salary) - y_train_pred)^2))
> RMSE_test  = sqrt(mean((log(df_test$Salary) - y_test_pred)^2))
> RMSE_train
[1] 0.5096182
> RMSE_test
[1] 0.6128059
```

How to control overfitting?

Observation:

- A large tree may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance.

How to control overfitting?

Observation:

- A large tree may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance.
- A smaller tree with fewer splits might lead to lower variance and better interpretation at the cost of a little bias.

How to control overfitting?

Observation:

- A large tree may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance.
- A smaller tree with fewer splits might lead to lower variance and better interpretation at the cost of a little bias.

How to control overfitting?

Observation:

- A large tree may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance.
- A smaller tree with fewer splits might lead to lower variance and better interpretation at the cost of a little bias.

Question: How can we control overfitting?

- **Idea 1:** Find the optimal subtree by cross validation.
  Not practical since there are too many possibilities!

How to control overfitting?

Observation:

- A large tree may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance.
- A smaller tree with fewer splits might lead to lower variance and better interpretation at the cost of a little bias.

Question: How can we control overfitting?

- **Idea 1:** Find the optimal subtree by cross validation.
  Not practical since there are too many possibilities!
- **Idea 2:** Stop growing the tree when the decrease in the RSS exceeds some (high) threshold.
  Short-sighted since It is possible to find good splits after bad ones.

Observation:

- A large tree may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance.
- A smaller tree with fewer splits might lead to lower variance and better interpretation at the cost of a little bias.

Question: How can we control overfitting?

- **Idea 1:** Find the optimal subtree by cross validation.
  Not practical since there are too many possibilities!
- **Idea 2:** Stop growing the tree when the decrease in the RSS exceeds some (high) threshold.
  Short-sighted since It is possible to find good splits after bad ones.
- **Idea 3:** Grow a very large tree T_0, stopping the splitting process only when some minimum node size (say 5) is reached, and then prune it back in order to obtain a subtree.

Cost-complexity pruning

- A subtree $T \subset T_0$ is any tree that can be obtained by pruning T_0, that is, collapsing any number of its internal nodes.

Cost-complexity pruning

- A subtree $T \subset T_0$ is any tree that can be obtained by pruning T_0, that is, collapsing any number of its internal nodes.
- For a subtree $T \subset T_0$, we define the cost-complexity criterion as

$$C_\alpha(T) = \mathrm{RSS}(T) + \alpha|T|$$

where $|T|$ indicates the number of terminal nodes of the tree T.

Cost-complexity pruning

- A subtree $T \subset T_0$ is any tree that can be obtained by pruning T_0, that is, collapsing any number of its internal nodes.
- For a subtree $T \subset T_0$, we define the cost-complexity criterion as

$$C_\alpha(T) = \mathrm{RSS}(T) + \alpha|T|$$

  where $|T|$ indicates the number of terminal nodes of the tree T.
- Recall that

$$\mathrm{RSS}(T) = \sum_{j=1}^{|T|} \sum_{i:x_i \in R_j} (y_i - \hat{y}_{R_j})^2$$

  where

Cost-complexity pruning

- A subtree $T \subset T_0$ is any tree that can be obtained by pruning T_0, that is, collapsing any number of its internal nodes.
- For a subtree $T \subset T_0$, we define the cost-complexity criterion as

$$C_\alpha(T) = \mathrm{RSS}(T) + \alpha|T|$$

  where $|T|$ indicates the number of terminal nodes of the tree T.
- Recall that

$$\mathrm{RSS}(T) = \sum_{j=1}^{|T|} \sum_{i: x_i \in R_j} (y_i - \hat{y}_{R_j})^2$$

  where
  - $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

Cost-complexity pruning

- A subtree $T \subset T_0$ is any tree that can be obtained by pruning T_0, that is, collapsing any number of its internal nodes.
- For a subtree $T \subset T_0$, we define the cost-complexity criterion as

$$C_\alpha(T) = \text{RSS}(T) + \alpha|T|$$

  where $|T|$ indicates the number of terminal nodes of the tree T.
- Recall that

$$\text{RSS}(T) = \sum_{j=1}^{|T|} \sum_{i: x_i \in R_j} (y_i - \hat{y}_{R_j})^2$$

  where
  - $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.
  - R_t is the region corresponding to the tth terminal node and $\hat{y}_{R_j}$ is the predicted response associated with R_t.

Cost-complexity pruning

- The idea is to find, for each α, the subtree $T(\alpha) \subset T_0$ to minimize $C_\alpha(T)$.

Cost-complexity pruning

- The idea is to find, for each α, the subtree $T(\alpha) \subset T_0$ to minimize $C_\alpha(T)$.
- The tuning parameter $\alpha \geq 0$ controls a trade-off between the subtree's complexity and its fit to the data

Cost-complexity pruning

- The idea is to find, for each α, the subtree $T(\alpha) \subset T_0$ to minimize $C_\alpha(T)$.
- The tuning parameter $\alpha \geq 0$ controls a trade-off between the subtree's complexity and its fit to the data
- Large values of α result in smaller trees T_α, and conversely for smaller values of α.

Cost-complexity pruning

- The idea is to find, for each α, the subtree $T(\alpha) \subset T_0$ to minimize $C_\alpha(T)$.
- The tuning parameter $\alpha \geq 0$ controls a trade-off between the subtree's complexity and its fit to the data
- Large values of α result in smaller trees T_α, and conversely for smaller values of α.
  - For $\alpha = 0$, the solution is the full tree T_0.

Cost-complexity pruning

- The idea is to find, for each α, the subtree $T(\alpha) \subset T_0$ to minimize $C_\alpha(T)$.
- The tuning parameter $\alpha \geq 0$ controls a trade-off between the subtree's complexity and its fit to the data
- Large values of α result in smaller trees T_α, and conversely for smaller values of α.
  - For $\alpha = 0$, the solution is the full tree T_0.
- For each α one can show that there is a unique smallest subtree $T(\alpha)$ that minimizes $C_\alpha(T)$.

Cost-complexity pruning

Questions:

- For a given α, how to find a tree that minimizes $C_\alpha(T)$?

Cost-complexity pruning

Questions:

- For a given α, how to find a tree that minimizes $C_\alpha(T)$?
- How to choose α?

Weakest link pruning

- We start from the full tree T_0.

Weakest link pruning

- We start from the full tree T_0.
- Substitute a subtree with a leaf node to obtain T_1 by minimizing

$$\frac{\mathrm{RSS}(T_1) - \mathrm{RSS}(T_0)}{|T_0| - |T_1|}$$

Weakest link pruning

- We start from the full tree T_0.
- Substitute a subtree with a leaf node to obtain T_1 by minimizing

$$\frac{\text{RSS}(T_1) - \text{RSS}(T_0)}{|T_0| - |T_1|}$$

- Iterate this process to obtain a sequence of nested trees:
  $\mathrm{T}_0 \succ T_1 \succ \ldots \succ T_{\text{root}}$

  where T_{root} is the root tree.

Weakest link pruning

- We start from the full tree T_0.
- Substitute a subtree with a leaf node to obtain T_1 by minimizing

$$\frac{\text{RSS}(T_1) - \text{RSS}(T_0)}{|T_0| - |T_1|}$$

- Iterate this process to obtain a sequence of nested trees:
  $\mathrm{T}_0 \succ T_1 \succ \ldots \succ T_{\text{root}}$

  where T_{root} is the root tree.
- The solution for each α is among $T_0, T_1, \ldots, T_{\text{root}}$.

Cross-validation

1. Build a sequence of nested trees $T_0, T_1, \ldots, T_{\text{root}}$ for a range of values for α

Cross-validation

1. Build a sequence of nested trees $T_0, T_1, \ldots, T_{\text{root}}$ for a range of values for α
2. Split the training data into K-folds

Cross-validation

1. Build a sequence of nested trees $T_0, T_1, \ldots, T_{\text{root}}$ for a range of values for α
2. Split the training data into K-folds
3. For $k = 1, \ldots, K$, do:

Cross-validation

1. Build a sequence of nested trees $T_0, T_1, \ldots, T_{\text{root}}$ for a range of values for α
2. Split the training data into K-folds
3. For $k = 1, \ldots, K$, do:
   - For each tree T_i, use every fold except the kth to estimate the averages in each region

Cross-validation

1. Build a sequence of nested trees $T_0, T_1, \ldots, T_{\text{root}}$ for a range of values for α
2. Split the training data into K-folds
3. For $k = 1, \ldots, K$, do:
    - For each tree T_i, use every fold except the kth to estimate the averages in each region
    - For each tree T_i, calculate the RSS in the test fold

Cross-validation

1. Build a sequence of nested trees $T_0, T_1, \ldots, T_{\text{root}}$ for a range of values for α
2. Split the training data into K-folds
3. For $k = 1, \ldots, K$, do:
   - For each tree T_i, use every fold except the kth to estimate the averages in each region
   - For each tree T_i, calculate the RSS in the test fold
4. For each tree T_i, average the K test errors and select the tree that minimizes the error.

Cross-validation

1. Build a sequence of nested trees $T_0, T_1, \ldots, T_{\text{root}}$ for a range of values for α
2. Split the training data into K-folds
3. For $k = 1, \ldots, K$, do:
    - For each tree T_i, use every fold except the kth to estimate the averages in each region
    - For each tree T_i, calculate the RSS in the test fold
4. For each tree T_i, average the K test errors and select the tree that minimizes the error.

Cross-validation

1. Build a sequence of nested trees $T_0, T_1, \ldots, T_{\text{root}}$ for a range of values for α
2. Split the training data into K-folds
3. For $k = 1, \ldots, K$, do:
   - For each tree T_i, use every fold except the kth to estimate the averages in each region
   - For each tree T_i, calculate the RSS in the test fold
4. For each tree T_i, average the K test errors and select the tree that minimizes the error.

WRONG WAY!

Cross-validation, the Right Way

1. Split the training data into K-folds

Cross-validation, the Right Way

1. Split the training data into K-folds
2. For $k = 1, \ldots, K$, do:

Cross-validation, the Right Way

1. Split the training data into K-folds
2. For $k = 1, \ldots, K$, do:
   - Using all data except the k-th fold, build a sequence of nested trees $T_0, T_1, \ldots, T_{\text{root}}$ for a range of values for α and calculate the average for each region in each one.

Cross-validation, the Right Way

1. Split the training data into K-folds
2. For $k = 1, \ldots, K$, do:
   - Using all data except the k-th fold, build a sequence of nested trees $T_0, T_1, \ldots, T_{\text{root}}$ for a range of values for α and calculate the average for each region in each one.
   - Select the parameter that minimizes the average test error

Cross-validation, the Right Way

1. Split the training data into K-folds
2. For $k = 1, \ldots, K$, do:
    - Using all data except the k-th fold, build a sequence of nested trees $T_0, T_1, \ldots, T_{\text{root}}$ for a range of values for α and calculate the average for each region in each one.
    - Select the parameter that minimizes the average test error

Cross-validation, the Right Way

1. Split the training data into K-folds
2. For $k = 1, \ldots, K$, do:
   - Using all data except the k-th fold, build a sequence of nested trees $T_0, T_1, \ldots, T_{\text{root}}$ for a range of values for α and calculate the average for each region in each one.
   - Select the parameter that minimizes the average test error

Note: In each iteration, only the training data (every fold except the k-th) is used to grow the large tree and then prune it back.