Lecture 13

Regression and classification trees

Julian Reif
Fall 2025

RStudio setup for this lecture

- Log into RStudio on your Amazon EC2 instance
  - Use AMI `FIN550-RStudio` with IAM role `BigDataEC2Role`

```
# Enter this command via RStudio Terminal
aws s3 cp --recursive s3://bigdata-fin550-reif/lecture-13 ~/fin550/lecture-13
```

Decision tree theory

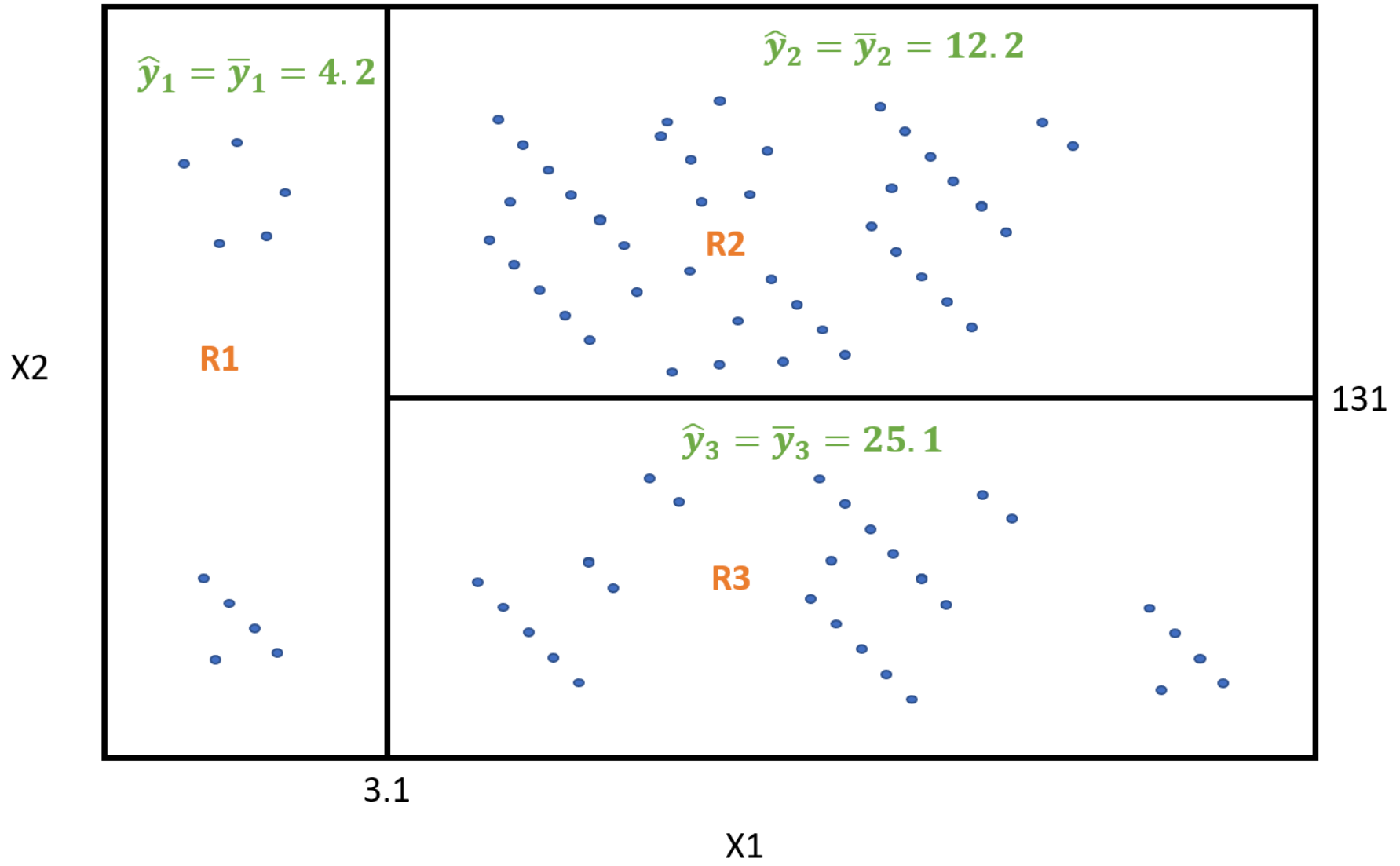
Decision trees

- Trees can be used for regression (continuous outcome) or classification analysis

- Building a tree consists of two main steps:
  1. Divide the predictor space up into different regions
  2. For every observation in a region, the prediction is the mean/mode of that region
     - Regression tree: mean
     - Classification tree: mode

- I will cover regression trees first

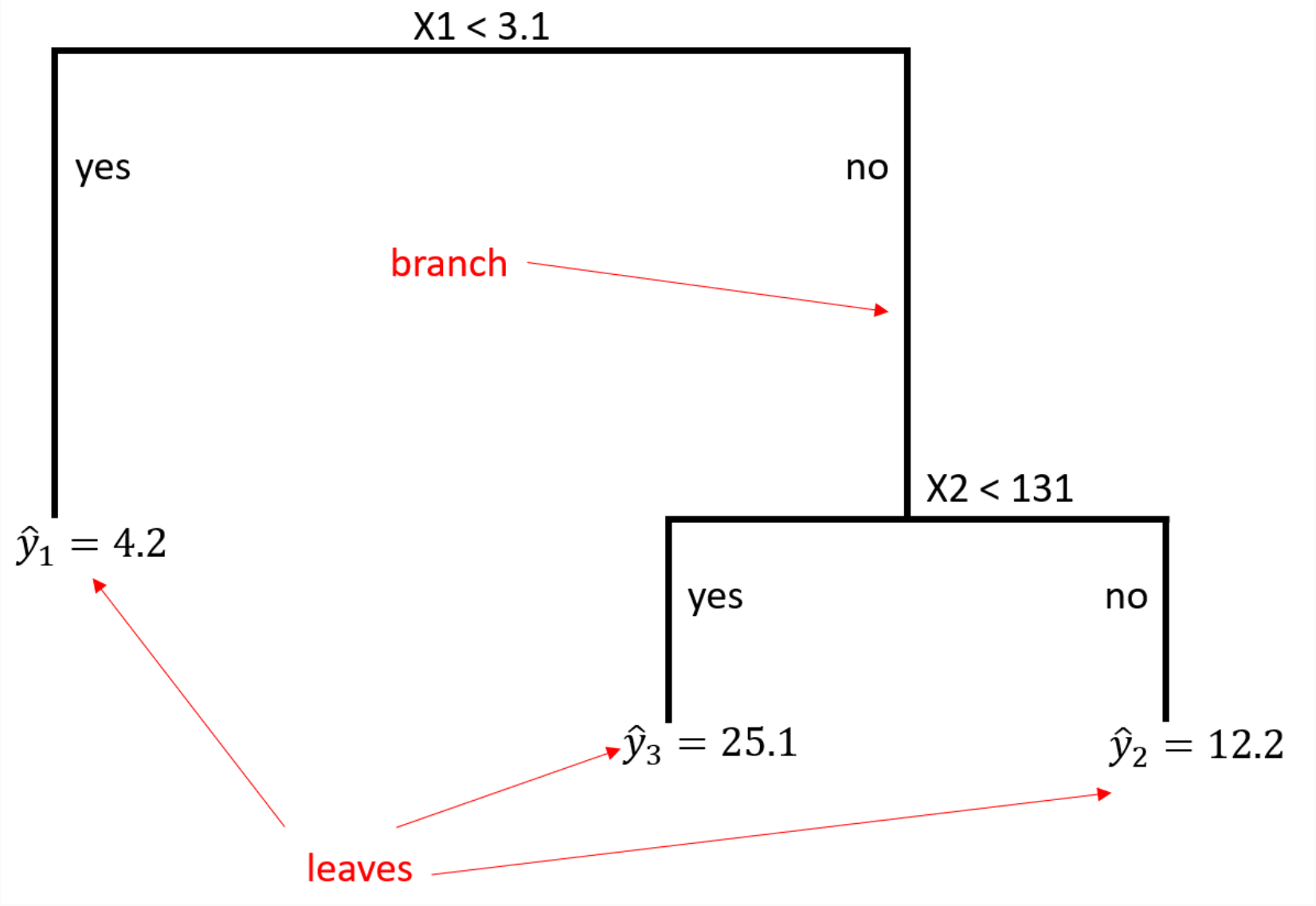
Example: dividing predictor space into three regions



The division is often represented in tree form

- Each region is called a "leaf" or a "terminal node"
- Connections between the internal nodes are called "branches"
- Size of tree, $|T|$, is equal to number of leaves (= # splits + 1)

Example: regression tree with three leaves



Use recursive binary splitting to grow a tree

- For each branch, we split the tree based on a predictor, X_j, and a value, s

- This split will produce two distinct regions, $R_1(j, s)$ and $R_2(j, s)$
  - For example, if $j = 2$ and $s = 131$, then we split based on $X_2 < 131$:
    - $R_1(2, 131)$ includes observations where $X_2 < 131$
    - $R_2(2, 131)$ includes observations where $X_2 \geq 131$

- We choose the split by solving for the j and s that minimize RSS:

$$RSS = \sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R2})^2$$

- Keep splitting until you reach a stopping criterion (e.g., $n < 5$ in each region)

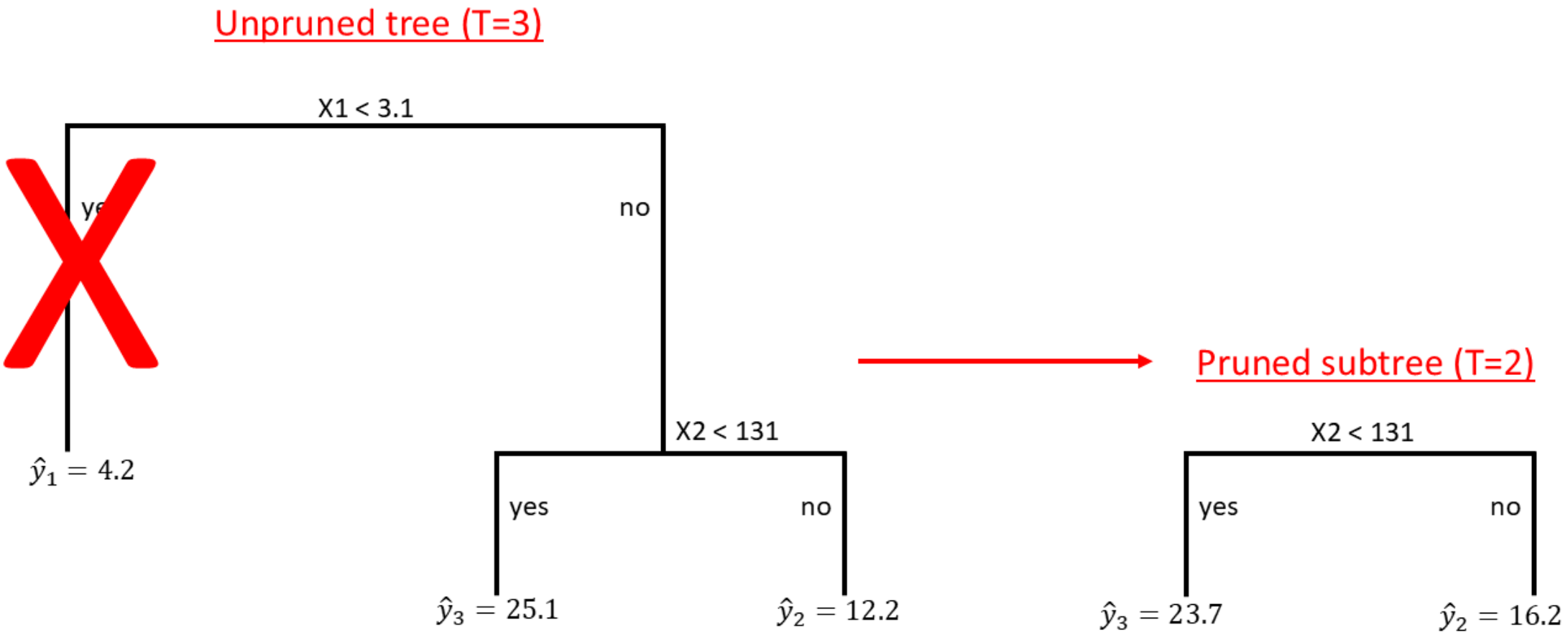
Pruning a tree

- Growing a tree by minimizing RSS usually leads to overfitting
  - Good fit to the training data
  - Poor fit to the test data
  - Note: linear regression has an analogous problem when there are too many variables

- Problem: some of the leaves are fitting to noise

- The solution is "tree pruning": dropping leaves with poor performance
  - Pruning produces a subtree

Pruning a tree with T=3 leaves



Pruning is accomplished by adding a penalty parameter

- Search through all possible subtrees, and find the one that minimizes:

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T| = RSS + \alpha|T|$$

where $|T|$ is the number of leaves in subtree T

- $\alpha \geq 0$ is a tuning parameter (aka "penalty" or "complexity" parameter)
  - When $\alpha = 0$, there is no pruning
  - As $\alpha \to \infty$, the number of leaves in the subtree goes to 1 (0 splits)
  - α is chosen using... cross-validation!

Algorithm: building a regression tree

1. Grow a tree using recursive binary splitting

2. Consider a set of different α's: $\alpha_1, \alpha_2, \ldots \alpha_k$

3. For each possible α, prune the tree to find the optimal subtree

4. For each possible α, calculate cross-validated mean-squared error for optimal subtree
    - Grow and prune a tree, using training data only
    - Calculate MSE using test data

5. Select the subtree from step 3 that corresponds to the "best" α (smallest CV MSE)

Classification trees are used with categorical outcomes

- In a regression tree, the prediction for a leaf is the **mean** of the observations
- In a classification tree, the prediction is the **mode**
- Example where y has $K = 2$ categories and leaf has 3 obs: {yes, yes, no}
  - $\hat{y} =$ "yes" for these observations
- Example where y has $K = 3$ categories and leaf has 5 obs: {MSF, MSF, MSF, MSFE, MSBA}
  - $\hat{y} =$ "MSF" for these observations
- Note: requires a tie-breaking rule for cases like {yes, yes, no, no}

Classification trees uses node impurity instead of RSS

- We choose the split by solving for the j and s that minimizes "weighted Gini impurity":

$$\text{Impurity} = \frac{N_1(j,s)}{N} G_1(j,s) + \frac{N_2(j,s)}{N} G_2(j,s)$$

where $N = N_1(j,s) + N_2(j,s)$ is total number of observations in leaves 1 and 2

- The impurity of node i is given by:

$$G_i = \sum_{k=1}^{K} \hat{p}_{ik}\left(1 - \hat{p}_{ik}\right)$$

where $\hat{p}_{ik}$ is the fraction of observations in leaf i belonging to class k

What is node impurity?

- Node impurity is a measure of classification error:

$$G_i = \sum_{k=1}^{K} \hat{p}_{ik}\left(1 - \hat{p}_{ik}\right)$$

- Consider $K = 2$ classes (yes/no), and let observations in the leaf be {yes, yes}
  - What is value of G for that leaf?

What are the pros/cons of trees?

- Regression (and lasso) assume Y is a linear function of X

$$Y = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

- Regression trees, by contrast, allow for complicated nonlinearities:

$$Y = \sum_{m=1}^{M} c_m 1_{(X \in R_m)}$$

  where 1 is indicator function, R_m is a region, and c_m is the mean of that region

- However, in practice trees often perform worse than other methods
  - But, there are ways to improve predictive performance further (e.g., random forests)

Regression trees in R

Try it: load and inspect the Boston housing dataset

```
library(tidyverse)
library(ggplot2)

# Housing information for 506 areas around Boston
housing <- read_csv("lecture-13-housing.csv") %>%
  select(MEDV, CRIM, NOX, RM, TAX)
nrow(housing)
ncol(housing)

sum(!complete.cases(housing)) # Check for missing values
```

```
# [1] 506
# [1] 5
# [1] 0
```

Variables come from the 1970 Census

Variable name	Definition
MEDV	Median value of owner-occupied homes in $1000's
CRIM	Per capita crime rate by census tract
NOX	Nitric oxides concentration (parts per 10 million)
RM	Average number of rooms per dwelling
TAX	Full-value property-tax rate per $10,000

```
summary(housing$MEDV)
```

```
#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#    5.00   17.02   21.20   22.53   25.00   50.00
```

Build decision trees using the rpart package

- Use `rpart` and `rpart.plot` libraries

- Key functions:
  - `mytree <- rpart()`: build the tree
  - `mytree$where`: where (which leaf) each observation was assigned to
  - `mytree$variable.importance`: list of most important predictors
  - `cp <- printcp(mytree)`: object containing tuning parameter values and CV MSE's
  - `prune(mytree, cp = myalpha)`: prune a tree for a particular value of alpha
  - `predict(mytree, newdata = mydata)`: use tree to form predictions
  - `prp(mytree)`: plot the tree

Try it: build a regression tree

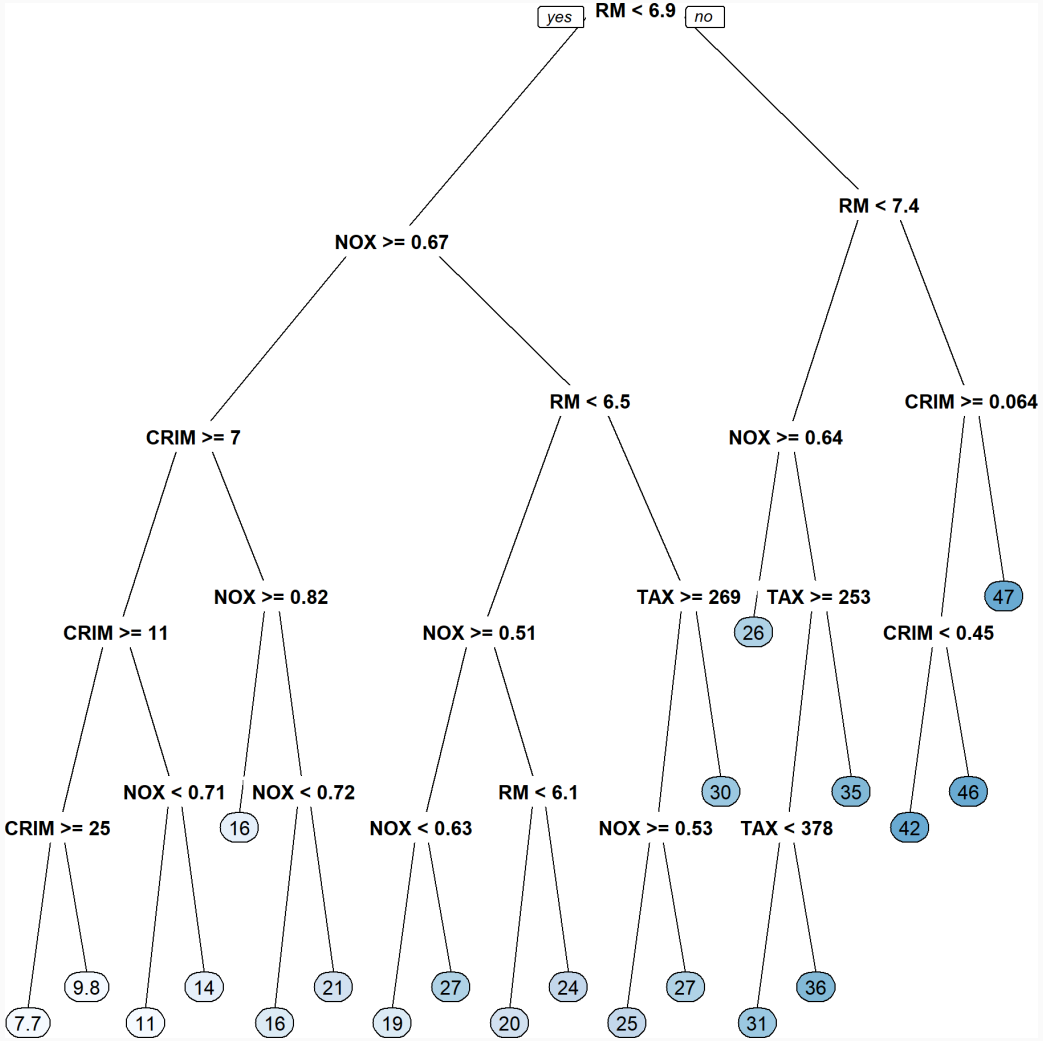
```r
library(rpart)
library(rpart.plot)

# method="anova" for regression trees
# Set cp (alpha) equal to 0 to grow a full tree
# (Optional) set maxdepth=5 to limit how deep the tree is
set.seed(1)
tree_housing <- rpart(MEDV ~ ., data = housing, cp = 0,
                      maxdepth = 5, method = "anova")

# Plot the full tree
prp(tree_housing, box.palette = "auto")
```

Plot the full tree (max depth = 5)

- What does tree look like if you omit `maxdepth=5` from `rpart()`?



Try it: number of leaves and variable importance

```
# Number of leaves/nodes
n_distinct(tree_housing$where) # `where` reports which leaf each obs was assigned to

# List variables that are most important for predicting outcomes
# Do these variables lie near the top or the bottom of the tree?
round(tree_housing$variable.importance,1)
```

```
# [1] 21
#      RM     NOX    CRIM     TAX
# 25867.4  7704.5  5017.5  2968.9
```

Try it: identify the vector of alpha's

```
cp <- printcp(tree_housing)
```

```
head(cp) # CP is alpha, xerror is CV MSE. What value is nsplit for the first CP? Why?
cat("\n")
as.vector(cp[,1]) # set of alpha's considered by `rpart`
```

```
#           CP nsplit rel error    xerror       xstd
# 1 0.45274420      0 1.0000000 1.0050753 0.08301497
# 2 0.11251782      1 0.5472558 0.5741582 0.05404496
# 3 0.07165784      2 0.4347380 0.4715515 0.06061093
# 4 0.05403671      3 0.3630801 0.4205866 0.05868427
# 5 0.02598598      4 0.3090434 0.3434092 0.05592466
# 6 0.01641021      5 0.2830575 0.3211350 0.05572784
#
#  [1] 0.4527442007 0.1125178160 0.0716578409 0.0540367050 0.0259859832
#  [6] 0.0164102099 0.0076138107 0.0073995566 0.0044752754 0.0042760000
# [11] 0.0028166156 0.0020539519 0.0019526403 0.0016295590 0.0012018315
# [16] 0.0006250947 0.0000000000
```

Try it: identify the min CV MSE and the optimal alpha

```
# Use cp object to identify minimum CV MSE
cvmse.min <-

# Use cp object to identify optimal alpha (hint: use `which.min()` to find the index)
index <-
alpha.best <-

print(paste("Min CV MSE is", round(cvmse.min,3)))
print(paste("Optimal alpha is", round(alpha.best,5)))
```

Identify the min CV MSE and the optimal alpha

```
# Use cp object to identify minimum CV MSE
cvmse.min <- min(cp[ , "xerror"])

# Use cp object to identify optimal alpha (hint: use `which.min()` to find the index)
index <- which.min(cp[ , "xerror"])
alpha.best <- cp[index,"CP"]

print(paste("Min CV MSE is", round(cvmse.min,3)))
print(paste("Optimal alpha is", round(alpha.best,5)))
```

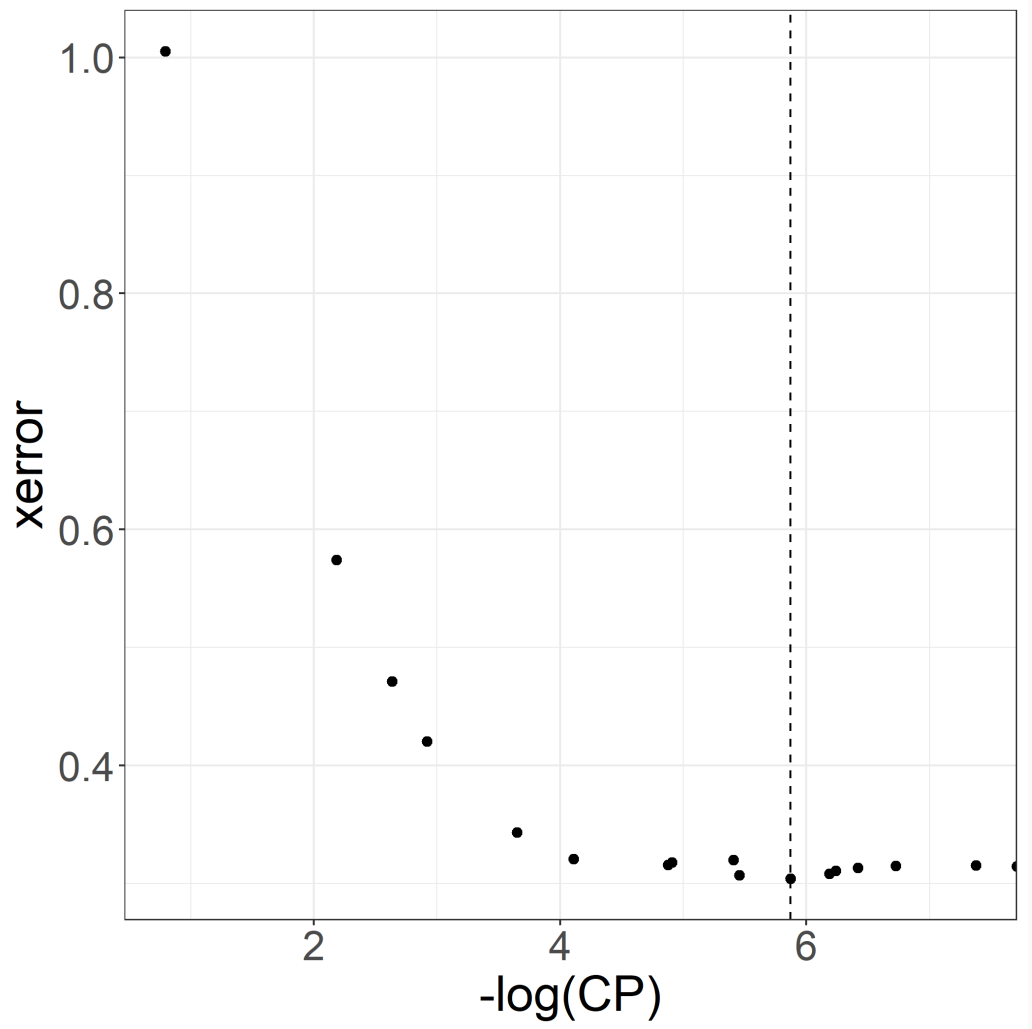
```
# [1] "Min CV MSE is 0.304"
# [1] "Optimal alpha is 0.00282"
```

Try it: plot CV MSE vs alpha

```
df <- as.data.frame(cp)
ggplot(df, aes(x=-log(CP), y=xerror)) +
  geom_point(size=2) +
  theme_bw() +
  theme(axis.title=element_text(size=24),
        axis.text=element_text(size=20))+
  geom_vline(xintercept =
               -log(alpha.best),
             linetype = "dashed")
```

```
-log(alpha.best)
```

```
# [1] 5.872219
```



Try it: prune the tree

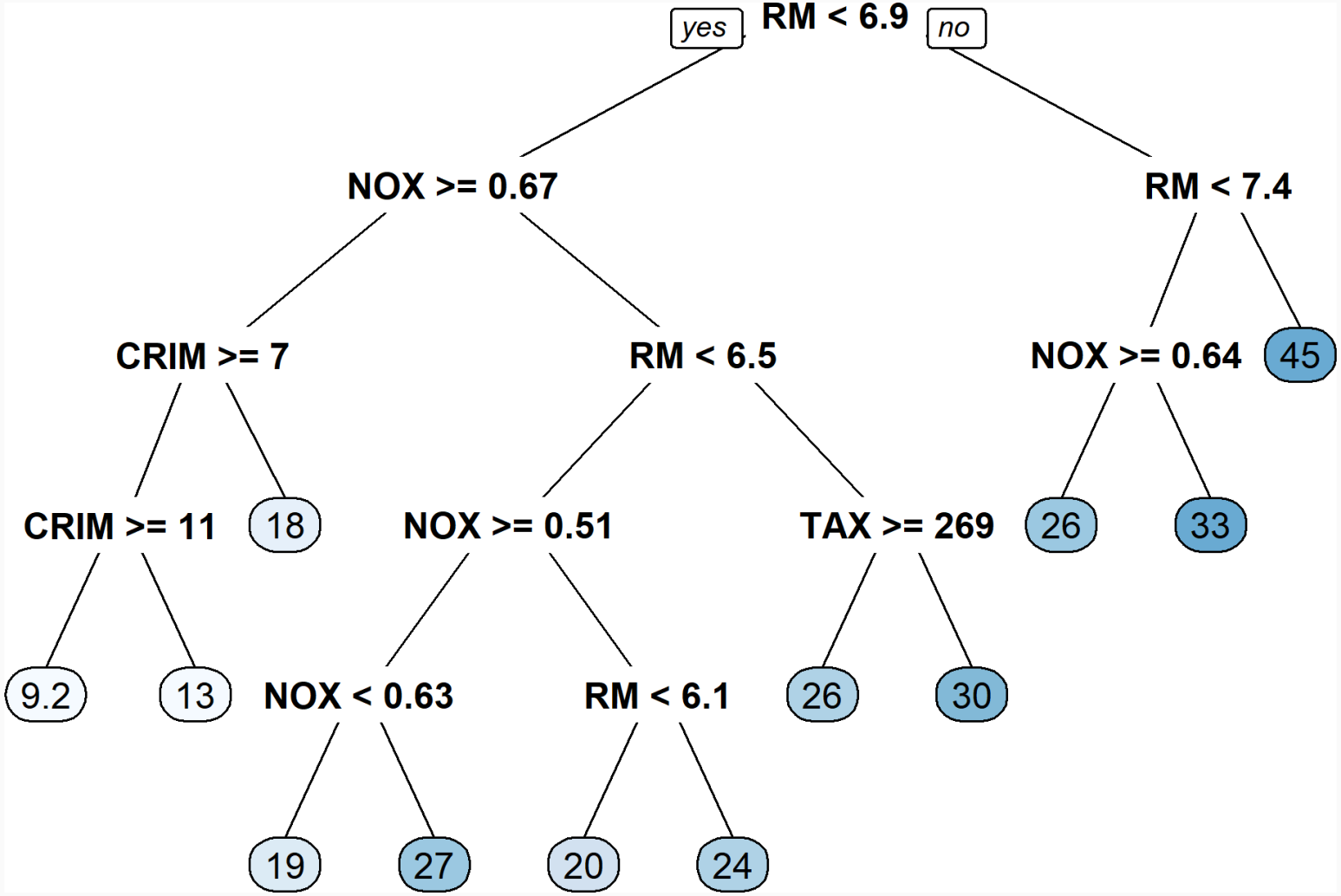
```
prune_housing <- prune(tree_housing, cp = alpha.best)

# What happened to the number of leaves?
n_distinct(tree_housing$where)
n_distinct(prune_housing$where)
```

```
# [1] 21
# [1] 12
```

```
# Plot the pruned tree
prp(prune_housing, box.palette = "auto")
```

Plot the pruned tree



Make predictions

```
# Create a vector of predicted housing values
medv_predict <- predict(prune_housing, newdata = housing)
length(medv_predict)
nrow(housing)

# MSE
mean((medv_predict - housing$MEDV)^2)
```

```
# [1] 506
# [1] 506
# [1] 19.1187
```

Summary

- Trees can be used for classification or regression problems
- They are useful in presence of complicated nonlinearities
- Otherwise, their predictive performance is often worse than other methods

- Lab-13 due Sunday at 11:59pm
- Reminder: midterm is **Wednesday, October 29**