Data Science Machine Learning

The textbook for the Data Science course series is freely available online.

Learning Objectives

- The basics of machine learning
- How to perform cross-validation to avoid overtraining
- Several popular machine learning algorithms
- How to build a recommendation system
- What regularization is and why it is useful

Course Overview

There are six major sections in this course: introduction to machine learning; machine learning basics; linear regression for prediction, smoothing, and working with matrices; distance, knn, cross validation, and generative models; classification with more than two classes and the caret package; and model fitting and recommendation systems.

Introduction to Machine Learning

In this section, you'll be introduced to some of the terminology and concepts you'll need going forward.

Machine Learning Basics

In this section, you'll learn how to start building a machine learning algorithm using training and test data sets and the importance of conditional probabilities for machine learning.

Linear Regression for Prediction, Smoothing, and Working with Matrices

In this section, you'll learn why linear regression is a useful baseline approach but is often insufficiently flexible for more complex analyses, how to smooth noisy data, and how to use matrices for machine learning.

Distance, Knn, Cross Validation, and Generative Models

In this section, you'll learn different types of discriminative and generative approaches for machine learning algorithms.

Classification with More than Two Classes and the Caret Package

In this section, you'll learn how to overcome the curse of dimensionality using methods that adapt to higher dimensions and how to use the caret package to implement many different machine learning algorithms.

Model Fitting and Recommendation Systems

In this section, you'll learn how to apply the machine learning algorithms you have learned.

Section 1 - Introduction to Machine Learning Overview

In the **Introduction to Machine Learning** section, you will be introduced to machine learning.

After completing this section, you will be able to:

- Explain the difference between the **outcome** and the **features**.
- Explain when to use **classification** and when to use **prediction**.
- Explain the importance of **prevalence**.
- Explain the difference between **sensitivity** and **specificity**.

This section has one part: **introduction to machine learning**.

Notation

There is a link to the relevant section of the textbook: Notation

Key points

- $X_1, ..., X_p$ denote the features, Y denotes the outcomes, and $\hat{Y}$ denotes the predictions.
- Machine learning prediction tasks can be divided into **categorical** and **continuous** outcomes. We refer to these as **classification** and **prediction**, respectively.

An Example

There is a link to the relevant section of the textbook: An Example

Key points

- Y_i = an outcome for observation or index i.
- We use boldface for $\mathbf{X_i}$ to distinguish the vector of predictors from the individual predictors $X_{i,1}, ..., X_{i,784}$.
- When referring to an arbitrary set of features and outcomes, we drop the index i and use Y and bold $\mathbf{X}$.
- Uppercase is used to refer to variables because we think of predictors as random variables.
- Lowercase is used to denote observed values. For example, $\mathbf{X} = \mathbf{x}$.

Comprehension Check - Introduction to Machine Learning

1. True or False: A key feature of machine learning is that the algorithms are built with data.

☒ A. True
☐ B. False

2. True or False: In machine learning, we build algorithms that take feature values (X) and train a model using known outcomes (Y) that is then used to predict outcomes when presented with features without known outcomes.

☒ A. True
☐ B. False

Section 2 - Machine Learning Basics Overview

In the **Machine Learning Basics** section, you will learn the basics of machine learning.

After completing this section, you will be able to:

- Start to use the **caret** package.
- Construct and interpret a **confusion matrix**.
- Use **conditional probabilities** in the context of machine learning.

This section has two parts: **basics of evaluating machine learning algorithms** and **conditional probabilities**.

Caret package, training and test sets, and overall accuracy

There is a link to the relevant sections of the textbook: Training and test sets and Overall accuracy

Key points

- Note: the `set.seed()` function is used to obtain reproducible results. If you have R 3.6 or later, please use the `sample.kind = "Rounding"` argument whenever you set the seed for this course.
- To mimic the ultimate evaluation process, we randomly split our data into two — a training set and a test set — and act as if we don't know the outcome of the test set. We develop algorithms using only the training set; the test set is used only for evaluation.
- The `createDataPartition()` function from the **caret** package can be used to generate indexes for randomly splitting data.
- Note: contrary to what the documentation says, this course will use the argument p as the percentage of data that goes to testing. The indexes made from `createDataPartition()` should be used to create the test set. Indexes should be created on the outcome and not a predictor.
- The simplest evaluation metric for categorical outcomes is overall accuracy: the proportion of cases that were correctly predicted in the test set.

Code

```
if(!require(tidyverse)) install.packages("tidyverse")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
if(!require(caret)) install.packages("caret")
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
if(!require(dslabs)) install.packages("dslabs")
```

```
## Loading required package: dslabs
```

```r
library(tidyverse)
library(caret)
library(dslabs)
data(heights)

# define the outcome and predictors
y <- heights$sex
x <- heights$height

# generate training and test sets
set.seed(2, sample.kind = "Rounding") # if using R 3.5 or earlier, remove the sample.kind argument
```

```
## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
test_set <- heights[test_index, ]
train_set <- heights[-test_index, ]

# guess the outcome
y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE)
y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE) %>%
  factor(levels = levels(test_set$sex))

# compute accuracy
mean(y_hat == test_set$sex)
```

```
## [1] 0.5238095
```

```r
heights %>% group_by(sex) %>% summarize(mean(height), sd(height))
```

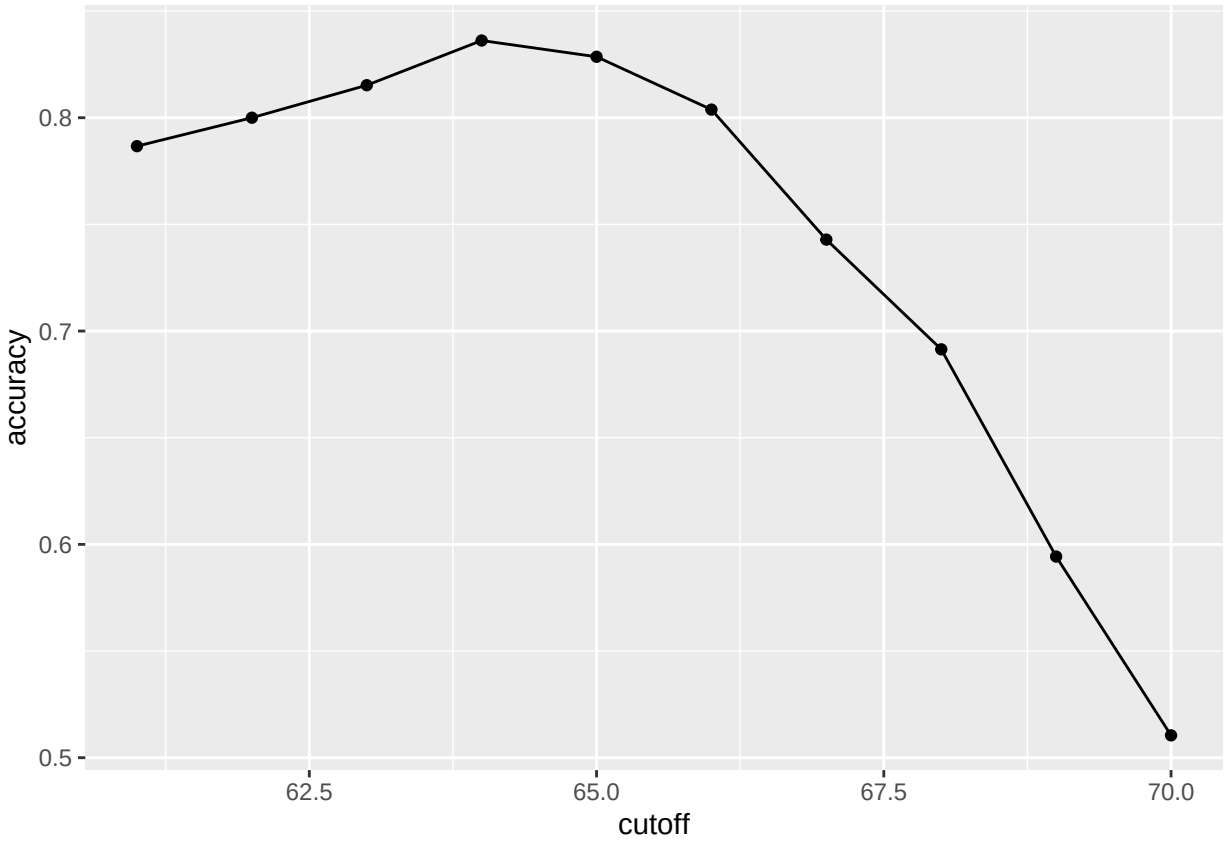
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 3
##   sex    `mean(height)` `sd(height)`
##   <fct>           <dbl>        <dbl>
## 1 Female           64.9         3.76
## 2 Male             69.3         3.61
```

```
y_hat <- ifelse(x > 62, "Male", "Female") %>% factor(levels = levels(test_set$sex))
mean(y == y_hat)
```

```
## [1] 0.7933333
```

```
# examine the accuracy of 10 cutoffs
cutoff <- seq(61, 70)
accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train_set$height > x, "Male", "Female") %>%
    factor(levels = levels(test_set$sex))
  mean(y_hat == train_set$sex)
})
data.frame(cutoff, accuracy) %>%
  ggplot(aes(cutoff, accuracy)) +
  geom_point() +
  geom_line()
```




```
max(accuracy)
```

```
## [1] 0.8361905
```

```
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff
```

```
## [1] 64
```

```
y_hat <- ifelse(test_set$height > best_cutoff, "Male", "Female") %>%
  factor(levels = levels(test_set$sex))
y_hat <- factor(y_hat)
mean(y_hat == test_set$sex)
```

```
## [1] 0.8171429
```

Comprehension Check - Basics of Evaluating Machine Learning Algorithms

1. For each of the following, indicate whether the outcome is continuous or categorical.

- Digit reader - categorical
- Height - continuous

- Spam filter - categorical
- Stock prices - continuous
- Sex - categorical

2. How many features are available to us for prediction in the `mnist` digits dataset?

You can download the `mnist` dataset using the `read_mnist()` function from the **dslabs** package.

```
mnist <- read_mnist()
ncol(mnist$train$images)
```

```
## [1] 784
```

Confusion matrix

There is a link to the relevant sections of the textbook: Confusion Matrix

Key points

- Overall accuracy can sometimes be a deceptive measure because of unbalanced classes.
- A general improvement to using overall accuracy is to study sensitivity and specificity separately. **Sensitivity**, also known as the true positive rate or recall, is the proportion of actual positive outcomes correctly identified as such. **Specificity**, also known as the true negative rate, is the proportion of actual negative outcomes that are correctly identified as such.
- A confusion matrix tabulates each combination of prediction and actual value. You can create a confusion matrix in R using the `table()` function or the `confusionMatrix()` function from the **caret** package.

Code

```r
# tabulate each combination of prediction and actual value
table(predicted = y_hat, actual = test_set$sex)
```

```
##          actual
## predicted Female Male
##    Female     50   27
##    Male       69  379
```

```r
test_set %>%
  mutate(y_hat = y_hat) %>%
  group_by(sex) %>%
  summarize(accuracy = mean(y_hat == sex))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   sex    accuracy
##   <fct>     <dbl>
## 1 Female    0.420
## 2 Male      0.933
```

```r
prev <- mean(y == "Male")

confusionMatrix(data = y_hat, reference = test_set$sex)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Female Male
##     Female     50   27
##     Male       69  379
##
##                Accuracy : 0.8171
##                  95% CI : (0.7814, 0.8493)
##     No Information Rate : 0.7733
##     P-Value [Acc > NIR] : 0.008354
##
##                   Kappa : 0.4041
##
##  Mcnemar's Test P-Value : 2.857e-05
##
##             Sensitivity : 0.42017
##             Specificity : 0.93350
##          Pos Pred Value : 0.64935
##          Neg Pred Value : 0.84598
##              Prevalence : 0.22667
##          Detection Rate : 0.09524
##    Detection Prevalence : 0.14667
##       Balanced Accuracy : 0.67683
##
##        'Positive' Class : Female
##
```

Balanced accuracy and F1 score

There is a link to the relevant sections of the textbook: Balanced accuracy and F1 Score

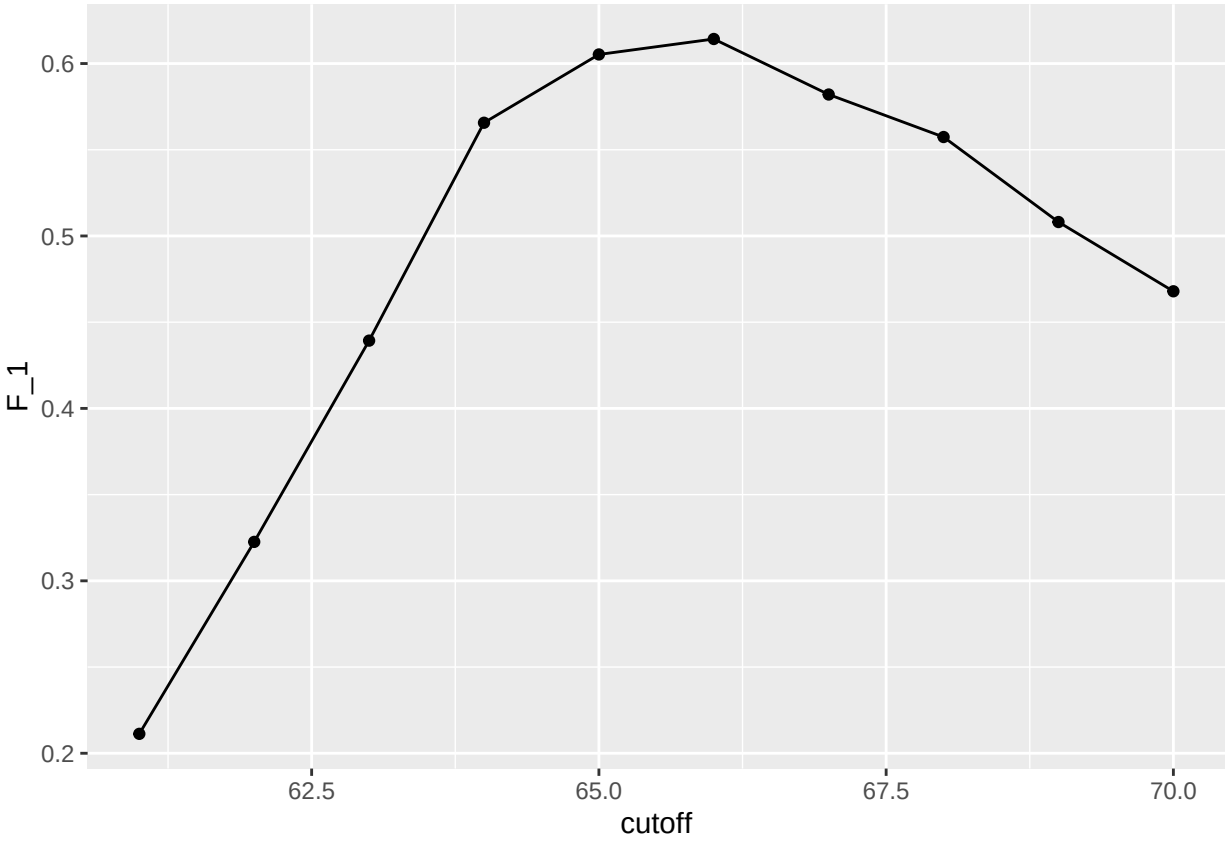
Key points

- For optimization purposes, sometimes it is more useful to have a one number summary than studying both specificity and sensitivity. One preferred metric is **balanced accuracy**. Because specificity and sensitivity are rates, it is more appropriate to compute the *harmonic* average. In fact, the **F1-score**, a widely used one-number summary, is the harmonic average of precision and recall.
- Depending on the context, some type of errors are more costly than others. The **F1-score** can be adapted to weigh specificity and sensitivity differently.
- You can compute the **F1-score** using the `F_meas()` function in the **caret** package.

Code

```
# maximize F-score
cutoff <- seq(61, 70)
F_1 <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train_set$height > x, "Male", "Female") %>%
    factor(levels = levels(test_set$sex))
  F_meas(data = y_hat, reference = factor(train_set$sex))
})

data.frame(cutoff, F_1) %>%
  ggplot(aes(cutoff, F_1)) +
  geom_point() +
  geom_line()
```



```r
max(F_1)
```

```
## [1] 0.6142322
```

```r
best_cutoff <- cutoff[which.max(F_1)]
best_cutoff
```

```
## [1] 66
```

```r
y_hat <- ifelse(test_set$height > best_cutoff, "Male", "Female") %>%
  factor(levels = levels(test_set$sex))
sensitivity(data = y_hat, reference = test_set$sex)
```

```
## [1] 0.6806723
```

```r
specificity(data = y_hat, reference = test_set$sex)
```

```
## [1] 0.8349754
```

Prevalence matters in practice

There is a link to the relevant sections of the textbook: Prevalence matters in practice

Key points

- A machine learning algorithm with very high sensitivity and specificity may not be useful in practice when prevalence is close to either 0 or 1. For example, if you develop an algorithm for disease diagnosis with very high sensitivity, but the prevalence of the disease is pretty low, then the precision of your algorithm is probably very low based on Bayes' theorem.

ROC and precision-recall curves

There is a link to the relevant sections of the textbook: ROC and precision-recall curves

Key points

- A very common approach to evaluating accuracy and F1-score is to compare them graphically by plotting both. A widely used plot that does this is the **receiver operating characteristic (ROC) curve**. The ROC curve plots sensitivity (TPR) versus 1 - specificity or the false positive rate (FPR).
- However, ROC curves have one weakness and it is that neither of the measures plotted depend on prevalence. In cases in which prevalence matters, we may instead make a **precision-recall plot**, which has a similar idea with ROC curve.

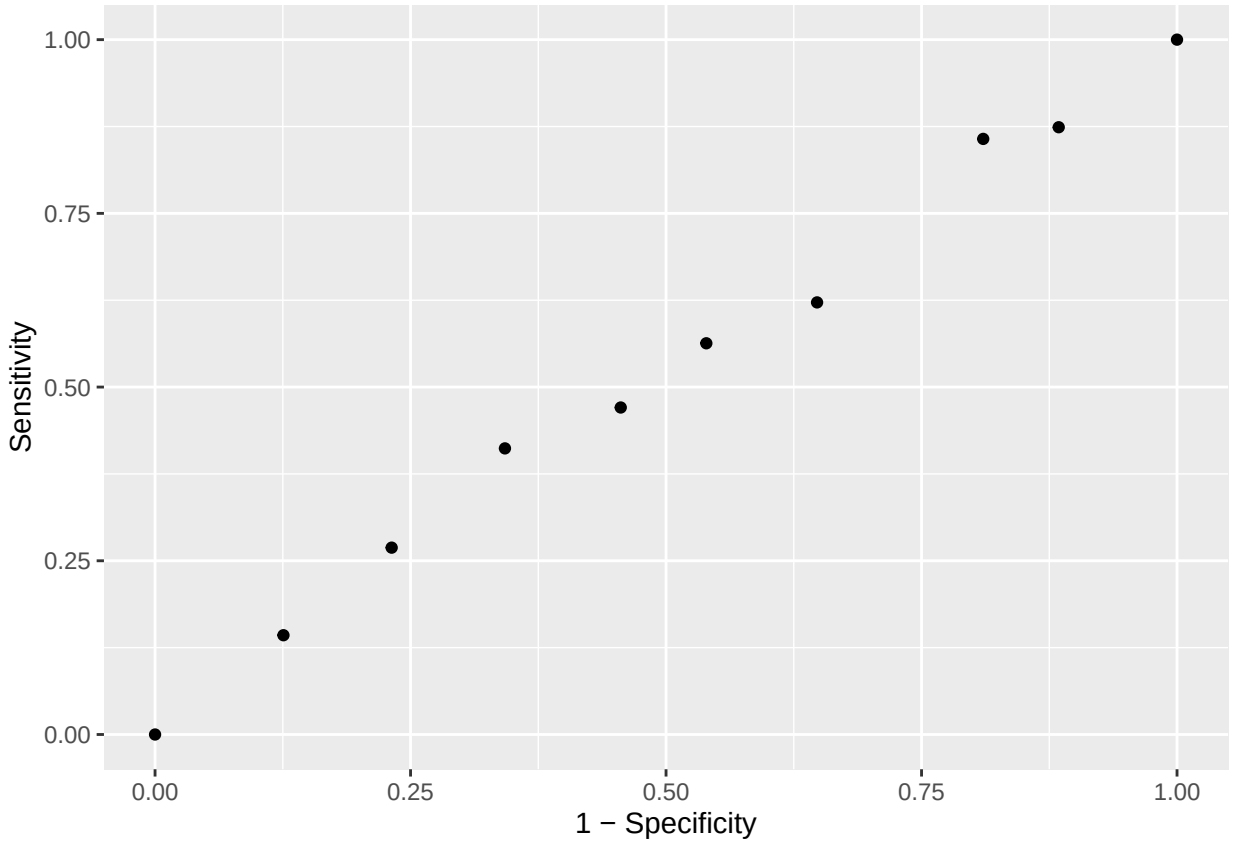
Code

Note: your results and plots may be slightly different.

```r
p <- 0.9
n <- length(test_index)
y_hat <- sample(c("Male", "Female"), n, replace = TRUE, prob=c(p, 1-p)) %>%
  factor(levels = levels(test_set$sex))
mean(y_hat == test_set$sex)
```

```
## [1] 0.7180952
```

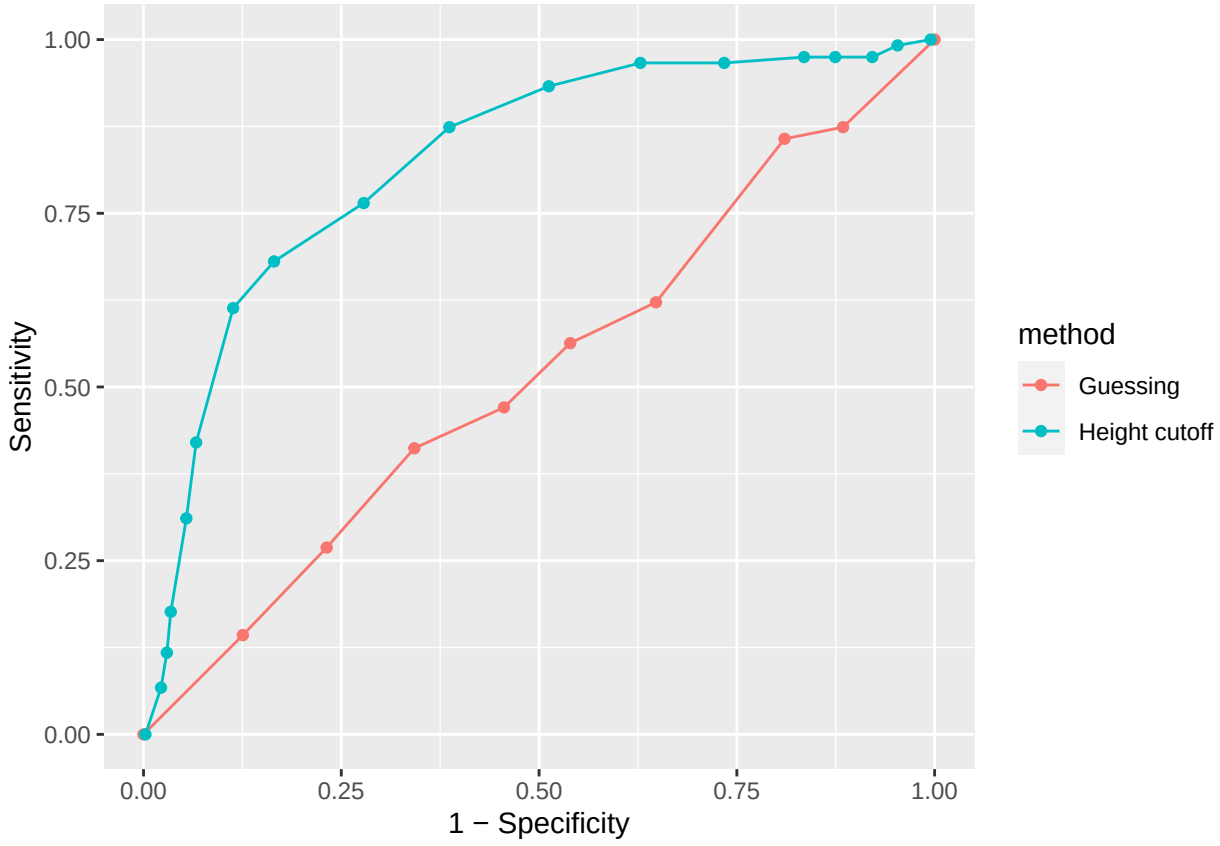
```
# ROC curve
probs <- seq(0, 1, length.out = 10)
guessing <- map_df(probs, function(p){
  y_hat <-
    sample(c("Male", "Female"), n, replace = TRUE, prob=c(p, 1-p)) %>%
    factor(levels = c("Female", "Male"))
  list(method = "Guessing",
       FPR = 1 - specificity(y_hat, test_set$sex),
       TPR = sensitivity(y_hat, test_set$sex))
})
guessing %>% qplot(FPR, TPR, data =., xlab = "1 - Specificity", ylab = "Sensitivity")
```




```
cutoffs <- c(50, seq(60, 75), 80)
height_cutoff <- map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Female", "Male"))
   list(method = "Height cutoff",
        FPR = 1-specificity(y_hat, test_set$sex),
        TPR = sensitivity(y_hat, test_set$sex))
})

# plot both curves together
bind_rows(guessing, height_cutoff) %>%
  ggplot(aes(FPR, TPR, color = method)) +
```

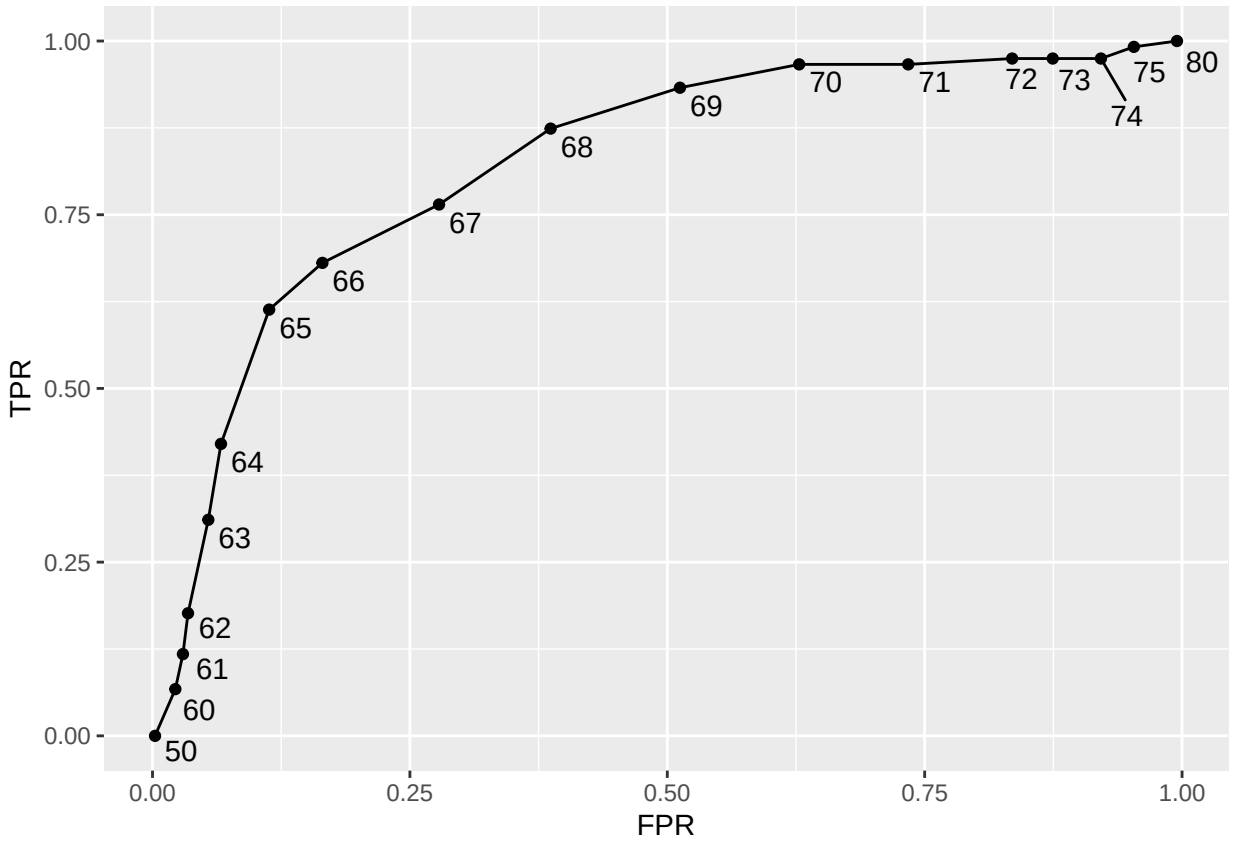
```
  geom_line() +
  geom_point() +
  xlab("1 - Specificity") +
  ylab("Sensitivity")
```




```
if(!require(ggrepel)) install.packages("ggrepel")
```

```
## Loading required package: ggrepel
```

```
library(ggrepel)
map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Female", "Male"))
   list(method = "Height cutoff",
        cutoff = x,
        FPR = 1-specificity(y_hat, test_set$sex),
        TPR = sensitivity(y_hat, test_set$sex))
}) %>%
  ggplot(aes(FPR, TPR, label = cutoff)) +
  geom_line() +
  geom_point() +
  geom_text_repel(nudge_x = 0.01, nudge_y = -0.01)
```



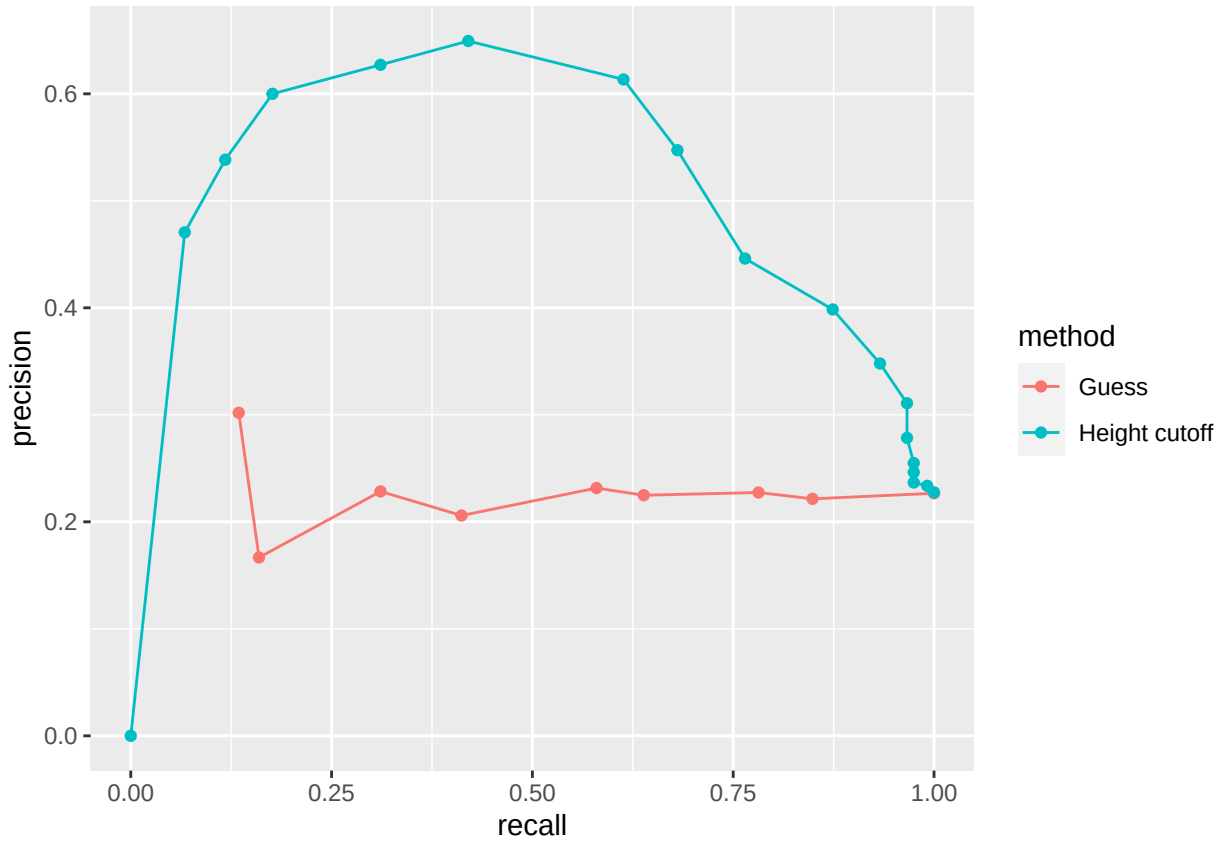

```
# plot precision against recall
guessing <- map_df(probs, function(p){
  y_hat <- sample(c("Male", "Female"), length(test_index),
                  replace = TRUE, prob=c(p, 1-p)) %>%
    factor(levels = c("Female", "Male"))
  list(method = "Guess",
    recall = sensitivity(y_hat, test_set$sex),
    precision = precision(y_hat, test_set$sex))
})

height_cutoff <- map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Female", "Male"))
  list(method = "Height cutoff",
       recall = sensitivity(y_hat, test_set$sex),
    precision = precision(y_hat, test_set$sex))
})

bind_rows(guessing, height_cutoff) %>%
  ggplot(aes(recall, precision, color = method)) +
  geom_line() +
  geom_point()
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

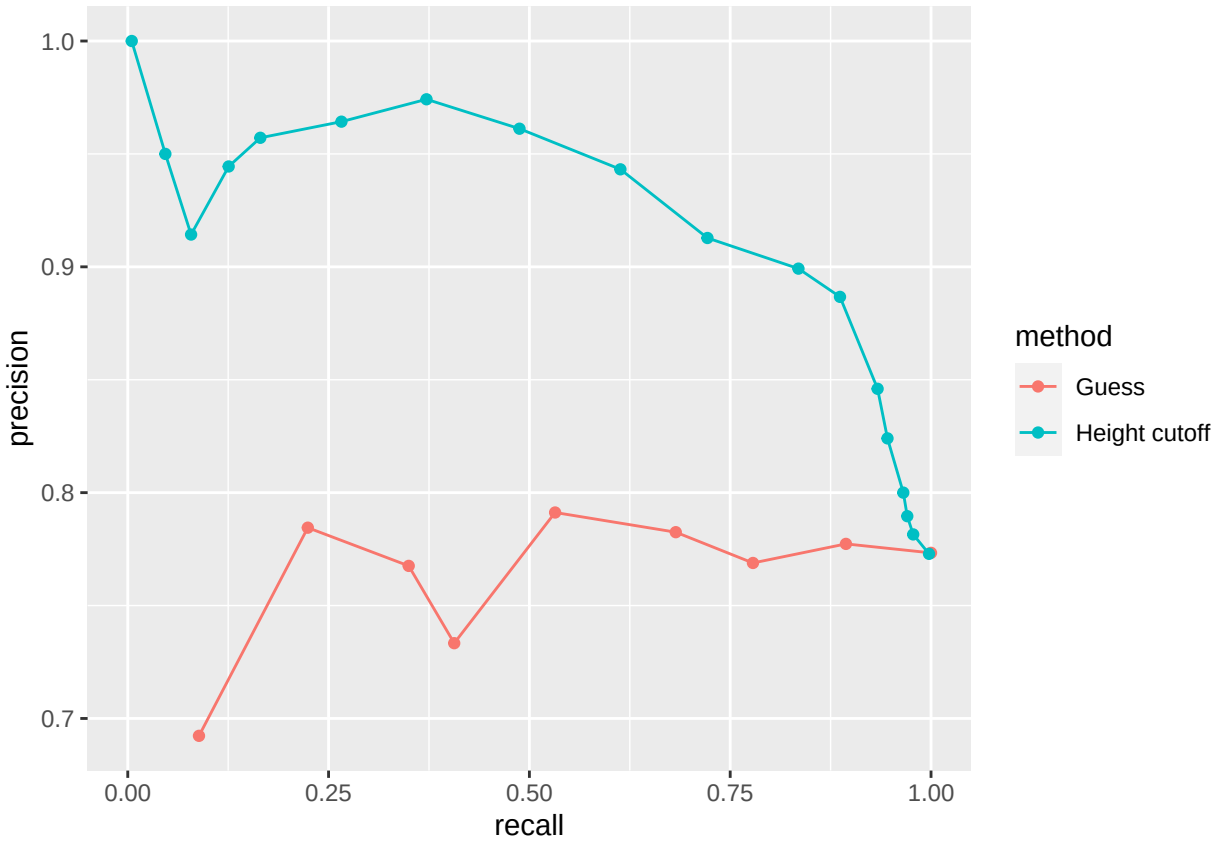



```
guessing <- map_df(probs, function(p){
  y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE,
                  prob=c(p, 1-p)) %>%
    factor(levels = c("Male", "Female"))
  list(method = "Guess",
    recall = sensitivity(y_hat, relevel(test_set$sex, "Male", "Female")),
    precision = precision(y_hat, relevel(test_set$sex, "Male", "Female")))
})

height_cutoff <- map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Male", "Female"))
  list(method = "Height cutoff",
       recall = sensitivity(y_hat, relevel(test_set$sex, "Male", "Female")),
    precision = precision(y_hat, relevel(test_set$sex, "Male", "Female")))
})
bind_rows(guessing, height_cutoff) %>%
  ggplot(aes(recall, precision, color = method)) +
  geom_line() +
  geom_point()
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).

## Warning: Removed 1 rows containing missing values (geom_point).
```



Comprehension Check - Practice with Machine Learning, Part 1

The following questions all ask you to work with the dataset described below.

The `reported_heights` and `heights` datasets were collected from three classes taught in the Departments of Computer Science and Biostatistics, as well as remotely through the Extension School. The Biostatistics class was taught in 2016 along with an online version offered by the Extension School. On 2016-01-25 at 8:15 AM, during one of the lectures, the instructors asked student to fill in the sex and height questionnaire that populated the `reported_heights` dataset. The online students filled out the survey during the next few days, after the lecture was posted online. We can use this insight to define a variable which we will call `type`, to denote the type of student, `inclass` or `online`.

The code below sets up the dataset for you to analyze in the following exercises:

```
if(!require(dplyr)) install.packages("dplyr")
if(!require(lubridate)) install.packages("lubridate")
```

```
## Loading required package: lubridate
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(dplyr)
library(lubridate)
data(reported_heights)

dat <- mutate(reported_heights, date_time = ymd_hms(time_stamp)) %>%
  filter(date_time >= make_date(2016, 01, 25) & date_time < make_date(2016, 02, 1)) %>%
  mutate(type = ifelse(day(date_time) == 25 & hour(date_time) == 8 & between(minute(date_time), 15, 30)
  select(sex, type)

y <- factor(dat$sex, c("Female", "Male"))
x <- dat$type
```

1. The `type` column of `dat` indicates whether students took classes in person ("inclass") or online ("online"). What proportion of the inclass group is female? What proportion of the online group is female?

Enter your answer as a percentage or decimal (eg "50%" or "0.50") to at least the hundredths place.

```
dat %>% group_by(type) %>% summarize(prop_female = mean(sex == "Female"))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   type    prop_female
##   <chr>         <dbl>
## 1 inclass       0.667
## 2 online        0.378
```

2. In the course videos, height cutoffs were used to predict sex. Instead of height, use the `type` variable to predict sex. Assume that for each class type the students are either all male or all female, based on the most prevalent sex in each class type you calculated in Q1. Report the accuracy of your prediction of sex based on type. You do not need to split the data into training and test sets.

Enter your accuracy as a percentage or decimal (eg "50%" or "0.50") to at least the hundredths place.

```
y_hat <- ifelse(x == "online", "Male", "Female") %>%
      factor(levels = levels(y))
mean(y_hat==y)
```

```
## [1] 0.6333333
```

3. Write a line of code using the `table()` function to show the confusion matrix between `y_hat` and `y`. Use the **exact** format `function(a, b)` for your answer and do not name the columns and rows. Your answer should have exactly one space.

```
table(y_hat, y)
```

```
##         y
## y_hat    Female Male
##   Female     26   13
##   Male       42   69
```

4. What is the sensitivity of this prediction? You can use the `sensitivity()` function from the **caret** package. Enter your answer as a percentage or decimal (eg "50%" or "0.50") to at least the hundredths place.

```
sensitivity(y_hat, y)
```

```
## [1] 0.3823529
```

5. What is the specificity of this prediction? You can use the `specificity()` function from the **caret** package. Enter your answer as a percentage or decimal (eg "50%" or "0.50") to at least the hundredths place.

```
specificity(y_hat, y)
```

```
## [1] 0.8414634
```

6. What is the prevalence (% of females) in the `dat` dataset defined above? Enter your answer as a percentage or decimal (eg "50%" or "0.50") to at least the hundredths place.

```
mean(y == "Female")
```

```
## [1] 0.4533333
```

Comprehension Check - Practice with Machine Learning, Part 2

We will practice building a machine learning algorithm using a new dataset, `iris`, that provides multiple predictors for us to use to train. To start, we will remove the `setosa` species and we will focus on the `versicolor` and `virginica` iris species using the following code:

```
data(iris)
iris <- iris[-which(iris$Species=='setosa'),]
y <- iris$Species
```

The following questions all involve work with this dataset.

7. First let us create an even split of the data into `train` and `test` partitions using `createDataPartition()` from the **caret** package. The code with a missing line is given below:

```
# set.seed(2) # if using R 3.5 or earlier
set.seed(2, sample.kind="Rounding") # if using R 3.6 or later
# line of code
test <- iris[test_index,]
train <- iris[-test_index,]
```

Which code should be used in place of # line of code above?

- ☐ A. test_index <- createDataPartition(y,times=1,p=0.5)
- ☐ B. test_index <- sample(2,length(y),replace=FALSE)
- ☒ C. test_index <- createDataPartition(y,times=1,p=0.5,list=FALSE)
- ☐ D. test_index <- rep(1,length(y))

```
# set.seed(2) # if using R 3.5 or earlier
set.seed(2, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y,times=1,p=0.5,list=FALSE)
```

```
## Warning in createDataPartition(y, times = 1, p = 0.5, list = FALSE): Some
## classes have no records ( setosa ) and these will be ignored
```

```
test <- iris[test_index,]
train <- iris[-test_index,]
```

8. Next we will figure out the singular feature in the dataset that yields the greatest overall accuracy when predicting species. You can use the code from the introduction and from Q7 to start your analysis.

Using only the `train` iris dataset, for each feature, perform a simple search to find the cutoff that produces the highest accuracy, predicting virginica if greater than the cutoff and versicolor otherwise. Use the `seq` function over the range of each feature by intervals of 0.1 for this search.

Which feature produces the highest accuracy?

```
foo <- function(x){
    rangedValues <- seq(range(x)[1],range(x)[2],by=0.1)
    sapply(rangedValues,function(i){
        y_hat <- ifelse(x>i,'virginica','versicolor')
        mean(y_hat==train$Species)
    })
}
predictions <- apply(train[,-5],2,foo)
sapply(predictions,max)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##         0.70         0.62         0.96         0.94
```

- ☐ A. Sepal.Length
- ☐ B. Sepal.Width
- ☒ C. Petal.Length
- ☐ D. Petal.Width

9. For the feature selected in Q8, use the smart cutoff value from the training data to calculate overall accuracy in the test data. What is the overall accuracy?

```
predictions <- foo(train[,3])
rangedValues <- seq(range(train[,3])[1],range(train[,3])[2],by=0.1)
cutoffs <-rangedValues[which(predictions==max(predictions))]

y_hat <- ifelse(test[,3]>cutoffs[1],'virginica','versicolor')
mean(y_hat==test$Species)
```

```
## [1] 0.9
```

10. Notice that we had an overall accuracy greater than 96% in the training data, but the overall accuracy was lower in the test data. This can happen often if we overtrain. In fact, it could be the case that a single feature is not the best choice. For example, a combination of features might be optimal. Using a single feature and optimizing the cutoff as we did on our training data can lead to overfitting.

Given that we know the test data, we can treat it like we did our training data to see if the same feature with a different cutoff will optimize our predictions.

Which feature best optimizes our overall accuracy?

```
foo <- function(x){
    rangedValues <- seq(range(x)[1],range(x)[2],by=0.1)
    sapply(rangedValues,function(i){
        y_hat <- ifelse(x>i,'virginica','versicolor')
        mean(y_hat==test$Species)
    })
}
predictions <- apply(test[,-5],2,foo)
sapply(predictions,max)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##         0.78         0.64         0.90         0.94
```

- ☐ A. Sepal.Length
- ☐ B. Sepal.Width
- ☐ C. Petal.Length
- ☒ D. Petal.Width

11. Now we will perform some exploratory data analysis on the data.

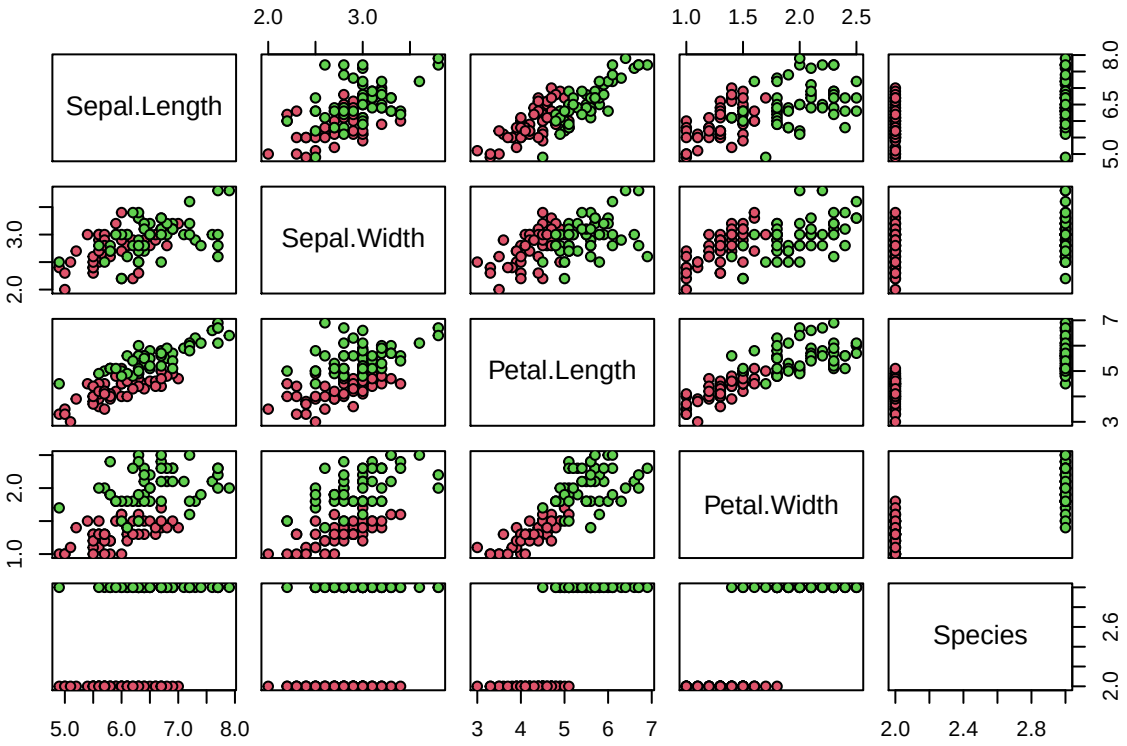
Notice that `Petal.Length` and `Petal.Width` in combination could potentially be more information than either feature alone.

Optimize the the cutoffs for `Petal.Length` and `Petal.Width` separately in the train dataset by using the `seq` function with increments of 0.1. Then, report the overall accuracy when applied to the test dataset by creating a rule that predicts virginica if `Petal.Length` is greater than the length cutoff OR `Petal.Width` is greater than the width cutoff, and versicolor otherwise.

What is the overall accuracy for the test data now?

```
data(iris)
iris <- iris[-which(iris$Species=='setosa'),]
y <- iris$Species

plot(iris,pch=21,bg=iris$Species)
```




```
# set.seed(2) # if using R 3.5 or earlier
set.seed(2, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y,times=1,p=0.5,list=FALSE)
```

```
## Warning in createDataPartition(y, times = 1, p = 0.5, list = FALSE): Some
## classes have no records ( setosa ) and these will be ignored
```

```
test <- iris[test_index,]
train <- iris[-test_index,]

petalLengthRange <- seq(range(train$Petal.Length)[1],range(train$Petal.Length)[2],by=0.1)
petalWidthRange <- seq(range(train$Petal.Width)[1],range(train$Petal.Width)[2],by=0.1)

length_predictions <- sapply(petalLengthRange,function(i){
        y_hat <- ifelse(train$Petal.Length>i,'virginica','versicolor')
        mean(y_hat==train$Species)
    })
length_cutoff <- petalLengthRange[which.max(length_predictions)] # 4.7

width_predictions <- sapply(petalWidthRange,function(i){
        y_hat <- ifelse(train$Petal.Width>i,'virginica','versicolor')
        mean(y_hat==train$Species)
    })
width_cutoff <- petalWidthRange[which.max(width_predictions)] # 1.5
```

```
y_hat <- ifelse(test$Petal.Length>length_cutoff | test$Petal.Width>width_cutoff,'virginica','versicolor
mean(y_hat==test$Species)
```

```
## [1] 0.88
```