Data Science Machine Learning

The textbook for the Data Science course series is [freely available online](https://rafalab.github.io/dsbook/).

# 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](https://rafalab.github.io/dsbook/introduction-to-machine-learning.html#notation-1)

**Key points**

* denote the features, denotes the outcomes, and 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](https://rafalab.github.io/dsbook/introduction-to-machine-learning.html#an-example)

**Key points**

* = an outcome for observation or index i.
* We use boldface for to distinguish the vector of predictors from the individual predictors .
* When referring to an arbitrary set of features and outcomes, we drop the index i and use and bold .
* Uppercase is used to refer to variables because we think of predictors as random variables.
* Lowercase is used to denote observed values. For example, .

## 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

1. 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](https://rafalab.github.io/dsbook/introduction-to-machine-learning.html#training-and-test-sets) and [Overall accuracy](https://rafalab.github.io/dsbook/introduction-to-machine-learning.html#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 ─────────────────────────────────────── tidyverse 1.3.0 ──

## ✓ ggplot2 3.3.2 ✓ purrr 0.3.4  
## ✓ tibble 3.0.4 ✓ dplyr 1.0.2  
## ✓ tidyr 1.1.2 ✓ stringr 1.4.0  
## ✓ readr 1.4.0 ✓ forcats 0.5.0

## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

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

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

## Loading required package: dslabs

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

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

heights %>% group\_by(sex) %>% summarize(mean(height), sd(height))

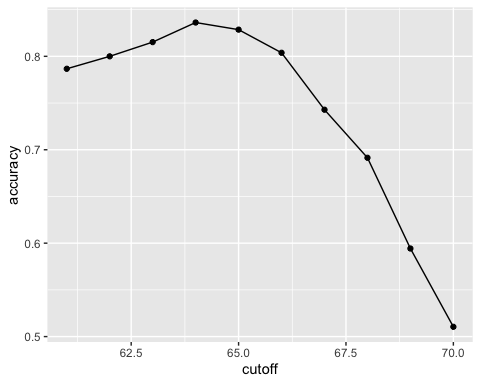
## `summarise()` ungrouping output (override with `.groups` argument)

## # 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

1. 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 section of the textbook: [Confusion Matrix](https://rafalab.github.io/dsbook/introduction-to-machine-learning.html#the-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*

# 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

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

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

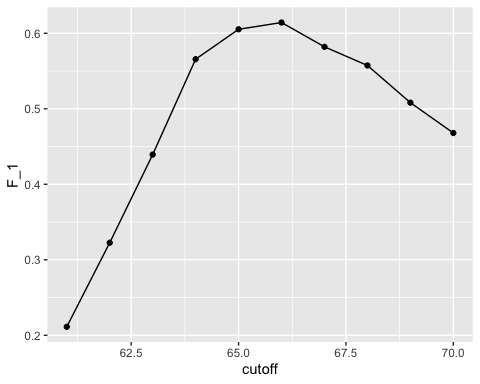
There is a link to the relevant section of the textbook: [Balanced accuracy and F1 Score](https://rafalab.github.io/dsbook/introduction-to-machine-learning.html#balanced-accuracy-and-f_1-score)

**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()



max(F\_1)

## [1] 0.6142322

best\_cutoff <- cutoff[which.max(F\_1)]  
best\_cutoff

## [1] 66

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

specificity(data = y\_hat, reference = test\_set$sex)

## [1] 0.8349754

## Prevalence matters in practice

There is a link to the relevant section of the textbook: [Prevalence matters in practice](https://rafalab.github.io/dsbook/introduction-to-machine-learning.html#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 section of the textbook: [ROC and precision-recall curves](https://rafalab.github.io/dsbook/introduction-to-machine-learning.html#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.

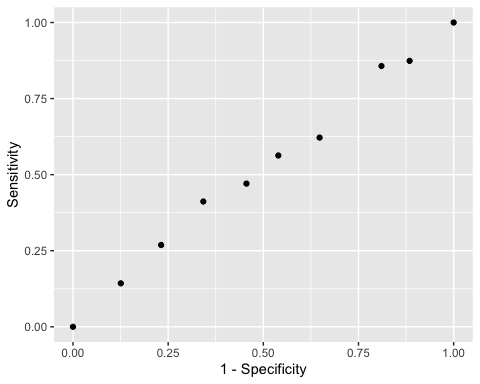
*Code*

Note: your results and plots may be slightly different.

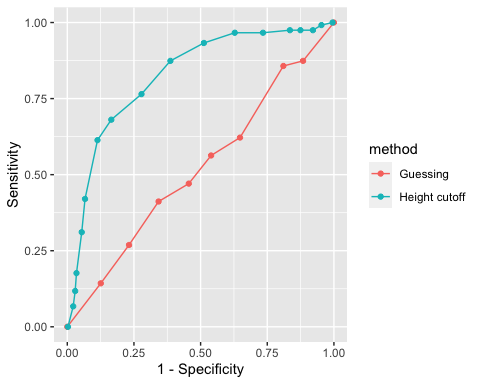
p <- 0.9  
n <- length(test\_index)  
y\_hat <- sample(c("Male", "Female"), n, replace = TRUE, prob=c(p, 1-p)) %>%   
 factor(levels = levels(test\_set$sex))  
mean(y\_hat == test\_set$sex)

## [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")



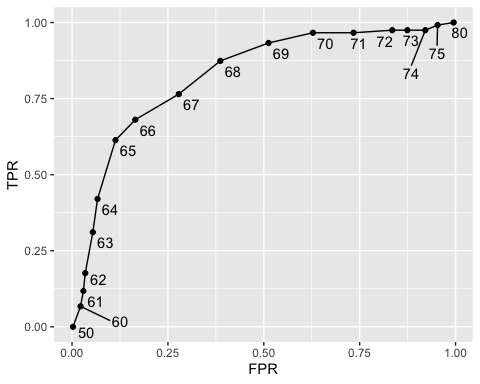
cutoffs <- c(50, seq(60, 75), 80)  
height\_cutoff <- map\_df(cutoffs, function(x){  
 y\_hat <- ifelse(test\_set$height > x, "Male", "Female") %>%   
 factor(levels = c("Female", "Male"))  
 list(method = "Height cutoff",  
 FPR = 1-specificity(y\_hat, test\_set$sex),  
 TPR = sensitivity(y\_hat, test\_set$sex))  
})  
  
# 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

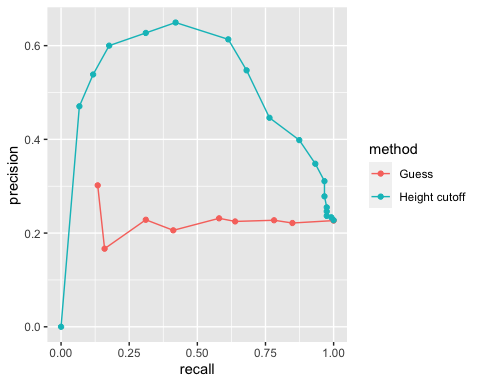
library(ggrepel)  
map\_df(cutoffs, function(x){  
 y\_hat <- ifelse(test\_set$height > x, "Male", "Female") %>%   
 factor(levels = c("Female", "Male"))  
 list(method = "Height cutoff",  
 cutoff = x,   
 FPR = 1-specificity(y\_hat, test\_set$sex),  
 TPR = sensitivity(y\_hat, test\_set$sex))  
}) %>%  
 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()

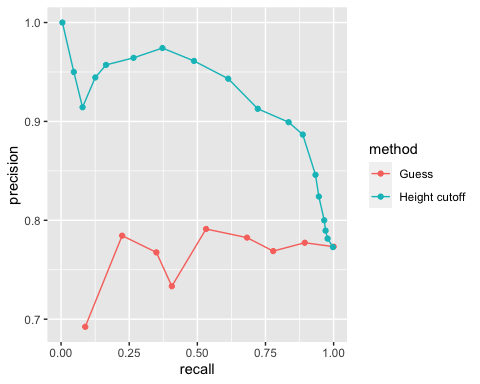
## Warning: Removed 1 row(s) containing missing values (geom\_path).

## 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), "inclass","online")) %>%  
 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

1. 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

1. 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

1. 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

1. 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

1. 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.

1. 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,]

1. 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

1. 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

1. 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

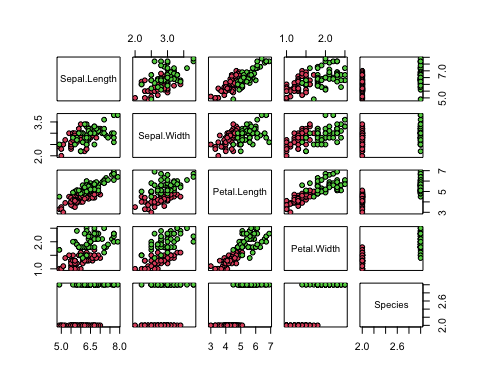
1. Now we will perform some exploratory data analysis on the data.

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

## Conditional probabilities

There is a link to the relevant section of the textbook: [Conditional probabilities](https://rafalab.github.io/dsbook/introduction-to-machine-learning.html#conditional-probabilities-1)

**Key points**

* Conditional probabilities for each class:
* In machine learning, this is referred to as **Bayes’ Rule**. This is a theoretical rule because in practice we don’t know . Having a good estimate of the will suffice for us to build optimal prediction models, since we can control the balance between specificity and sensitivity however we wish. In fact, estimating these conditional probabilities can be thought of as the main challenge of machine learning.

## Conditional expectations and loss function

There is a link to the relevant sections of the textbook: [Conditional expectations](https://rafalab.github.io/dsbook/introduction-to-machine-learning.html#conditional-expectations) and [Loss functions](https://rafalab.github.io/dsbook/introduction-to-machine-learning.html#conditional-expectation-minimizes-squared-loss-function)

**Key points**

* Due to the connection between **conditional probabilities** and **conditional expectations**:

we often only use the expectation to denote both the conditional probability and conditional expectation.

* For continuous outcomes, we define a loss function to evaluate the model. The most commonly used one is **MSE (Mean Squared Error)**. The reason why we care about the conditional expectation in machine learning is that the expected value minimizes the MSE:

Due to this property, a succinct description of the main task of machine learning is that we use data to estimate for any set of features. **The main way in which competing machine learning algorithms differ is in their approach to estimating this expectation**.

## Comprehension Check - Conditional Probabilities, Part 1

1. In a previous module, we covered Bayes’ theorem and the Bayesian paradigm. Conditional probabilities are a fundamental part of this previous covered rule.

We first review a simple example to go over conditional probabilities.

Assume a patient comes into the doctor’s office to test whether they have a particular disease.

* The test is positive 85% of the time when tested on a patient with the disease (high sensitivity):
* The test is negative 90% of the time when tested on a healthy patient (high specificity):
* The disease is prevalent in about 2% of the community:

Using Bayes’ theorem, calculate the probability that you have the disease if the test is positive.

The following 4 questions (Q2-Q5) all relate to implementing this calculation using R.

We have a hypothetical population of 1 million individuals with the following conditional probabilities as described below:

* The test is positive 85% of the time when tested on a patient with the disease (high sensitivity):
* The test is negative 90% of the time when tested on a healthy patient (high specificity):
* The disease is prevalent in about 2% of the community:

Here is some sample code to get you started:

# set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

disease <- sample(c(0,1), size=1e6, replace=TRUE, prob=c(0.98,0.02))  
test <- rep(NA, 1e6)  
test[disease==0] <- sample(c(0,1), size=sum(disease==0), replace=TRUE, prob=c(0.90,0.10))  
test[disease==1] <- sample(c(0,1), size=sum(disease==1), replace=TRUE, prob=c(0.15, 0.85))

1. What is the probability that a test is positive?

mean(test)

## [1] 0.114509

1. What is the probability that an individual has the disease if the test is negative?

mean(disease[test==0])

## [1] 0.003461356

1. What is the probability that you have the disease if the test is positive? Remember: calculate the conditional probability the disease is positive assuming a positive test.

mean(disease[test==1]==1)

## [1] 0.1471762

1. Compare the prevalence of disease in people who test positive to the overall prevalence of disease.

If a patient’s test is positive, by how many times does that increase their risk of having the disease? First calculate the probability of having the disease given a positive test, then divide by the probability of having the disease.

mean(disease[test==1]==1)/mean(disease==1)

## [1] 7.389106

## Comprehension Check - Conditional Probabilities, Part 2

1. We are now going to write code to compute conditional probabilities for being male in the heights dataset. Round the heights to the closest inch. Plot the estimated conditional probability .

Part of the code is provided here:

data("heights")  
# MISSING CODE  
 qplot(height, p, data =.)

Which of the following blocks of code can be used to replace **# MISSING CODE** to make the correct plot?

* ☐ A.

heights %>%   
 group\_by(height) %>%  
 summarize(p = mean(sex == "Male")) %>%

* ☐ B.

heights %>%   
 mutate(height = round(height)) %>%  
 group\_by(height) %>%  
 summarize(p = mean(sex == "Female")) %>%

* ☐ C.

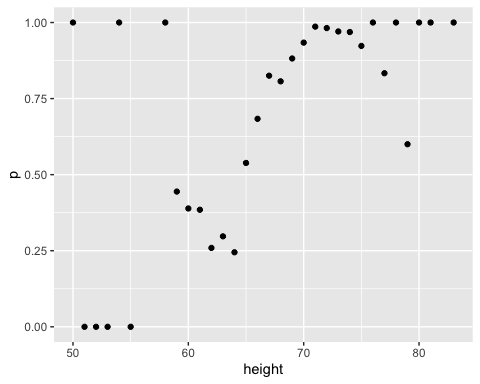
heights %>%   
 mutate(height = round(height)) %>%  
 summarize(p = mean(sex == "Male")) %>%

* ☒ D.

heights %>%   
 mutate(height = round(height)) %>%  
 group\_by(height) %>%  
 summarize(p = mean(sex == "Male")) %>%

data("heights")  
heights %>%   
 mutate(height = round(height)) %>%  
 group\_by(height) %>%  
 summarize(p = mean(sex == "Male")) %>%  
 qplot(height, p, data =.)

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



1. In the plot we just made in Q6 we see high variability for low values of height. This is because we have few data points. This time use the quantile and the cut() function to assure each group has the same number of points. Note that for any numeric vector x, you can create groups based on quantiles like this: cut(x, quantile(x, seq(0, 1, 0.1)), include.lowest = TRUE).

Part of the code is provided here:

ps <- seq(0, 1, 0.1)  
heights %>%   
 # MISSING CODE  
 group\_by(g) %>%  
 summarize(p = mean(sex == "Male"), height = mean(height)) %>%  
 qplot(height, p, data =.)

Which of the following lines of code can be used to replace **# MISSING CODE** to make the correct plot?

* ☐ A.

mutate(g = cut(male, quantile(height, ps), include.lowest = TRUE)) %>%

* ☒ B.

mutate(g = cut(height, quantile(height, ps), include.lowest = TRUE)) %>%

* ☐ C.

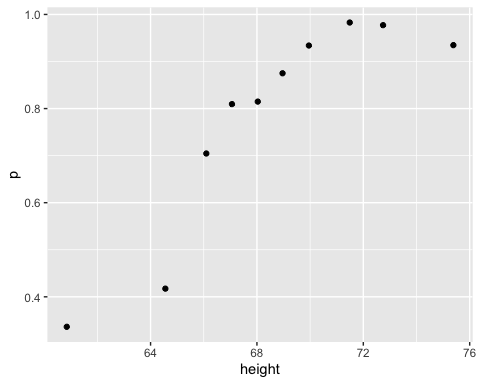
mutate(g = cut(female, quantile(height, ps), include.lowest = TRUE)) %>%

* ☐ D.

mutate(g = cut(height, quantile(height, ps))) %>%

ps <- seq(0, 1, 0.1)  
heights %>%   
 mutate(g = cut(height, quantile(height, ps), include.lowest = TRUE)) %>%  
 group\_by(g) %>%  
 summarize(p = mean(sex == "Male"), height = mean(height)) %>%  
 qplot(height, p, data =.)

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



1. You can generate data from a bivariate normal distrubution using the **MASS** package using the following code:

if(!require(MASS)) install.packages("MASS")

## Loading required package: MASS

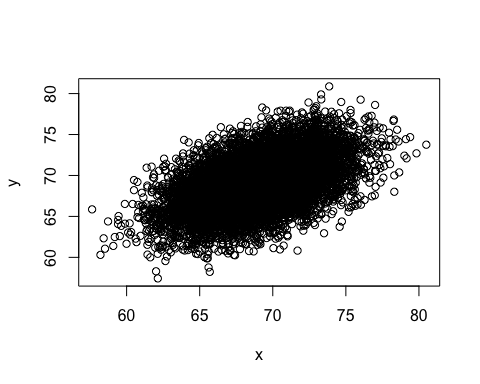
##   
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':  
##   
## select

Sigma <- 9\*matrix(c(1,0.5,0.5,1), 2, 2)  
dat <- MASS::mvrnorm(n = 10000, c(69, 69), Sigma) %>%  
 data.frame() %>% setNames(c("x", "y"))

And you can make a quick plot using plot(dat).

plot(dat)



Using an approach similar to that used in the previous exercise, let’s estimate the conditional expectations and make a plot. Part of the code has again been provided for you:

ps <- seq(0, 1, 0.1)  
dat %>%   
 # MISSING CODE  
 qplot(x, y, data =.)

Which of the following blocks of code can be used to replace **# MISSING CODE** to make the correct plot?

* ☒ A.

mutate(g = cut(x, quantile(x, ps), include.lowest = TRUE)) %>%  
group\_by(g) %>%  
summarize(y = mean(y), x = mean(x)) %>%

* ☐ B.

mutate(g = cut(x, quantile(x, ps))) %>%  
group\_by(g) %>%  
summarize(y = mean(y), x = mean(x)) %>%

* ☐ C.

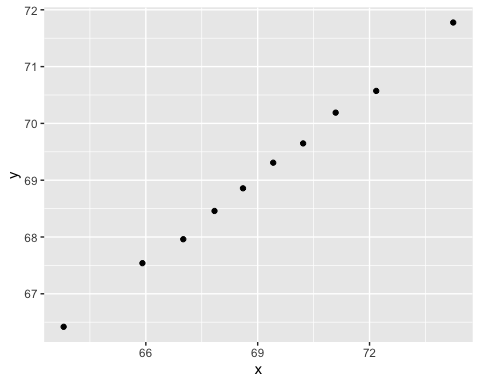
mutate(g = cut(x, quantile(x, ps), include.lowest = TRUE)) %>%  
summarize(y = mean(y), x = mean(x)) %>%

* ☐ D.

mutate(g = cut(x, quantile(x, ps), include.lowest = TRUE)) %>%  
group\_by(g) %>%  
summarize(y =(y), x =(x)) %>%

ps <- seq(0, 1, 0.1)  
dat %>%   
 mutate(g = cut(x, quantile(x, ps), include.lowest = TRUE)) %>%  
 group\_by(g) %>%  
 summarize(y = mean(y), x = mean(x)) %>%  
 qplot(x, y, data =.)

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



# Section 3 - Linear Regression for Prediction, Smoothing, and Working with Matrices Overview

In the **Linear Regression for Prediction, Smoothing, and Working with Matrices Overview** section, you will 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.

After completing this section, you will be able to:

* Use **linear regression for prediction** as a baseline approach.
* Use **logistic regression** for categorical data.
* Detect trends in noisy data using **smoothing** (also known as **curve fitting** or **low pass filtering**).
* Convert predictors to **matrices** and outcomes to **vectors** when all predictors are numeric (or can be converted to numerics in a meaningful way).
* Perform basic **matrix algebra** calculations.

This section has three parts: **linear regression for prediction**, **smoothing**, and **working with matrices**.

## Linear Regression for Prediction

There is a link to the relevant section of the textbook: [Linear regression for prediction](https://rafalab.github.io/dsbook/examples-of-algorithms.html#linear-regression)

**Key points**

* Linear regression can be considered a machine learning algorithm. Although it can be too rigid to be useful, it works rather well for some challenges. It also serves as a baseline approach: if you can’t beat it with a more complex approach, you probably want to stick to linear regression.

*Code*

Note: the seed was not set before createDataPartition so your results may be different.

if(!require(HistData)) install.packages("HistData")

## Loading required package: HistData

library(HistData)  
  
galton\_heights <- GaltonFamilies %>%  
 filter(childNum == 1 & gender == "male") %>%  
 dplyr::select(father, childHeight) %>%  
 rename(son = childHeight)  
  
y <- galton\_heights$son  
test\_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)  
  
train\_set <- galton\_heights %>% slice(-test\_index)  
test\_set <- galton\_heights %>% slice(test\_index)  
  
avg <- mean(train\_set$son)  
avg

## [1] 70.50114

mean((avg - test\_set$son)^2)

## [1] 6.034931

# fit linear regression model  
fit <- lm(son ~ father, data = train\_set)  
fit$coef

## (Intercept) father   
## 34.8934373 0.5170499

y\_hat <- fit$coef[1] + fit$coef[2]\*test\_set$father  
mean((y\_hat - test\_set$son)^2)

## [1] 4.632629

## Predict Function

There is a link to the relevant section of the textbook: [Predict function](https://rafalab.github.io/dsbook/examples-of-algorithms.html#the-predict-function)

**Key points**

* The predict() function takes a fitted object from functions such as lm() or glm() and a data frame with the new predictors for which to predict. We can use predict like this:

y\_hat <- predict(fit, test\_set)

* predict() is a generic function in R that calls other functions depending on what kind of object it receives. To learn about the specifics, you can read the help files using code like this:

?predict.lm # or ?predict.glm

*Code*

y\_hat <- predict(fit, test\_set)  
mean((y\_hat - test\_set$son)^2)

## [1] 4.632629

# read help files  
?predict.lm  
?predict.glm

## Comprehension Check - Linear Regression

1. Create a data set using the following code:

# set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

n <- 100  
Sigma <- 9\*matrix(c(1.0, 0.5, 0.5, 1.0), 2, 2)  
dat <- MASS::mvrnorm(n = 100, c(69, 69), Sigma) %>%  
 data.frame() %>% setNames(c("x", "y"))

We will build 100 linear models using the data above and calculate the mean and standard deviation of the combined models. First, set the seed to 1 again (make sure to use sample.kind="Rounding" if your R is version 3.6 or later). Then, within a replicate() loop, (1) partition the dataset into test and training sets with p = 0.5 and using dat$y to generate your indices, (2) train a linear model predicting y from x, (3) generate predictions on the test set, and (4) calculate the RMSE of that model. Then, report the mean and standard deviation (SD) of the RMSEs from all 100 models.

Report all answers to at least 3 significant digits.

# set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

rmse <- replicate(100, {  
 test\_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)  
 train\_set <- dat %>% slice(-test\_index)  
 test\_set <- dat %>% slice(test\_index)  
 fit <- lm(y ~ x, data = train\_set)  
 y\_hat <- predict(fit, newdata = test\_set)  
 sqrt(mean((y\_hat-test\_set$y)^2))  
})  
  
mean(rmse)

## [1] 2.488661

sd(rmse)

## [1] 0.1243952

1. Now we will repeat the exercise above but using larger datasets. Write a function that takes a size n, then (1) builds a dataset using the code provided at the top of Q1 but with n observations instead of 100 and without the set.seed(1), (2) runs the replicate() loop that you wrote to answer Q1, which builds 100 linear models and returns a vector of RMSEs, and (3) calculates the mean and standard deviation of the 100 RMSEs.

Set the seed to 1 (if using R 3.6 or later, use the argument sample.kind="Rounding") and then use sapply() or map() to apply your new function to n <- c(100, 500, 1000, 5000, 10000).

Hint: You only need to set the seed once before running your function; do not set a seed within your function. Also be sure to use sapply() or map() as you will get different answers running the simulations individually due to setting the seed.

# set.seed(1) # if R 3.5 or earlier  
set.seed(1, sample.kind="Rounding") # if R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

n <- c(100, 500, 1000, 5000, 10000)  
res <- sapply(n, function(n){  
 Sigma <- 9\*matrix(c(1.0, 0.5, 0.5, 1.0), 2, 2)  
 dat <- MASS::mvrnorm(n, c(69, 69), Sigma) %>%  
 data.frame() %>% setNames(c("x", "y"))  
 rmse <- replicate(100, {  
 test\_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)  
 train\_set <- dat %>% slice(-test\_index)  
 test\_set <- dat %>% slice(test\_index)  
 fit <- lm(y ~ x, data = train\_set)  
 y\_hat <- predict(fit, newdata = test\_set)  
 sqrt(mean((y\_hat-test\_set$y)^2))  
 })  
 c(avg = mean(rmse), sd = sd(rmse))  
})  
  
res

## [,1] [,2] [,3] [,4] [,5]  
## avg 2.4977540 2.72095125 2.55554451 2.62482800 2.61844227  
## sd 0.1180821 0.08002108 0.04560258 0.02309673 0.01689205

1. What happens to the RMSE as the size of the dataset becomes larger?

* ☒ A. On average, the RMSE does not change much as n gets larger, but the variability of the RMSE decreases.
* ☐ B. Because of the law of large numbers the RMSE decreases; more data means more precise estimates.
* ☐ C. n = 10000 is not sufficiently large. To see a decrease in the RMSE we would need to make it larger.
* ☐ D. The RMSE is not a random variable.

1. Now repeat the exercise from Q1, this time making the correlation between x and y larger, as in the following code:

# set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

n <- 100  
Sigma <- 9\*matrix(c(1.0, 0.95, 0.95, 1.0), 2, 2)  
dat <- MASS::mvrnorm(n = 100, c(69, 69), Sigma) %>%  
 data.frame() %>% setNames(c("x", "y"))

Note what happens to RMSE - set the seed to 1 as before.

# set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

rmse <- replicate(100, {  
 test\_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)  
 train\_set <- dat %>% slice(-test\_index)  
 test\_set <- dat %>% slice(test\_index)  
 fit <- lm(y ~ x, data = train\_set)  
 y\_hat <- predict(fit, newdata = test\_set)  
 sqrt(mean((y\_hat-test\_set$y)^2))  
})  
  
mean(rmse)

## [1] 0.9099808

sd(rmse)

## [1] 0.06244347

1. Which of the following best explains why the RMSE in question 4 is so much lower than the RMSE in question 1?

* ☐ A. It is just luck. If we do it again, it will be larger.
* ☐ B. The central limit theorem tells us that the RMSE is normal.
* ☒ C. When we increase the correlation between x and y, x has more predictive power and thus provides a better estimate of y.
* ☐ D. These are both examples of regression so the RMSE has to be the same.

1. Create a data set using the following code.

# set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

Sigma <- matrix(c(1.0, 0.75, 0.75, 0.75, 1.0, 0.25, 0.75, 0.25, 1.0), 3, 3)  
dat <- MASS::mvrnorm(n = 100, c(0, 0, 0), Sigma) %>%  
 data.frame() %>% setNames(c("y", "x\_1", "x\_2"))

Note that y is correlated with both x\_1 and x\_2 but the two predictors are independent of each other, as seen by cor(dat).

Set the seed to 1, then use the **caret** package to partition into test and training sets with p = 0.5. Compare the RMSE when using just x\_1, just x\_2 and both x\_1 and x\_2. Train a single linear model for each (not 100 like in the previous questions).

Which of the three models performs the best (has the lowest RMSE)?

# set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

test\_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)  
train\_set <- dat %>% slice(-test\_index)  
test\_set <- dat %>% slice(test\_index)  
  
fit <- lm(y ~ x\_1, data = train\_set)  
y\_hat <- predict(fit, newdata = test\_set)  
sqrt(mean((y\_hat-test\_set$y)^2))

## [1] 0.600666

fit <- lm(y ~ x\_2, data = train\_set)  
y\_hat <- predict(fit, newdata = test\_set)  
sqrt(mean((y\_hat-test\_set$y)^2))

## [1] 0.630699

fit <- lm(y ~ x\_1 + x\_2, data = train\_set)  
y\_hat <- predict(fit, newdata = test\_set)  
sqrt(mean((y\_hat-test\_set$y)^2))

## [1] 0.3070962

* ☐ A. x\_1
* ☐ B. x\_2
* ☒ C. x\_1 and x\_2

1. Report the lowest RMSE of the three models tested in Q6.

fit <- lm(y ~ x\_1 + x\_2, data = train\_set)  
y\_hat <- predict(fit, newdata = test\_set)  
sqrt(mean((y\_hat-test\_set$y)^2))

## [1] 0.3070962

1. Repeat the exercise from Q6 but now create an example in which x\_1 and x\_2 are highly correlated.

# set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

Sigma <- matrix(c(1.0, 0.75, 0.75, 0.75, 1.0, 0.95, 0.75, 0.95, 1.0), 3, 3)  
dat <- MASS::mvrnorm(n = 100, c(0, 0, 0), Sigma) %>%  
 data.frame() %>% setNames(c("y", "x\_1", "x\_2"))

Set the seed to 1, then use the **caret** package to partition into a test and training set of equal size. Compare the RMSE when using just x\_1, just x\_2, and both x\_1 and x\_2.

Compare the results from Q6 and Q8. What can you conclude?

# set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

test\_index <- createDataPartition(dat$y, times = 1, p = 0.5, list = FALSE)  
train\_set <- dat %>% slice(-test\_index)  
test\_set <- dat %>% slice(test\_index)  
  
fit <- lm(y ~ x\_1, data = train\_set)  
y\_hat <- predict(fit, newdata = test\_set)  
sqrt(mean((y\_hat-test\_set$y)^2))

## [1] 0.6592608

fit <- lm(y ~ x\_2, data = train\_set)  
y\_hat <- predict(fit, newdata = test\_set)  
sqrt(mean((y\_hat-test\_set$y)^2))

## [1] 0.640081

fit <- lm(y ~ x\_1 + x\_2, data = train\_set)  
y\_hat <- predict(fit, newdata = test\_set)  
sqrt(mean((y\_hat-test\_set$y)^2))

## [1] 0.6597865

* ☐ A. Unless we include all predictors we have no predictive power.
* ☐ B. Adding extra predictors improves RMSE regardless of whether the added predictors are correlated with other predictors or not.
* ☐ C. Adding extra predictors results in over fitting.
* ☒ D. Adding extra predictors can improve RMSE substantially, but not when the added predictors are highly correlated with other predictors.

## Regression for a Categorical Outcome

There is a link to the relevant section of the textbook: [Regression for a categorical outcome](https://rafalab.github.io/dsbook/examples-of-algorithms.html#logistic-regression)

**Key points**

* The regression approach can be extended to categorical data. For example, we can try regression to estimate the conditional probability:
* Once we have estimates and , we can obtain an actual prediction . Then we can define a specific decision rule to form a prediction.

*Code*

data("heights")  
y <- heights$height  
  
set.seed(2) #if you are using R 3.5 or earlier  
set.seed(2, sample.kind = "Rounding") #if you are using R 3.6 or later

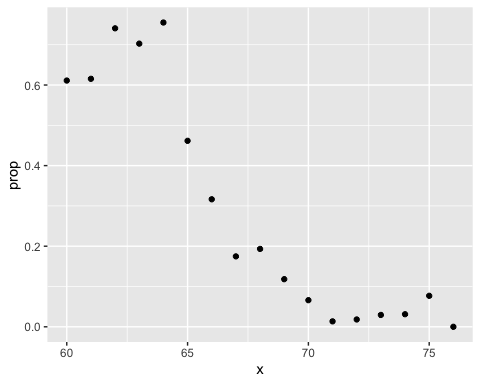
## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

test\_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)  
train\_set <- heights %>% slice(-test\_index)  
test\_set <- heights %>% slice(test\_index)  
  
train\_set %>%   
 filter(round(height)==66) %>%  
 summarize(y\_hat = mean(sex=="Female"))

## y\_hat  
## 1 0.2424242

heights %>%   
 mutate(x = round(height)) %>%  
 group\_by(x) %>%  
 filter(n() >= 10) %>%  
 summarize(prop = mean(sex == "Female")) %>%  
 ggplot(aes(x, prop)) +  
 geom\_point()

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



lm\_fit <- mutate(train\_set, y = as.numeric(sex == "Female")) %>% lm(y ~ height, data = .)  
p\_hat <- predict(lm\_fit, test\_set)  
y\_hat <- ifelse(p\_hat > 0.5, "Female", "Male") %>% factor()  
confusionMatrix(y\_hat, test\_set$sex)$overall["Accuracy"]

## Accuracy   
## 0.7851711

## Logistic Regression

There is a link to the relevant section of the textbook: [Logistic regression](https://rafalab.github.io/dsbook/examples-of-algorithms.html#logistic-regression)

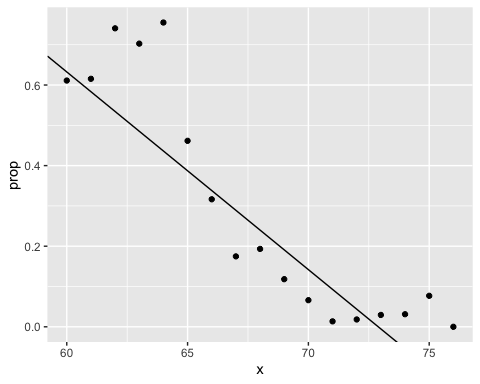
**Key points**

* **Logistic regression** is an extension of linear regression that assures that the estimate of conditional probability is between 0 and 1. This approach makes use of the logistic transformation:
* With logistic regression, we model the conditional probability directly with:
* Note that with this model, we can no longer use least squares. Instead we compute the **maximum likelihood estimate (MLE)**.
* In R, we can fit the logistic regression model with the function glm() (generalized linear models). If we want to compute the conditional probabilities, we want type="response" since the default is to return the logistic transformed values.

*Code*

heights %>%   
 mutate(x = round(height)) %>%  
 group\_by(x) %>%  
 filter(n() >= 10) %>%  
 summarize(prop = mean(sex == "Female")) %>%  
 ggplot(aes(x, prop)) +  
 geom\_point() +   
 geom\_abline(intercept = lm\_fit$coef[1], slope = lm\_fit$coef[2])

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



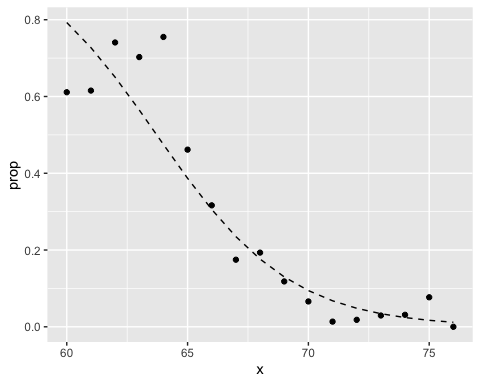
range(p\_hat)

## [1] -0.397868 1.123309

# fit logistic regression model  
glm\_fit <- train\_set %>%   
 mutate(y = as.numeric(sex == "Female")) %>%  
 glm(y ~ height, data=., family = "binomial")  
  
p\_hat\_logit <- predict(glm\_fit, newdata = test\_set, type = "response")  
  
tmp <- heights %>%   
 mutate(x = round(height)) %>%  
 group\_by(x) %>%  
 filter(n() >= 10) %>%  
 summarize(prop = mean(sex == "Female"))

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

logistic\_curve <- data.frame(x = seq(min(tmp$x), max(tmp$x))) %>%  
 mutate(p\_hat = plogis(glm\_fit$coef[1] + glm\_fit$coef[2]\*x))  
tmp %>%   
 ggplot(aes(x, prop)) +  
 geom\_point() +  
 geom\_line(data = logistic\_curve, mapping = aes(x, p\_hat), lty = 2)



y\_hat\_logit <- ifelse(p\_hat\_logit > 0.5, "Female", "Male") %>% factor  
confusionMatrix(y\_hat\_logit, test\_set$sex)$overall[["Accuracy"]]

## [1] 0.7984791

## Case Study: 2 or 7

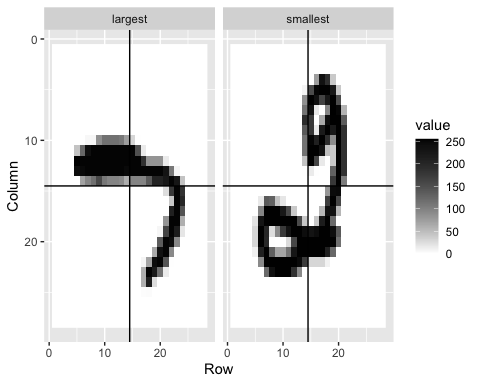
There is a link to the relevant section of the textbook: [Case study: 2 or 7](https://rafalab.github.io/dsbook/examples-of-algorithms.html#logistic-regression-with-more-than-one-predictor)

**Key points**

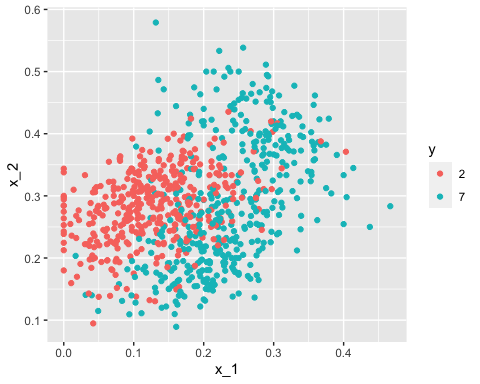
* In this case study we apply logistic regression to classify whether a digit is two or seven. We are interested in estimating a conditional probability that depends on two variables:
* Through this case, we know that logistic regression forces our estimates to be a **plane** and our boundary to be a **line**. This implies that a logistic regression approach has no chance of capturing the **non-linear** nature of the true . Therefore, we need other more flexible methods that permit other shapes.

*Code*

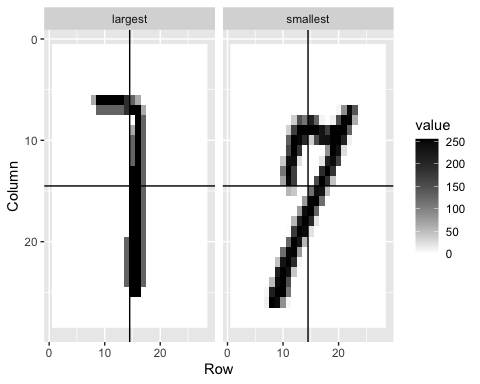
mnist <- read\_mnist()  
is <- mnist\_27$index\_train[c(which.min(mnist\_27$train$x\_1), which.max(mnist\_27$train$x\_1))]  
titles <- c("smallest","largest")  
tmp <- lapply(1:2, function(i){  
 expand.grid(Row=1:28, Column=1:28) %>%  
 mutate(label=titles[i],  
 value = mnist$train$images[is[i],])  
})  
tmp <- Reduce(rbind, tmp)  
tmp %>% ggplot(aes(Row, Column, fill=value)) +  
 geom\_raster() +  
 scale\_y\_reverse() +  
 scale\_fill\_gradient(low="white", high="black") +  
 facet\_grid(.~label) +  
 geom\_vline(xintercept = 14.5) +  
 geom\_hline(yintercept = 14.5)



data("mnist\_27")  
mnist\_27$train %>% ggplot(aes(x\_1, x\_2, color = y)) + geom\_point()



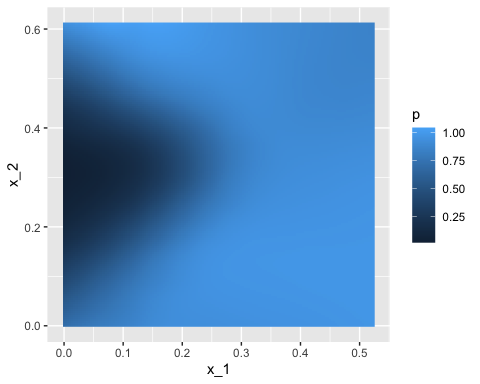
is <- mnist\_27$index\_train[c(which.min(mnist\_27$train$x\_2), which.max(mnist\_27$train$x\_2))]  
titles <- c("smallest","largest")  
tmp <- lapply(1:2, function(i){  
 expand.grid(Row=1:28, Column=1:28) %>%  
 mutate(label=titles[i],  
 value = mnist$train$images[is[i],])  
})  
tmp <- Reduce(rbind, tmp)  
tmp %>% ggplot(aes(Row, Column, fill=value)) +  
 geom\_raster() +  
 scale\_y\_reverse() +  
 scale\_fill\_gradient(low="white", high="black") +  
 facet\_grid(.~label) +  
 geom\_vline(xintercept = 14.5) +  
 geom\_hline(yintercept = 14.5)



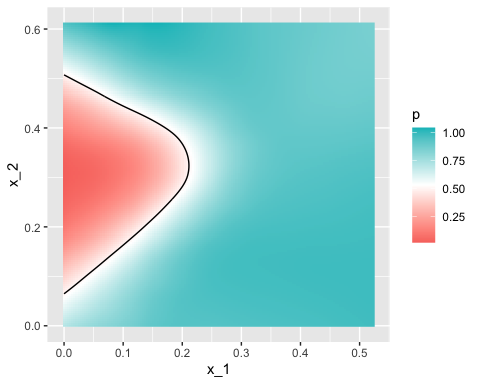
fit\_glm <- glm(y ~ x\_1 + x\_2, data=mnist\_27$train, family = "binomial")  
p\_hat\_glm <- predict(fit\_glm, mnist\_27$test)  
y\_hat\_glm <- factor(ifelse(p\_hat\_glm > 0.5, 7, 2))  
confusionMatrix(data = y\_hat\_glm, reference = mnist\_27$test$y)$overall["Accuracy"]

## Accuracy   
## 0.76

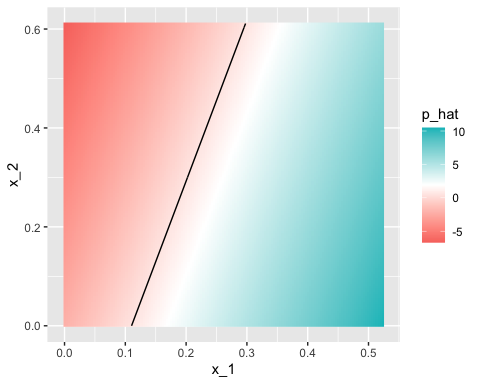
mnist\_27$true\_p %>% ggplot(aes(x\_1, x\_2, fill=p)) +  
 geom\_raster()



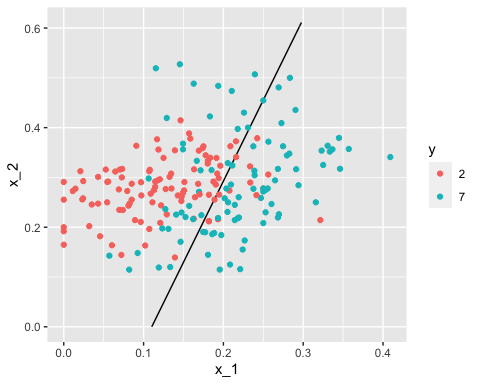
mnist\_27$true\_p %>% ggplot(aes(x\_1, x\_2, z=p, fill=p)) +  
 geom\_raster() +  
 scale\_fill\_gradientn(colors=c("#F8766D","white","#00BFC4")) +  
 stat\_contour(breaks=c(0.5), color="black")



p\_hat <- predict(fit\_glm, newdata = mnist\_27$true\_p)  
mnist\_27$true\_p %>%  
 mutate(p\_hat = p\_hat) %>%  
 ggplot(aes(x\_1, x\_2, z=p\_hat, fill=p\_hat)) +  
 geom\_raster() +  
 scale\_fill\_gradientn(colors=c("#F8766D","white","#00BFC4")) +  
 stat\_contour(breaks=c(0.5),color="black")



p\_hat <- predict(fit\_glm, newdata = mnist\_27$true\_p)  
mnist\_27$true\_p %>%  
 mutate(p\_hat = p\_hat) %>%  
 ggplot() +  
 stat\_contour(aes(x\_1, x\_2, z=p\_hat), breaks=c(0.5), color="black") +  
 geom\_point(mapping = aes(x\_1, x\_2, color=y), data = mnist\_27$test)



## Comprehension Check - Logistic Regression

1. Define a dataset using the following code:

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

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

make\_data <- function(n = 1000, p = 0.5,   
 mu\_0 = 0, mu\_1 = 2,   
 sigma\_0 = 1, sigma\_1 = 1){  
  
y <- rbinom(n, 1, p)  
f\_0 <- rnorm(n, mu\_0, sigma\_0)  
f\_1 <- rnorm(n, mu\_1, sigma\_1)  
x <- ifelse(y == 1, f\_1, f\_0)  
   
test\_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)  
  
list(train = data.frame(x = x, y = as.factor(y)) %>% slice(-test\_index),  
 test = data.frame(x = x, y = as.factor(y)) %>% slice(test\_index))  
}  
dat <- make\_data()

Note that we have defined a variable x that is predictive of a binary outcome y:

dat$train %>% ggplot(aes(x, color = y)) + geom\_density().

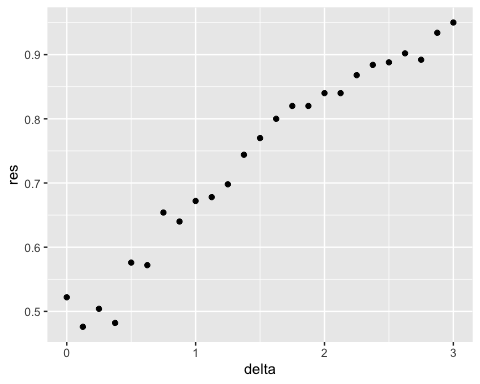
Set the seed to 1, then use the make\_data() function defined above to generate 25 different datasets with mu\_1 <- seq(0, 3, len=25). Perform logistic regression on each of the 25 different datasets (predict 1 if p > 0.5) and plot accuracy (res in the figures) vs mu\_1 (delta in the figures).

Which is the correct plot?

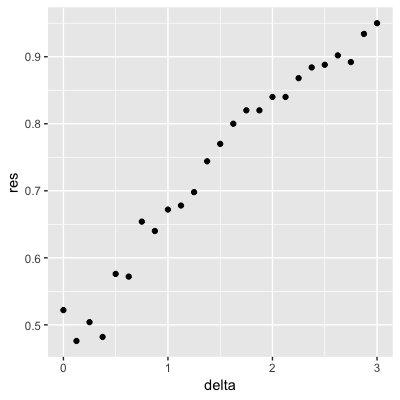
set.seed(1) #if you are using R 3.5 or earlier  
set.seed(1, sample.kind="Rounding") #if you are using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

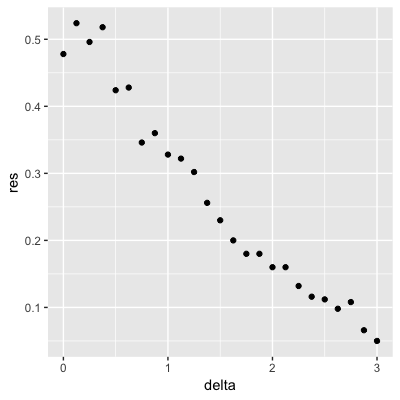
delta <- seq(0, 3, len = 25)  
res <- sapply(delta, function(d){  
 dat <- make\_data(mu\_1 = d)  
 fit\_glm <- dat$train %>% glm(y ~ x, family = "binomial", data = .)  
 y\_hat\_glm <- ifelse(predict(fit\_glm, dat$test) > 0.5, 1, 0) %>% factor(levels = c(0, 1))  
 mean(y\_hat\_glm == dat$test$y)  
})  
qplot(delta, res)



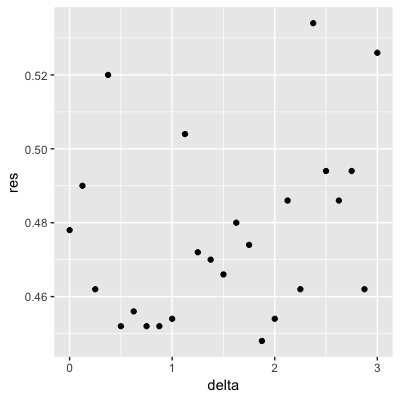
* ☒ A.



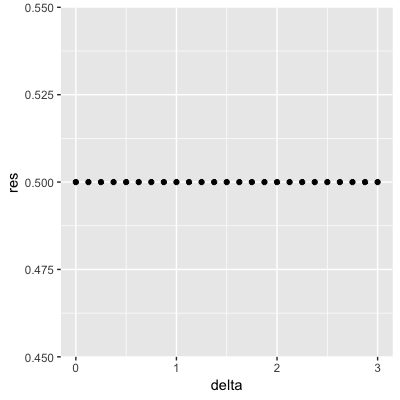
* ☐ B.



* ☐ C.



* ☐ D.



## Introduction to Smoothing

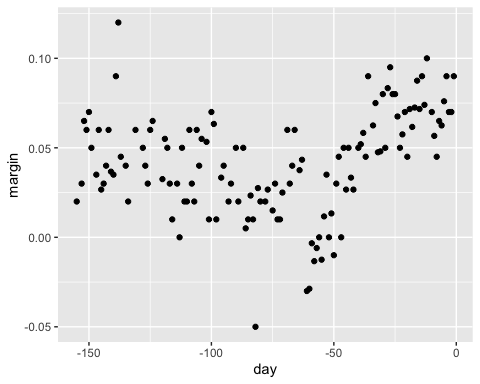
There is a link to the relevant section of the textbook: [Smoothing](https://rafalab.github.io/dsbook/smoothing.html)

**Key points**

* **Smoothing** is a very powerful technique used all across data analysis. It is designed to detect trends in the presence of noisy data in cases in which the shape of the trend is unknown.
* The concepts behind smoothing techniques are extremely useful in machine learning because **conditional expectations/probabilities** can be thought of as **trends** of unknown shapes that we need to estimate in the presence of uncertainty.

*Code*

data("polls\_2008")  
qplot(day, margin, data = polls\_2008)



## Bin Smoothing and Kernels

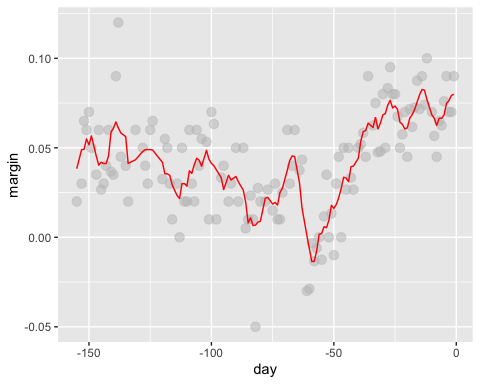
There is a link to the relevant sections of the textbook: [Bin smoothing](https://rafalab.github.io/dsbook/smoothing.html#bin-smoothing) and [Kernels](https://rafalab.github.io/dsbook/smoothing.html#kernels)

**Key points**

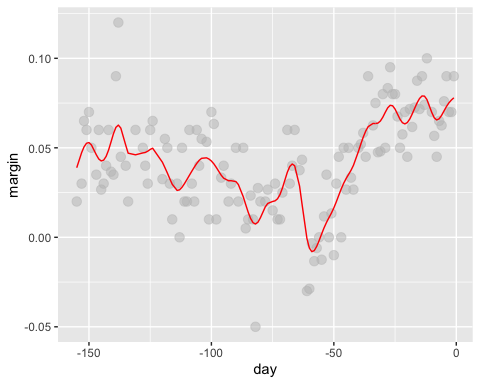
* The general idea of smoothing is to group data points into strata in which the value of can be assumed to be constant. We can make this assumption because we think changes slowly and, as a result, is almost constant in small windows of time.
* This assumption implies that a good estimate for is the average of the values in the window. The estimate is:
* In smoothing, we call the size of the interval satisfying the particular condition the window size, bandwidth or span.

*Code*

# bin smoothers  
span <- 7   
fit <- with(polls\_2008,ksmooth(day, margin, x.points = day, kernel="box", bandwidth =span))  
polls\_2008 %>% mutate(smooth = fit$y) %>%  
 ggplot(aes(day, margin)) +  
 geom\_point(size = 3, alpha = .5, color = "grey") +   
 geom\_line(aes(day, smooth), color="red")



# kernel  
span <- 7  
fit <- with(polls\_2008, ksmooth(day, margin, x.points = day, kernel="normal", bandwidth = span))  
polls\_2008 %>% mutate(smooth = fit$y) %>%  
 ggplot(aes(day, margin)) +  
 geom\_point(size = 3, alpha = .5, color = "grey") +   
 geom\_line(aes(day, smooth), color="red")



## Local Weighted Regression (loess)

There is a link to the relevant section of the textbook: [Local weighted regression](https://rafalab.github.io/dsbook/smoothing.html#local-weighted-regression-loess)

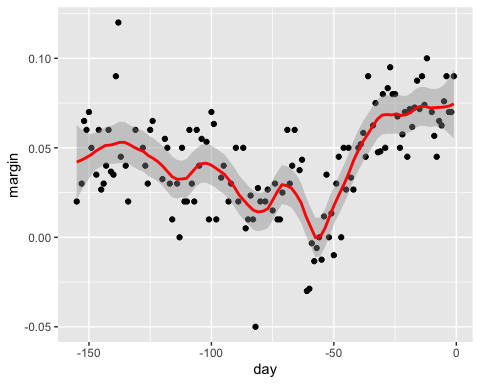
**Key points**

* A limitation of the bin smoothing approach is that we need small windows for the approximately constant assumptions to hold which may lead to imprecise estimates of . **Local weighted regression (loess)** permits us to consider larger window sizes.
* One important difference between loess and bin smoother is that we assume the smooth function is locally **linear** in a window instead of constant.
* The result of loess is a smoother fit than bin smoothing because we use larger sample sizes to estimate our local parameters.

*Code*

polls\_2008 %>% ggplot(aes(day, margin)) +  
 geom\_point() +   
 geom\_smooth(color="red", span = 0.15, method = "loess", method.args = list(degree=1))

## `geom\_smooth()` using formula 'y ~ x'



## Comprehension Check - Smoothing

1. In the Wrangling course of this series, PH125.6x, we used the following code to obtain mortality counts for Puerto Rico for 2015-2018:

if(!require(purrr)) install.packages("purrr")  
if(!require(pdftools)) install.packages("pdftools")

## Loading required package: pdftools

## Using poppler version 0.73.0

library(tidyverse)  
library(lubridate)  
library(purrr)  
library(pdftools)  
   
fn <- system.file("extdata", "RD-Mortality-Report\_2015-18-180531.pdf", package="dslabs")  
dat <- map\_df(str\_split(pdf\_text(fn), "\n"), function(s){  
 s <- str\_trim(s)  
 header\_index <- str\_which(s, "2015")[1]  
 tmp <- str\_split(s[header\_index], "\\s+", simplify = TRUE)  
 month <- tmp[1]  
 header <- tmp[-1]  
 tail\_index <- str\_which(s, "Total")  
 n <- str\_count(s, "\\d+")  
 out <- c(1:header\_index, which(n==1), which(n>=28), tail\_index:length(s))  
 s[-out] %>%  
 str\_remove\_all("[^\\d\\s]") %>%  
 str\_trim() %>%  
 str\_split\_fixed("\\s+", n = 6) %>%  
 .[,1:5] %>%  
 as\_data\_frame() %>%   
 setNames(c("day", header)) %>%  
 mutate(month = month,  
 day = as.numeric(day)) %>%  
 gather(year, deaths, -c(day, month)) %>%  
 mutate(deaths = as.numeric(deaths))  
}) %>%  
 mutate(month = recode(month, "JAN" = 1, "FEB" = 2, "MAR" = 3, "APR" = 4, "MAY" = 5, "JUN" = 6,   
 "JUL" = 7, "AGO" = 8, "SEP" = 9, "OCT" = 10, "NOV" = 11, "DEC" = 12)) %>%  
 mutate(date = make\_date(year, month, day)) %>%  
 dplyr::filter(date <= "2018-05-01")

## Warning: `as\_data\_frame()` is deprecated as of tibble 2.0.0.  
## Please use `as\_tibble()` instead.  
## The signature and semantics have changed, see `?as\_tibble`.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_warnings()` to see where this warning was generated.

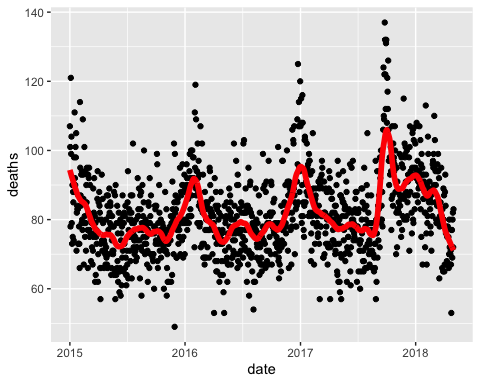
## Warning: The `x` argument of `as\_tibble.matrix()` must have unique column names if `.name\_repair` is omitted as of tibble 2.0.0.  
## Using compatibility `.name\_repair`.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_warnings()` to see where this warning was generated.

Use the loess() function to obtain a smooth estimate of the expected number of deaths as a function of date. Plot this resulting smooth function. Make the span about two months long.

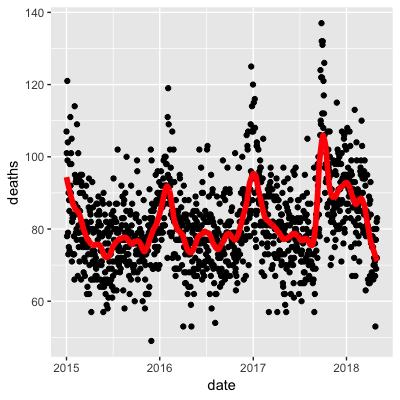
Which of the following plots is correct?

span <- 60 / as.numeric(diff(range(dat$date)))  
fit <- dat %>% mutate(x = as.numeric(date)) %>% loess(deaths ~ x, data = ., span = span, degree = 1)  
dat %>% mutate(smooth = predict(fit, as.numeric(date))) %>%  
 ggplot() +  
 geom\_point(aes(date, deaths)) +  
 geom\_line(aes(date, smooth), lwd = 2, col = "red")

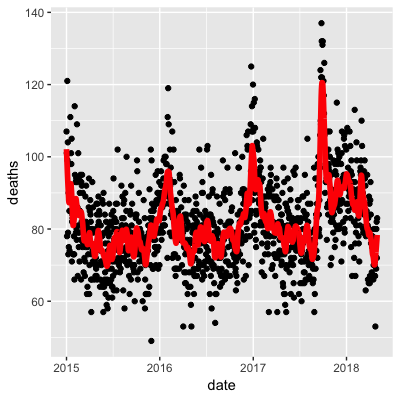
## Warning: Removed 1 rows containing missing values (geom\_point).



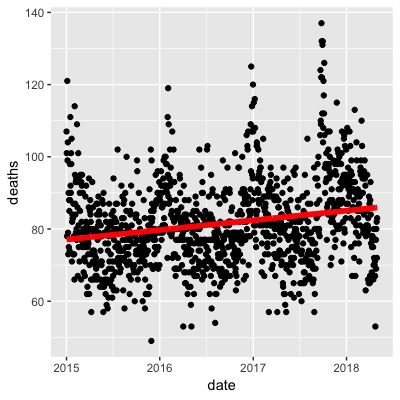
* ☒ A.



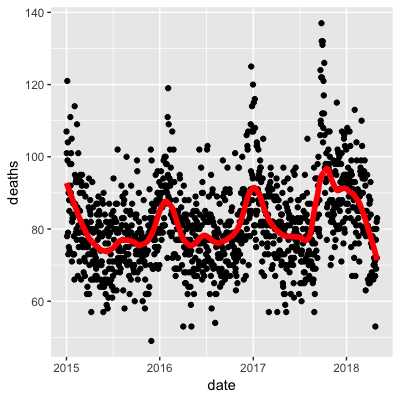
* ☐ B.



* ☐ C.



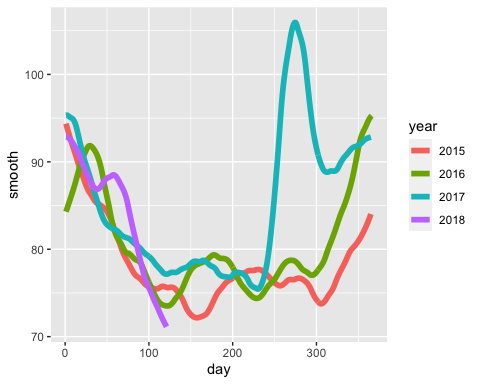
* ☐ D.



1. Work with the same data as in Q1 to plot smooth estimates against day of the year, all on the same plot, but with different colors for each year.

Which code produces the desired plot?

dat %>%   
 mutate(smooth = predict(fit, as.numeric(date)), day = yday(date), year = as.character(year(date))) %>%  
 ggplot(aes(day, smooth, col = year)) +  
 geom\_line(lwd = 2)



* ☐ A.

dat %>%   
 mutate(smooth = predict(fit), day = yday(date), year = as.character(year(date))) %>%  
 ggplot(aes(day, smooth, col = year)) +  
 geom\_line(lwd = 2)

* ☐ B.

dat %>%   
 mutate(smooth = predict(fit, as.numeric(date)), day = mday(date), year = as.character(year(date))) %>%  
 ggplot(aes(day, smooth, col = year)) +  
 geom\_line(lwd = 2)

* ☐ C.

dat %>%   
 mutate(smooth = predict(fit, as.numeric(date)), day = yday(date), year = as.character(year(date))) %>%  
 ggplot(aes(day, smooth)) +  
 geom\_line(lwd = 2)

* ☒ D.

dat %>%   
 mutate(smooth = predict(fit, as.numeric(date)), day = yday(date), year = as.character(year(date))) %>%  
 ggplot(aes(day, smooth, col = year)) +  
 geom\_line(lwd = 2)

1. Suppose we want to predict 2s and 7s in the mnist\_27 dataset with just the second covariate. Can we do this? On first inspection it appears the data does not have much predictive power.

In fact, if we fit a regular logistic regression the coefficient for x\_2 is not significant!

This can be seen using this code:

if(!require(broom)) install.packages("broom")

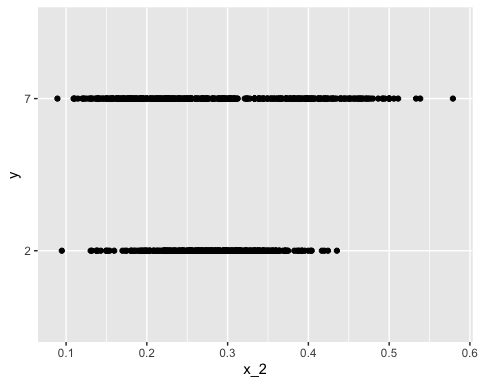
## Loading required package: broom

library(broom)  
mnist\_27$train %>% glm(y ~ x\_2, family = "binomial", data = .) %>% tidy()

## # A tibble: 2 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) -0.0907 0.247 -0.368 0.713  
## 2 x\_2 0.685 0.827 0.829 0.407

Plotting a scatterplot here is not useful since y is binary:

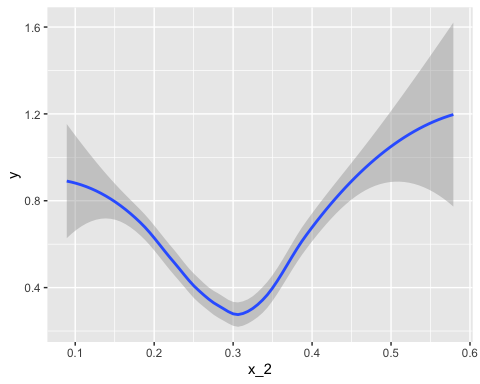
qplot(x\_2, y, data = mnist\_27$train)



Fit a loess line to the data above and plot the results. What do you observe?

mnist\_27$train %>%   
 mutate(y = ifelse(y=="7", 1, 0)) %>%  
 ggplot(aes(x\_2, y)) +   
 geom\_smooth(method = "loess")

## `geom\_smooth()` using formula 'y ~ x'



* ☐ A. There is no predictive power and the conditional probability is linear.
* ☐ B. There is no predictive power and the conditional probability is non-linear.
* ☐ C. There is predictive power and the conditional probability is linear.
* ☒ D. There is predictive power and the conditional probability is non-linear.

## Matrices

There is a link to the relevant section of the textbook: [Matrices](https://rafalab.github.io/dsbook/large-datasets.html#matrix-algebra)

**Key points**

* The main reason for using matrices is that certain mathematical operations needed to develop efficient code can be performed using techniques from a branch of mathematics called **linear algebra**.
* **Linear algebra** and **matrix notation** are key elements of the language used in academic papers describing machine learning techniques.

*Code*

if(!exists("mnist")) mnist <- read\_mnist()  
  
class(mnist$train$images)

## [1] "matrix" "array"

x <- mnist$train$images[1:1000,]   
y <- mnist$train$labels[1:1000]

## Matrix Notation

There is a link to the relevant section of the textbook: [Matrix notation](https://rafalab.github.io/dsbook/large-datasets.html#notation-2)

**Key points**

* In matrix algebra, we have three main types of objects: **scalars**, **vectors**, and **matrices**.
  + **Scalar:**
  + **Vector:**
  + **Matrix:**
* In R, we can extract the dimension of a matrix with the function dim(). We can convert a vector into a matrix using the function as.matrix().

*Code*

length(x[,1])

## [1] 1000

x\_1 <- 1:5  
x\_2 <- 6:10  
cbind(x\_1, x\_2)

## x\_1 x\_2  
## [1,] 1 6  
## [2,] 2 7  
## [3,] 3 8  
## [4,] 4 9  
## [5,] 5 10

dim(x)

## [1] 1000 784

dim(x\_1)

## NULL

dim(as.matrix(x\_1))

## [1] 5 1

dim(x)

## [1] 1000 784

## Converting a Vector to a Matrix

There is a link to the relevant section of the textbook: [Converting a vector to a matrix](https://rafalab.github.io/dsbook/large-datasets.html#converting-a-vector-to-a-matrix)

**Key points**

* In R, we can **convert a vector into a matrix** with the matrix() function. The matrix is filled in by column, but we can fill by row by using the byrow argument. The function t() can be used to directly transpose a matrix.
* Note that the matrix function **recycles values in the vector** without warning if the product of columns and rows does not match the length of the vector.

*Code*

my\_vector <- 1:15  
  
# fill the matrix by column  
mat <- matrix(my\_vector, 5, 3)  
mat

## [,1] [,2] [,3]  
## [1,] 1 6 11  
## [2,] 2 7 12  
## [3,] 3 8 13  
## [4,] 4 9 14  
## [5,] 5 10 15

# fill by row  
mat\_t <- matrix(my\_vector, 3, 5, byrow = TRUE)  
mat\_t

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 1 2 3 4 5  
## [2,] 6 7 8 9 10  
## [3,] 11 12 13 14 15

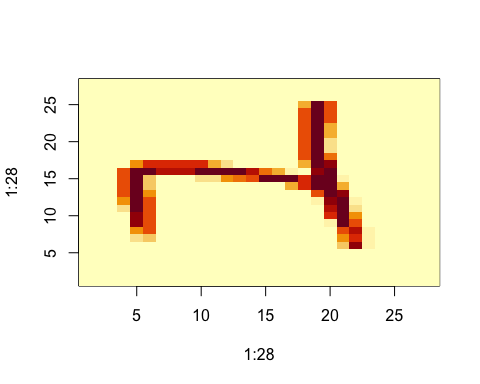
identical(t(mat), mat\_t)

## [1] TRUE

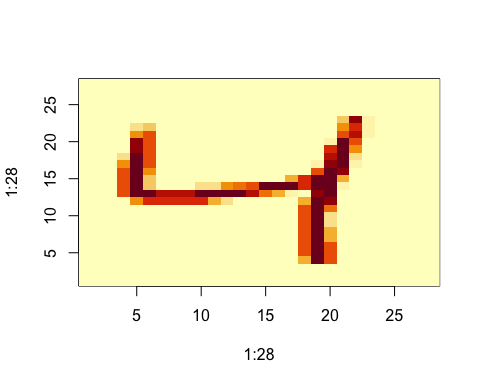
matrix(my\_vector, 5, 5)

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 1 6 11 1 6  
## [2,] 2 7 12 2 7  
## [3,] 3 8 13 3 8  
## [4,] 4 9 14 4 9  
## [5,] 5 10 15 5 10

grid <- matrix(x[3,], 28, 28)  
image(1:28, 1:28, grid)



# flip the image back  
image(1:28, 1:28, grid[, 28:1])



## Row and Column Summaries and Apply

There is a link to the relevant section of the textbook: [Row and column summaries](https://rafalab.github.io/dsbook/large-datasets.html#row-and-column-summaries)

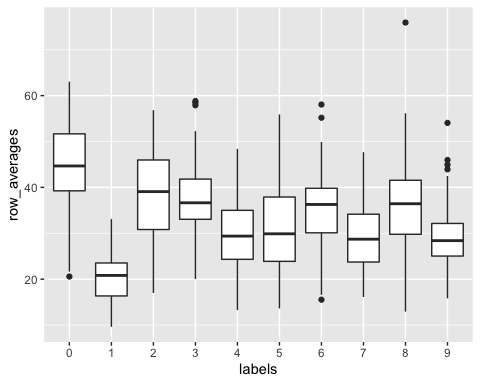
**Key points**

* The function rowSums() computes the sum of each row.
* The function rowMeans() computes the average of each row.
* We can compute the column sums and averages using the functions colSums() and colMeans().
* The **matrixStats** package adds functions that performs operations on each row or column very efficiently, including the functions rowSds() and colSds().
* The apply() function lets you apply any function to a matrix. The first argument is the **matrix**, the second is the **dimension** (1 for rows, 2 for columns), and the third is the **function**.

*Code*

sums <- rowSums(x)  
avg <- rowMeans(x)  
  
data\_frame(labels = as.factor(y), row\_averages = avg) %>%  
 qplot(labels, row\_averages, data = ., geom = "boxplot")

## Warning: `data\_frame()` is deprecated as of tibble 1.1.0.  
## Please use `tibble()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_warnings()` to see where this warning was generated.



avgs <- apply(x, 1, mean)  
sds <- apply(x, 2, sd)

## Filtering Columns Based on Summaries

There is a link to the relevant section of the textbook: [Filtering columns based on summaries](https://rafalab.github.io/dsbook/large-datasets.html#filtering-columns-based-on-summaries)

**Key points**

* The operations used to extract columns: x[,c(351,352)].
* The operations used to extract rows: x[c(2,3),].
* We can also use logical indexes to determine which columns or rows to keep: new\_x <- x[ ,colSds(x) > 60].
* **Important note:** if you select only one column or only one row, the result is no longer a matrix but a **vector**. We can **preserve the matrix class** by using the argument drop=FALSE.

*Code*

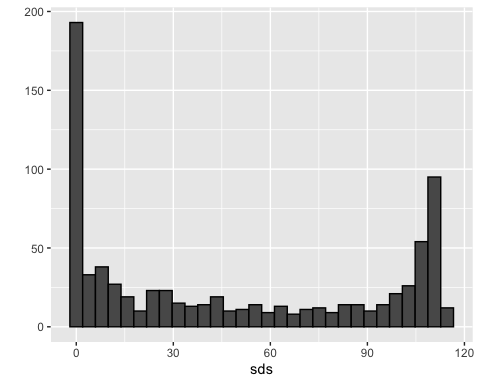
if(!require(matrixStats)) install.packages("matrixStats")

## Loading required package: matrixStats

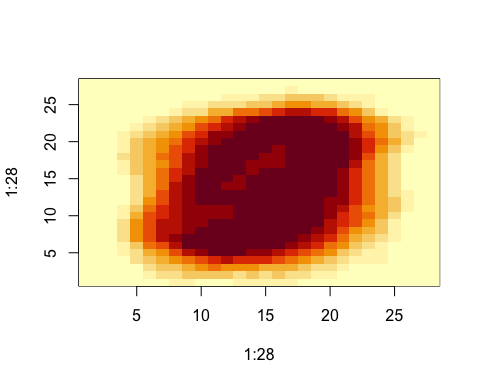
##   
## Attaching package: 'matrixStats'

## The following object is masked from 'package:dplyr':  
##   
## count

library(matrixStats)  
  
sds <- colSds(x)  
qplot(sds, bins = "30", color = I("black"))



image(1:28, 1:28, matrix(sds, 28, 28)[, 28:1])



#extract columns and rows  
x[ ,c(351,352)]

## [,1] [,2]  
## [1,] 70 0  
## [2,] 0 0  
## [3,] 0 0  
## [4,] 205 253  
## [5,] 8 78  
## [6,] 0 0  
## [7,] 253 253  
## [8,] 91 212  
## [9,] 254 143  
## [10,] 0 0  
## [11,] 254 254  
## [12,] 78 79  
## [13,] 254 248  
## [14,] 0 114  
## [15,] 254 109  
## [16,] 0 0  
## [17,] 0 0  
## [18,] 80 223  
## [19,] 0 0  
## [20,] 8 43  
## [21,] 109 109  
## [22,] 96 204  
## [23,] 0 0  
## [24,] 142 255  
## [25,] 32 254  
## [26,] 250 253  
## [27,] 0 0  
## [28,] 253 253  
## [29,] 0 0  
## [30,] 2 0  
## [31,] 253 253  
## [32,] 253 253  
## [33,] 0 0  
## [34,] 228 216  
## [35,] 225 0  
## [36,] 141 86  
## [37,] 107 0  
## [38,] 0 0  
## [39,] 0 15  
## [40,] 0 0  
## [41,] 253 253  
## [42,] 232 233  
## [43,] 0 182  
## [44,] 71 173  
## [45,] 253 203  
## [46,] 44 199  
## [47,] 0 154  
## [48,] 0 0  
## [49,] 169 254  
## [50,] 252 176  
## [51,] 254 254  
## [52,] 0 0  
## [53,] 0 0  
## [54,] 24 242  
## [55,] 71 122  
## [56,] 0 186  
## [57,] 0 0  
## [58,] 0 0  
## [59,] 111 189  
## [60,] 229 254  
## [61,] 0 0  
## [62,] 0 227  
## [63,] 0 0  
## [64,] 253 251  
## [65,] 0 0  
## [66,] 216 151  
## [67,] 128 128  
## [68,] 254 254  
## [69,] 0 0  
## [70,] 29 0  
## [71,] 253 122  
## [72,] 69 0  
## [73,] 254 204  
## [74,] 17 179  
## [75,] 253 252  
## [76,] 182 15  
## [77,] 254 254  
## [78,] 251 253  
## [79,] 173 253  
## [80,] 10 0  
## [81,] 252 253  
## [82,] 0 0  
## [83,] 0 0  
## [84,] 0 128  
## [85,] 0 0  
## [86,] 253 253  
## [87,] 253 253  
## [88,] 21 52  
## [89,] 0 0  
## [90,] 0 0  
## [91,] 0 0  
## [92,] 53 53  
## [93,] 0 0  
## [94,] 70 236  
## [95,] 38 0  
## [96,] 0 0  
## [97,] 0 26  
## [98,] 38 38  
## [99,] 253 240  
## [100,] 69 253  
## [101,] 0 0  
## [102,] 66 0  
## [103,] 254 95  
## [104,] 0 0  
## [105,] 251 0  
## [106,] 253 253  
## [107,] 0 0  
## [108,] 191 255  
## [109,] 0 0  
## [110,] 163 8  
## [111,] 78 253  
## [112,] 55 139  
## [113,] 252 253  
## [114,] 252 252  
## [115,] 0 0  
## [116,] 0 0  
## [117,] 0 15  
## [118,] 253 253  
## [119,] 0 0  
## [120,] 14 0  
## [121,] 0 0  
## [122,] 0 0  
## [123,] 0 150  
## [124,] 0 0  
## [125,] 253 233  
## [126,] 254 178  
## [127,] 0 0  
## [128,] 61 1  
## [129,] 253 253  
## [130,] 192 252  
## [131,] 254 247  
## [132,] 0 5  
## [133,] 253 253  
## [134,] 141 240  
## [135,] 253 251  
## [136,] 252 252  
## [137,] 254 179  
## [138,] 255 255  
## [139,] 244 253  
## [140,] 0 0  
## [141,] 0 0  
## [142,] 131 44  
## [143,] 0 0  
## [144,] 162 255  
## [145,] 72 142  
## [146,] 0 0  
## [147,] 0 34  
## [148,] 0 0  
## [149,] 0 0  
## [150,] 252 252  
## [151,] 221 254  
## [152,] 0 0  
## [153,] 232 254  
## [154,] 5 89  
## [155,] 253 213  
## [156,] 0 36  
## [157,] 0 0  
## [158,] 179 242  
## [159,] 50 50  
## [160,] 0 90  
## [161,] 254 254  
## [162,] 229 254  
## [163,] 0 0  
## [164,] 76 243  
## [165,] 0 0  
## [166,] 63 167  
## [167,] 0 0  
## [168,] 0 0  
## [169,] 253 252  
## [170,] 105 4  
## [171,] 37 168  
## [172,] 69 168  
## [173,] 255 152  
## [174,] 170 0  
## [175,] 252 253  
## [176,] 185 8  
## [177,] 254 253  
## [178,] 251 253  
## [179,] 0 0  
## [180,] 59 106  
## [181,] 0 178  
## [182,] 0 0  
## [183,] 176 253  
## [184,] 0 64  
## [185,] 253 226  
## [186,] 0 0  
## [187,] 0 0  
## [188,] 254 254  
## [189,] 0 0  
## [190,] 252 252  
## [191,] 167 254  
## [192,] 0 0  
## [193,] 0 0  
## [194,] 32 32  
## [195,] 0 0  
## [196,] 148 149  
## [197,] 0 0  
## [198,] 250 225  
## [199,] 104 252  
## [200,] 0 11  
## [201,] 253 169  
## [202,] 157 252  
## [203,] 100 247  
## [204,] 162 216  
## [205,] 0 0  
## [206,] 253 251  
## [207,] 0 0  
## [208,] 0 0  
## [209,] 253 253  
## [210,] 0 0  
## [211,] 0 0  
## [212,] 253 254  
## [213,] 199 253  
## [214,] 0 20  
## [215,] 0 0  
## [216,] 253 253  
## [217,] 0 0  
## [218,] 0 0  
## [219,] 106 239  
## [220,] 181 84  
## [221,] 0 0  
## [222,] 0 31  
## [223,] 152 244  
## [224,] 0 0  
## [225,] 0 61  
## [226,] 253 227  
## [227,] 0 136  
## [228,] 0 0  
## [229,] 0 0  
## [230,] 0 0  
## [231,] 0 0  
## [232,] 253 251  
## [233,] 0 0  
## [234,] 0 0  
## [235,] 0 2  
## [236,] 253 253  
## [237,] 0 0  
## [238,] 0 0  
## [239,] 0 0  
## [240,] 98 88  
## [241,] 253 252  
## [242,] 0 0  
## [243,] 254 254  
## [244,] 0 0  
## [245,] 0 169  
## [246,] 255 255  
## [247,] 0 0  
## [248,] 0 2  
## [249,] 254 252  
## [250,] 0 0  
## [251,] 0 1  
## [252,] 253 253  
## [253,] 253 252  
## [254,] 0 0  
## [255,] 254 254  
## [256,] 253 253  
## [257,] 253 171  
## [258,] 0 0  
## [259,] 0 0  
## [260,] 254 231  
## [261,] 0 0  
## [262,] 0 0  
## [263,] 0 0  
## [264,] 0 0  
## [265,] 0 0  
## [266,] 236 62  
## [267,] 77 0  
## [268,] 0 90  
## [269,] 0 93  
## [270,] 253 253  
## [271,] 251 57  
## [272,] 0 0  
## [273,] 125 168  
## [274,] 127 127  
## [275,] 232 8  
## [276,] 0 0  
## [277,] 191 254  
## [278,] 0 0  
## [279,] 245 254  
## [280,] 0 128  
## [281,] 0 51  
## [282,] 253 255  
## [283,] 0 0  
## [284,] 0 0  
## [285,] 253 253  
## [286,] 0 0  
## [287,] 253 253  
## [288,] 254 251  
## [289,] 0 0  
## [290,] 0 0  
## [291,] 252 253  
## [292,] 253 253  
## [293,] 2 45  
## [294,] 0 0  
## [295,] 0 0  
## [296,] 133 160  
## [297,] 0 0  
## [298,] 0 0  
## [299,] 253 253  
## [300,] 0 155  
## [301,] 42 235  
## [302,] 0 0  
## [303,] 0 0  
## [304,] 0 0  
## [305,] 29 29  
## [306,] 0 0  
## [307,] 100 176  
## [308,] 0 0  
## [309,] 0 0  
## [310,] 232 253  
## [311,] 235 254  
## [312,] 0 0  
## [313,] 183 102  
## [314,] 0 35  
## [315,] 0 0  
## [316,] 243 253  
## [317,] 255 255  
## [318,] 0 0  
## [319,] 241 224  
## [320,] 0 5  
## [321,] 0 0  
## [322,] 230 253  
## [323,] 0 0  
## [324,] 0 0  
## [325,] 0 0  
## [326,] 0 0  
## [327,] 0 0  
## [328,] 253 253  
## [329,] 45 0  
## [330,] 0 0  
## [331,] 70 70  
## [332,] 0 0  
## [333,] 0 0  
## [334,] 184 184  
## [335,] 0 183  
## [336,] 211 86  
## [337,] 0 0  
## [338,] 0 0  
## [339,] 0 0  
## [340,] 0 0  
## [341,] 0 64  
## [342,] 253 255  
## [343,] 132 152  
## [344,] 252 241  
## [345,] 0 0  
## [346,] 158 254  
## [347,] 8 134  
## [348,] 0 0  
## [349,] 205 254  
## [350,] 0 0  
## [351,] 0 3  
## [352,] 180 253  
## [353,] 253 207  
## [354,] 0 0  
## [355,] 0 102  
## [356,] 254 254  
## [357,] 253 253  
## [358,] 211 253  
## [359,] 254 95  
## [360,] 0 0  
## [361,] 253 253  
## [362,] 160 252  
## [363,] 0 0  
## [364,] 0 96  
## [365,] 0 0  
## [366,] 0 0  
## [367,] 253 217  
## [368,] 0 0  
## [369,] 254 254  
## [370,] 0 0  
## [371,] 253 253  
## [372,] 0 0  
## [373,] 0 43  
## [374,] 0 0  
## [375,] 121 252  
## [376,] 0 0  
## [377,] 0 0  
## [378,] 0 0  
## [379,] 0 0  
## [380,] 0 3  
## [381,] 0 0  
## [382,] 0 0  
## [383,] 254 84  
## [384,] 0 0  
## [385,] 0 56  
## [386,] 0 52  
## [387,] 252 240  
## [388,] 0 0  
## [389,] 0 0  
## [390,] 0 0  
## [391,] 38 233  
## [392,] 197 173  
## [393,] 53 232  
## [394,] 64 64  
## [395,] 181 0  
## [396,] 0 0  
## [397,] 0 0  
## [398,] 207 252  
## [399,] 253 158  
## [400,] 27 0  
## [401,] 0 0  
## [402,] 0 0  
## [403,] 0 0  
## [404,] 105 0  
## [405,] 253 253  
## [406,] 93 239  
## [407,] 253 58  
## [408,] 42 27  
## [409,] 254 195  
## [410,] 0 0  
## [411,] 229 253  
## [412,] 0 0  
## [413,] 0 100  
## [414,] 0 0  
## [415,] 0 70  
## [416,] 0 0  
## [417,] 253 251  
## [418,] 58 0  
## [419,] 7 221  
## [420,] 0 45  
## [421,] 252 253  
## [422,] 0 0  
## [423,] 0 77  
## [424,] 0 0  
## [425,] 253 253  
## [426,] 23 29  
## [427,] 252 252  
## [428,] 0 0  
## [429,] 135 246  
## [430,] 0 0  
## [431,] 0 0  
## [432,] 0 0  
## [433,] 0 0  
## [434,] 253 253  
## [435,] 0 0  
## [436,] 0 0  
## [437,] 0 0  
## [438,] 40 8  
## [439,] 0 34  
## [440,] 254 254  
## [441,] 0 0  
## [442,] 0 47  
## [443,] 0 0  
## [444,] 99 253  
## [445,] 222 246  
## [446,] 252 209  
## [447,] 0 0  
## [448,] 172 253  
## [449,] 12 161  
## [450,] 0 0  
## [451,] 251 180  
## [452,] 0 0  
## [453,] 254 253  
## [454,] 0 0  
## [455,] 254 223  
## [456,] 237 252  
## [457,] 252 252  
## [458,] 0 0  
## [459,] 0 0  
## [460,] 49 159  
## [461,] 0 0  
## [462,] 0 0  
## [463,] 0 0  
## [464,] 0 0  
## [465,] 0 0  
## [466,] 0 0  
## [467,] 98 254  
## [468,] 0 0  
## [469,] 0 0  
## [470,] 0 0  
## [471,] 0 0  
## [472,] 51 51  
## [473,] 154 250  
## [474,] 0 0  
## [475,] 0 0  
## [476,] 211 253  
## [477,] 0 0  
## [478,] 0 0  
## [479,] 114 253  
## [480,] 254 253  
## [481,] 0 0  
## [482,] 0 0  
## [483,] 0 0  
## [484,] 0 0  
## [485,] 253 132  
## [486,] 0 0  
## [487,] 67 0  
## [488,] 0 9  
## [489,] 254 255  
## [490,] 0 0  
## [491,] 253 250  
## [492,] 0 255  
## [493,] 252 250  
## [494,] 0 0  
## [495,] 0 0  
## [496,] 253 253  
## [497,] 202 203  
## [498,] 0 0  
## [499,] 0 0  
## [500,] 130 76  
## [501,] 0 0  
## [502,] 0 0  
## [503,] 0 0  
## [504,] 115 34  
## [505,] 105 0  
## [506,] 0 0  
## [507,] 0 0  
## [508,] 143 253  
## [509,] 254 254  
## [510,] 160 253  
## [511,] 253 224  
## [512,] 12 118  
## [513,] 0 0  
## [514,] 0 0  
## [515,] 148 237  
## [516,] 0 0  
## [517,] 0 0  
## [518,] 24 0  
## [519,] 0 7  
## [520,] 0 0  
## [521,] 0 0  
## [522,] 128 25  
## [523,] 0 0  
## [524,] 0 0  
## [525,] 0 0  
## [526,] 0 0  
## [527,] 0 0  
## [528,] 12 0  
## [529,] 221 62  
## [530,] 0 51  
## [531,] 0 0  
## [532,] 0 0  
## [533,] 253 253  
## [534,] 18 246  
## [535,] 204 252  
## [536,] 128 253  
## [537,] 0 0  
## [538,] 156 127  
## [539,] 254 254  
## [540,] 0 42  
## [541,] 114 0  
## [542,] 0 0  
## [543,] 151 0  
## [544,] 0 0  
## [545,] 189 112  
## [546,] 0 164  
## [547,] 252 253  
## [548,] 0 15  
## [549,] 0 0  
## [550,] 82 202  
## [551,] 0 8  
## [552,] 0 0  
## [553,] 215 254  
## [554,] 206 252  
## [555,] 251 253  
## [556,] 0 0  
## [557,] 253 253  
## [558,] 253 253  
## [559,] 115 0  
## [560,] 110 231  
## [561,] 0 136  
## [562,] 254 254  
## [563,] 0 0  
## [564,] 0 23  
## [565,] 0 0  
## [566,] 113 206  
## [567,] 0 71  
## [568,] 0 0  
## [569,] 0 0  
## [570,] 0 22  
## [571,] 0 0  
## [572,] 25 119  
## [573,] 255 255  
## [574,] 246 253  
## [575,] 253 128  
## [576,] 21 22  
## [577,] 194 113  
## [578,] 0 0  
## [579,] 0 0  
## [580,] 0 0  
## [581,] 43 225  
## [582,] 253 253  
## [583,] 0 0  
## [584,] 112 166  
## [585,] 0 0  
## [586,] 0 0  
## [587,] 0 0  
## [588,] 253 253  
## [589,] 70 254  
## [590,] 0 0  
## [591,] 0 157  
## [592,] 0 0  
## [593,] 0 6  
## [594,] 179 253  
## [595,] 221 253  
## [596,] 0 32  
## [597,] 0 0  
## [598,] 252 82  
## [599,] 0 0  
## [600,] 0 0  
## [601,] 111 245  
## [602,] 0 0  
## [603,] 253 65  
## [604,] 64 0  
## [605,] 47 254  
## [606,] 0 14  
## [607,] 10 168  
## [608,] 7 160  
## [609,] 0 0  
## [610,] 252 252  
## [611,] 0 0  
## [612,] 23 172  
## [613,] 0 0  
## [614,] 253 247  
## [615,] 0 0  
## [616,] 0 0  
## [617,] 0 0  
## [618,] 0 0  
## [619,] 253 0  
## [620,] 0 0  
## [621,] 252 253  
## [622,] 0 0  
## [623,] 253 255  
## [624,] 50 7  
## [625,] 0 0  
## [626,] 0 0  
## [627,] 0 0  
## [628,] 0 0  
## [629,] 182 253  
## [630,] 206 253  
## [631,] 68 41  
## [632,] 0 0  
## [633,] 47 5  
## [634,] 18 0  
## [635,] 0 80  
## [636,] 0 0  
## [637,] 0 0  
## [638,] 193 254  
## [639,] 254 177  
## [640,] 0 0  
## [641,] 84 19  
## [642,] 236 253  
## [643,] 0 0  
## [644,] 253 253  
## [645,] 254 254  
## [646,] 253 253  
## [647,] 164 253  
## [648,] 0 0  
## [649,] 229 254  
## [650,] 5 0  
## [651,] 88 211  
## [652,] 0 0  
## [653,] 252 229  
## [654,] 0 0  
## [655,] 0 9  
## [656,] 0 0  
## [657,] 5 0  
## [658,] 0 0  
## [659,] 0 0  
## [660,] 8 128  
## [661,] 25 0  
## [662,] 0 29  
## [663,] 19 0  
## [664,] 0 0  
## [665,] 0 10  
## [666,] 235 239  
## [667,] 0 0  
## [668,] 255 128  
## [669,] 0 0  
## [670,] 0 0  
## [671,] 14 51  
## [672,] 253 253  
## [673,] 0 0  
## [674,] 0 0  
## [675,] 244 89  
## [676,] 253 253  
## [677,] 254 230  
## [678,] 20 0  
## [679,] 253 253  
## [680,] 239 249  
## [681,] 0 0  
## [682,] 0 0  
## [683,] 0 0  
## [684,] 0 0  
## [685,] 0 0  
## [686,] 254 254  
## [687,] 0 0  
## [688,] 0 0  
## [689,] 13 221  
## [690,] 0 0  
## [691,] 0 0  
## [692,] 206 253  
## [693,] 131 178  
## [694,] 57 144  
## [695,] 73 253  
## [696,] 252 252  
## [697,] 0 47  
## [698,] 0 0  
## [699,] 253 253  
## [700,] 237 165  
## [701,] 0 0  
## [702,] 0 0  
## [703,] 0 0  
## [704,] 0 0  
## [705,] 17 65  
## [706,] 253 253  
## [707,] 49 189  
## [708,] 51 92  
## [709,] 133 254  
## [710,] 0 0  
## [711,] 253 72  
## [712,] 252 252  
## [713,] 180 0  
## [714,] 0 55  
## [715,] 113 254  
## [716,] 254 253  
## [717,] 249 127  
## [718,] 0 0  
## [719,] 253 254  
## [720,] 251 253  
## [721,] 253 246  
## [722,] 0 0  
## [723,] 8 0  
## [724,] 0 0  
## [725,] 0 0  
## [726,] 252 252  
## [727,] 254 218  
## [728,] 0 0  
## [729,] 0 51  
## [730,] 0 0  
## [731,] 0 0  
## [732,] 253 253  
## [733,] 209 253  
## [734,] 0 0  
## [735,] 122 198  
## [736,] 0 0  
## [737,] 255 29  
## [738,] 32 0  
## [739,] 254 59  
## [740,] 0 5  
## [741,] 254 139  
## [742,] 0 0  
## [743,] 0 0  
## [744,] 7 0  
## [745,] 226 226  
## [746,] 73 0  
## [747,] 0 219  
## [748,] 176 253  
## [749,] 194 71  
## [750,] 9 0  
## [751,] 0 29  
## [752,] 253 254  
## [753,] 252 252  
## [754,] 0 0  
## [755,] 0 0  
## [756,] 0 0  
## [757,] 208 208  
## [758,] 246 230  
## [759,] 251 252  
## [760,] 0 0  
## [761,] 243 40  
## [762,] 177 8  
## [763,] 0 0  
## [764,] 0 0  
## [765,] 0 57  
## [766,] 253 253  
## [767,] 203 204  
## [768,] 254 200  
## [769,] 208 199  
## [770,] 252 253  
## [771,] 0 0  
## [772,] 110 110  
## [773,] 15 178  
## [774,] 0 0  
## [775,] 0 0  
## [776,] 60 100  
## [777,] 0 0  
## [778,] 241 101  
## [779,] 0 0  
## [780,] 253 252  
## [781,] 253 252  
## [782,] 7 0  
## [783,] 0 0  
## [784,] 253 253  
## [785,] 224 252  
## [786,] 0 0  
## [787,] 0 0  
## [788,] 0 0  
## [789,] 0 0  
## [790,] 254 254  
## [791,] 0 0  
## [792,] 218 253  
## [793,] 242 78  
## [794,] 0 0  
## [795,] 7 0  
## [796,] 0 54  
## [797,] 24 0  
## [798,] 0 10  
## [799,] 0 0  
## [800,] 253 254  
## [801,] 0 103  
## [802,] 132 253  
## [803,] 0 78  
## [804,] 0 6  
## [805,] 0 0  
## [806,] 254 254  
## [807,] 0 15  
## [808,] 144 254  
## [809,] 252 154  
## [810,] 253 252  
## [811,] 116 137  
## [812,] 253 253  
## [813,] 0 54  
## [814,] 0 131  
## [815,] 141 210  
## [816,] 203 223  
## [817,] 0 0  
## [818,] 254 254  
## [819,] 0 0  
## [820,] 0 0  
## [821,] 0 0  
## [822,] 253 253  
## [823,] 2 41  
## [824,] 13 126  
## [825,] 0 135  
## [826,] 0 0  
## [827,] 0 0  
## [828,] 0 0  
## [829,] 0 0  
## [830,] 5 0  
## [831,] 252 253  
## [832,] 137 184  
## [833,] 255 253  
## [834,] 253 252  
## [835,] 0 0  
## [836,] 253 252  
## [837,] 82 223  
## [838,] 254 254  
## [839,] 252 253  
## [840,] 0 0  
## [841,] 253 204  
## [842,] 0 0  
## [843,] 253 253  
## [844,] 254 253  
## [845,] 0 0  
## [846,] 249 253  
## [847,] 0 0  
## [848,] 0 0  
## [849,] 0 0  
## [850,] 64 0  
## [851,] 0 0  
## [852,] 0 0  
## [853,] 59 0  
## [854,] 0 0  
## [855,] 0 0  
## [856,] 0 0  
## [857,] 254 253  
## [858,] 252 252  
## [859,] 0 0  
## [860,] 0 0  
## [861,] 0 0  
## [862,] 253 134  
## [863,] 0 190  
## [864,] 77 254  
## [865,] 159 254  
## [866,] 242 253  
## [867,] 0 0  
## [868,] 253 253  
## [869,] 0 0  
## [870,] 8 0  
## [871,] 253 253  
## [872,] 240 254  
## [873,] 0 0  
## [874,] 0 0  
## [875,] 253 253  
## [876,] 253 253  
## [877,] 44 249  
## [878,] 0 0  
## [879,] 243 174  
## [880,] 97 97  
## [881,] 0 0  
## [882,] 6 86  
## [883,] 0 0  
## [884,] 0 0  
## [885,] 82 253  
## [886,] 197 253  
## [887,] 114 0  
## [888,] 1 25  
## [889,] 0 0  
## [890,] 0 0  
## [891,] 252 253  
## [892,] 240 253  
## [893,] 181 20  
## [894,] 0 0  
## [895,] 203 254  
## [896,] 254 253  
## [897,] 0 0  
## [898,] 0 0  
## [899,] 0 0  
## [900,] 24 0  
## [901,] 6 191  
## [902,] 0 0  
## [903,] 0 0  
## [904,] 0 0  
## [905,] 0 0  
## [906,] 104 254  
## [907,] 0 152  
## [908,] 0 8  
## [909,] 67 160  
## [910,] 253 253  
## [911,] 0 0  
## [912,] 0 0  
## [913,] 0 0  
## [914,] 37 167  
## [915,] 0 0  
## [916,] 35 0  
## [917,] 7 108  
## [918,] 0 0  
## [919,] 71 241  
## [920,] 254 254  
## [921,] 253 253  
## [922,] 0 0  
## [923,] 1 0  
## [924,] 0 64  
## [925,] 198 198  
## [926,] 0 170  
## [927,] 0 0  
## [928,] 0 0  
## [929,] 0 0  
## [930,] 0 0  
## [931,] 0 0  
## [932,] 0 0  
## [933,] 123 254  
## [934,] 251 225  
## [935,] 0 0  
## [936,] 14 69  
## [937,] 89 253  
## [938,] 0 0  
## [939,] 190 252  
## [940,] 94 0  
## [941,] 0 0  
## [942,] 150 254  
## [943,] 163 238  
## [944,] 7 0  
## [945,] 168 169  
## [946,] 0 0  
## [947,] 75 231  
## [948,] 1 0  
## [949,] 128 254  
## [950,] 0 0  
## [951,] 116 253  
## [952,] 241 254  
## [953,] 0 0  
## [954,] 254 254  
## [955,] 0 0  
## [956,] 0 0  
## [957,] 74 53  
## [958,] 8 0  
## [959,] 253 253  
## [960,] 253 253  
## [961,] 0 0  
## [962,] 234 254  
## [963,] 0 0  
## [964,] 98 253  
## [965,] 222 25  
## [966,] 0 0  
## [967,] 241 189  
## [968,] 0 0  
## [969,] 0 46  
## [970,] 0 0  
## [971,] 6 6  
## [972,] 0 0  
## [973,] 0 0  
## [974,] 23 0  
## [975,] 231 254  
## [976,] 254 254  
## [977,] 0 32  
## [978,] 15 0  
## [979,] 155 0  
## [980,] 6 0  
## [981,] 135 243  
## [982,] 0 0  
## [983,] 253 201  
## [984,] 198 254  
## [985,] 0 0  
## [986,] 22 0  
## [987,] 3 171  
## [988,] 0 0  
## [989,] 0 0  
## [990,] 0 0  
## [991,] 0 0  
## [992,] 221 151  
## [993,] 254 172  
## [994,] 156 253  
## [995,] 0 0  
## [996,] 254 254  
## [997,] 0 0  
## [998,] 0 0  
## [999,] 103 64  
## [1000,] 139 0

x[c(2,3),]

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]  
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]  
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]  
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]  
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62]  
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74]  
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86]  
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98]  
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0  
## [,99] [,100] [,101] [,102] [,103] [,104] [,105] [,106] [,107] [,108]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,109] [,110] [,111] [,112] [,113] [,114] [,115] [,116] [,117] [,118]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,119] [,120] [,121] [,122] [,123] [,124] [,125] [,126] [,127] [,128]  
## [1,] 0 0 0 0 0 0 0 0 0 51  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,129] [,130] [,131] [,132] [,133] [,134] [,135] [,136] [,137] [,138]  
## [1,] 159 253 159 50 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,139] [,140] [,141] [,142] [,143] [,144] [,145] [,146] [,147] [,148]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,149] [,150] [,151] [,152] [,153] [,154] [,155] [,156] [,157] [,158]  
## [1,] 0 0 0 0 0 0 48 238 252 252  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,159] [,160] [,161] [,162] [,163] [,164] [,165] [,166] [,167] [,168]  
## [1,] 252 237 0 0 0 0 0 0 0 0  
## [2,] 0 0 67 232 39 0 0 0 0 0  
## [,169] [,170] [,171] [,172] [,173] [,174] [,175] [,176] [,177] [,178]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 62 81 0 0 0 0  
## [,179] [,180] [,181] [,182] [,183] [,184] [,185] [,186] [,187] [,188]  
## [1,] 0 0 0 54 227 253 252 239 233 252  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,189] [,190] [,191] [,192] [,193] [,194] [,195] [,196] [,197] [,198]  
## [1,] 57 6 0 0 0 0 0 0 0 0  
## [2,] 120 180 39 0 0 0 0 0 0 0  
## [,199] [,200] [,201] [,202] [,203] [,204] [,205] [,206] [,207] [,208]  
## [1,] 0 0 0 0 0 0 0 0 0 10  
## [2,] 0 0 126 163 0 0 0 0 0 0  
## [,209] [,210] [,211] [,212] [,213] [,214] [,215] [,216] [,217] [,218]  
## [1,] 60 224 252 253 252 202 84 252 253 122  
## [2,] 0 0 0 0 0 0 0 2 153 210  
## [,219] [,220] [,221] [,222] [,223] [,224] [,225] [,226] [,227] [,228]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 40 0 0 0 0 0 0 0 0 0  
## [,229] [,230] [,231] [,232] [,233] [,234] [,235] [,236] [,237] [,238]  
## [1,] 0 0 0 0 0 0 0 163 252 252  
## [2,] 220 163 0 0 0 0 0 0 0 0  
## [,239] [,240] [,241] [,242] [,243] [,244] [,245] [,246] [,247] [,248]  
## [1,] 252 253 252 252 96 189 253 167 0 0  
## [2,] 0 0 0 0 0 27 254 162 0 0  
## [,249] [,250] [,251] [,252] [,253] [,254] [,255] [,256] [,257] [,258]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 222 163  
## [,259] [,260] [,261] [,262] [,263] [,264] [,265] [,266] [,267] [,268]  
## [1,] 0 0 0 0 51 238 253 253 190 114  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,269] [,270] [,271] [,272] [,273] [,274] [,275] [,276] [,277] [,278]  
## [1,] 253 228 47 79 255 168 0 0 0 0  
## [2,] 0 0 0 183 254 125 0 0 0 0  
## [,279] [,280] [,281] [,282] [,283] [,284] [,285] [,286] [,287] [,288]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 46 245 163 0 0  
## [,289] [,290] [,291] [,292] [,293] [,294] [,295] [,296] [,297] [,298]  
## [1,] 0 48 238 252 252 179 12 75 121 21  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,299] [,300] [,301] [,302] [,303] [,304] [,305] [,306] [,307] [,308]  
## [1,] 0 0 253 243 50 0 0 0 0 0  
## [2,] 0 198 254 56 0 0 0 0 0 0  
## [,309] [,310] [,311] [,312] [,313] [,314] [,315] [,316] [,317] [,318]  
## [1,] 0 0 0 0 0 0 0 0 38 165  
## [2,] 0 0 0 120 254 163 0 0 0 0  
## [,319] [,320] [,321] [,322] [,323] [,324] [,325] [,326] [,327] [,328]  
## [1,] 253 233 208 84 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 23 231  
## [,329] [,330] [,331] [,332] [,333] [,334] [,335] [,336] [,337] [,338]  
## [1,] 253 252 165 0 0 0 0 0 0 0  
## [2,] 254 29 0 0 0 0 0 0 0 0  
## [,339] [,340] [,341] [,342] [,343] [,344] [,345] [,346] [,347] [,348]  
## [1,] 0 0 0 0 0 7 178 252 240 71  
## [2,] 0 159 254 120 0 0 0 0 0 0  
## [,349] [,350] [,351] [,352] [,353] [,354] [,355] [,356] [,357] [,358]  
## [1,] 19 28 0 0 0 0 0 0 253 252  
## [2,] 0 0 0 0 0 0 163 254 216 16  
## [,359] [,360] [,361] [,362] [,363] [,364] [,365] [,366] [,367] [,368]  
## [1,] 195 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 159  
## [,369] [,370] [,371] [,372] [,373] [,374] [,375] [,376] [,377] [,378]  
## [1,] 0 0 0 57 252 252 63 0 0 0  
## [2,] 254 67 0 0 0 0 0 0 0 0  
## [,379] [,380] [,381] [,382] [,383] [,384] [,385] [,386] [,387] [,388]  
## [1,] 0 0 0 0 0 0 253 252 195 0  
## [2,] 0 14 86 178 248 254 91 0 0 0  
## [,389] [,390] [,391] [,392] [,393] [,394] [,395] [,396] [,397] [,398]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 159 254 85  
## [,399] [,400] [,401] [,402] [,403] [,404] [,405] [,406] [,407] [,408]  
## [1,] 0 198 253 190 0 0 0 0 0 0  
## [2,] 0 0 0 47 49 116 144 150 241 243  
## [,409] [,410] [,411] [,412] [,413] [,414] [,415] [,416] [,417] [,418]  
## [1,] 0 0 0 0 255 253 196 0 0 0  
## [2,] 234 179 241 252 40 0 0 0 0 0  
## [,419] [,420] [,421] [,422] [,423] [,424] [,425] [,426] [,427] [,428]  
## [1,] 0 0 0 0 0 0 0 0 76 246  
## [2,] 0 0 0 0 0 150 253 237 207 207  
## [,429] [,430] [,431] [,432] [,433] [,434] [,435] [,436] [,437] [,438]  
## [1,] 252 112 0 0 0 0 0 0 0 0  
## [2,] 207 253 254 250 240 198 143 91 28 5  
## [,439] [,440] [,441] [,442] [,443] [,444] [,445] [,446] [,447] [,448]  
## [1,] 0 0 253 252 148 0 0 0 0 0  
## [2,] 233 250 0 0 0 0 0 0 0 0  
## [,449] [,450] [,451] [,452] [,453] [,454] [,455] [,456] [,457] [,458]  
## [1,] 0 0 0 0 0 0 85 252 230 25  
## [2,] 0 0 0 0 119 177 177 177 177 177  
## [,459] [,460] [,461] [,462] [,463] [,464] [,465] [,466] [,467] [,468]  
## [1,] 0 0 0 0 0 0 0 0 7 135  
## [2,] 98 56 0 0 0 0 0 102 254 220  
## [,469] [,470] [,471] [,472] [,473] [,474] [,475] [,476] [,477] [,478]  
## [1,] 253 186 12 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,479] [,480] [,481] [,482] [,483] [,484] [,485] [,486] [,487] [,488]  
## [1,] 0 0 0 0 85 252 223 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,489] [,490] [,491] [,492] [,493] [,494] [,495] [,496] [,497] [,498]  
## [1,] 0 0 0 0 0 7 131 252 225 71  
## [2,] 0 0 0 0 0 169 254 137 0 0  
## [,499] [,500] [,501] [,502] [,503] [,504] [,505] [,506] [,507] [,508]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,509] [,510] [,511] [,512] [,513] [,514] [,515] [,516] [,517] [,518]  
## [1,] 0 0 85 252 145 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,519] [,520] [,521] [,522] [,523] [,524] [,525] [,526] [,527] [,528]  
## [1,] 0 0 48 165 252 173 0 0 0 0  
## [2,] 0 0 0 169 254 57 0 0 0 0  
## [,529] [,530] [,531] [,532] [,533] [,534] [,535] [,536] [,537] [,538]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,539] [,540] [,541] [,542] [,543] [,544] [,545] [,546] [,547] [,548]  
## [1,] 86 253 225 0 0 0 0 0 0 114  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,549] [,550] [,551] [,552] [,553] [,554] [,555] [,556] [,557] [,558]  
## [1,] 238 253 162 0 0 0 0 0 0 0  
## [2,] 0 169 254 57 0 0 0 0 0 0  
## [,559] [,560] [,561] [,562] [,563] [,564] [,565] [,566] [,567] [,568]  
## [1,] 0 0 0 0 0 0 0 0 85 252  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,569] [,570] [,571] [,572] [,573] [,574] [,575] [,576] [,577] [,578]  
## [1,] 249 146 48 29 85 178 225 253 223 167  
## [2,] 0 0 0 0 0 0 0 0 0 169  
## [,579] [,580] [,581] [,582] [,583] [,584] [,585] [,586] [,587] [,588]  
## [1,] 56 0 0 0 0 0 0 0 0 0  
## [2,] 255 94 0 0 0 0 0 0 0 0  
## [,589] [,590] [,591] [,592] [,593] [,594] [,595] [,596] [,597] [,598]  
## [1,] 0 0 0 0 0 0 85 252 252 252  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,599] [,600] [,601] [,602] [,603] [,604] [,605] [,606] [,607] [,608]  
## [1,] 229 215 252 252 252 196 130 0 0 0  
## [2,] 0 0 0 0 0 0 0 169 254 96  
## [,609] [,610] [,611] [,612] [,613] [,614] [,615] [,616] [,617] [,618]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,619] [,620] [,621] [,622] [,623] [,624] [,625] [,626] [,627] [,628]  
## [1,] 0 0 0 0 28 199 252 252 253 252  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,629] [,630] [,631] [,632] [,633] [,634] [,635] [,636] [,637] [,638]  
## [1,] 252 233 145 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 169 254 153 0 0  
## [,639] [,640] [,641] [,642] [,643] [,644] [,645] [,646] [,647] [,648]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,649] [,650] [,651] [,652] [,653] [,654] [,655] [,656] [,657] [,658]  
## [1,] 0 0 0 25 128 252 253 252 141 37  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,659] [,660] [,661] [,662] [,663] [,664] [,665] [,666] [,667] [,668]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 169 255 153 0 0 0 0  
## [,669] [,670] [,671] [,672] [,673] [,674] [,675] [,676] [,677] [,678]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,679] [,680] [,681] [,682] [,683] [,684] [,685] [,686] [,687] [,688]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,689] [,690] [,691] [,692] [,693] [,694] [,695] [,696] [,697] [,698]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 96 254 153 0 0 0 0 0 0  
## [,699] [,700] [,701] [,702] [,703] [,704] [,705] [,706] [,707] [,708]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,709] [,710] [,711] [,712] [,713] [,714] [,715] [,716] [,717] [,718]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,719] [,720] [,721] [,722] [,723] [,724] [,725] [,726] [,727] [,728]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,729] [,730] [,731] [,732] [,733] [,734] [,735] [,736] [,737] [,738]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,739] [,740] [,741] [,742] [,743] [,744] [,745] [,746] [,747] [,748]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,749] [,750] [,751] [,752] [,753] [,754] [,755] [,756] [,757] [,758]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,759] [,760] [,761] [,762] [,763] [,764] [,765] [,766] [,767] [,768]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,769] [,770] [,771] [,772] [,773] [,774] [,775] [,776] [,777] [,778]  
## [1,] 0 0 0 0 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0 0 0 0 0  
## [,779] [,780] [,781] [,782] [,783] [,784]  
## [1,] 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0

new\_x <- x[ ,colSds(x) > 60]  
dim(new\_x)

## [1] 1000 314

class(x[,1])

## [1] "integer"

dim(x[1,])

## NULL

#preserve the matrix class  
class(x[ , 1, drop=FALSE])

## [1] "matrix" "array"

dim(x[, 1, drop=FALSE])

## [1] 1000 1

## Indexing with Matrices and Binarizing the Data

There is a link to the relevant sections of the textbook: [Indexing with matrices](https://rafalab.github.io/dsbook/large-datasets.html#indexing-with-matrices) and [Binarizing the data](https://rafalab.github.io/dsbook/large-datasets.html#binarizing-the-data)

**Key points**

* We can use logical operations with matrices:

mat <- matrix(1:15, 5, 3)  
mat[mat > 6 & mat < 12] <- 0

* We can also binarize the data using just matrix operations:

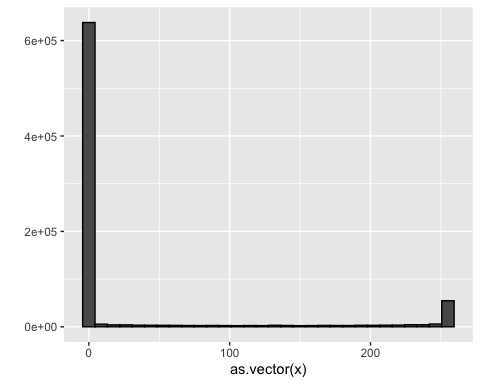
bin\_x <- x  
bin\_x[bin\_x < 255/2] <- 0   
bin\_x[bin\_x > 255/2] <- 1

*Code*

#index with matrices  
mat <- matrix(1:15, 5, 3)  
as.vector(mat)

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

qplot(as.vector(x), bins = 30, color = I("black"))



new\_x <- x  
new\_x[new\_x < 50] <- 0  
  
mat <- matrix(1:15, 5, 3)  
mat[mat < 3] <- 0  
mat

## [,1] [,2] [,3]  
## [1,] 0 6 11  
## [2,] 0 7 12  
## [3,] 3 8 13  
## [4,] 4 9 14  
## [5,] 5 10 15

mat <- matrix(1:15, 5, 3)  
mat[mat > 6 & mat < 12] <- 0  
mat

## [,1] [,2] [,3]  
## [1,] 1 6 0  
## [2,] 2 0 12  
## [3,] 3 0 13  
## [4,] 4 0 14  
## [5,] 5 0 15

#binarize the data  
bin\_x <- x  
bin\_x[bin\_x < 255/2] <- 0  
bin\_x[bin\_x > 255/2] <- 1  
bin\_X <- (x > 255/2)\*1

## Vectorization for Matrices and Matrix Algebra Operations

There is a link to the relevant sections of the textbook: [Vectorization for matrices](https://rafalab.github.io/dsbook/large-datasets.html#vectorization-for-matrices) and [Matrix algebra operations](https://rafalab.github.io/dsbook/large-datasets.html#matrix-algebra-operations)

**Key points**

* We can scale each row of a matrix using this line of code:

(x - rowMeans(x)) / rowSds(x)

* To scale each column of a matrix, we use this code:

t(t(X) - colMeans(X))

* We can also use a function called sweep() that works similarly to apply(). It takes each entry of a vector and subtracts it from the corresponding row or column:

X\_mean\_0 <- sweep(x, 2, colMeans(x))

* Matrix multiplication: t(x) %\*% x
* The cross product: crossprod(x)
* The inverse of a function: solve(crossprod(x))
* The QR decomposition: qr(x)

*Code*

#scale each row of a matrix  
(x - rowMeans(x)) / rowSds(x)  
  
#scale each column  
t(t(x) - colMeans(x))

#take each entry of a vector and subtracts it from the corresponding row or column  
x\_mean\_0 <- sweep(x, 2, colMeans(x))  
  
#divide by the standard deviation  
x\_mean\_0 <- sweep(x, 2, colMeans(x))  
x\_standardized <- sweep(x\_mean\_0, 2, colSds(x), FUN = "/")

## Comprehension Check - Working with Matrices

1. Which line of code correctly creates a 100 by 10 matrix of randomly generated normal numbers and assigns it to x?

* ☐ A. x <- matrix(rnorm(1000), 100, 100)
* ☒ B. x <- matrix(rnorm(100\*10), 100, 10)
* ☐ C. x <- matrix(rnorm(100\*10), 10, 10)
* ☐ D. x <- matrix(rnorm(100\*10), 10, 100)

1. Write the line of code that would give you the specified information about the matrix x that you generated in q1. Do not include any spaces in your line of code.

Dimension of x: dim(x)

Number of rows of x: nrow(x) or dim(x)[1] or length(x[,1])

Number of columns of x: ncol(x) or dim(x)[2] or length(x[1,])

1. Which of the following lines of code would add the scalar 1 to row 1, the scalar 2 to row 2, and so on, for the matrix x? Select ALL that apply.

* ☒ A. x <- x + seq(nrow(x))
* ☐ B. x <- 1:nrow(x)
* ☐ C. x <- sweep(x, 2, 1:nrow(x),"+")
* ☒ D. x <- sweep(x, 1, 1:nrow(x),"+")

1. Which of the following lines of code would add the scalar 1 to column 1, the scalar 2 to column 2, and so on, for the matrix x? Select ALL that apply.

* ☐ A. x <- 1:ncol(x)
* ☐ B. x <- 1:col(x)
* ☒ C. x <- sweep(x, 2, 1:ncol(x), FUN = "+")
* ☐ D. x <- -x

1. Which code correctly computes the average of each row of x?

* ☐ A. mean(x)
* ☐ B. rowMedians(x)
* ☐ C. sapply(x,mean)
* ☐ D. rowSums(x)
* ☒ E. rowMeans(x)

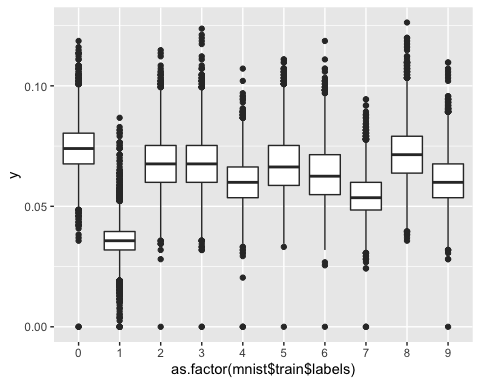
Which code correctly computes the average of each column of x?

* ☐ A. mean(x)
* ☐ B. sapply(x,mean)
* ☒ C. colMeans(x)
* ☐ D. colMedians(x)
* ☐ C. colSums(x)

1. For each observation in the mnist training data, compute the proportion of pixels that are in the **grey area**, defined as values between 50 and 205 (but not including 50 and 205). (To visualize this, you can make a boxplot by digit class.)

What proportion of the 60000\*784 pixels in the mnist training data are in the grey area overall, defined as values between 50 and 205? Report your answer to at least 3 significant digits.

mnist <- read\_mnist()  
y <- rowMeans(mnist$train$images>50 & mnist$train$images<205)  
qplot(as.factor(mnist$train$labels), y, geom = "boxplot")



mean(y) # proportion of pixels

## [1] 0.06183703

# Section 4 - Distance, Knn, Cross Validation, and Generative Models

In the **Distance, kNN, Cross Validation, and Generative Models** section, you will learn about different types of discriminative and generative approaches for machine learning algorithms.

After completing this section, you will be able to:

* Use the **k-nearest neighbors (kNN)** algorithm.
* Understand the problems of **overtraining** and **oversmoothing**.
* Use **cross-validation** to reduce the **true error** and the **apparent error**.
* Use **generative models** such as **naive Bayes, quadratic discriminant analysis (qda),** and **linear discriminant analysis (lda)** for machine learning.

This section has three parts: **nearest neighbors, cross-validation,** and **generative models**.

## Distance

There is a link to the relevant section of the textbook: [Distance](https://rafalab.github.io/dsbook/large-datasets.html#distance)

**Key points**

* Most clustering and machine learning techniques rely on being able to define distance between observations, using features or predictors.
* With high dimensional data, a quick way to compute all the distances at once is to use the function dist(), which computes the distance between each row and produces an object of class dist():

d <- dist(x)

* We can also compute distances between predictors. If is the number of observations, the distance between two predictors, say 1 and 2, is:
* To compute the distance between all pairs of the 784 predictors, we can transpose the matrix first and then use dist():

d <- dist(t(x))

*Code*

if(!exists("mnist")) mnist <- read\_mnist()  
set.seed(0) # if using R 3.5 or earlier  
set.seed(0, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(0, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

ind <- which(mnist$train$labels %in% c(2,7)) %>% sample(500)  
  
#the predictors are in x and the labels in y  
x <- mnist$train$images[ind,]  
y <- mnist$train$labels[ind]  
  
y[1:3]

## [1] 7 7 2

x\_1 <- x[1,]  
x\_2 <- x[2,]  
x\_3 <- x[3,]  
  
#distance between two numbers  
sqrt(sum((x\_1 - x\_2)^2))

## [1] 2079.753

sqrt(sum((x\_1 - x\_3)^2))

## [1] 2252.129

sqrt(sum((x\_2 - x\_3)^2))

## [1] 2642.906

#compute distance using matrix algebra  
sqrt(crossprod(x\_1 - x\_2))

## [,1]  
## [1,] 2079.753

sqrt(crossprod(x\_1 - x\_3))

## [,1]  
## [1,] 2252.129

sqrt(crossprod(x\_2 - x\_3))

## [,1]  
## [1,] 2642.906

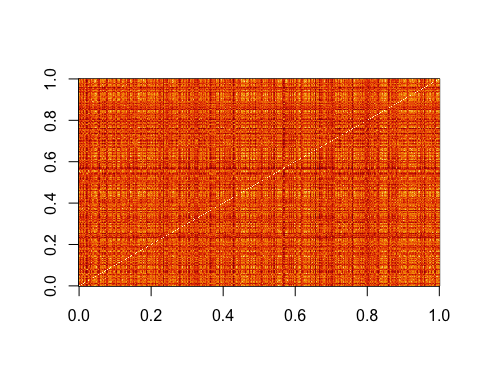
#compute distance between each row  
d <- dist(x)  
class(d)

## [1] "dist"

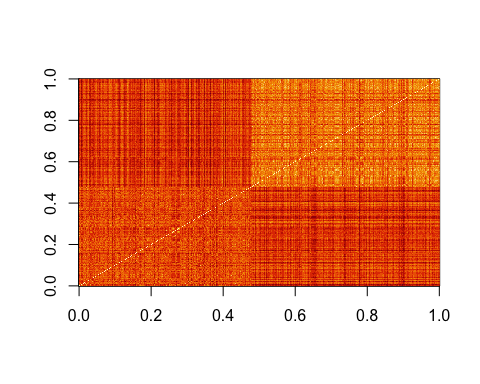
as.matrix(d)[1:3,1:3]

## 1 2 3  
## 1 0.000 2079.753 2252.129  
## 2 2079.753 0.000 2642.906  
## 3 2252.129 2642.906 0.000

#visualize these distances  
image(as.matrix(d))



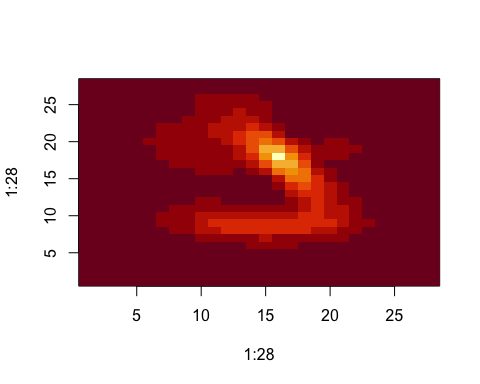
#order the distance by labels  
image(as.matrix(d)[order(y), order(y)])



#compute distance between predictors  
d <- dist(t(x))  
dim(as.matrix(d))

## [1] 784 784

d\_492 <- as.matrix(d)[492,]  
  
image(1:28, 1:28, matrix(d\_492, 28, 28))



## Comprehension Check - Distance

1. Load the following dataset:

data(tissue\_gene\_expression)

This dataset includes a matrix x:

dim(tissue\_gene\_expression$x)

## [1] 189 500

This matrix has the gene expression levels of 500 genes from 189 biological samples representing seven different tissues. The tissue type is stored in y:

table(tissue\_gene\_expression$y)

##   
## cerebellum colon endometrium hippocampus kidney liver   
## 38 34 15 31 39 26   
## placenta   
## 6

Which of the following lines of code computes the Euclidean distance between each observation and stores it in the object d?

d <- dist(tissue\_gene\_expression$x)

* ☐ A. d <- dist(tissue\_gene\_expression$x, distance='maximum')
* ☐ B. d <- dist(tissue\_gene\_expression)
* ☒ C. d <- dist(tissue\_gene\_expression$x)
* ☐ D. d <- cor(tissue\_gene\_expression$x)

1. Using the dataset from Q1, compare the distances between observations 1 and 2 (both cerebellum), observations 39 and 40 (both colon), and observations 73 and 74 (both endometrium).

Distance-wise, are samples from tissues of the same type closer to each other than tissues of different type?

ind <- c(1, 2, 39, 40, 73, 74)  
as.matrix(d)[ind,ind]

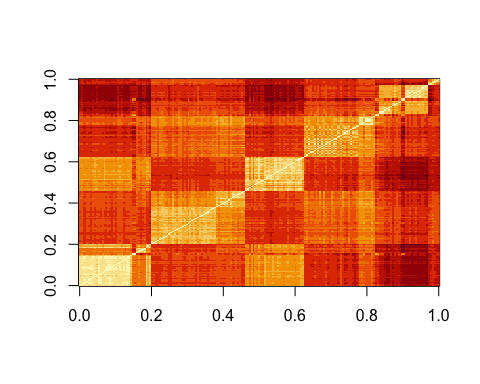
## cerebellum\_1 cerebellum\_2 colon\_1 colon\_2 endometrium\_1  
## cerebellum\_1 0.000000 7.005922 22.694801 22.699755 21.12763  
## cerebellum\_2 7.005922 0.000000 22.384821 22.069557 20.87910  
## colon\_1 22.694801 22.384821 0.000000 8.191935 14.99672  
## colon\_2 22.699755 22.069557 8.191935 0.000000 14.80355  
## endometrium\_1 21.127629 20.879099 14.996715 14.803545 0.00000  
## endometrium\_2 21.780792 20.674802 18.089213 17.004456 14.29405  
## endometrium\_2  
## cerebellum\_1 21.78079  
## cerebellum\_2 20.67480  
## colon\_1 18.08921  
## colon\_2 17.00446  
## endometrium\_1 14.29405  
## endometrium\_2 0.00000

* ☐ A. No, the samples from the same tissue type are not necessarily closer.
* ☐ B. The two colon samples are close to each other, but the samples from the other two tissues are not.
* ☐ C. The two cerebellum samples are close to each other, but the samples from the other two tissues are not.
* ☒ D. Yes, the samples from the same tissue type are closer to each other.

1. Make a plot of all the distances using the image() function to see if the pattern you observed in Q2 is general.

Which code would correctly make the desired plot?

image(as.matrix(d))



* ☐ A. image(d)
* ☒ B. image(as.matrix(d))
* ☐ C. d
* ☐ D. image()

## Knn

There is a link to the relevant section of the textbook: [k-nearest neighbors](https://rafalab.github.io/dsbook/examples-of-algorithms.html#k-nearest-neighbors)

**Key points**

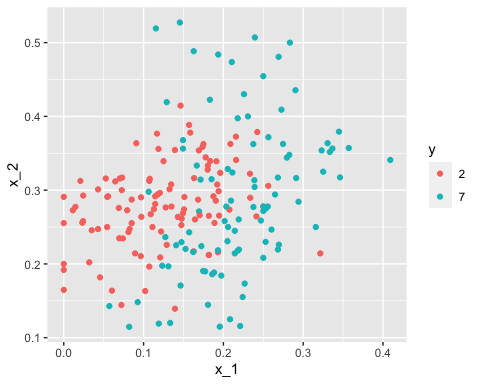
* **K-nearest neighbors (kNN)** estimates the conditional probabilities in a similar way to bin smoothing. However, kNN is easier to adapt to multiple dimensions.
* Using kNN, for any point for which we want an estimate of , we look for the **k nearest points** to and take an average of the 0s and 1s associated with these points. We refer to the set of points used to compute the average as the **neighborhood**. Larger values of k result in smoother estimates, while smaller values of k result in more flexible and more wiggly estimates.
* To implement the algorithm, we can use the knn3() function from the **caret** package. There are two ways to call this function:

1. We need to specify a formula and a data frame. The formula looks like this: . The predict() function for knn3 produces a probability for each class.
2. We can also call the function with the first argument being the matrix predictors and the second a vector of outcomes, like this:

x <- as.matrix(mnist\_27$train[,2:3])  
y <- mnist\_27$train$y  
knn\_fit <- knn3(x,y)

*Code*

data("mnist\_27")  
mnist\_27$test %>% ggplot(aes(x\_1, x\_2, color = y)) + geom\_point()



#logistic regression  
library(caret)  
fit\_glm <- glm(y~x\_1+x\_2, data=mnist\_27$train, family="binomial")  
p\_hat\_logistic <- predict(fit\_glm, mnist\_27$test)  
y\_hat\_logistic <- factor(ifelse(p\_hat\_logistic > 0.5, 7, 2))  
confusionMatrix(data = y\_hat\_logistic, reference = mnist\_27$test$y)$overall[1]

## Accuracy   
## 0.76

#fit knn model  
knn\_fit <- knn3(y ~ ., data = mnist\_27$train)  
  
x <- as.matrix(mnist\_27$train[,2:3])  
y <- mnist\_27$train$y  
knn\_fit <- knn3(x, y)  
  
knn\_fit <- knn3(y ~ ., data = mnist\_27$train, k=5)  
  
y\_hat\_knn <- predict(knn\_fit, mnist\_27$test, type = "class")  
confusionMatrix(data = y\_hat\_knn, reference = mnist\_27$test$y)$overall["Accuracy"]

## Accuracy   
## 0.815

## Over-training and Over-smoothing

There is a link to the relevant sections of the textbook: [Over-training](https://rafalab.github.io/dsbook/cross-validation.html#over-training) and [Over-smoothing](https://rafalab.github.io/dsbook/cross-validation.html#over-smoothing)

**Key points**

* **Over-training** is the reason that we have higher accuracy in the train set compared to the test set. Over-training is at its worst when we set . With , the estimate for each in the training set is obtained with just the corresponding to that point.
* When we try a larger , the might be so large that it does not permit enough flexibility. We call this **over-smoothing**.
* Note that if we use the test set to pick this , we should not expect the accompanying accuracy estimate to extrapolate to the real world. This is because even here we broke a golden rule of machine learning: **we selected the using the test set. Cross validation** also provides an estimate that takes this into account.

*Code*

y\_hat\_knn <- predict(knn\_fit, mnist\_27$train, type = "class")   
confusionMatrix(data = y\_hat\_knn, reference = mnist\_27$train$y)$overall["Accuracy"]

## Accuracy   
## 0.8825

y\_hat\_knn <- predict(knn\_fit, mnist\_27$test, type = "class")   
confusionMatrix(data = y\_hat\_knn, reference = mnist\_27$test$y)$overall["Accuracy"]

## Accuracy   
## 0.815

#fit knn with k=1  
knn\_fit\_1 <- knn3(y ~ ., data = mnist\_27$train, k = 1)  
y\_hat\_knn\_1 <- predict(knn\_fit\_1, mnist\_27$train, type = "class")  
confusionMatrix(data=y\_hat\_knn\_1, reference=mnist\_27$train$y)$overall[["Accuracy"]]

## [1] 0.995

y\_hat\_knn\_1 <- predict(knn\_fit\_1, mnist\_27$test, type = "class")  
confusionMatrix(data=y\_hat\_knn\_1, reference=mnist\_27$test$y)$overall[["Accuracy"]]

## [1] 0.74

#fit knn with k=401  
knn\_fit\_401 <- knn3(y ~ ., data = mnist\_27$train, k = 401)  
y\_hat\_knn\_401 <- predict(knn\_fit\_401, mnist\_27$test, type = "class")  
confusionMatrix(data=y\_hat\_knn\_401, reference=mnist\_27$test$y)$overall["Accuracy"]

## Accuracy   
## 0.79

#pick the k in knn  
ks <- seq(3, 251, 2)  
library(purrr)  
accuracy <- map\_df(ks, function(k){  
 fit <- knn3(y ~ ., data = mnist\_27$train, k = k)  
 y\_hat <- predict(fit, mnist\_27$train, type = "class")  
 cm\_train <- confusionMatrix(data = y\_hat, reference = mnist\_27$train$y)  
 train\_error <- cm\_train$overall["Accuracy"]  
 y\_hat <- predict(fit, mnist\_27$test, type = "class")  
 cm\_test <- confusionMatrix(data = y\_hat, reference = mnist\_27$test$y)  
 test\_error <- cm\_test$overall["Accuracy"]  
   
tibble(train = train\_error, test = test\_error)  
})  
  
  
#pick the k that maximizes accuracy using the estimates built on the test data  
ks[which.max(accuracy$test)]

## [1] 41

max(accuracy$test)

## [1] 0.86

## Comprehension Check - Nearest Neighbors

1. Previously, we used logistic regression to predict sex based on height. Now we are going to use knn to do the same. Set the seed to 1, then use the **caret** package to partition the **dslabs** heights data into a training and test set of equal size. Use the sapply() function to perform knn with k values of seq(1, 101, 3) and calculate F1 scores with the F\_meas() function using the default value of the relevant argument.

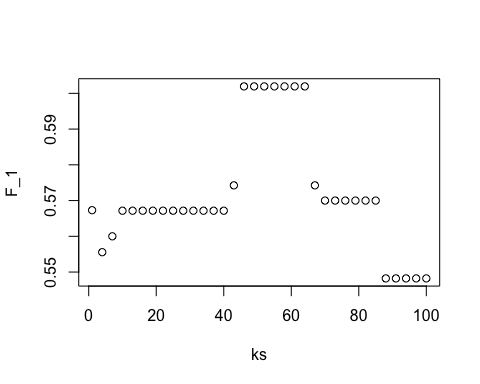
What is the max value of F\_1?

At what value of k does the max occur?

data("heights")  
  
# set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

test\_index <- createDataPartition(heights$sex, times = 1, p = 0.5, list = FALSE)  
test\_set <- heights[test\_index, ]  
train\_set <- heights[-test\_index, ]   
   
ks <- seq(1, 101, 3)  
F\_1 <- sapply(ks, function(k){  
 fit <- knn3(sex ~ height, data = train\_set, k = k)  
 y\_hat <- predict(fit, test\_set, type = "class") %>%   
 factor(levels = levels(train\_set$sex))  
 F\_meas(data = y\_hat, reference = test\_set$sex)  
})  
plot(ks, F\_1)



max(F\_1)

## [1] 0.6019417

ks[which.max(F\_1)]

## [1] 46

1. Next we will use the same gene expression example used in the Comprehension Check: Distance exercises. You can load it like this:

library(dslabs)  
library(caret)  
data("tissue\_gene\_expression")

First, set the seed to 1 and split the data into training and test sets with p = 0.5. Then, report the accuracy you obtain from predicting tissue type using KNN with k = seq(1, 11, 2) using sapply() or map\_df(). Note: use the createDataPartition() function outside of sapply() or map\_df().

# set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

y <- tissue\_gene\_expression$y  
x <- tissue\_gene\_expression$x  
test\_index <- createDataPartition(y, list = FALSE)  
sapply(seq(1, 11, 2), function(k){  
 fit <- knn3(x[-test\_index,], y[-test\_index], k = k)  
 y\_hat <- predict(fit, newdata = data.frame(x=x[test\_index,]),  
 type = "class")  
mean(y\_hat == y[test\_index])  
})

## [1] 0.9895833 0.9687500 0.9479167 0.9166667 0.9166667 0.9062500

## K-fold cross validation

There is a link to the relevant section of the textbook: [K-fold cross validation](https://rafalab.github.io/dsbook/cross-validation.html#k-fold-cross-validation)

**Key points**

* For **-fold cross validation**, we divide the dataset into a training set and a test set. We train our algorithm exclusively on the training set and use the test set only for evaluation purposes.
* For each set of algorithm parameters being considered, we want an **estimate of the MSE and then we will choose the parameters with the smallest MSE**. In -fold cross validation, we randomly split the observations into non-overlapping sets, and repeat the calculation for MSE for each of these sets. Then, we compute the average MSE and obtain an estimate of our loss. Finally, we can select the optimal parameter that minimized the MSE.
* In terms of how to select for cross validation, **larger values of are preferable but they will also take much more** computational time. For this reason, the choices of and are common.

## Comprehension Check - Cross-validation

1. Generate a set of random predictors and outcomes using the following code:

# set.seed(1996) #if you are using R 3.5 or earlier  
set.seed(1996, sample.kind="Rounding") #if you are using R 3.6 or later

## Warning in set.seed(1996, sample.kind = "Rounding"): non-uniform 'Rounding'  
## sampler used

n <- 1000  
p <- 10000  
x <- matrix(rnorm(n\*p), n, p)  
colnames(x) <- paste("x", 1:ncol(x), sep = "\_")  
y <- rbinom(n, 1, 0.5) %>% factor()  
  
x\_subset <- x[ ,sample(p, 100)]

Because x and y are completely independent, you should not be able to predict y using x with accuracy greater than 0.5. Confirm this by running cross-validation using logistic regression to fit the model. Because we have so many predictors, we selected a random sample x\_subset. Use the subset when training the model.

Which code correctly performs this cross-validation?

fit <- train(x\_subset, y, method = "glm")  
fit$results

## parameter Accuracy Kappa AccuracySD KappaSD  
## 1 none 0.5078406 0.01318925 0.02336971 0.04626366

* ☐ A.

fit <- train(x\_subset, y)  
fit$results

* ☒ B.

fit <- train(x\_subset, y, method = "glm")  
fit$results

* ☐ C.

fit <- train(y, x\_subset, method = "glm")  
fit$results

* ☐ D.

fit <- test(x\_subset, y, method = "glm")  
fit$results

1. Now, instead of using a random selection of predictors, we are going to search for those that are most predictive of the outcome. We can do this by comparing the values for the group to those in the group, for each predictor, using a t-test. You can do perform this step like this:

if(!require(BiocManager)) install.packages("BiocManager")

## Loading required package: BiocManager

## Bioconductor version 3.11 (BiocManager 1.30.10), ?BiocManager::install for help

## Bioconductor version '3.11' is out-of-date; the current release version '3.12'  
## is available with R version '4.0'; see https://bioconductor.org/install

BiocManager::install("genefilter")

## Bioconductor version 3.11 (BiocManager 1.30.10), R 4.0.2 (2020-06-22)

## Installing package(s) 'genefilter'

##   
## The downloaded binary packages are in  
## /var/folders/6m/nz2p76pn679b692c99t644bm0000gn/T//Rtmp43TP2j/downloaded\_packages

library(genefilter)

##   
## Attaching package: 'genefilter'

## The following objects are masked from 'package:matrixStats':  
##   
## rowSds, rowVars

## The following object is masked from 'package:MASS':  
##   
## area

## The following object is masked from 'package:readr':  
##   
## spec

tt <- colttests(x, y)

Which of the following lines of code correctly creates a vector of the p-values called pvals?

pvals <- tt$p.value

* ☐ A. pvals <- tt$dm
* ☐ B. pvals <- tt$statistic
* ☐ C. pvals <- tt
* ☒ D. pvals <- tt$p.value

1. Create an index ind with the column numbers of the predictors that were “statistically significantly” associated with y. Use a p-value cutoff of 0.01 to define “statistically significantly.”

How many predictors survive this cutoff?

ind <- which(pvals <= 0.01)  
length(ind)

## [1] 108

1. Now re-run the cross-validation after redefinining x\_subset to be the subset of x defined by the columns showing “statistically significant” association with y.

What is the accuracy now?

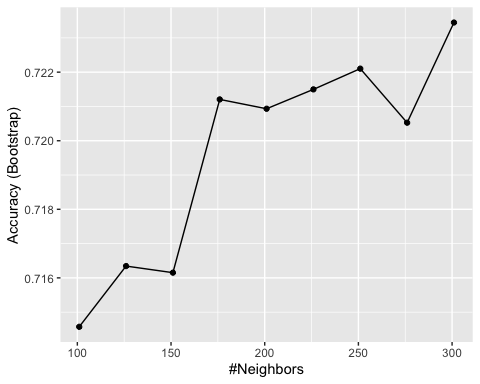
x\_subset <- x[,ind]  
fit <- train(x\_subset, y, method = "glm")  
fit$results

## parameter Accuracy Kappa AccuracySD KappaSD  
## 1 none 0.7571395 0.5134142 0.01922097 0.03805696

1. Re-run the cross-validation again, but this time using kNN. Try out the following grid k = seq(101, 301, 25) of tuning parameters. Make a plot of the resulting accuracies.

Which code is correct?

fit <- train(x\_subset, y, method = "knn", tuneGrid = data.frame(k = seq(101, 301, 25)))  
ggplot(fit)



* ☒ A.

fit <- train(x\_subset, y, method = "knn", tuneGrid = data.frame(k = seq(101, 301, 25)))  
ggplot(fit)

* ☐ B.

fit <- train(x\_subset, y, method = "knn")  
ggplot(fit)

* ☐ C.

fit <- train(x\_subset, y, method = "knn", tuneGrid = data.frame(k = seq(103, 301, 25)))  
ggplot(fit)

* ☐ D.

fit <- train(x\_subset, y, method = "knn", tuneGrid = data.frame(k = seq(101, 301, 5)))  
ggplot(fit)

1. In the previous exercises, we see that despite the fact that x and y are completely independent, we were able to predict y with accuracy higher than 70%. We must be doing something wrong then.

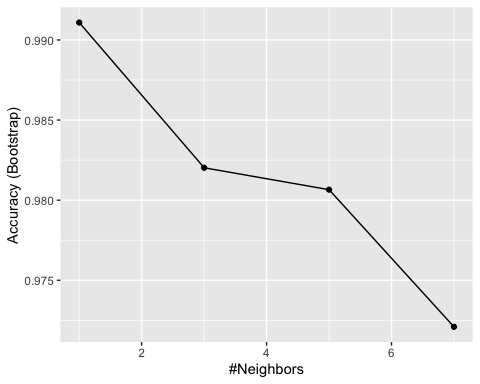
What is it?

* ☐ A. The function train() estimates accuracy on the same data it uses to train the algorithm.
* ☐ B. We are overfitting the model by including 100 predictors.
* ☒ C. We used the entire dataset to select the columns used in the model.
* ☐ D. The high accuracy is just due to random variability.

1. Use the train() function with kNN to select the best k for predicting tissue from gene expression on the tissue\_gene\_expression dataset from **dslabs**. Try k = seq(1,7,2) for tuning parameters. For this question, do not split the data into test and train sets (understand this can lead to overfitting, but ignore this for now).

What value of k results in the highest accuracy?

data("tissue\_gene\_expression")  
fit <- with(tissue\_gene\_expression, train(x, y, method = "knn", tuneGrid = data.frame( k = seq(1, 7, 2))))  
ggplot(fit)



fit$results

## k Accuracy Kappa AccuracySD KappaSD  
## 1 1 0.9910881 0.9892456 0.01127300 0.01349575  
## 2 3 0.9820243 0.9781738 0.01389797 0.01683721  
## 3 5 0.9806558 0.9765964 0.02484299 0.02995977  
## 4 7 0.9720962 0.9660514 0.03007035 0.03650127

## Bootstrap

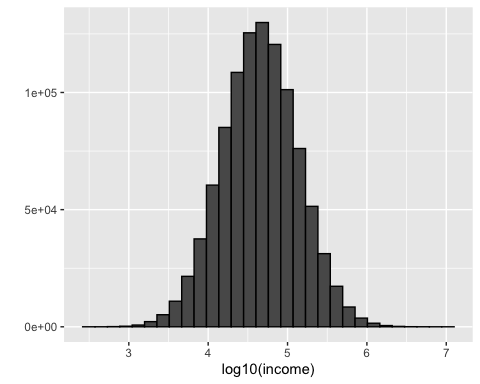
There is a link to the relevant section of the textbook: [Bootstrap](https://rafalab.github.io/dsbook/cross-validation.html#bootstrap)

**Key points**

* When we don’t have access to the entire population, we can use **bootstrap** to estimate the population median .
* The bootstrap permits us to **approximate a Monte Carlo simulation** without access to the entire distribution. The general idea is relatively simple. We act as if the observed sample is the population. We then sample datasets (with replacement) of the same sample size as the original dataset. Then we compute the summary statistic, in this case the median, on this bootstrap sample.
* Note that we can use ideas similar to those used in the bootstrap in **cross validation**: instead of dividing the data into equal partitions, we simply bootstrap many times.

*Code*

n <- 10^6  
income <- 10^(rnorm(n, log10(45000), log10(3)))  
qplot(log10(income), bins = 30, color = I("black"))



m <- median(income)  
m

## [1] 44986.86

set.seed(1)  
#use set.seed(1, sample.kind="Rounding") instead if using R 3.6 or later  
N <- 250  
X <- sample(income, N)  
M<- median(X)  
M

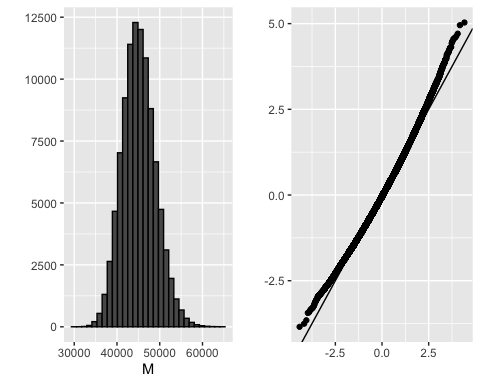
## [1] 47024.18

library(gridExtra)

##   
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':  
##   
## combine

B <- 10^5  
M <- replicate(B, {  
 X <- sample(income, N)  
 median(X)  
})  
p1 <- qplot(M, bins = 30, color = I("black"))  
p2 <- qplot(sample = scale(M)) + geom\_abline()  
grid.arrange(p1, p2, ncol = 2)



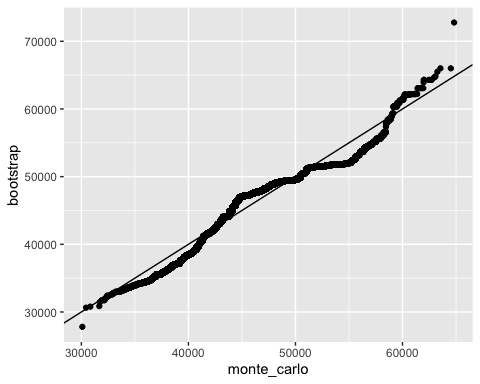
mean(M)

## [1] 45132.14

sd(M)

## [1] 3912.368

B <- 10^5  
M\_star <- replicate(B, {  
 X\_star <- sample(X, N, replace = TRUE)  
 median(X\_star)  
})  
  
tibble(monte\_carlo = sort(M), bootstrap = sort(M\_star)) %>%  
 qplot(monte\_carlo, bootstrap, data = .) +   
 geom\_abline()



quantile(M, c(0.05, 0.95))

## 5% 95%   
## 38996.50 51811.42

quantile(M\_star, c(0.05, 0.95))

## 5% 95%   
## 37112.39 51462.43

median(X) + 1.96 \* sd(X) / sqrt(N) \* c(-1, 1)

## [1] 33154.08 60894.28

mean(M) + 1.96 \* sd(M) \* c(-1,1)

## [1] 37463.90 52800.38

mean(M\_star) + 1.96 \* sd(M\_star) \* c(-1, 1)

## [1] 36913.52 53897.73

## Comprehension Check - Bootstrap

1. The createResample() function can be used to create bootstrap samples. For example, we can create the indexes for 10 bootstrap samples for the mnist\_27 dataset like this:

data(mnist\_27)  
# set.seed(1995) # if R 3.5 or earlier  
set.seed(1995, sample.kind="Rounding") # if R 3.6 or later

## Warning in set.seed(1995, sample.kind = "Rounding"): non-uniform 'Rounding'  
## sampler used

indexes <- createResample(mnist\_27$train$y, 10)

How many times do 3, 4, and 7 appear in the first resampled index?

sum(indexes[[1]] == 3)

## [1] 1

sum(indexes[[1]] == 4)

## [1] 4

sum(indexes[[1]] == 7)

## [1] 0

1. We see that some numbers appear more than once and others appear no times. This has to be this way for each dataset to be independent. Repeat the exercise for all the resampled indexes.

What is the total number of times that 3 appears in all of the resampled indexes?

x=sapply(indexes, function(ind){  
 sum(ind == 3)  
})  
sum(x)

## [1] 11

1. Generate a random dataset using the following code:

y <- rnorm(100, 0, 1)

Estimate the 75th quantile, which we know is qnorm(0.75), with the sample quantile: quantile(y, 0.75).

Now, set the seed to 1 and perform a Monte Carlo simulation with 10,000 repetitions, generating the random dataset and estimating the 75th quantile each time. What is the expected value and standard error of the 75th quantile?

Report all answers to at least 3 decimal digits.

# set.seed(1) # # if R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

B <- 10000  
q\_75 <- replicate(B, {  
 y <- rnorm(100, 0, 1)  
 quantile(y, 0.75)  
})  
  
mean(q\_75)

## [1] 0.6656107

sd(q\_75)

## [1] 0.1353809

1. In practice, we can’t run a Monte Carlo simulation. Use the sample:

# set.seed(1) # if R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

y <- rnorm(100, 0, 1)

Set the seed to 1 again after generating y and use 10 bootstrap samples to estimate the expected value and standard error of the 75th quantile.

# set.seed(1) # if R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

y <- rnorm(100, 0, 1)  
  
# set.seed(1) # if R 3.5 or earlier  
set.seed(1, sample.kind="Rounding") # if R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

indexes <- createResample(y, 10)  
q\_75\_star <- sapply(indexes, function(ind){  
 y\_star <- y[ind]  
 quantile(y\_star, 0.75)  
})  
mean(q\_75\_star)

## [1] 0.7312648

sd(q\_75\_star)

## [1] 0.07419278

1. Repeat the exercise from Q4 but with 10,000 bootstrap samples instead of 10. Set the seed to 1 first.

# set.seed(1) # # if R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

indexes <- createResample(y, 10000)  
q\_75\_star <- sapply(indexes, function(ind){  
 y\_star <- y[ind]  
 quantile(y\_star, 0.75)  
})  
mean(q\_75\_star)

## [1] 0.6737512

sd(q\_75\_star)

## [1] 0.0930575

1. When doing bootstrap sampling, the simulated samples are drawn from the empirical distribution of the original data.

True or False: The bootstrap is particularly useful in situations when we do not have access to the distribution or it is unknown.

* ☒ A. True
* ☐ B. False

## Generative Models

There is a link to the relevant section of the textbook: [Generative models](https://rafalab.github.io/dsbook/examples-of-algorithms.html#generative-models)

\*\*Key points

* **Discriminative approaches** estimate the conditional probability directly and do not consider the distribution of the predictors.
* **Generative models** are methods that model the joint distribution and (we model how the entire data, and , are generated).

## Naive Bayes

There is a link to the relevant section of the textbook: [Naive Bayes](https://rafalab.github.io/dsbook/examples-of-algorithms.html#naive-bayes)

**Key points**

* Bayes’ rule:

with and representing the distribution functions of the predictor for the two classes and .

* The **Naive Bayes** approach is similar to the logistic regression prediction mathematically. However, we leave the demonstration to a more advanced text, such as [The Elements of Statistical Learning by Hastie, Tibshirani, and Friedman](https://web.stanford.edu/~hastie/Papers/ESLII.pdf).

*Code*

# Generating train and test set  
data("heights")  
y <- heights$height  
set.seed(2)  
test\_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)  
train\_set <- heights %>% slice(-test\_index)  
test\_set <- heights %>% slice(test\_index)  
  
# Estimating averages and standard deviations  
params <- train\_set %>%  
 group\_by(sex) %>%  
 summarize(avg = mean(height), sd = sd(height))

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

params

## # A tibble: 2 x 3  
## sex avg sd  
## <fct> <dbl> <dbl>  
## 1 Female 64.5 4.02  
## 2 Male 69.3 3.52

# Estimating the prevalence  
pi <- train\_set %>% summarize(pi=mean(sex=="Female")) %>% pull(pi)  
pi

## [1] 0.2290076

# Getting an actual rule  
x <- test\_set$height  
f0 <- dnorm(x, params$avg[2], params$sd[2])  
f1 <- dnorm(x, params$avg[1], params$sd[1])  
p\_hat\_bayes <- f1\*pi / (f1\*pi + f0\*(1 - pi))

## Controlling Prevalence

There is a link to the relevant section of the textbook: [Controlling prevalence](https://rafalab.github.io/dsbook/examples-of-algorithms.html#controlling-prevalence)

**Key points**

* The Naive Bayes approach includes a **parameter to account for differences in prevalence** . If we use hats to denote the estimates, we can write as:
* The Naive Bayes approach gives us a direct way to correct the imbalance between sensitivity and specificity by simply forcing to be whatever value we want it to be in order to better **balance specificity and sensitivity**.

*Code*

# Computing sensitivity  
y\_hat\_bayes <- ifelse(p\_hat\_bayes > 0.5, "Female", "Male")  
sensitivity(data = factor(y\_hat\_bayes), reference = factor(test\_set$sex))

## [1] 0.2627119

# Computing specificity  
specificity(data = factor(y\_hat\_bayes), reference = factor(test\_set$sex))

## [1] 0.9534314

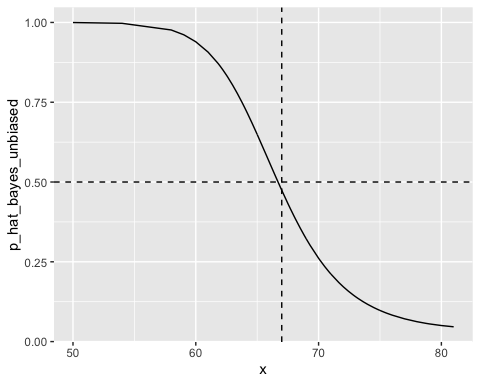
# Changing the cutoff of the decision rule  
p\_hat\_bayes\_unbiased <- f1 \* 0.5 / (f1 \* 0.5 + f0 \* (1 - 0.5))  
y\_hat\_bayes\_unbiased <- ifelse(p\_hat\_bayes\_unbiased > 0.5, "Female", "Male")  
sensitivity(data = factor(y\_hat\_bayes\_unbiased), reference = factor(test\_set$sex))

## [1] 0.7118644

specificity(data = factor(y\_hat\_bayes\_unbiased), reference = factor(test\_set$sex))

## [1] 0.8210784

# Draw plot  
qplot(x, p\_hat\_bayes\_unbiased, geom = "line") +  
 geom\_hline(yintercept = 0.5, lty = 2) +  
 geom\_vline(xintercept = 67, lty = 2)



## qda and lda

There is a link to the relevant sections of the textbook: [Quadratic discriminant analysis](https://rafalab.github.io/dsbook/examples-of-algorithms.html#quadratic-discriminant-analysis) and [Linear discriminant analysis](https://rafalab.github.io/dsbook/examples-of-algorithms.html#linear-discriminant-analysis)

**Key points**

* **Quadratic discriminant analysis (QDA)** is a version of Naive Bayes in which we assume that the distributions and are multivariate normal.
* QDA can work well with a few predictors, but it becomes **harder to use as the number of predictors increases**. Once the number of parameters approaches the size of our data, the method becomes impractical due to overfitting.
* Forcing the assumption that all predictors share the same standard deviations and correlations, the boundary will be a line, just as with logistic regression. For this reason, we call the method **linear discriminant analysis (LDA)**.
* In the case of LDA, the lack of flexibility **does not permit us to capture the non-linearity** in the true conditional probability function.

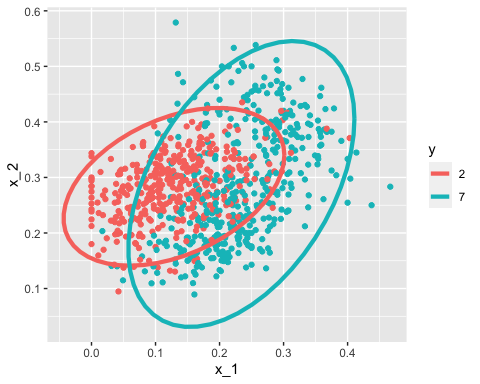
*Code*

**QDA**

# Load data  
data("mnist\_27")  
  
# Estimate parameters from the data  
params <- mnist\_27$train %>%  
 group\_by(y) %>%  
 summarize(avg\_1 = mean(x\_1), avg\_2 = mean(x\_2),  
 sd\_1 = sd(x\_1), sd\_2 = sd(x\_2),  
 r = cor(x\_1, x\_2))

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

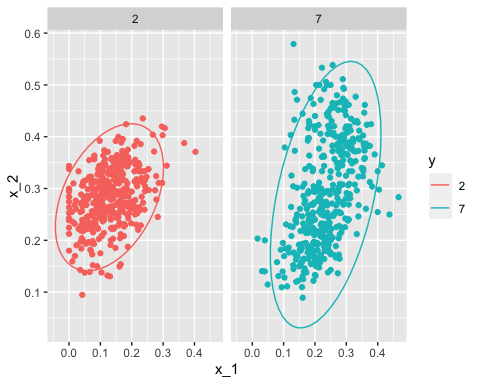
# Contour plots  
mnist\_27$train %>% mutate(y = factor(y)) %>%  
 ggplot(aes(x\_1, x\_2, fill = y, color = y)) +  
 geom\_point(show.legend = FALSE) +  
 stat\_ellipse(type="norm", lwd = 1.5)



# Fit model  
library(caret)  
train\_qda <- train(y ~., method = "qda", data = mnist\_27$train)  
# Obtain predictors and accuracy  
y\_hat <- predict(train\_qda, mnist\_27$test)  
confusionMatrix(data = y\_hat, reference = mnist\_27$test$y)$overall["Accuracy"]

## Accuracy   
## 0.82

# Draw separate plots for 2s and 7s  
mnist\_27$train %>% mutate(y = factor(y)) %>%  
 ggplot(aes(x\_1, x\_2, fill = y, color = y)) +  
 geom\_point(show.legend = FALSE) +  
 stat\_ellipse(type="norm") +  
 facet\_wrap(~y)



**LDA**

params <- mnist\_27$train %>%  
 group\_by(y) %>%  
 summarize(avg\_1 = mean(x\_1), avg\_2 = mean(x\_2),  
 sd\_1 = sd(x\_1), sd\_2 = sd(x\_2),  
 r = cor(x\_1, x\_2))

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

params <- params %>% mutate(sd\_1 = mean(sd\_1), sd\_2 = mean(sd\_2), r = mean(r))  
train\_lda <- train(y ~., method = "lda", data = mnist\_27$train)  
y\_hat <- predict(train\_lda, mnist\_27$test)  
confusionMatrix(data = y\_hat, reference = mnist\_27$test$y)$overall["Accuracy"]

## Accuracy   
## 0.75

## Case Study - More than Three Classes

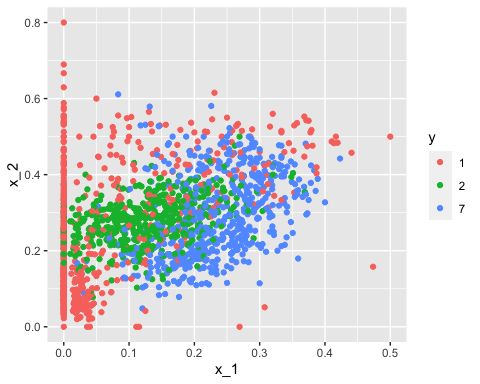
There is a link to the relevant section of the textbook: [Case study: more than three classes](https://rafalab.github.io/dsbook/examples-of-algorithms.html#case-study-more-than-three-classes)

**Key points**

* In this case study, we will briefly give a slightly more complex example: one with **3 classes instead of 2**. Then we will fit QDA, LDA, and KNN models for prediction.
* Generative models can be very powerful, but only when we are able to **successfully approximate the joint distribution** of predictors conditioned on each class.

*Code*

if(!exists("mnist"))mnist <- read\_mnist()  
  
set.seed(3456) #use set.seed(3456, sample.kind="Rounding") in R 3.6 or later  
index\_127 <- sample(which(mnist$train$labels %in% c(1,2,7)), 2000)  
y <- mnist$train$labels[index\_127]   
x <- mnist$train$images[index\_127,]  
index\_train <- createDataPartition(y, p=0.8, list = FALSE)  
  
# get the quadrants  
# temporary object to help figure out the quadrants  
row\_column <- expand.grid(row=1:28, col=1:28)  
upper\_left\_ind <- which(row\_column$col <= 14 & row\_column$row <= 14)  
lower\_right\_ind <- which(row\_column$col > 14 & row\_column$row > 14)  
  
# binarize the values. Above 200 is ink, below is no ink  
x <- x > 200   
  
# cbind proportion of pixels in upper right quadrant and proportion of pixels in lower right quadrant  
x <- cbind(rowSums(x[ ,upper\_left\_ind])/rowSums(x),  
 rowSums(x[ ,lower\_right\_ind])/rowSums(x))   
  
train\_set <- data.frame(y = factor(y[index\_train]),  
 x\_1 = x[index\_train,1],  
 x\_2 = x[index\_train,2])  
  
test\_set <- data.frame(y = factor(y[-index\_train]),  
 x\_1 = x[-index\_train,1],  
 x\_2 = x[-index\_train,2])  
  
train\_set %>% ggplot(aes(x\_1, x\_2, color=y)) + geom\_point()



train\_qda <- train(y ~ ., method = "qda", data = train\_set)  
predict(train\_qda, test\_set, type = "prob") %>% head()

## 1 2 7  
## 1 0.22232613 0.6596410 0.11803290  
## 2 0.19256640 0.4535212 0.35391242  
## 3 0.62749331 0.3220448 0.05046191  
## 4 0.04623381 0.1008304 0.85293583  
## 5 0.21671529 0.6229295 0.16035523  
## 6 0.12669776 0.3349700 0.53833219

predict(train\_qda, test\_set) %>% head()

## [1] 2 2 1 7 2 7  
## Levels: 1 2 7

confusionMatrix(predict(train\_qda, test\_set), test\_set$y)$table

## Reference  
## Prediction 1 2 7  
## 1 111 17 7  
## 2 14 80 17  
## 7 19 25 109

confusionMatrix(predict(train\_qda, test\_set), test\_set$y)$overall["Accuracy"]

## Accuracy   
## 0.7518797

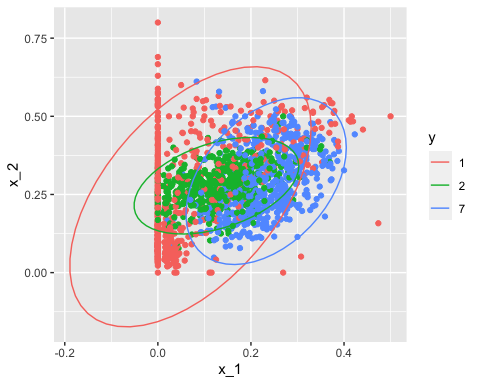
train\_lda <- train(y ~ ., method = "lda", data = train\_set)  
confusionMatrix(predict(train\_lda, test\_set), test\_set$y)$overall["Accuracy"]

## Accuracy   
## 0.6641604

train\_knn <- train(y ~ ., method = "knn", tuneGrid = data.frame(k = seq(15, 51, 2)),  
 data = train\_set)  
confusionMatrix(predict(train\_knn, test\_set), test\_set$y)$overall["Accuracy"]

## Accuracy   
## 0.7719298

train\_set %>% mutate(y = factor(y)) %>% ggplot(aes(x\_1, x\_2, fill = y, color=y)) + geom\_point(show.legend = FALSE) + stat\_ellipse(type="norm")



## Comprehension Check - Generative Models

In the following exercises, we are going to apply LDA and QDA to the tissue\_gene\_expression dataset from **dslabs**. We will start with simple examples based on this dataset and then develop a realistic example.

1. Create a dataset of samples from just cerebellum and hippocampus, two parts of the brain, and a predictor matrix with 10 randomly selected columns using the following code:

data("tissue\_gene\_expression")  
   
# set.seed(1993) #if using R 3.5 or earlier  
set.seed(1993, sample.kind="Rounding") # if using R 3.6 or later

## Warning in set.seed(1993, sample.kind = "Rounding"): non-uniform 'Rounding'  
## sampler used

ind <- which(tissue\_gene\_expression$y %in% c("cerebellum", "hippocampus"))  
y <- droplevels(tissue\_gene\_expression$y[ind])  
x <- tissue\_gene\_expression$x[ind, ]  
x <- x[, sample(ncol(x), 10)]

Use the train() function to estimate the accuracy of LDA. For this question, use the version of x and y created with the code above: do not split them or tissue\_gene\_expression into training and test sets (understand this can lead to overfitting). Report the accuracy from the train() results (do not make predictions).

What is the accuracy? Enter your answer as a percentage or decimal (eg “50%” or “0.50”) to at least the thousandths place.

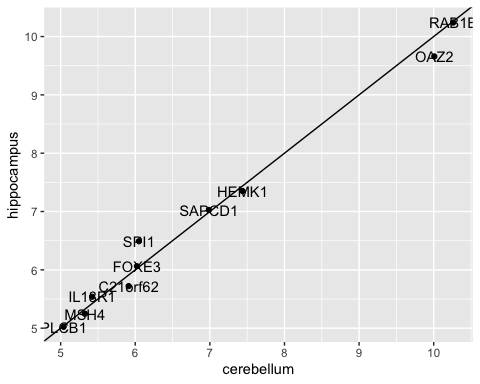
fit\_lda <- train(x, y, method = "lda")  
fit\_lda$results["Accuracy"]

## Accuracy  
## 1 0.8707879

1. In this case, LDA fits two 10-dimensional normal distributions. Look at the fitted model by looking at the finalModel component of the result of train(). Notice there is a component called means that includes the estimated means of both distributions. Plot the mean vectors against each other and determine which predictors (genes) appear to be driving the algorithm.

Which TWO genes appear to be driving the algorithm (i.e. the two genes with the highest means)?

t(fit\_lda$finalModel$means) %>% data.frame() %>%  
 mutate(predictor\_name = rownames(.)) %>%  
 ggplot(aes(cerebellum, hippocampus, label = predictor\_name)) +  
 geom\_point() +  
 geom\_text() +  
 geom\_abline()



* ☐ A. PLCB1
* ☒ B. RAB1B
* ☐ C. MSH4
* ☒ D. OAZ2
* ☐ E. SPI1
* ☐ F. SAPCD1
* ☐ G. HEMK1

1. Repeat the exercise in Q1 with QDA.

Create a dataset of samples from just cerebellum and hippocampus, two parts of the brain, and a predictor matrix with 10 randomly selected columns using the following code:

data("tissue\_gene\_expression")  
   
set.seed(1993) #set.seed(1993, sample.kind="Rounding") if using R 3.6 or later  
ind <- which(tissue\_gene\_expression$y %in% c("cerebellum", "hippocampus"))  
y <- droplevels(tissue\_gene\_expression$y[ind])  
x <- tissue\_gene\_expression$x[ind, ]  
x <- x[, sample(ncol(x), 10)]

Use the train() function to estimate the accuracy of QDA. For this question, use the version of x and y created above instead of the default from tissue\_gene\_expression. Do not split them into training and test sets (understand this can lead to overfitting).

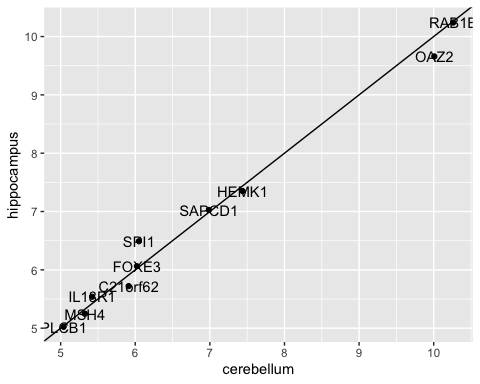
What is the accuracy?

fit\_qda <- train(x, y, method = "qda")  
fit\_qda$results["Accuracy"]

## Accuracy  
## 1 0.8147954

1. Which TWO genes drive the algorithm when using QDA instead of LDA (i.e. the two genes with the highest means)?

t(fit\_qda$finalModel$means) %>% data.frame() %>%  
 mutate(predictor\_name = rownames(.)) %>%  
 ggplot(aes(cerebellum, hippocampus, label = predictor\_name)) +  
 geom\_point() +  
 geom\_text() +  
 geom\_abline()



* ☐ A. PLCB1
* ☒ B. RAB1B
* ☐ C. MSH4
* ☒ D. OAZ2
* ☐ E. SPI1
* ☐ F. SAPCD1
* ☐ G. HEMK1

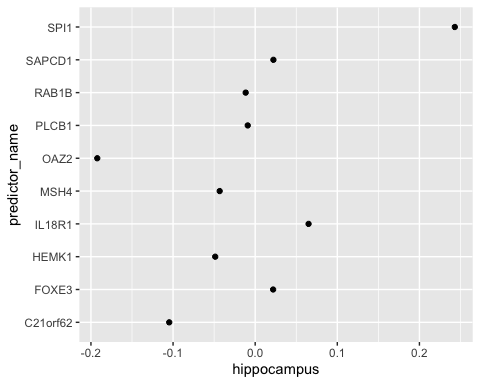
1. One thing we saw in the previous plots is that the values of the predictors correlate in both groups: some predictors are low in both groups and others high in both groups. The mean value of each predictor found in colMeans(x) is not informative or useful for prediction and often for purposes of interpretation, it is useful to center or scale each column. This can be achieved with the preProcess argument in train(). Re-run LDA with preProcess = "center". Note that accuracy does not change, but it is now easier to identify the predictors that differ more between groups than based on the plot made in Q2.

Which TWO genes drive the algorithm after performing the scaling?

fit\_lda <- train(x, y, method = "lda", preProcess = "center")  
fit\_lda$results["Accuracy"]

## Accuracy  
## 1 0.8595389

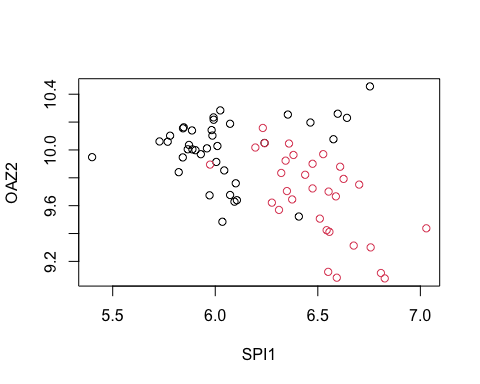
t(fit\_lda$finalModel$means) %>% data.frame() %>%  
 mutate(predictor\_name = rownames(.)) %>%  
 ggplot(aes(predictor\_name, hippocampus)) +  
 geom\_point() +  
 coord\_flip()



* ☐ A. C21orf62
* ☐ B. PLCB1
* ☐ C. RAB1B
* ☐ D. MSH4
* ☒ E. OAZ2
* ☒ F. SPI1
* ☐ G. SAPCD1
* ☐ H. IL18R1

You can see that it is different genes driving the algorithm now. This is because the predictor means change. In the previous exercises we saw that both LDA and QDA approaches worked well. For further exploration of the data, you can plot the predictor values for the two genes with the largest differences between the two groups in a scatter plot to see how they appear to follow a bivariate distribution as assumed by the LDA and QDA approaches, coloring the points by the outcome, using the following code:

d <- apply(fit\_lda$finalModel$means, 2, diff)  
ind <- order(abs(d), decreasing = TRUE)[1:2]  
plot(x[, ind], col = y)



1. Now we are going to increase the complexity of the challenge slightly. Repeat the LDA analysis from Q5 but using all tissue types. Use the following code to create your dataset:

data("tissue\_gene\_expression")  
   
# set.seed(1993) # if using R 3.5 or earlier  
set.seed(1993, sample.kind="Rounding") # if using R 3.6 or later

## Warning in set.seed(1993, sample.kind = "Rounding"): non-uniform 'Rounding'  
## sampler used

y <- tissue\_gene\_expression$y  
x <- tissue\_gene\_expression$x  
x <- x[, sample(ncol(x), 10)]

What is the accuracy using LDA?

fit\_lda <- train(x, y, method = "lda", preProcess = c("center"))  
fit\_lda$results["Accuracy"]

## Accuracy  
## 1 0.8194837

# Section 5 - Classification with More than Two Classes and the Caret Package

In the **Classification with More than Two Classes and the Caret Package** section, you will 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.

After completing this section, you will be able to:

* Use **classification and regression trees**.
* Use **classification (decision) trees**.
* Apply **random forests** to address the shortcomings of decision trees.
* Use the **caret** package to implement a variety of machine learning algorithms.

This section has three parts: **classification with more than two classes, caret package**, and a **set of exercises** on the Titanic.

## Trees Motivation

There is a link to the relevant section of the textbook: [The curse of dimensionality](https://rafalab.github.io/dsbook/examples-of-algorithms.html#the-curse-of-dimensionality)

**Key points**

* LDA and QDA are not **meant to be used with many predictors**  because the number of parameters needed to be estimated becomes too large.
* **Curse of dimensionality:** For kernel methods such as kNN or local regression, when they have multiple predictors used, the span/neighborhood/window made to include a given percentage of the data become large. With larger neighborhoods, our methods lose flexibility. The dimension here refers to the fact that when we have predictors, the distance between two observations is computed in -dimensional space.

## Classification and Regression Trees (CART)

There is a link to the relevant sections of the textbook: [CART motivation](https://rafalab.github.io/dsbook/examples-of-algorithms.html#cart-motivation) and [Regression trees](https://rafalab.github.io/dsbook/examples-of-algorithms.html#regression-trees)

**Key points**

* A tree is basically a **flow chart of yes or no questions**. The general idea of the methods we are describing is to define an algorithm that uses data to create these trees with predictions at the ends, referred to as nodes.
* When the outcome is continuous, we call the decision tree method a **regression tree**.
* Regression and decision trees operate by predicting an outcome variable by **partitioning the predictors**.
* The general idea here is to **build a decision tree** and, at end of each node, obtain a predictor . Mathematically, we are **partitioning the predictor space** into non-overlapping regions, , , …, and then for any predictor that falls within region , estimate with the average of the training observations for which the associated predictor in also in .
* To pick and its value , we find the pair that **minimizes the residual sum of squares (RSS)**:
* To fit the regression tree model, we can use the rpart() function in the **rpart** package.
* Two common parameters used for partition decision are the **complexity parameter** (cp) and the **minimum number of observations required in a partition** before partitioning it further (minsplit in the **rpart** package).
* If we already have a tree and want to apply a higher cp value, we can use the prune() function. We call this pruning a tree because we are snipping off partitions that do not meet a cp criterion.

*Code*

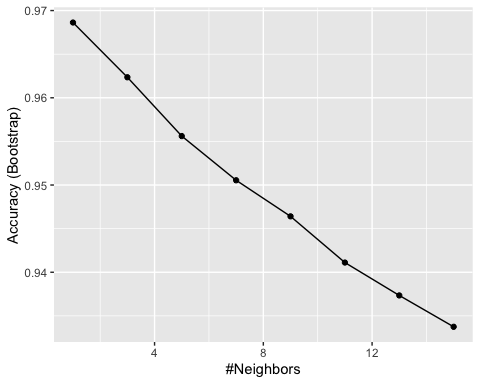
# Load data  
data("olive")  
olive %>% as\_tibble()

## # A tibble: 572 x 10  
## region area palmitic palmitoleic stearic oleic linoleic linolenic arachidic  
## <fct> <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 South… Nort… 10.8 0.75 2.26 78.2 6.72 0.36 0.6   
## 2 South… Nort… 10.9 0.73 2.24 77.1 7.81 0.31 0.61  
## 3 South… Nort… 9.11 0.54 2.46 81.1 5.49 0.31 0.63  
## 4 South… Nort… 9.66 0.570 2.4 79.5 6.19 0.5 0.78  
## 5 South… Nort… 10.5 0.67 2.59 77.7 6.72 0.5 0.8   
## 6 South… Nort… 9.11 0.49 2.68 79.2 6.78 0.51 0.7   
## 7 South… Nort… 9.22 0.66 2.64 79.9 6.18 0.49 0.56  
## 8 South… Nort… 11 0.61 2.35 77.3 7.34 0.39 0.64  
## 9 South… Nort… 10.8 0.6 2.39 77.4 7.09 0.46 0.83  
## 10 South… Nort… 10.4 0.55 2.13 79.4 6.33 0.26 0.52  
## # … with 562 more rows, and 1 more variable: eicosenoic <dbl>

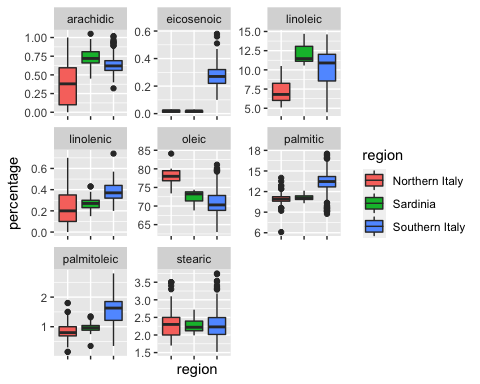
table(olive$region)

##   
## Northern Italy Sardinia Southern Italy   
## 151 98 323

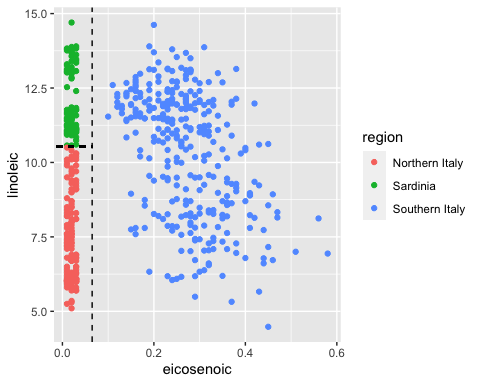
olive <- dplyr::select(olive, -area)  
  
# Predict region using KNN  
fit <- train(region ~ ., method = "knn",   
 tuneGrid = data.frame(k = seq(1, 15, 2)),   
 data = olive)  
ggplot(fit)



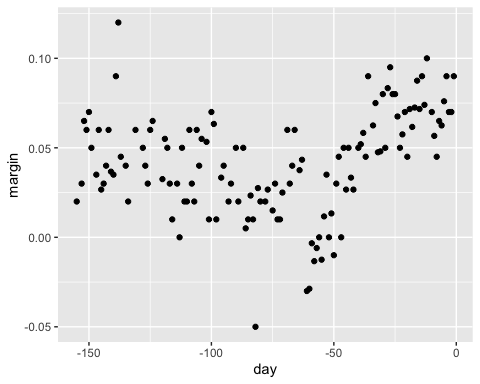
# Plot distribution of each predictor stratified by region  
olive %>% gather(fatty\_acid, percentage, -region) %>%  
 ggplot(aes(region, percentage, fill = region)) +  
 geom\_boxplot() +  
 facet\_wrap(~fatty\_acid, scales = "free") +  
 theme(axis.text.x = element\_blank())



# plot values for eicosenoic and linoleic  
p <- olive %>%   
 ggplot(aes(eicosenoic, linoleic, color = region)) +   
 geom\_point()  
p + geom\_vline(xintercept = 0.065, lty = 2) +   
 geom\_segment(x = -0.2, y = 10.54, xend = 0.065, yend = 10.54, color = "black", lty = 2)



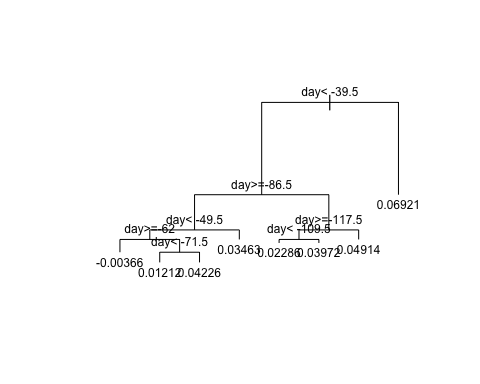
# load data for regression tree  
data("polls\_2008")  
qplot(day, margin, data = polls\_2008)



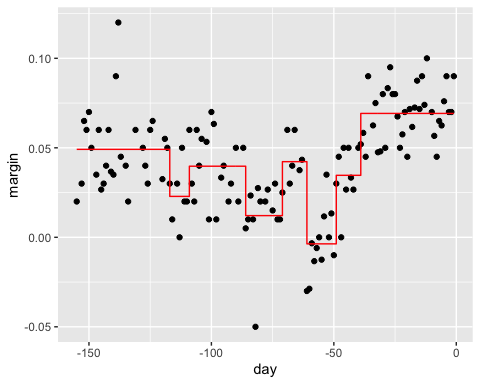
if(!require(rpart)) install.packages("rpart")

## Loading required package: rpart

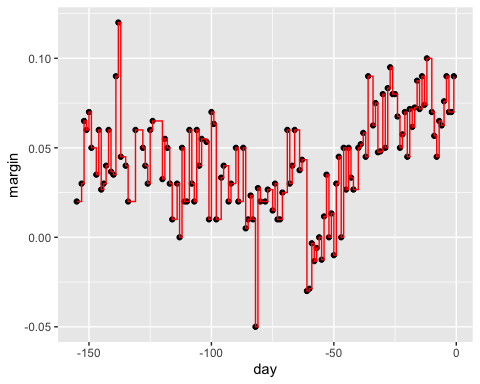
library(rpart)  
fit <- rpart(margin ~ ., data = polls\_2008)  
  
# visualize the splits   
plot(fit, margin = 0.1)  
text(fit, cex = 0.75)



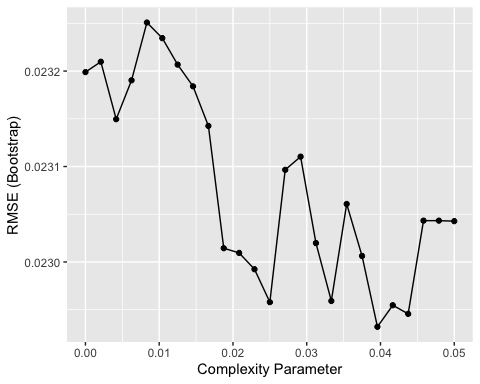
polls\_2008 %>%   
 mutate(y\_hat = predict(fit)) %>%   
 ggplot() +  
 geom\_point(aes(day, margin)) +  
 geom\_step(aes(day, y\_hat), col="red")



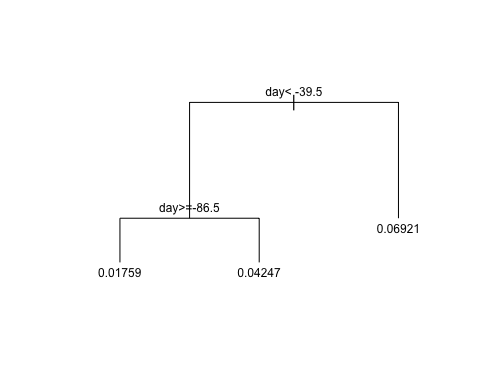
# change parameters  
fit <- rpart(margin ~ ., data = polls\_2008, control = rpart.control(cp = 0, minsplit = 2))  
polls\_2008 %>%   
 mutate(y\_hat = predict(fit)) %>%   
 ggplot() +  
 geom\_point(aes(day, margin)) +  
 geom\_step(aes(day, y\_hat), col="red")



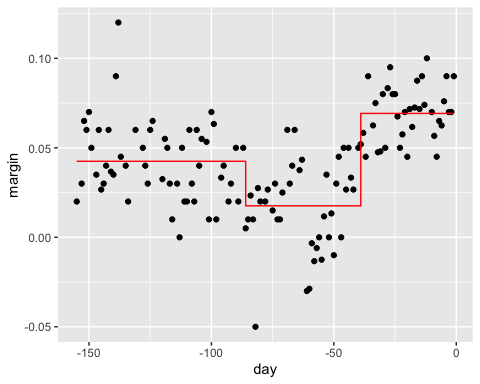
# use cross validation to choose cp  
train\_rpart <- train(margin ~ ., method = "rpart", tuneGrid = data.frame(cp = seq(0, 0.05, len = 25)), data = polls\_2008)  
ggplot(train\_rpart)



# access the final model and plot it  
plot(train\_rpart$finalModel, margin = 0.1)  
text(train\_rpart$finalModel, cex = 0.75)



polls\_2008 %>%   
 mutate(y\_hat = predict(train\_rpart)) %>%   
 ggplot() +  
 geom\_point(aes(day, margin)) +  
 geom\_step(aes(day, y\_hat), col="red")



# prune the tree   
pruned\_fit <- prune(fit, cp = 0.01)

## Classification (Decision) Trees

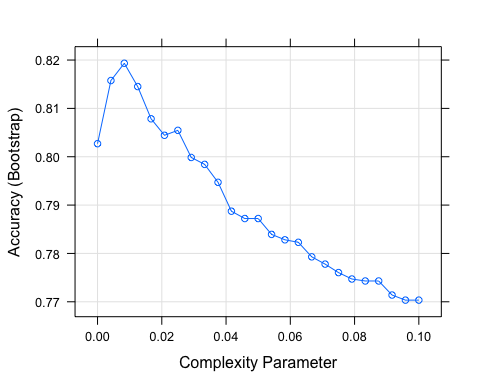
There is a link to the relevant section of the textbook: [Classification (decision) trees](https://rafalab.github.io/dsbook/examples-of-algorithms.html#classification-decision-trees)

**Key points**

* **Classification trees**, or decision trees, are used in prediction problems where the **outcome is categorical**.
* Decision trees form predictions by calculating **which class is the most common** among the training set observations within the partition, rather than taking the average in each partition.
* Two of the more popular metrics to choose the partitions are the **Gini index** and **entropy**.
* Pros: Classification trees are highly interpretable and easy to visualize.They can model human decision processes and don’t require use of dummy predictors for categorical variables.
* Cons: The approach via recursive partitioning can easily over-train and is therefore a bit harder to train than. Furthermore, in terms of accuracy, it is rarely the best performing method since it is not very flexible and is highly unstable to changes in training data.

*Code*

# fit a classification tree and plot it  
train\_rpart <- train(y ~ .,  
 method = "rpart",  
 tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 25)),  
 data = mnist\_27$train)  
plot(train\_rpart)



# compute accuracy  
confusionMatrix(predict(train\_rpart, mnist\_27$test), mnist\_27$test$y)$overall["Accuracy"]

## Accuracy   
## 0.82

## Random Forests

There is a link to the relevant section of the textbook: [Random forests](https://rafalab.github.io/dsbook/examples-of-algorithms.html#random-forests)

**Key points**

* **Random forests** are a very popular machine learning approach that addresses the shortcomings of decision trees. The goal is to improve prediction performance and reduce instability by **averaging multiple decision trees** (a forest of trees constructed with randomness).
* The general idea of random forests is to generate many predictors, each using regression or classification trees, and then **forming a final prediction based on the average prediction of all these trees**. To assure that the individual trees are not the same, we use the **bootstrap to induce randomness**.
* A **disadvantage** of random forests is that we **lose interpretability**.
* An approach that helps with interpretability is to examine **variable importance**. To define variable importance we **count how often a predictor is used in the individual trees**. The **caret** package includes the function varImp that extracts variable importance from any model in which the calculation is implemented.

*Code*

if(!require(randomForest)) install.packages("randomForest")

## Loading required package: randomForest

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:gridExtra':  
##   
## combine

## The following object is masked from 'package:dplyr':  
##   
## combine

## The following object is masked from 'package:ggplot2':  
##   
## margin

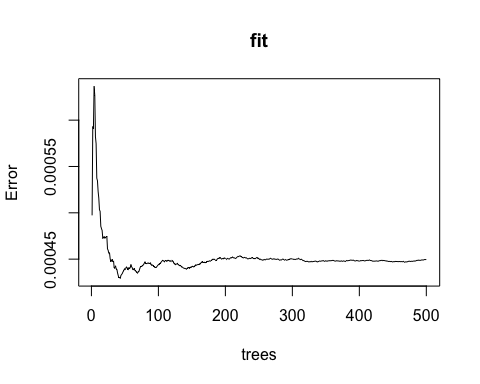
if(!require(Rborist)) install.packages("Rborist")

## Loading required package: Rborist

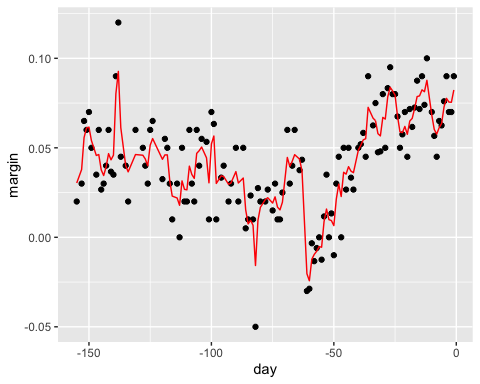
## Rborist 0.2-3

## Type RboristNews() to see new features/changes/bug fixes.

library(randomForest)  
fit <- randomForest(margin~., data = polls\_2008)   
plot(fit)



polls\_2008 %>%  
 mutate(y\_hat = predict(fit, newdata = polls\_2008)) %>%   
 ggplot() +  
 geom\_point(aes(day, margin)) +  
 geom\_line(aes(day, y\_hat), col="red")



train\_rf <- randomForest(y ~ ., data=mnist\_27$train)  
confusionMatrix(predict(train\_rf, mnist\_27$test), mnist\_27$test$y)$overall["Accuracy"]

## Accuracy   
## 0.785

# use cross validation to choose parameter  
train\_rf\_2 <- train(y ~ .,  
 method = "Rborist",  
 tuneGrid = data.frame(predFixed = 2, minNode = c(3, 50)),  
 data = mnist\_27$train)  
confusionMatrix(predict(train\_rf\_2, mnist\_27$test), mnist\_27$test$y)$overall["Accuracy"]

## Accuracy   
## 0.8

## Comprehension Check - Trees and Random Forests

1. Create a simple dataset where the outcome grows 0.75 units on average for every increase in a predictor, using this code:

n <- 1000  
sigma <- 0.25  
# set.seed(1) # if using R 3.5 or ealier  
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

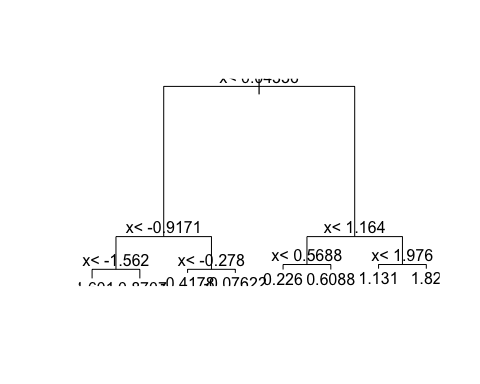
x <- rnorm(n, 0, 1)  
y <- 0.75 \* x + rnorm(n, 0, sigma)  
dat <- data.frame(x = x, y = y)

Which code correctly uses rpart() to fit a regression tree and saves the result to fit?

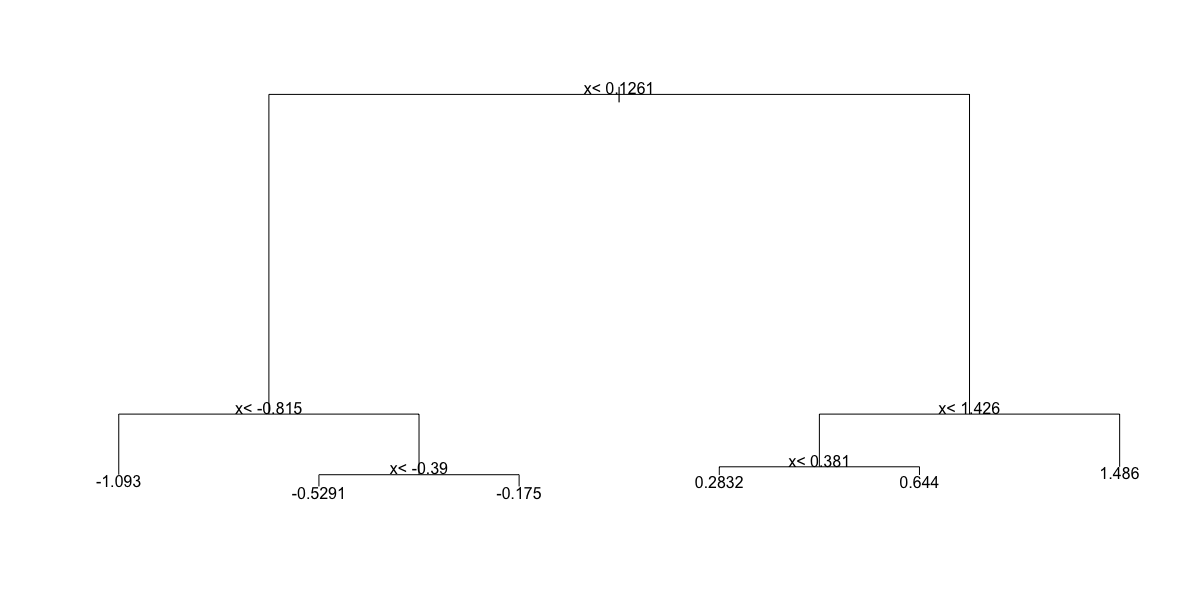
* ☐ A. fit <- rpart(y ~ .)
* ☐ B. fit <- rpart(y, ., data = dat)
* ☐ C. fit <- rpart(x ~ ., data = dat)
* ☒ D. fit <- rpart(y ~ ., data = dat)

1. Which of the following plots has the same tree shape obtained in Q1?

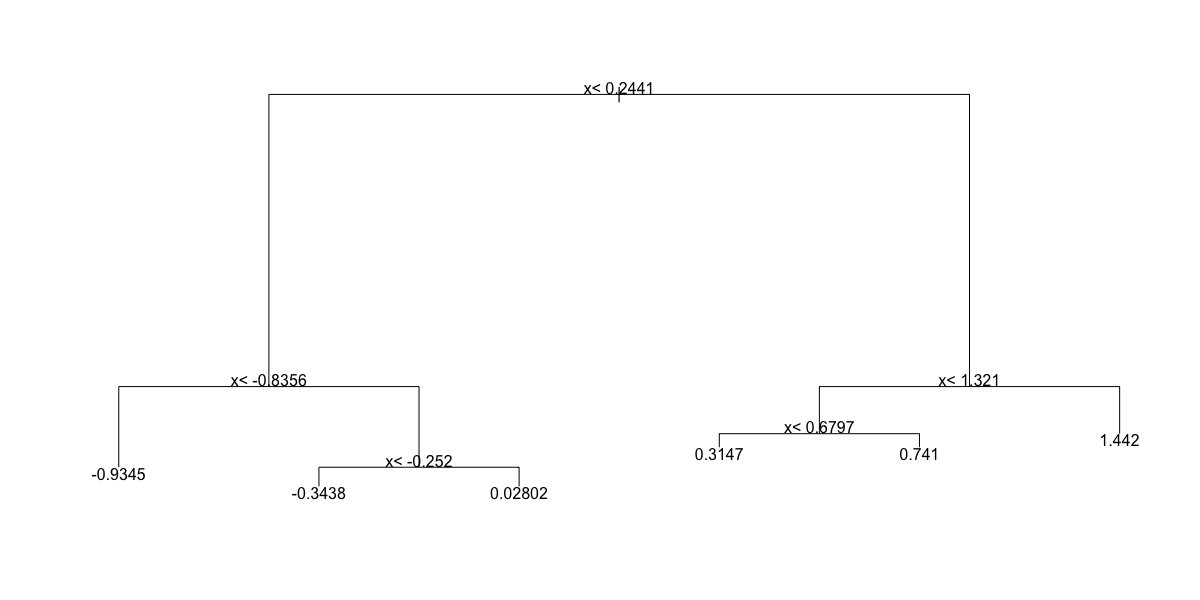
fit <- rpart(y ~ ., data = dat)  
plot(fit)  
text(fit)



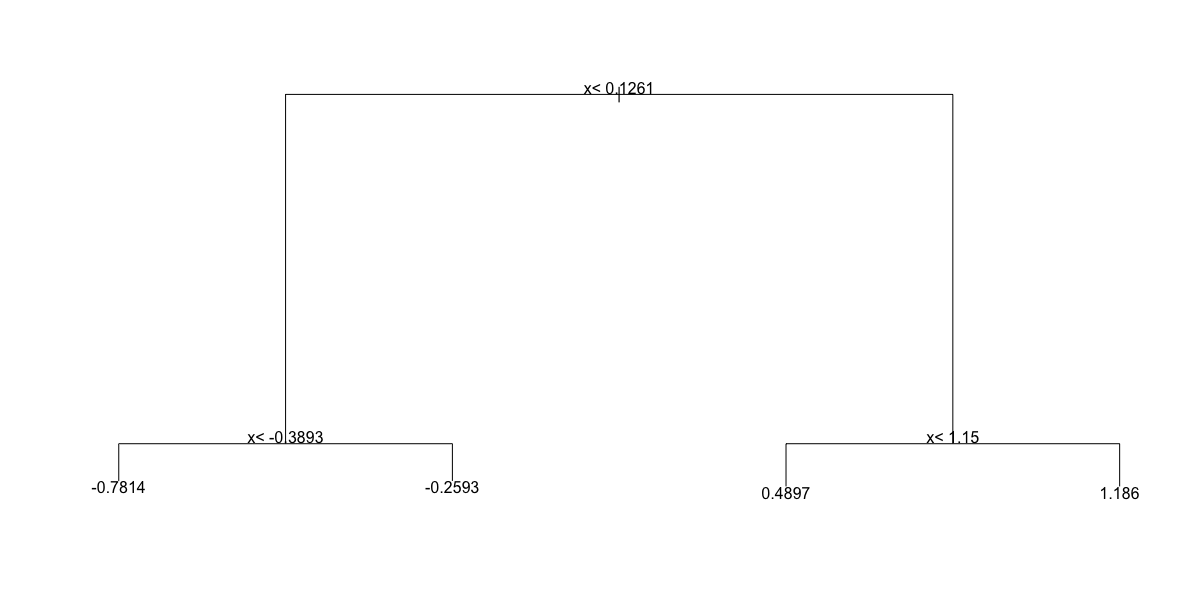
* ☐ A.



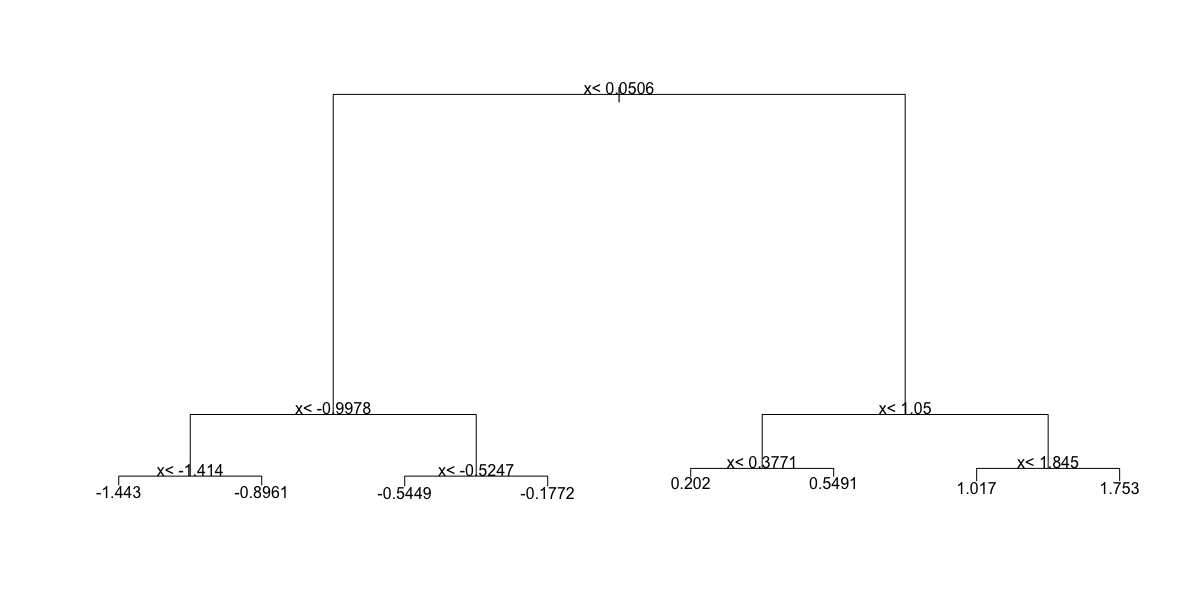
* ☐ B.



* ☐ C.



* ☒ D.

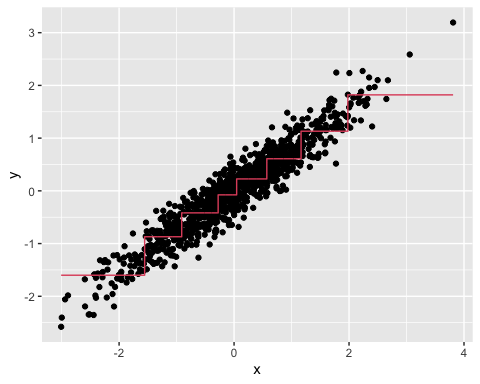


1. Below is most of the code to make a scatter plot of y versus x along with the predicted values based on the fit.

dat %>%   
mutate(y\_hat = predict(fit)) %>%   
ggplot() +  
geom\_point(aes(x, y)) +  
#BLANK

Which line of code should be used to replace #BLANK in the code above?

dat %>%   
mutate(y\_hat = predict(fit)) %>%   
ggplot() +  
geom\_point(aes(x, y)) +  
geom\_step(aes(x, y\_hat), col=2)



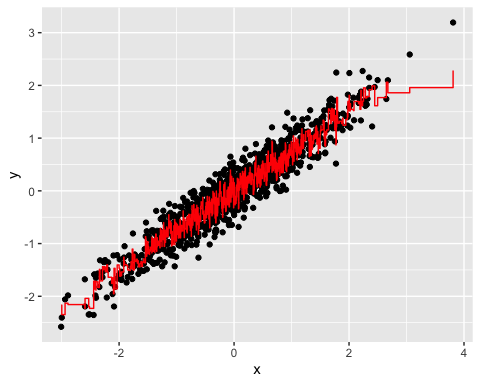
* ☒ A. geom\_step(aes(x, y\_hat), col=2)
* ☐ B. geom\_smooth(aes(y\_hat, x), col=2)
* ☐ C. geom\_quantile(aes(x, y\_hat), col=2)
* ☐ D. geom\_step(aes(y\_hat, x), col=2)

1. Now run Random Forests instead of a regression tree using randomForest() from the **randomForest** package, and remake the scatterplot with the prediction line. Part of the code is provided for you below.

library(randomForest)  
fit <- #BLANK   
dat %>%   
 mutate(y\_hat = predict(fit)) %>%   
 ggplot() +  
 geom\_point(aes(x, y)) +  
 geom\_step(aes(x, y\_hat), col = "red")

What code should replace #BLANK in the provided code?

library(randomForest)  
fit <- randomForest(y ~ x, data = dat)   
dat %>%   
 mutate(y\_hat = predict(fit)) %>%   
 ggplot() +  
 geom\_point(aes(x, y)) +  
 geom\_step(aes(x, y\_hat), col = "red")

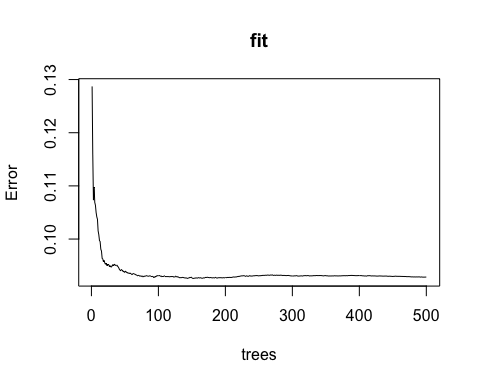


* ☒ A. randomForest(y ~ x, data = dat)
* ☐ B. randomForest(x ~ y, data = dat)
* ☐ C. randomForest(y ~ x, data = data)
* ☐ D. randomForest(x ~ y)

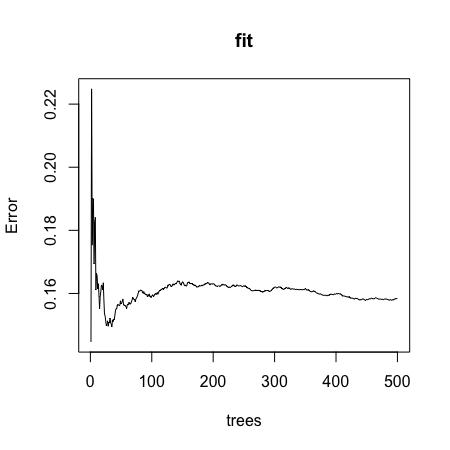
1. Use the plot() function to see if the Random Forest from Q4 has converged or if we need more trees.

Which of these graphs is most similar to the one produced by plotting the random forest? Note that there may be slight differences due to the seed not being set.

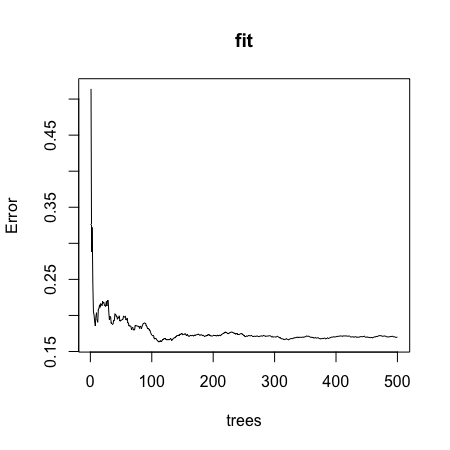
plot(fit)



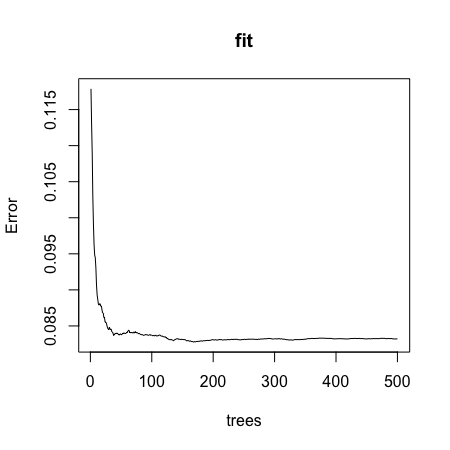
* ☐ A.



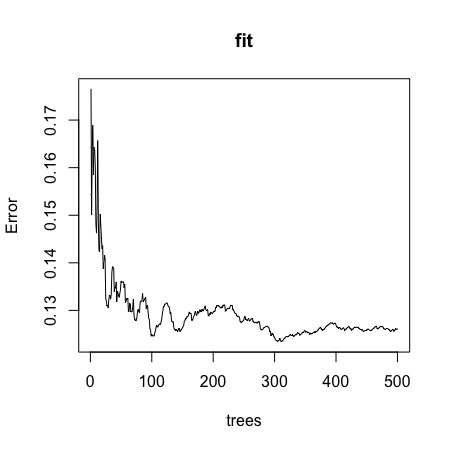
* ☐ B.



* ☒ C.



* ☐ D.



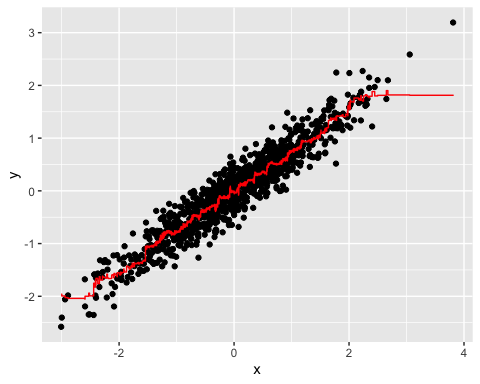
1. It seems that the default values for the Random Forest result in an estimate that is too flexible (unsmooth). Re-run the Random Forest but this time with a node size of 50 and a maximum of 25 nodes. Remake the plot.

Part of the code is provided for you below.

library(randomForest)  
fit <- #BLANK  
dat %>%   
 mutate(y\_hat = predict(fit)) %>%   
 ggplot() +  
 geom\_point(aes(x, y)) +  
 geom\_step(aes(x, y\_hat), col = "red")

What code should replace #BLANK in the provided code?

library(randomForest)  
fit <- randomForest(y ~ x, data = dat, nodesize = 50, maxnodes = 25)  
dat %>%   
 mutate(y\_hat = predict(fit)) %>%   
 ggplot() +  
 geom\_point(aes(x, y)) +  
 geom\_step(aes(x, y\_hat), col = "red")



* ☐ A. randomForest(y ~ x, data = dat, nodesize = 25, maxnodes = 25)
* ☐ B. randomForest(y ~ x, data = dat, nodes = 50, max = 25)
* ☐ C. randomForest(x ~ y, data = dat, nodes = 50, max = 25)
* ☒ D. randomForest(y ~ x, data = dat, nodesize = 50, maxnodes = 25)
* ☐ E. randomForest(x ~ y, data = dat, nodesize = 50, maxnodes = 25)

## Caret Package

There is a link to the relevant section of the textbook: [The caret package](https://rafalab.github.io/dsbook/caret.html)

**Caret package links**

<http://topepo.github.io/caret/available-models.html>

<http://topepo.github.io/caret/train-models-by-tag.html>

**Key points**

* The **caret** package helps provides a uniform interface and standardized syntax for the many different machine learning packages in R. Note that **caret** does not automatically install the packages needed.

*Code*

data("mnist\_27")  
  
train\_glm <- train(y ~ ., method = "glm", data = mnist\_27$train)  
train\_knn <- train(y ~ ., method = "knn", data = mnist\_27$train)  
  
y\_hat\_glm <- predict(train\_glm, mnist\_27$test, type = "raw")  
y\_hat\_knn <- predict(train\_knn, mnist\_27$test, type = "raw")  
  
confusionMatrix(y\_hat\_glm, mnist\_27$test$y)$overall[["Accuracy"]]

## [1] 0.75

confusionMatrix(y\_hat\_knn, mnist\_27$test$y)$overall[["Accuracy"]]

## [1] 0.84

## Tuning Parameters with Caret

There is a link to the relevant section of the textbook: [Cross validation](https://rafalab.github.io/dsbook/caret.html#caret-cv)

**Key points**

* The train() function automatically uses cross-validation to decide among a few default values of a tuning parameter.
* The getModelInfo() and modelLookup() functions can be used to learn more about a model and the parameters that can be optimized.
* We can use the tunegrid() parameter in the train() function to select a grid of values to be compared.
* The trControl parameter and trainControl() function can be used to change the way cross-validation is performed.
* Note that **not all parameters in machine learning algorithms are tuned**. We use the train() function to only optimize parameters that are tunable.

*Code*

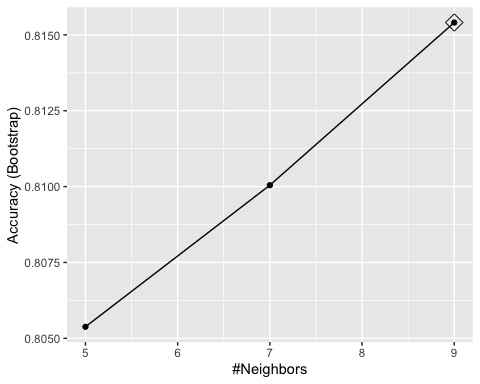
getModelInfo("knn")

## $kknn  
## $kknn$label  
## [1] "k-Nearest Neighbors"  
##   
## $kknn$library  
## [1] "kknn"  
##   
## $kknn$loop  
## NULL  
##   
## $kknn$type  
## [1] "Regression" "Classification"  
##   
## $kknn$parameters  
## parameter class label  
## 1 kmax numeric Max. #Neighbors  
## 2 distance numeric Distance  
## 3 kernel character Kernel  
##   
## $kknn$grid  
## function(x, y, len = NULL, search = "grid") {  
## if(search == "grid") {  
## out <- data.frame(kmax = (5:((2 \* len)+4))[(5:((2 \* len)+4))%%2 > 0],   
## distance = 2,   
## kernel = "optimal")  
## } else {  
## by\_val <- if(is.factor(y)) length(levels(y)) else 1  
## kerns <- c("rectangular", "triangular", "epanechnikov", "biweight", "triweight",   
## "cos", "inv", "gaussian")  
## out <- data.frame(kmax = sample(seq(1, floor(nrow(x)/3), by = by\_val), size = len, replace = TRUE),  
## distance = runif(len, min = 0, max = 3),  
## kernel = sample(kerns, size = len, replace = TRUE))  
## }  
## out  
## }  
##   
## $kknn$fit  
## function(x, y, wts, param, lev, last, classProbs, ...) {  
## dat <- if(is.data.frame(x)) x else as.data.frame(x, stringsAsFactors = TRUE)  
## dat$.outcome <- y  
## kknn::train.kknn(.outcome ~ ., data = dat,  
## kmax = param$kmax,  
## distance = param$distance,  
## kernel = as.character(param$kernel), ...)  
## }  
##   
## $kknn$predict  
## function(modelFit, newdata, submodels = NULL) {  
## if(!is.data.frame(newdata)) newdata <- as.data.frame(newdata, stringsAsFactors = TRUE)  
## predict(modelFit, newdata)  
## }  
##   
## $kknn$levels  
## function(x) x$obsLevels  
##   
## $kknn$tags  
## [1] "Prototype Models"  
##   
## $kknn$prob  
## function(modelFit, newdata, submodels = NULL) {  
## if(!is.data.frame(newdata)) newdata <- as.data.frame(newdata, stringsAsFactors = TRUE)  
## predict(modelFit, newdata, type = "prob")  
## }  
##   
## $kknn$sort  
## function(x) x[order(-x[,1]),]  
##   
##   
## $knn  
## $knn$label  
## [1] "k-Nearest Neighbors"  
##   
## $knn$library  
## NULL  
##   
## $knn$loop  
## NULL  
##   
## $knn$type  
## [1] "Classification" "Regression"   
##   
## $knn$parameters  
## parameter class label  
## 1 k numeric #Neighbors  
##   
## $knn$grid  
## function(x, y, len = NULL, search = "grid"){  
## if(search == "grid") {  
## out <- data.frame(k = (5:((2 \* len)+4))[(5:((2 \* len)+4))%%2 > 0])  
## } else {  
## by\_val <- if(is.factor(y)) length(levels(y)) else 1  
## out <- data.frame(k = sample(seq(1, floor(nrow(x)/3), by = by\_val), size = len, replace = TRUE))  
## }  
## }  
##   
## $knn$fit  
## function(x, y, wts, param, lev, last, classProbs, ...) {  
## if(is.factor(y))  
## {  
## knn3(as.matrix(x), y, k = param$k, ...)  
## } else {  
## knnreg(as.matrix(x), y, k = param$k, ...)  
## }  
## }  
##   
## $knn$predict  
## function(modelFit, newdata, submodels = NULL) {  
## if(modelFit$problemType == "Classification")  
## {  
## out <- predict(modelFit, newdata, type = "class")  
## } else {  
## out <- predict(modelFit, newdata)  
## }  
## out  
## }  
##   
## $knn$predictors  
## function(x, ...) colnames(x$learn$X)  
##   
## $knn$tags  
## [1] "Prototype Models"  
##   
## $knn$prob  
## function(modelFit, newdata, submodels = NULL)  
## predict(modelFit, newdata, type = "prob")  
##   
## $knn$levels  
## function(x) levels(x$learn$y)  
##   
## $knn$sort  
## function(x) x[order(-x[,1]),]

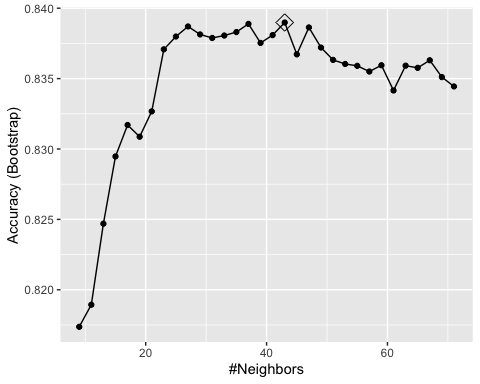
modelLookup("knn")

## model parameter label forReg forClass probModel  
## 1 knn k #Neighbors TRUE TRUE TRUE

train\_knn <- train(y ~ ., method = "knn", data = mnist\_27$train)  
ggplot(train\_knn, highlight = TRUE)



train\_knn <- train(y ~ ., method = "knn",   
 data = mnist\_27$train,  
 tuneGrid = data.frame(k = seq(9, 71, 2)))  
ggplot(train\_knn, highlight = TRUE)



train\_knn$bestTune

## k  
## 18 43

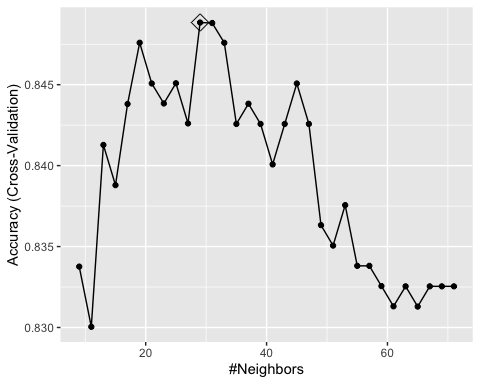
train\_knn$finalModel

## 43-nearest neighbor model  
## Training set outcome distribution:  
##   
## 2 7   
## 379 421

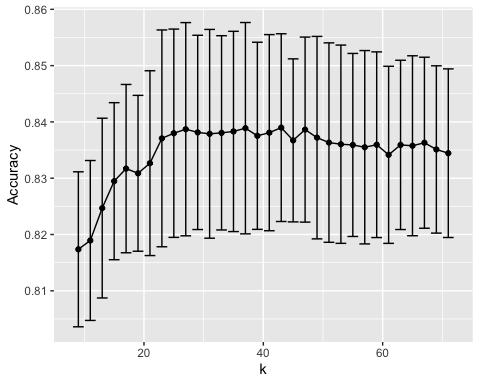
confusionMatrix(predict(train\_knn, mnist\_27$test, type = "raw"),  
 mnist\_27$test$y)$overall["Accuracy"]

## Accuracy   
## 0.855

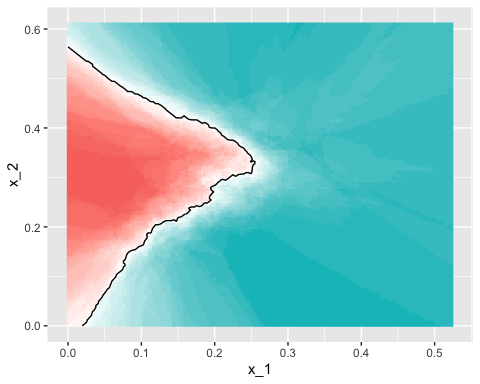
control <- trainControl(method = "cv", number = 10, p = .9)  
train\_knn\_cv <- train(y ~ ., method = "knn",   
 data = mnist\_27$train,  
 tuneGrid = data.frame(k = seq(9, 71, 2)),  
 trControl = control)  
ggplot(train\_knn\_cv, highlight = TRUE)



train\_knn$results %>%   
 ggplot(aes(x = k, y = Accuracy)) +  
 geom\_line() +  
 geom\_point() +  
 geom\_errorbar(aes(x = k,   
 ymin = Accuracy - AccuracySD,  
 ymax = Accuracy + AccuracySD))



plot\_cond\_prob <- function(p\_hat=NULL){  
 tmp <- mnist\_27$true\_p  
 if(!is.null(p\_hat)){  
 tmp <- mutate(tmp, p=p\_hat)  
 }  
 tmp %>% ggplot(aes(x\_1, x\_2, z=p, fill=p)) +  
 geom\_raster(show.legend = FALSE) +  
 scale\_fill\_gradientn(colors=c("#F8766D","white","#00BFC4")) +  
 stat\_contour(breaks=c(0.5),color="black")  
}  
  
plot\_cond\_prob(predict(train\_knn, mnist\_27$true\_p, type = "prob")[,2])



if(!require(gam)) install.packages("gam")

## Loading required package: gam

## Loading required package: splines

## Loading required package: foreach

##   
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':  
##   
## accumulate, when

## Loaded gam 1.20

modelLookup("gamLoess")

## model parameter label forReg forClass probModel  
## 1 gamLoess span Span TRUE TRUE TRUE  
## 2 gamLoess degree Degree TRUE TRUE TRUE

grid <- expand.grid(span = seq(0.15, 0.65, len = 10), degree = 1)  
  
train\_loess <- train(y ~ .,   
 method = "gamLoess",  
 tuneGrid=grid,  
 data = mnist\_27$train)  
ggplot(train\_loess, highlight = TRUE)  
  
confusionMatrix(data = predict(train\_loess, mnist\_27$test),   
 reference = mnist\_27$test$y)$overall["Accuracy"]  
  
p1 <- plot\_cond\_prob(predict(train\_loess, mnist\_27$true\_p, type = "prob")[,2])  
p1

## Comprehension Check - Caret Package

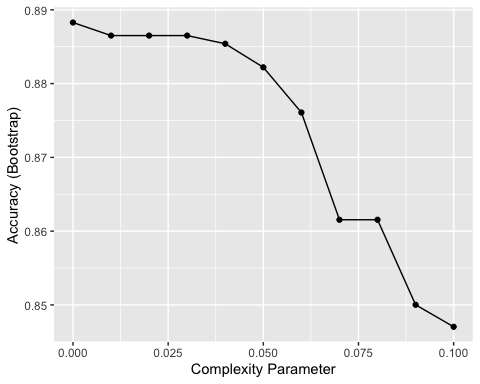
1. Load the **rpart** package and then use the caret::train() function with method = "rpart" to fit a classification tree to the tissue\_gene\_expression dataset. Try out cp values of seq(0, 0.1, 0.01). Plot the accuracies to report the results of the best model. Set the seed to 1991.

Which value of cp gives the highest accuracy? 0

set.seed(1991, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1991, sample.kind = "Rounding"): non-uniform 'Rounding'  
## sampler used

data("tissue\_gene\_expression")  
   
fit <- with(tissue\_gene\_expression,   
 train(x, y, method = "rpart",  
 tuneGrid = data.frame(cp = seq(0, 0.1, 0.01))))  
   
ggplot(fit)



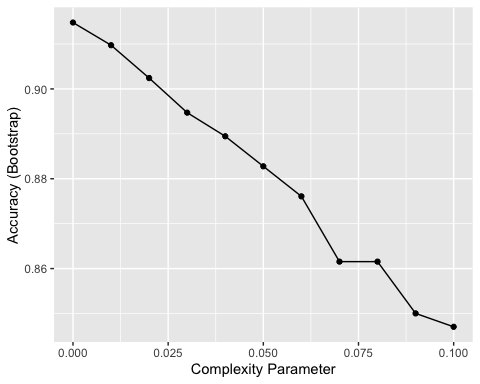
1. Note that there are only 6 placentas in the dataset. By default, rpart requires 20 observations before splitting a node. That means that it is difficult to have a node in which placentas are the majority. Rerun the analysis you did in Q1 with caret::train(), but this time with method = "rpart" and allow it to split any node by using the argument control = rpart.control(minsplit = 0). Look at the confusion matrix again to determine whether the accuracy increases. Again, set the seed to 1991.

What is the accuracy now?

# set.seed(1991) # if using R 3.5 or earlier  
set.seed(1991, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1991, sample.kind = "Rounding"): non-uniform 'Rounding'  
## sampler used

fit\_rpart <- with(tissue\_gene\_expression,   
 train(x, y, method = "rpart",  
 tuneGrid = data.frame(cp = seq(0, 0.10, 0.01)),  
 control = rpart.control(minsplit = 0)))  
ggplot(fit\_rpart)



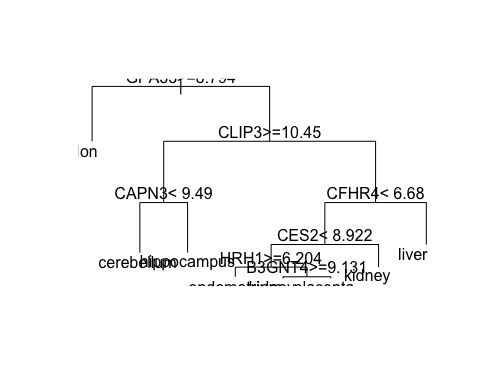
confusionMatrix(fit\_rpart)

## Bootstrapped (25 reps) Confusion Matrix   
##   
## (entries are percentual average cell counts across resamples)  
##   
## Reference  
## Prediction cerebellum colon endometrium hippocampus kidney liver placenta  
## cerebellum 19.5 0.0 0.2 0.9 0.4 0.0 0.1  
## colon 0.3 16.5 0.1 0.0 0.1 0.0 0.1  
## endometrium 0.1 0.2 6.4 0.1 0.9 0.1 0.5  
## hippocampus 0.2 0.0 0.0 15.6 0.1 0.0 0.0  
## kidney 0.3 0.3 0.9 0.1 19.1 0.5 0.3  
## liver 0.0 0.0 0.3 0.0 0.3 12.6 0.2  
## placenta 0.1 0.1 0.5 0.0 0.6 0.1 1.8  
##   
## Accuracy (average) : 0.9141

1. Plot the tree from the best fitting model of the analysis you ran in Q2.

Which gene is at the first split?

plot(fit\_rpart$finalModel)  
text(fit\_rpart$finalModel)



* ☐ A. B3GNT4
* ☐ B. CAPN3
* ☐ C. CES2
* ☐ D. CFHR4
* ☐ E. CLIP3
* ☒ F. GPA33
* ☐ G. HRH1

1. We can see that with just seven genes, we are able to predict the tissue type. Now let’s see if we can predict the tissue type with even fewer genes using a Random Forest. Use the train() function and the rf method to train a Random Forest model and save it to an object called fit. Try out values of mtry ranging from seq(50, 200, 25) (you can also explore other values on your own). What mtry value maximizes accuracy? To permit small nodesize to grow as we did with the classification trees, use the following argument: nodesize = 1.

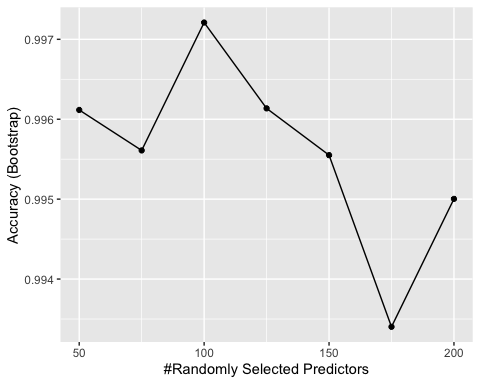
Note: This exercise will take some time to run. If you want to test out your code first, try using smaller values with ntree. Set the seed to 1991 again.

What value of mtry maximizes accuracy? 100

# set.seed(1991) # if using R 3.5 or earlier  
set.seed(1991, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1991, sample.kind = "Rounding"): non-uniform 'Rounding'  
## sampler used

library(randomForest)  
fit <- with(tissue\_gene\_expression,   
 train(x, y, method = "rf",   
 nodesize = 1,  
 tuneGrid = data.frame(mtry = seq(50, 200, 25))))  
   
ggplot(fit)



1. Use the function varImp() on the output of train() and save it to an object called imp:

imp <- #BLANK  
imp

What should replace #BLANK in the code above?

imp <- varImp(fit)  
imp

## rf variable importance  
##   
## only 20 most important variables shown (out of 500)  
##   
## Overall  
## GPA33 100.00  
## BIN1 64.65  
## GPM6B 62.35  
## KIF2C 62.15  
## CLIP3 52.09  
## COLGALT2 46.48  
## CFHR4 35.03  
## SHANK2 34.90  
## TFR2 33.61  
## GALNT11 30.70  
## CEP55 30.49  
## TCN2 27.96  
## CAPN3 27.52  
## CYP4F11 25.74  
## GTF2IRD1 24.89  
## KCTD2 24.34  
## FCN3 22.68  
## SUSD6 22.24  
## DOCK4 22.02  
## RARRES2 21.53

1. The rpart() model we ran above in Q2 produced a tree that used just seven predictors. Extracting the predictor names is not straightforward, but can be done. If the output of the call to train was fit\_rpart, we can extract the names like this: 1/1 point (graded)

tree\_terms <- as.character(unique(fit\_rpart$finalModel$frame$var[!(fit\_rpart$finalModel$frame$var == "<leaf>")]))  
tree\_terms

## [1] "GPA33" "CLIP3" "CAPN3" "CFHR4" "CES2" "HRH1" "B3GNT4"

Calculate the variable importance in the Random Forest call from Q4 for these seven predictors and examine where they rank.

What is the importance of the CFHR4 gene in the Random Forest call? 35.0

What is the rank of the CFHR4 gene in the Random Forest call? 7

data\_frame(term = rownames(imp$importance),   
 importance = imp$importance$Overall) %>%  
mutate(rank = rank(-importance)) %>% arrange(desc(importance)) %>%  
filter(term %in% tree\_terms)

## # A tibble: 7 x 3  
## term importance rank  
## <chr> <dbl> <dbl>  
## 1 GPA33 100 1  
## 2 CLIP3 52.1 5  
## 3 CFHR4 35.0 7  
## 4 CAPN3 27.5 13  
## 5 CES2 20.0 22  
## 6 HRH1 2.35 97  
## 7 B3GNT4 0.136 343

## Titanic Exercises - Part 1

These exercises cover everything you have learned in this course so far. You will use the background information to provided to train a number of different types of models on this dataset.

**Background**

The Titanic was a British ocean liner that struck an iceberg and sunk on its maiden voyage in 1912 from the United Kingdom to New York. More than 1,500 of the estimated 2,224 passengers and crew died in the accident, making this one of the largest maritime disasters ever outside of war. The ship carried a wide range of passengers of all ages and both genders, from luxury travelers in first-class to immigrants in the lower classes. However, not all passengers were equally likely to survive the accident. You will use real data about a selection of 891 passengers to predict which passengers survived.

if(!require(titanic)) install.packages("titanic")

## Loading required package: titanic

library(titanic) # loads titanic\_train data frame  
  
# 3 significant digits  
options(digits = 3)  
  
# clean the data - `titanic\_train` is loaded with the titanic package  
titanic\_clean <- titanic\_train %>%  
 mutate(Survived = factor(Survived),  
 Embarked = factor(Embarked),  
 Age = ifelse(is.na(Age), median(Age, na.rm = TRUE), Age), # NA age to median age  
 FamilySize = SibSp + Parch + 1) %>% # count family members  
 dplyr::select(Survived, Sex, Pclass, Age, Fare, SibSp, Parch, FamilySize, Embarked)

1. Training and test sets

Split titanic\_clean into test and training sets - after running the setup code, it should have 891 rows and 9 variables.

Set the seed to 42, then use the **caret** package to create a 20% data partition based on the Survived column. Assign the 20% partition to test\_set and the remaining 80% partition to train\_set.

How many observations are in the training set?

#set.seed(42) # if using R 3.5 or earlier  
set.seed(42, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(42, sample.kind = "Rounding"): non-uniform 'Rounding'  
## sampler used

test\_index <- createDataPartition(titanic\_clean$Survived, times = 1, p = 0.2, list = FALSE) # create a 20% test set  
test\_set <- titanic\_clean[test\_index,]  
train\_set <- titanic\_clean[-test\_index,]  
  
nrow(train\_set)

## [1] 712

How many observations are in the test set?

nrow(test\_set)

## [1] 179

What proportion of individuals in the training set survived?

mean(train\_set$Survived == 1)

## [1] 0.383

1. Baseline prediction by guessing the outcome

The simplest prediction method is randomly guessing the outcome without using additional predictors. These methods will help us determine whether our machine learning algorithm performs better than chance. How accurate are two methods of guessing Titanic passenger survival?

Set the seed to 3. For each individual in the test set, randomly guess whether that person survived or not by sampling from the vector c(0,1) (Note: use the default argument setting of prob from the sample function).

What is the accuracy of this guessing method?

#set.seed(3)  
set.seed(3, sample.kind = "Rounding")

## Warning in set.seed(3, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

# guess with equal probability of survival  
guess <- sample(c(0,1), nrow(test\_set), replace = TRUE)  
mean(guess == test\_set$Survived)

## [1] 0.475

3a. Predicting survival by sex

Use the training set to determine whether members of a given sex were more likely to survive or die. Apply this insight to generate survival predictions on the test set.

What proportion of training set females survived?

train\_set %>%  
 group\_by(Sex) %>%  
 summarize(Survived = mean(Survived == 1)) %>%  
 filter(Sex == "female") %>%  
 pull(Survived)

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

## [1] 0.731

What proportion of training set males survived?

train\_set %>%  
 group\_by(Sex) %>%  
 summarize(Survived = mean(Survived == 1)) %>%  
 filter(Sex == "male") %>%  
 pull(Survived)

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

## [1] 0.197

3b. Predicting survival by sex

Predict survival using sex on the test set: if the survival rate for a sex is over 0.5, predict survival for all individuals of that sex, and predict death if the survival rate for a sex is under 0.5.

What is the accuracy of this sex-based prediction method on the test set?

sex\_model <- ifelse(test\_set$Sex == "female", 1, 0) # predict Survived=1 if female, 0 if male  
mean(sex\_model == test\_set$Survived) # calculate accuracy

## [1] 0.821

4a. Predicting survival by passenger class

In the training set, which class(es) (Pclass) were passengers more likely to survive than die?

train\_set %>%  
 group\_by(Pclass) %>%  
 summarize(Survived = mean(Survived == 1))

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

## # A tibble: 3 x 2  
## Pclass Survived  
## <int> <dbl>  
## 1 1 0.619  
## 2 2 0.5   
## 3 3 0.242

Select ALL that apply.

* ☒ A. 1
* ☐ B. 2
* ☐ C. 3

4b. Predicting survival by passenger class

Predict survival using passenger class on the test set: predict survival if the survival rate for a class is over 0.5, otherwise predict death.

What is the accuracy of this class-based prediction method on the test set?

class\_model <- ifelse(test\_set$Pclass == 1, 1, 0) # predict survival only if first class  
mean(class\_model == test\_set$Survived) # calculate accuracy

## [1] 0.704

4c. Predicting survival by passenger class

Use the training set to group passengers by both sex and passenger class.

Which sex and class combinations were more likely to survive than die?

train\_set %>%  
 group\_by(Sex, Pclass) %>%  
 summarize(Survived = mean(Survived == 1)) %>%  
 filter(Survived > 0.5)

## `summarise()` regrouping output by 'Sex' (override with `.groups` argument)

## # A tibble: 2 x 3  
## # Groups: Sex [1]  
## Sex Pclass Survived  
## <chr> <int> <dbl>  
## 1 female 1 0.957  
## 2 female 2 0.919

Select ALL that apply.

* ☒ A. female 1st class
* ☒ B. female 2nd class
* ☐ C. female 3rd class
* ☐ D. male 1st class
* ☐ E. male 2nd class
* ☐ F. male 3rd class

4d. Predicting survival by passenger class

Predict survival using both sex and passenger class on the test set. Predict survival if the survival rate for a sex/class combination is over 0.5, otherwise predict death.

What is the accuracy of this sex- and class-based prediction method on the test set?

sex\_class\_model <- ifelse(test\_set$Sex == "female" & test\_set$Pclass != 3, 1, 0)  
mean(sex\_class\_model == test\_set$Survived)

## [1] 0.821

5a. Confusion matrix

Use the confusionMatrix() function to create confusion matrices for the sex model, class model, and combined sex and class model. You will need to convert predictions and survival status to factors to use this function.

confusionMatrix(data = factor(sex\_model), reference = factor(test\_set$Survived))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 96 18  
## 1 14 51  
##   
## Accuracy : 0.821   
## 95% CI : (0.757, 0.874)  
## No Information Rate : 0.615   
## P-Value [Acc > NIR] : 1.72e-09   
##   
## Kappa : 0.619   
##   
## Mcnemar's Test P-Value : 0.596   
##   
## Sensitivity : 0.873   
## Specificity : 0.739   
## Pos Pred Value : 0.842   
## Neg Pred Value : 0.785   
## Prevalence : 0.615   
## Detection Rate : 0.536   
## Detection Prevalence : 0.637   
## Balanced Accuracy : 0.806   
##   
## 'Positive' Class : 0   
##

confusionMatrix(data = factor(class\_model), reference = factor(test\_set$Survived))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 94 37  
## 1 16 32  
##   
## Accuracy : 0.704   
## 95% CI : (0.631, 0.77)  
## No Information Rate : 0.615   
## P-Value [Acc > NIR] : 0.00788   
##   
## Kappa : 0.337   
##   
## Mcnemar's Test P-Value : 0.00601   
##   
## Sensitivity : 0.855   
## Specificity : 0.464   
## Pos Pred Value : 0.718   
## Neg Pred Value : 0.667   
## Prevalence : 0.615   
## Detection Rate : 0.525   
## Detection Prevalence : 0.732   
## Balanced Accuracy : 0.659   
##   
## 'Positive' Class : 0   
##

confusionMatrix(data = factor(sex\_class\_model), reference = factor(test\_set$Survived))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 109 31  
## 1 1 38  
##   
## Accuracy : 0.821   
## 95% CI : (0.757, 0.874)  
## No Information Rate : 0.615   
## P-Value [Acc > NIR] : 1.72e-09   
##   
## Kappa : 0.589   
##   
## Mcnemar's Test P-Value : 2.95e-07   
##   
## Sensitivity : 0.991   
## Specificity : 0.551   
## Pos Pred Value : 0.779   
## Neg Pred Value : 0.974   
## Prevalence : 0.615   
## Detection Rate : 0.609   
## Detection Prevalence : 0.782   
## Balanced Accuracy : 0.771   
##   
## 'Positive' Class : 0   
##

What is the “positive” class used to calculate confusion matrix metrics?

* ☒ A. 0
* ☐ B. 1

Which model has the highest sensitivity?

* ☐ A. sex only
* ☐ B. class only
* ☒ C. sex and class combined

Which model has the highest specificity?

* ☒ A. sex only
* ☐ B. class only
* ☐ C. sex and class combined

Which model has the highest balanced accuracy?

* ☒ A. sex only
* ☐ B. class only
* ☐ C. sex and class combined

5b. Confusion matrix

What is the maximum value of balanced accuracy? 0.806

1. F1 scores

Use the F\_meas() function to calculate scores for the sex model, class model, and combined sex and class model. You will need to convert predictions to factors to use this function.

Which model has the highest score?

F\_meas(data = factor(sex\_model), reference = test\_set$Survived)

## [1] 0.857

F\_meas(data = factor(class\_model), reference = test\_set$Survived)

## [1] 0.78

F\_meas(data = factor(sex\_class\_model), reference = test\_set$Survived)

## [1] 0.872

* ☐ A. sex only
* ☐ B. class only
* ☒ C. sex and class combined

What is the maximum value of the score? 0.872

## Titanic Exercises - Part 2

1. Survival by fare - LDA and QDA

Set the seed to 1. Train a model using linear discriminant analysis (LDA) with the **caret** lda method using fare as the only predictor.

What is the accuracy on the test set for the LDA model?

#set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

train\_lda <- train(Survived ~ Fare, method = "lda", data = train\_set)  
lda\_preds <- predict(train\_lda, test\_set)  
mean(lda\_preds == test\_set$Survived)

## [1] 0.693

Set the seed to 1. Train a model using quadratic discriminant analysis (QDA) with the **caret** qda method using fare as the only predictor.

What is the accuracy on the test set for the QDA model?

#set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

train\_qda <- train(Survived ~ Fare, method = "qda", data = train\_set)  
qda\_preds <- predict(train\_qda, test\_set)  
mean(qda\_preds == test\_set$Survived)

## [1] 0.693

Note: when training models for Titanic Exercises Part 2, please use the S3 method for class formula rather than the default S3 method of **caret** train() (see ?caret::train for details).

1. Logistic regression models

Set the seed to 1. Train a logistic regression model with the **caret** glm method using age as the only predictor.

What is the accuracy of your model (using age as the only predictor) on the test set ?

#set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

train\_glm\_age <- train(Survived ~ Age, method = "glm", data = train\_set)  
glm\_preds\_age <- predict(train\_glm\_age, test\_set)  
mean(glm\_preds\_age == test\_set$Survived)

## [1] 0.615

Set the seed to 1. Train a logistic regression model with the **caret** glm method using four predictors: sex, class, fare, and age.

What is the accuracy of your model (using these four predictors) on the test set?

#set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

train\_glm <- train(Survived ~ Sex + Pclass + Fare + Age, method = "glm", data = train\_set)  
glm\_preds <- predict(train\_glm, test\_set)  
mean(glm\_preds == test\_set$Survived)

## [1] 0.849

Set the seed to 1. Train a logistic regression model with the **caret** glm method using all predictors. Ignore warnings about rank-deficient fit.

What is the accuracy of your model (using all predictors) on the test set?

#set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

train\_glm\_all <- train(Survived ~ ., method = "glm", data = train\_set)

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading  
  
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

glm\_all\_preds <- predict(train\_glm\_all, test\_set)

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

mean(glm\_all\_preds == test\_set$Survived)

## [1] 0.849

9a. kNN model

Set the seed to 6. Train a kNN model on the training set using the **caret** train function. Try tuning with k = seq(3, 51, 2).

What is the optimal value of the number of neighbors k?

#set.seed(6)  
set.seed(6, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(6, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

train\_knn <- train(Survived ~ .,  
 method = "knn",  
 data = train\_set,  
 tuneGrid = data.frame(k = seq(3, 51, 2)))  
train\_knn$bestTune

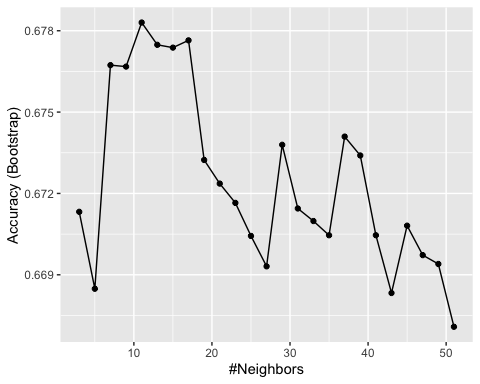
## k  
## 5 11

9b. kNN model

Plot the kNN model to investigate the relationship between the number of neighbors and accuracy on the training set.

Of these values of , which yields the highest accuracy?

ggplot(train\_knn)



* ☐ A. 7
* ☒ B. 11
* ☐ C. 17
* ☐ D. 21

9c. kNN model

What is the accuracy of the kNN model on the test set?

knn\_preds <- predict(train\_knn, test\_set)  
mean(knn\_preds == test\_set$Survived)

## [1] 0.709

1. Cross-validation

Set the seed to 8 and train a new kNN model. Instead of the default training control, use 10-fold cross-validation where each partition consists of 10% of the total. Try tuning with k = seq(3, 51, 2).

What is the optimal value of k using cross-validation?

#set.seed(8)  
set.seed(8, sample.kind = "Rounding") # simulate R 3.5

## Warning in set.seed(8, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

train\_knn\_cv <- train(Survived ~ .,  
 method = "knn",  
 data = train\_set,  
 tuneGrid = data.frame(k = seq(3, 51, 2)),  
 trControl = trainControl(method = "cv", number = 10, p = 0.9))  
train\_knn\_cv$bestTune

## k  
## 2 5

What is the accuracy on the test set using the cross-validated kNN model?

knn\_cv\_preds <- predict(train\_knn\_cv, test\_set)  
mean(knn\_cv\_preds == test\_set$Survived)

## [1] 0.648

11a. Classification tree model

Set the seed to 10. Use **caret** to train a decision tree with the rpart method. Tune the complexity parameter with cp = seq(0, 0.05, 0.002).

What is the optimal value of the complexity parameter (cp)?

#set.seed(10)  
set.seed(10, sample.kind = "Rounding") # simulate R 3.5

## Warning in set.seed(10, sample.kind = "Rounding"): non-uniform 'Rounding'  
## sampler used

train\_rpart <- train(Survived ~ .,   
 method = "rpart",  
 tuneGrid = data.frame(cp = seq(0, 0.05, 0.002)),  
 data = train\_set)  
train\_rpart$bestTune

## cp  
## 9 0.016

What is the accuracy of the decision tree model on the test set?

rpart\_preds <- predict(train\_rpart, test\_set)  
mean(rpart\_preds == test\_set$Survived)

## [1] 0.838

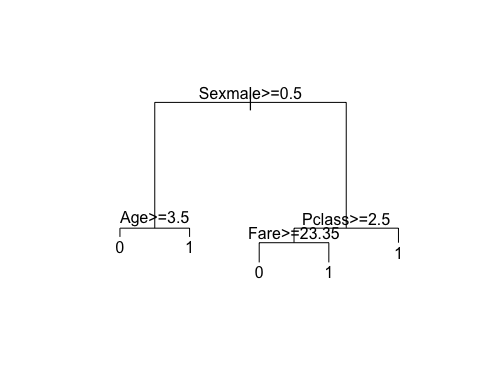
11b. Classification tree model

Inspect the final model and plot the decision tree.

train\_rpart$finalModel # inspect final model

## n= 712   
##   
## node), split, n, loss, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 712 273 0 (0.6166 0.3834)   
## 2) Sexmale>=0.5 463 91 0 (0.8035 0.1965)   
## 4) Age>=3.5 449 80 0 (0.8218 0.1782) \*  
## 5) Age< 3.5 14 3 1 (0.2143 0.7857) \*  
## 3) Sexmale< 0.5 249 67 1 (0.2691 0.7309)   
## 6) Pclass>=2.5 118 59 0 (0.5000 0.5000)   
## 12) Fare>=23.4 24 3 0 (0.8750 0.1250) \*  
## 13) Fare< 23.4 94 38 1 (0.4043 0.5957) \*  
## 7) Pclass< 2.5 131 8 1 (0.0611 0.9389) \*

# make plot of decision tree  
plot(train\_rpart$finalModel, margin = 0.1)  
text(train\_rpart$finalModel)



Which variables are used in the decision tree?

Select ALL that apply.

* ☐ A. Survived
* ☒ B. Sex
* ☒ C. Pclass
* ☒ D. Age
* ☒ E. Fare
* ☐ F. Parch
* ☐ G. Embarked

11c. Classification tree model

Using the decision rules generated by the final model, predict whether the following individuals would survive.

* A 28-year-old male would NOT survive
* A female in the second passenger class would survive
* A third-class female who paid a fare of $8 would survive
* A 5-year-old male with 4 siblings would NOT survive
* A third-class female who paid a fare of $25 would NOT survive
* A first-class 17-year-old female with 2 siblings would survive
* A first-class 17-year-old male with 2 siblings would NOT survive

1. Random forest model

Set the seed to 14. Use the **caret** train() function with the rf method to train a random forest. Test values of mtry = seq(1:7). Set ntree to 100.

What mtry value maximizes accuracy?

#set.seed(14)  
set.seed(14, sample.kind = "Rounding") # simulate R 3.5

## Warning in set.seed(14, sample.kind = "Rounding"): non-uniform 'Rounding'  
## sampler used

train\_rf <- train(Survived ~ .,  
 data = train\_set,  
 method = "rf",  
 ntree = 100,  
 tuneGrid = data.frame(mtry = seq(1:7)))  
train\_rf$bestTune

## mtry  
## 2 2

What is the accuracy of the random forest model on the test set?

rf\_preds <- predict(train\_rf, test\_set)  
mean(rf\_preds == test\_set$Survived)

## [1] 0.844

Use varImp() on the random forest model object to determine the importance of various predictors to the random forest model.

What is the most important variable?

varImp(train\_rf) # first row

## rf variable importance  
##   
## Overall  
## Sexmale 100.000  
## Fare 65.091  
## Age 45.533  
## Pclass 32.529  
## FamilySize 18.275  
## SibSp 7.881  
## Parch 7.150  
## EmbarkedS 2.839  
## EmbarkedQ 0.122  
## EmbarkedC 0.000

# Section 6 - Model Fitting and Recommendation Systems Overview

In the **Model Fitting and Recommendation Systems** section, you will learn how to apply the machine learning algorithms you have learned.

After completing this section, you will be able to:

* Apply the methods we have learned to an example, the **MNIST digits**.
* Build a **movie recommendation system** using machine learning.
* Penalize large estimates that come from small sample sizes using **regularization**.

This section has three parts: **case study: MNIST, recommendation systems**, and **regularization**.

## Case Study: MNIST

There is a link to the relevant section of the textbook: [Machine learning in practice](https://rafalab.github.io/dsbook/machine-learning-in-practice.html)

**Key points**

* We will apply what we have learned in the course on the Modified National Institute of Standards and Technology database (MNIST) digits, a popular dataset used in machine learning competitions.

*Code*

mnist <- read\_mnist()  
  
names(mnist)

## [1] "train" "test"

dim(mnist$train$images)

## [1] 60000 784

class(mnist$train$labels)

## [1] "integer"

table(mnist$train$labels)

##   
## 0 1 2 3 4 5 6 7 8 9   
## 5923 6742 5958 6131 5842 5421 5918 6265 5851 5949

# sample 10k rows from training set, 1k rows from test set  
set.seed(123)  
index <- sample(nrow(mnist$train$images), 10000)  
x <- mnist$train$images[index,]  
y <- factor(mnist$train$labels[index])  
  
index <- sample(nrow(mnist$test$images), 1000)  
#note that the line above is the corrected code - code in video at 0:52 is incorrect  
x\_test <- mnist$test$images[index,]  
y\_test <- factor(mnist$test$labels[index])

## Preprocessing MNIST Data

There is a link to the relevant section of the textbook: [Preprocessing](https://rafalab.github.io/dsbook/machine-learning-in-practice.html#preprocessing)

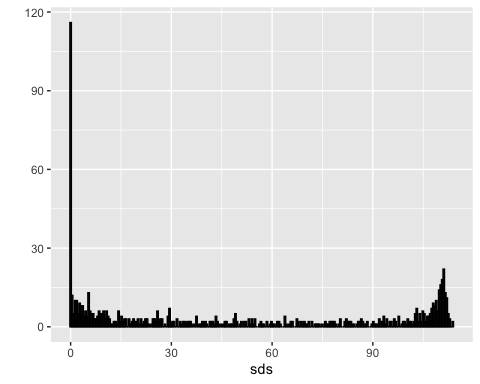
**Key points**

* Common \*\*preprocessing steps include:

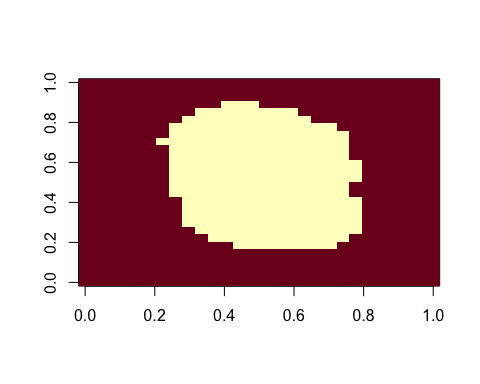
1. standardizing or transforming predictors and
2. removing predictors that are not useful, are highly correlated with others, have very few non-unique values, or have close to zero variation.

*Code*

sds <- colSds(x)  
qplot(sds, bins = 256, color = I("black"))



nzv <- nearZeroVar(x)  
image(matrix(1:784 %in% nzv, 28, 28))



col\_index <- setdiff(1:ncol(x), nzv)  
length(col\_index)

## [1] 252

## Model Fitting for MNIST Data

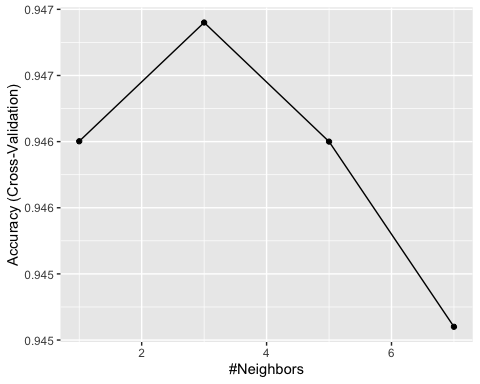
There is a link to the relevant section of the textbook: [k-nearest neighbor and random forest](https://rafalab.github.io/dsbook/machine-learning-in-practice.html#k-nearest-neighbor-and-random-forest)

**Key points**

* The **caret** package requires that we **add column names** to the feature matrices.
* In general, it is a good idea to **test out a small subset of the data** first to get an idea of how long your code will take to run.

*Code*

colnames(x) <- 1:ncol(mnist$train$images)  
colnames(x\_test) <- colnames(x)  
  
control <- trainControl(method = "cv", number = 10, p = .9)  
train\_knn <- train(x[,col\_index], y,  
 method = "knn",   
 tuneGrid = data.frame(k = c(1,3,5,7)),  
 trControl = control)  
ggplot(train\_knn)



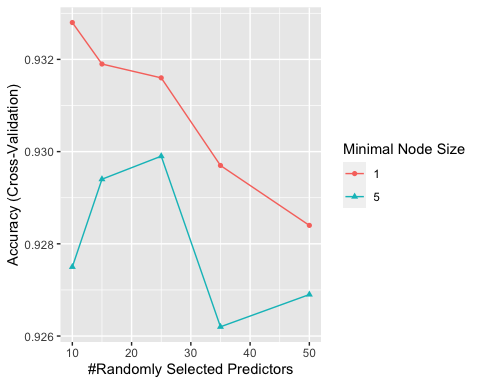
n <- 1000  
b <- 2  
index <- sample(nrow(x), n)  
control <- trainControl(method = "cv", number = b, p = .9)  
train\_knn <- train(x[index ,col\_index], y[index],  
 method = "knn",  
 tuneGrid = data.frame(k = c(3,5,7)),  
 trControl = control)  
fit\_knn <- knn3(x[ ,col\_index], y, k = 3)  
  
y\_hat\_knn <- predict(fit\_knn,  
 x\_test[, col\_index],  
 type="class")  
cm <- confusionMatrix(y\_hat\_knn, factor(y\_test))  
cm$overall["Accuracy"]

## Accuracy   
## 0.955

cm$byClass[,1:2]

## Sensitivity Specificity  
## Class: 0 1.000 0.998  
## Class: 1 1.000 0.992  
## Class: 2 0.953 0.999  
## Class: 3 0.917 0.993  
## Class: 4 0.936 0.996  
## Class: 5 0.971 0.991  
## Class: 6 0.990 0.998  
## Class: 7 0.945 0.994  
## Class: 8 0.846 0.998  
## Class: 9 0.971 0.991

control <- trainControl(method="cv", number = 5, p = 0.8)  
grid <- expand.grid(minNode = c(1,5) , predFixed = c(10, 15, 25, 35, 50))  
train\_rf <- train(x[, col\_index], y,  
 method = "Rborist",  
 nTree = 50,  
 trControl = control,  
 tuneGrid = grid,  
 nSamp = 5000)  
ggplot(train\_rf)



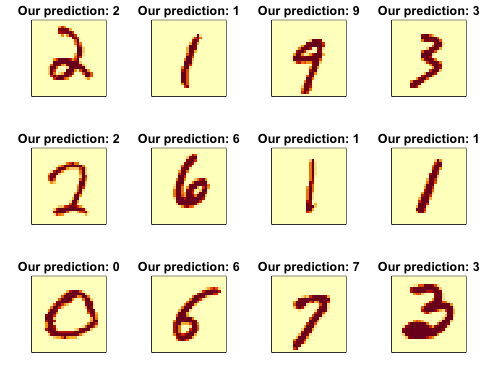
train\_rf$bestTune

## predFixed minNode  
## 1 10 1

fit\_rf <- Rborist(x[, col\_index], y,  
 nTree = 1000,  
 minNode = train\_rf$bestTune$minNode,  
 predFixed = train\_rf$bestTune$predFixed)  
  
y\_hat\_rf <- factor(levels(y)[predict(fit\_rf, x\_test[ ,col\_index])$yPred])  
cm <- confusionMatrix(y\_hat\_rf, y\_test)  
cm$overall["Accuracy"]

## Accuracy   
## 0.959

rafalib::mypar(3,4)  
for(i in 1:12){  
 image(matrix(x\_test[i,], 28, 28)[, 28:1],   
 main = paste("Our prediction:", y\_hat\_rf[i]),  
 xaxt="n", yaxt="n")  
}



## Variable Importance

There is a link to the relevant sections of the textbook: [Variable importance](https://rafalab.github.io/dsbook/machine-learning-in-practice.html#variable-importance) and [Visual assessments](https://rafalab.github.io/dsbook/machine-learning-in-practice.html#visual-assessments)

**Key points**

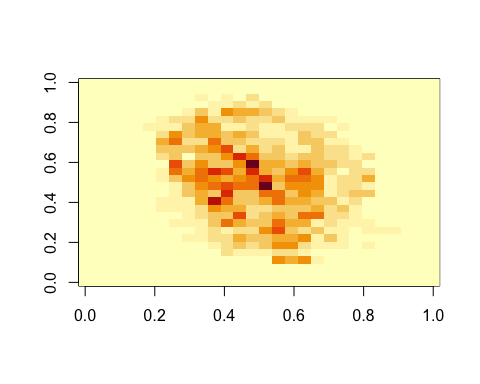
* The **Rborist** package does not currently support variable importance calculations, but the **randomForest** package does.
* An important part of data science is visualizing results to determine why we are failing.

*Code*

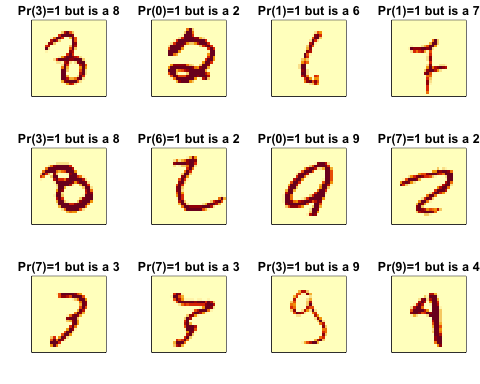
x <- mnist$train$images[index,]  
y <- factor(mnist$train$labels[index])  
rf <- randomForest(x, y, ntree = 50)  
imp <- importance(rf)  
imp

## MeanDecreaseGini  
## 1 0.0000  
## 2 0.0000  
## 3 0.0000  
## 4 0.0000  
## 5 0.0000  
## 6 0.0000  
## 7 0.0000  
## 8 0.0000  
## 9 0.0000  
## 10 0.0000  
## 11 0.0000  
## 12 0.0000  
## 13 0.0000  
## 14 0.0000  
## 15 0.0000  
## 16 0.0000  
## 17 0.0000  
## 18 0.0000  
## 19 0.0000  
## 20 0.0000  
## 21 0.0000  
## 22 0.0000  
## 23 0.0000  
## 24 0.0000  
## 25 0.0000  
## 26 0.0000  
## 27 0.0000  
## 28 0.0000  
## 29 0.0000  
## 30 0.0000  
## 31 0.0000  
## 32 0.0000  
## 33 0.0000  
## 34 0.0000  
## 35 0.0000  
## 36 0.0000  
## 37 0.0000  
## 38 0.0000  
## 39 0.0000  
## 40 0.0000  
## 41 0.0000  
## 42 0.0000  
## 43 0.0000  
## 44 0.0000  
## 45 0.0000  
## 46 0.0000  
## 47 0.0000  
## 48 0.0000  
## 49 0.0000  
## 50 0.0000  
## 51 0.0000  
## 52 0.0000  
## 53 0.0000  
## 54 0.0000  
## 55 0.0000  
## 56 0.0000  
## 57 0.0000  
## 58 0.0000  
## 59 0.0000  
## 60 0.0000  
## 61 0.0000  
## 62 0.0000  
## 63 0.0000  
## 64 0.0000  
## 65 0.0000  
## 66 0.0000  
## 67 0.0000  
## 68 0.0000  
## 69 0.0000  
## 70 0.0200  
## 71 0.0386  
## 72 0.3364  
## 73 0.4292  
## 74 0.1083  
## 75 0.1228  
## 76 0.0000  
## 77 0.0000  
## 78 0.0359  
## 79 0.0000  
## 80 0.0000  
## 81 0.0000  
## 82 0.0000  
## 83 0.0000  
## 84 0.0000  
## 85 0.0000  
## 86 0.0000  
## 87 0.0000  
## 88 0.0000  
## 89 0.0000  
## 90 0.0000  
## 91 0.0000  
## 92 0.0000  
## 93 0.0267  
## 94 0.0702  
## 95 0.0267  
## 96 0.1533  
## 97 0.5302  
## 98 0.1691  
## 99 0.1951  
## 100 4.3825  
## 101 3.7575  
## 102 4.0716  
## 103 1.4450  
## 104 0.5788  
## 105 0.0756  
## 106 0.0300  
## 107 0.0916  
## 108 0.0000  
## 109 0.0000  
## 110 0.0000  
## 111 0.0000  
## 112 0.0000  
## 113 0.0000  
## 114 0.0000  
## 115 0.0000  
## 116 0.0000  
## 117 0.0000  
## 118 0.0000  
## 119 0.0368  
## 120 0.0958  
## 121 0.0368  
## 122 0.4054  
## 123 0.1888  
## 124 1.6623  
## 125 1.0255  
## 126 0.9706  
## 127 0.9350  
## 128 1.8896  
## 129 2.3448  
## 130 0.9726  
## 131 0.7841  
## 132 0.3058  
## 133 0.2913  
## 134 0.0611  
## 135 0.4770  
## 136 0.0000  
## 137 0.0000  
## 138 0.0000  
## 139 0.0000  
## 140 0.0000  
## 141 0.0000  
## 142 0.0000  
## 143 0.0000  
## 144 0.0000  
## 145 0.0450  
## 146 0.4217  
## 147 0.1030  
## 148 0.4381  
## 149 0.2826  
## 150 0.6646  
## 151 1.4041  
## 152 2.1603  
## 153 3.1023  
## 154 1.7377  
## 155 2.9828  
## 156 4.4697  
## 157 4.6632  
## 158 1.9789  
## 159 1.1770  
## 160 1.2593  
## 161 1.1914  
## 162 0.4314  
## 163 0.9320  
## 164 0.5088  
## 165 0.0583  
## 166 0.0000  
## 167 0.0000  
## 168 0.0000  
## 169 0.0000  
## 170 0.0000  
## 171 0.0000  
## 172 0.0337  
## 173 0.0000  
## 174 0.0467  
## 175 0.0971  
## 176 0.2638  
## 177 0.8443  
## 178 1.3889  
## 179 2.3951  
## 180 1.8932  
## 181 3.7141  
## 182 3.1491  
## 183 2.5722  
## 184 3.5550  
## 185 3.7543  
## 186 4.1136  
## 187 1.2190  
## 188 2.7119  
## 189 1.3368  
## 190 0.7848  
## 191 0.5944  
## 192 0.6998  
## 193 0.0367  
## 194 0.0000  
## 195 0.0560  
## 196 0.0000  
## 197 0.0000  
## 198 0.0000  
## 199 0.0000  
## 200 0.0000  
## 201 0.0653  
## 202 0.1618  
## 203 0.2514  
## 204 0.1467  
## 205 0.7132  
## 206 1.0696  
## 207 1.8813  
## 208 1.5488  
## 209 1.6265  
## 210 2.3821  
## 211 4.1416  
## 212 6.0898  
## 213 2.8040  
## 214 1.9544  
## 215 2.9735  
## 216 1.1595  
## 217 1.2301  
## 218 0.7179  
## 219 0.8997  
## 220 1.4020  
## 221 0.8376  
## 222 0.0376  
## 223 0.0000  
## 224 0.0000  
## 225 0.0000  
## 226 0.0000  
## 227 0.0000  
## 228 0.0000  
## 229 0.1500  
## 230 0.1951  
## 231 0.6163  
## 232 1.3442  
## 233 0.8332  
## 234 1.1122  
## 235 3.0582  
## 236 4.9129  
## 237 3.2573  
## 238 2.7814  
## 239 2.9401  
## 240 5.4603  
## 241 3.9843  
## 242 3.9568  
## 243 1.1594  
## 244 1.9290  
## 245 1.5714  
## 246 1.1573  
## 247 0.9894  
## 248 0.7398  
## 249 0.2346  
## 250 0.5157  
## 251 0.0000  
## 252 0.0000  
## 253 0.0000  
## 254 0.0000  
## 255 0.0000  
## 256 0.0000  
## 257 0.0000  
## 258 0.0722  
## 259 0.6696  
## 260 0.3971  
## 261 1.1764  
## 262 2.2870  
## 263 2.6467  
## 264 3.0094  
## 265 5.8341  
## 266 2.1984  
## 267 3.1962  
## 268 3.5770  
## 269 2.7636  
## 270 5.0814  
## 271 4.8756  
## 272 2.4102  
## 273 2.2899  
## 274 1.2372  
## 275 0.3960  
## 276 0.7806  
## 277 0.2840  
## 278 0.0000  
## 279 0.0000  
## 280 0.0000  
## 281 0.0000  
## 282 0.0000  
## 283 0.0000  
## 284 0.0000  
## 285 0.1978  
## 286 0.0691  
## 287 0.8360  
## 288 0.8459  
## 289 0.9408  
## 290 2.0882  
## 291 4.3131  
## 292 3.5580  
## 293 3.2671  
## 294 1.9374  
## 295 1.9242  
## 296 2.6329  
## 297 3.0550  
## 298 2.8851  
## 299 3.3400  
## 300 2.2500  
## 301 2.8778  
## 302 1.3096  
## 303 0.5058  
## 304 0.1055  
## 305 0.1202  
## 306 0.0000  
## 307 0.0000  
## 308 0.0000  
## 309 0.0000  
## 310 0.0000  
## 311 0.0000  
## 312 0.0000  
## 313 0.0267  
## 314 0.1652  
## 315 1.0535  
## 316 0.9770  
## 317 1.1757  
## 318 3.9662  
## 319 7.4847  
## 320 5.0866  
## 321 3.2152  
## 322 2.9141  
## 323 3.5169  
## 324 4.8595  
## 325 3.6001  
## 326 3.6972  
## 327 2.4491  
## 328 3.2116  
## 329 1.3368  
## 330 2.0959  
## 331 0.6248  
## 332 0.1734  
## 333 0.1204  
## 334 0.0000  
## 335 0.0000  
## 336 0.0000  
## 337 0.0000  
## 338 0.0000  
## 339 0.0000  
## 340 0.0669  
## 341 0.0589  
## 342 0.0710  
## 343 0.7515  
## 344 1.5224  
## 345 2.9044  
## 346 3.4698  
## 347 2.9629  
## 348 6.6917  
## 349 2.8665  
## 350 2.5272  
## 351 5.2107  
## 352 5.2579  
## 353 2.5862  
## 354 4.0516  
## 355 3.9797  
## 356 1.2102  
## 357 1.9677  
## 358 2.8926  
## 359 2.4807  
## 360 0.2659  
## 361 0.0710  
## 362 0.0000  
## 363 0.0000  
## 364 0.0000  
## 365 0.0000  
## 366 0.0000  
## 367 0.0000  
## 368 0.0000  
## 369 0.0267  
## 370 0.1961  
## 371 0.6116  
## 372 0.9917  
## 373 2.6019  
## 374 4.5573  
## 375 5.0599  
## 376 6.0905  
## 377 5.3284  
## 378 5.1077  
## 379 9.6768  
## 380 3.0461  
## 381 4.7315  
## 382 4.3859  
## 383 4.5496  
## 384 1.2225  
## 385 2.1867  
## 386 1.7976  
## 387 1.3636  
## 388 0.2294  
## 389 0.0000  
## 390 0.0000  
## 391 0.0000  
## 392 0.0000  
## 393 0.0000  
## 394 0.0000  
## 395 0.0000  
## 396 0.0000  
## 397 0.2786  
## 398 0.3010  
## 399 1.2454  
## 400 3.1789  
## 401 4.4449  
## 402 5.5182  
## 403 4.3270  
## 404 4.0243  
## 405 4.0694  
## 406 5.5033  
## 407 6.6132  
## 408 3.8076  
## 409 5.1868  
## 410 5.2291  
## 411 4.3761  
## 412 1.2487  
## 413 1.6620  
## 414 1.7047  
## 415 3.3018  
## 416 0.3135  
## 417 0.0667  
## 418 0.0000  
## 419 0.0000  
## 420 0.0000  
## 421 0.0000  
## 422 0.0000  
## 423 0.0000  
## 424 0.0200  
## 425 0.1010  
## 426 0.3706  
## 427 0.8750  
## 428 5.2063  
## 429 3.6503  
## 430 5.5588  
## 431 6.5687  
## 432 6.3710  
## 433 3.7244  
## 434 6.4584  
## 435 3.8925  
## 436 3.1450  
## 437 4.6127  
## 438 5.8932  
## 439 3.6514  
## 440 1.8678  
## 441 0.7452  
## 442 2.3169  
## 443 1.7684  
## 444 0.3237  
## 445 0.0000  
## 446 0.0000  
## 447 0.0000  
## 448 0.0000  
## 449 0.0000  
## 450 0.0000  
## 451 0.0000  
## 452 0.0384  
## 453 0.0814  
## 454 0.5199  
## 455 0.5373  
## 456 5.9110  
## 457 2.8719  
## 458 4.4087  
## 459 2.8772  
## 460 2.8043  
## 461 4.5564  
## 462 9.2761  
## 463 3.5203  
## 464 3.9495  
## 465 3.0245  
## 466 3.5809  
## 467 2.6407  
## 468 2.9175  
## 469 1.9749  
## 470 2.2785  
## 471 0.5547  
## 472 0.2392  
## 473 0.1860  
## 474 0.0200  
## 475 0.0000  
## 476 0.0000  
## 477 0.0000  
## 478 0.0000  
## 479 0.0000  
## 480 0.0383  
## 481 0.0387  
## 482 0.4292  
## 483 1.6728  
## 484 2.5022  
## 485 0.4138  
## 486 2.9169  
## 487 3.0419  
## 488 4.1365  
## 489 7.1352  
## 490 4.9019  
## 491 2.8327  
## 492 2.5211  
## 493 1.7125  
## 494 2.7378  
## 495 2.8248  
## 496 2.0614  
## 497 2.3113  
## 498 0.9727  
## 499 1.6279  
## 500 0.5343  
## 501 0.3333  
## 502 0.0000  
## 503 0.0000  
## 504 0.0000  
## 505 0.0000  
## 506 0.0000  
## 507 0.0000  
## 508 0.0676  
## 509 0.2275  
## 510 0.2708  
## 511 2.4200  
## 512 2.5823  
## 513 3.0054  
## 514 3.4622  
## 515 4.5320  
## 516 6.1263  
## 517 2.3824  
## 518 3.3455  
## 519 1.9886  
## 520 2.9348  
## 521 1.1133  
## 522 1.4845  
## 523 3.0486  
## 524 1.7594  
## 525 2.0075  
## 526 1.0956  
## 527 0.7642  
## 528 0.5527  
## 529 0.0702  
## 530 0.0000  
## 531 0.0000  
## 532 0.0000  
## 533 0.0000  
## 534 0.0000  
## 535 0.0000  
## 536 0.0000  
## 537 0.1836  
## 538 0.8058  
## 539 3.7220  
## 540 5.5971  
## 541 1.8936  
## 542 2.1503  
## 543 5.3189  
## 544 3.1706  
## 545 2.5217  
## 546 2.2154  
## 547 1.6559  
## 548 2.3495  
## 549 0.9677  
## 550 2.5048  
## 551 2.7026  
## 552 1.4848  
## 553 1.0656  
## 554 0.5196  
## 555 0.4745  
## 556 0.5605  
## 557 0.1946  
## 558 0.0000  
## 559 0.0000  
## 560 0.0000  
## 561 0.0000  
## 562 0.0000  
## 563 0.0000  
## 564 0.0000  
## 565 0.0360  
## 566 0.7484  
## 567 2.0237  
## 568 4.3082  
## 569 3.1404  
## 570 4.0156  
## 571 3.2594  
## 572 3.2163  
## 573 3.2371  
## 574 2.6207  
## 575 1.3211  
## 576 1.4396  
## 577 1.4215  
## 578 2.6131  
## 579 2.1551  
## 580 1.6976  
## 581 0.4295  
## 582 0.7656  
## 583 0.1415  
## 584 0.1012  
## 585 0.0653  
## 586 0.1405  
## 587 0.0000  
## 588 0.0000  
## 589 0.0000  
## 590 0.0000  
## 591 0.0000  
## 592 0.0000  
## 593 0.3101  
## 594 0.8712  
## 595 1.2101  
## 596 1.5286  
## 597 3.0302  
## 598 3.8308  
## 599 3.8574  
## 600 1.4988  
## 601 1.4851  
## 602 2.2346  
## 603 1.6009  
## 604 1.5888  
## 605 1.7945  
## 606 1.9097  
## 607 1.8448  
## 608 0.7688  
## 609 1.4031  
## 610 0.4461  
## 611 0.1067  
## 612 0.2739  
## 613 0.0000  
## 614 0.0000  
## 615 0.0000  
## 616 0.0000  
## 617 0.0000  
## 618 0.0000  
## 619 0.0000  
## 620 0.0390  
## 621 0.1751  
## 622 0.1036  
## 623 1.4516  
## 624 2.0503  
## 625 1.8557  
## 626 4.5113  
## 627 2.0373  
## 628 1.6867  
## 629 2.8683  
## 630 2.0734  
## 631 1.8517  
## 632 2.4817  
## 633 1.4786  
## 634 1.3862  
## 635 1.1019  
## 636 1.0241  
## 637 0.4047  
## 638 0.3250  
## 639 0.0655  
## 640 0.0000  
## 641 0.0400  
## 642 0.0000  
## 643 0.0000  
## 644 0.0000  
## 645 0.0000  
## 646 0.0000  
## 647 0.0000  
## 648 0.0000  
## 649 0.0000  
## 650 0.0360  
## 651 0.5241  
## 652 0.7703  
## 653 1.3069  
## 654 2.9215  
## 655 1.3210  
## 656 4.7766  
## 657 3.5148  
## 658 3.5579  
## 659 2.7827  
## 660 2.0031  
## 661 1.1806  
## 662 0.6780  
## 663 0.4173  
## 664 0.5286  
## 665 0.0000  
## 666 0.0840  
## 667 0.1122  
## 668 0.1322  
## 669 0.0644  
## 670 0.0000  
## 671 0.0000  
## 672 0.0000  
## 673 0.0000  
## 674 0.0000  
## 675 0.0000  
## 676 0.0000  
## 677 0.0923  
## 678 0.1728  
## 679 0.2596  
## 680 0.2985  
## 681 0.2241  
## 682 0.5979  
## 683 1.1140  
## 684 1.2162  
## 685 1.9263  
## 686 0.9836  
## 687 1.6218  
## 688 0.6831  
## 689 0.4048  
## 690 0.4089  
## 691 0.4024  
## 692 0.0845  
## 693 0.1489  
## 694 0.0533  
## 695 0.0000  
## 696 0.0394  
## 697 0.0000  
## 698 0.0000  
## 699 0.0000  
## 700 0.0000  
## 701 0.0000  
## 702 0.0000  
## 703 0.0000  
## 704 0.0000  
## 705 0.0000  
## 706 0.0378  
## 707 0.0745  
## 708 0.0460  
## 709 0.0400  
## 710 0.8688  
## 711 0.5995  
## 712 1.3124  
## 713 0.3276  
## 714 2.1420  
## 715 0.5888  
## 716 0.1989  
## 717 0.6024  
## 718 0.1311  
## 719 0.1512  
## 720 0.0356  
## 721 0.0000  
## 722 0.0000  
## 723 0.1434  
## 724 0.0000  
## 725 0.0000  
## 726 0.0000  
## 727 0.0000  
## 728 0.0000  
## 729 0.0000  
## 730 0.0000  
## 731 0.0000  
## 732 0.0000  
## 733 0.0000  
## 734 0.0000  
## 735 0.0000  
## 736 0.0367  
## 737 0.0000  
## 738 0.2851  
## 739 0.5083  
## 740 0.2420  
## 741 0.0676  
## 742 0.0320  
## 743 0.0709  
## 744 0.2129  
## 745 0.0382  
## 746 0.0350  
## 747 0.0326  
## 748 0.0000  
## 749 0.0000  
## 750 0.0393  
## 751 0.0000  
## 752 0.0000  
## 753 0.0000  
## 754 0.0000  
## 755 0.0000  
## 756 0.0000  
## 757 0.0000  
## 758 0.0000  
## 759 0.0000  
## 760 0.0000  
## 761 0.0000  
## 762 0.0000  
## 763 0.0000  
## 764 0.0000  
## 765 0.0000  
## 766 0.0000  
## 767 0.0000  
## 768 0.0000  
## 769 0.0000  
## 770 0.0000  
## 771 0.0371  
## 772 0.0000  
## 773 0.0000  
## 774 0.0000  
## 775 0.0000  
## 776 0.0000  
## 777 0.0000  
## 778 0.0000  
## 779 0.0000  
## 780 0.0000  
## 781 0.0000  
## 782 0.0000  
## 783 0.0000  
## 784 0.0000

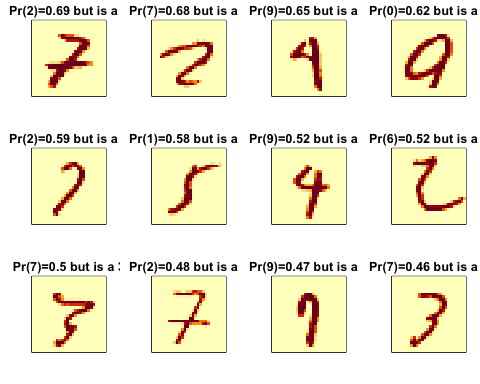
image(matrix(imp, 28, 28))



p\_max <- predict(fit\_knn, x\_test[,col\_index])  
p\_max <- apply(p\_max, 1, max)  
ind <- which(y\_hat\_knn != y\_test)  
ind <- ind[order(p\_max[ind], decreasing = TRUE)]  
rafalib::mypar(3,4)  
for(i in ind[1:12]){  
 image(matrix(x\_test[i,], 28, 28)[, 28:1],  
 main = paste0("Pr(",y\_hat\_knn[i],")=",round(p\_max[i], 2),  
 " but is a ",y\_test[i]),  
 xaxt="n", yaxt="n")  
}



p\_max <- predict(fit\_rf, x\_test[,col\_index])$census   
p\_max <- p\_max / rowSums(p\_max)  
p\_max <- apply(p\_max, 1, max)  
ind <- which(y\_hat\_rf != y\_test)  
ind <- ind[order(p\_max[ind], decreasing = TRUE)]  
rafalib::mypar(3,4)  
for(i in ind[1:12]){  
 image(matrix(x\_test[i,], 28, 28)[, 28:1],   
 main = paste0("Pr(",y\_hat\_rf[i],")=",round(p\_max[i], 2),  
 " but is a ",y\_test[i]),  
 xaxt="n", yaxt="n")  
}



## Ensembles

There is a link to the relevant sections of the textbook: [Ensembles](https://rafalab.github.io/dsbook/machine-learning-in-practice.html#ensembles)

**Key points**

* **Ensembles** combine multiple machine learning algorithms into one model to improve predictions.

*Code*

p\_rf <- predict(fit\_rf, x\_test[,col\_index])$census  
p\_rf <- p\_rf / rowSums(p\_rf)  
p\_knn <- predict(fit\_knn, x\_test[,col\_index])  
p <- (p\_rf + p\_knn)/2  
y\_pred <- factor(apply(p, 1, which.max)-1)  
confusionMatrix(y\_pred, y\_test)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1 2 3 4 5 6 7 8 9  
## 0 102 0 1 0 0 0 0 0 0 1  
## 1 0 121 1 0 1 1 1 2 0 0  
## 2 0 0 102 1 0 0 0 3 0 0  
## 3 0 0 0 78 0 1 0 0 3 2  
## 4 0 0 0 0 102 0 0 1 1 0  
## 5 0 0 0 2 0 68 0 0 5 0  
## 6 0 0 1 0 1 0 101 0 0 0  
## 7 0 0 1 2 0 0 0 102 0 0  
## 8 0 0 0 1 0 0 0 0 81 0  
## 9 0 0 0 0 5 0 0 2 1 102  
##   
## Overall Statistics  
##   
## Accuracy : 0.959   
## 95% CI : (0.945, 0.97)  
## No Information Rate : 0.121   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.954   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5  
## Sensitivity 1.000 1.000 0.962 0.929 0.936 0.971  
## Specificity 0.998 0.993 0.996 0.993 0.998 0.992  
## Pos Pred Value 0.981 0.953 0.962 0.929 0.981 0.907  
## Neg Pred Value 1.000 1.000 0.996 0.993 0.992 0.998  
## Prevalence 0.102 0.121 0.106 0.084 0.109 0.070  
## Detection Rate 0.102 0.121 0.102 0.078 0.102 0.068  
## Detection Prevalence 0.104 0.127 0.106 0.084 0.104 0.075  
## Balanced Accuracy 0.999 0.997 0.979 0.961 0.967 0.982  
## Class: 6 Class: 7 Class: 8 Class: 9  
## Sensitivity 0.990 0.927 0.890 0.971  
## Specificity 0.998 0.997 0.999 0.991  
## Pos Pred Value 0.981 0.971 0.988 0.927  
## Neg Pred Value 0.999 0.991 0.989 0.997  
## Prevalence 0.102 0.110 0.091 0.105  
## Detection Rate 0.101 0.102 0.081 0.102  
## Detection Prevalence 0.103 0.105 0.082 0.110  
## Balanced Accuracy 0.994 0.962 0.945 0.981

## Comprehension Check - Ensembles

1. Use the training set to build a model with several of the models available from the caret package. We will test out 10 of the most common machine learning models in this exercise:

models <- c("glm", "lda", "naive\_bayes", "svmLinear", "knn", "gamLoess", "multinom", "qda", "rf", "adaboost")

Apply all of these models using train() with all the default parameters. You may need to install some packages. Keep in mind that you will probably get some warnings. Also, it will probably take a while to train all of the models - be patient!

Run the following code to train the various models:

# set.seed(1) # if using R 3.5 or earlier  
set.seed(1, sample.kind = "Rounding") # if using R 3.6 or later

## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler  
## used

data("mnist\_27")  
  
fits <- lapply(models, function(model){   
 print(model)  
 train(y ~ ., method = model, data = mnist\_27$train)  
})

## [1] "glm"  
## [1] "lda"  
## [1] "naive\_bayes"  
## [1] "svmLinear"  
## [1] "knn"  
## [1] "gamLoess"

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.41586

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.4375

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.41586

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.10703

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.094737

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.10703

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.092518

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.57895

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.54068

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.43969

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.43969

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.092518

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.57895

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.54068

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.43969

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.092316

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.092316

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.53846

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.53555

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.57895

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.53555

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.43969

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.10703

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.094737

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.10703

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.57895

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.54071

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.43969

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.092316

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.40628

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.41379

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.40628

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.4375

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.40628

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.092518

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.57895

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.54068

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.10703

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.094737

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.10703

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.43969

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.402

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.40323

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.402

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.41379

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.402

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.40426

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.402

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.4375

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.402

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.41586

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.4375

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.41586

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.43969

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.57895

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.54071

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.41586

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.4375

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.41586

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.092316

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.092518

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.57895

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.54068

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.10877

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.094737

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.10877

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.089286

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## lowerlimit 0.092316

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.46667

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.43969

## Warning in gam.lo(data[["lo(x\_1, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.53846

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.50797

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.51111

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.50797

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.57895

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.50797

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## eval 0.53333

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## upperlimit 0.50797

## Warning in gam.lo(data[["lo(x\_2, span = 0.5, degree = 1)"]], z, w, span = 0.5, :  
## extrapolation not allowed with blending

## [1] "multinom"  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 384.794809  
## final value 384.794775   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 421.251454   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 384.848555  
## final value 384.848522   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 358.466023  
## final value 358.466014   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 400.257332   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 358.528966  
## final value 358.528958   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 345.361326  
## final value 345.361319   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 389.162400   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 345.427631  
## final value 345.427624   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 370.819967  
## iter 10 value 370.819967  
## iter 10 value 370.819967  
## final value 370.819967   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 411.520894   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 370.881269  
## iter 10 value 370.881269  
## iter 10 value 370.881269  
## final value 370.881269   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 338.339240  
## final value 337.642174   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 389.552735   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 337.725860  
## final value 337.725851   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 362.651997  
## iter 10 value 362.651996  
## iter 10 value 362.651996  
## final value 362.651996   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 404.947235   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 362.716896  
## iter 10 value 362.716895  
## iter 10 value 362.716894  
## final value 362.716894   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 353.360649   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 396.615883   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 353.427369   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 331.505876  
## final value 331.505837   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 382.233327   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 331.587049  
## final value 331.587010   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 364.158073  
## iter 10 value 364.158073  
## iter 10 value 364.158073  
## final value 364.158073   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 400.438283   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 364.210111  
## iter 10 value 364.210111  
## iter 10 value 364.210111  
## final value 364.210111   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 343.760429  
## final value 343.760410   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 387.083157   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 343.826126  
## final value 343.826108   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 377.277862  
## iter 10 value 377.277862  
## iter 10 value 377.277861  
## final value 377.277861   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 413.479657   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 377.330740  
## iter 10 value 377.330739  
## iter 10 value 377.330738  
## final value 377.330738   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 363.527477  
## final value 363.527449   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 405.904614   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 363.591426  
## final value 363.591399   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 346.706756  
## iter 10 value 346.706754  
## iter 10 value 346.706754  
## final value 346.706754   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 393.064300   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 346.778579  
## iter 10 value 346.778577  
## iter 10 value 346.778577  
## final value 346.778577   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 350.308158  
## final value 350.308124   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 394.686750   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 350.376208  
## final value 350.376174   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 365.423988  
## final value 365.423967   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 407.046095   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 365.486830  
## final value 365.486809   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 375.942875  
## final value 375.942868   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 412.738783   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 375.996860  
## final value 375.996853   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 369.004020  
## final value 369.003531   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 407.374841   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 369.060934  
## final value 369.060455   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 360.551961  
## iter 10 value 360.551959  
## iter 10 value 360.551959  
## final value 360.551959   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 400.866217   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 360.611945  
## iter 10 value 360.611943  
## iter 10 value 360.611943  
## final value 360.611943   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 370.467778  
## final value 370.414135   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 406.680836   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 370.519928  
## final value 370.466715   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 355.236387  
## final value 355.236347   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 401.370189   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 355.308279  
## final value 355.308240   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 364.714111  
## final value 364.714051   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 407.312950   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 364.779508  
## final value 364.779448   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 347.812292  
## final value 347.812150   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 389.764148  
## iter 10 value 389.764145  
## iter 10 value 389.764145  
## final value 389.764145   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 347.875247  
## final value 347.875105   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 319.870357  
## final value 319.870338   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 372.994080   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 319.955663  
## final value 319.955644   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 312.576095  
## final value 312.576064   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 367.284329  
## iter 10 value 367.284329  
## iter 10 value 367.284329  
## final value 367.284329   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 312.666550  
## final value 312.666520   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 363.313712  
## iter 10 value 363.313712  
## iter 10 value 363.313712  
## final value 363.313712   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## final value 403.175943   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 363.373575  
## iter 10 value 363.373575  
## iter 10 value 363.373575  
## final value 363.373575   
## converged  
## # weights: 4 (3 variable)  
## initial value 554.517744   
## iter 10 value 358.900453  
## iter 10 value 358.900452  
## iter 10 value 358.900452  
## final value 358.900452   
## converged  
## [1] "qda"  
## [1] "rf"  
## note: only 1 unique complexity parameters in default grid. Truncating the grid to 1 .  
##   
## [1] "adaboost"

names(fits) <- models

Did you train all of the models?

* ☒ A. Yes
* ☐ B. No

1. Now that you have all the trained models in a list, use sapply() or map() to create a matrix of predictions for the test set. You should end up with a matrix with length(mnist\_27$test$y) rows and length(models) columns.

What are the dimensions of the matrix of predictions?

pred <- sapply(fits, function(object)   
 predict(object, newdata = mnist\_27$test))  
dim(pred)

## [1] 200 10

1. Now compute accuracy for each model on the test set.

Report the mean accuracy across all models.

acc <- colMeans(pred == mnist\_27$test$y)  
acc

## glm lda naive\_bayes svmLinear knn gamLoess   
## 0.750 0.750 0.795 0.755 0.840 0.845   
## multinom qda rf adaboost   
## 0.750 0.820 0.780 0.805

mean(acc)

## [1] 0.789

1. Next, build an ensemble prediction by majority vote and compute the accuracy of the ensemble. Vote 7 if more than 50% of the models are predicting a 7, and 2 otherwise.

What is the accuracy of the ensemble?

votes <- rowMeans(pred == "7")  
y\_hat <- ifelse(votes > 0.5, "7", "2")  
mean(y\_hat == mnist\_27$test$y)

## [1] 0.815

1. In Q3, we computed the accuracy of each method on the test set and noticed that the individual accuracies varied.

How many of the individual methods do better than the ensemble?

Which individual methods perform better than the ensemble?

ind <- acc > mean(y\_hat == mnist\_27$test$y)  
sum(ind)

## [1] 3

models[ind]

## [1] "knn" "gamLoess" "qda"

* ☐ A. glm
* ☐ B. lda
* ☐ C. naive\_bayes
* ☐ D. svmLinear
* ☒ E. knn
* ☒ F. gamLoess
* ☐ G. multinom
* ☒ H. qda
* ☐ I. rf
* ☐ J. adaboost

1. It is tempting to remove the methods that do not perform well and re-do the ensemble. The problem with this approach is that we are using the test data to make a decision. However, we could use the minimum accuracy estimates obtained from cross validation with the training data for each model from fit$results$Accuracy. Obtain these estimates and save them in an object. Report the mean of these training set accuracy estimates.

What is the mean of these training set accuracy estimates?

acc\_hat <- sapply(fits, function(fit) min(fit$results$Accuracy))  
mean(acc\_hat)

## [1] 0.809

1. Now let’s only consider the methods with an estimated accuracy of greater than or equal to 0.8 when constructing the ensemble. Vote 7 if 50% or more of the models are predicting a 7, and 2 otherwise.

What is the accuracy of the ensemble now?

ind <- acc\_hat >= 0.8  
votes <- rowMeans(pred[,ind] == "7")  
y\_hat <- ifelse(votes>=0.5, 7, 2)  
mean(y\_hat == mnist\_27$test$y)

## [1] 0.825

## Recommendation Systems