# Machine Learning

## Section 1: Introduction to Machine Learning

## Notation

### Key points

* + X1,...,Xp denote the features, Y denotes the outcomes, and Y^ denotes the predictions.
  + Machine learning prediction tasks can be divided into **categorical** and **continuous** outcomes. We refer to these as **classification** and **prediction**, respectively.

## An Example

### Key points

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

## Section 2: Machine Learning Basics

## 2.1 Basics of Evaluating Machine Learning Algorithms

## Caret package, training and test sets, and overall accuracy

### Key points

* + 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

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(2007)

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)

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

y\_hat <- ifelse(x > 62, "Male", "Female") %>% factor(levels = levels(test\_set$sex))

mean(y == y\_hat)

# 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)

})

max(accuracy)

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

best\_cutoff

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)

## 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)

test\_set %>%

mutate(y\_hat = y\_hat) %>%

group\_by(sex) %>%

summarize(accuracy = mean(y\_hat == sex))

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

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

## Balanced accuracy and F1 score

### Key points

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

### Code

# maximize F-score

cutoff <- seq(61, 70)

F\_1 <- map\_dbl(cutoff, function(x){

y\_hat <- ifelse(train\_set$height > x, "Male", "Female") %>%

factor(levels = levels(test\_set$sex))

F\_meas(data = y\_hat, reference = factor(train\_set$sex))

})

max(F\_1)

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

y\_hat <- ifelse(test\_set$height > best\_cutoff, "Male", "Female") %>%

factor(levels = levels(test\_set$sex))

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

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

## 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

### 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

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)

# 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")

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()

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()

## 2.2 Conditional Probabilities

## Conditional probabilities

### Key points

* + Conditional probabilities for each class:

pk(x)=Pr(Y=k|X=x),fork=1,...,K

* + In machine learning, this is referred to as **Bayes' Rule**. This is a theoretical rule because in practice we don't know p(x). Having a good estimate of the p(x) 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

### Key points

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

pk(x)=Pr(Y=k|X=x),fork=1,...,K

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:

Y^=E(Y|X=x)minimizesE{(Y^−Y)2|X=x}

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.**

## Conditional expectations and loss function

### Key points

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

pk(x)=Pr(Y=k|X=x),fork=1,...,K

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:

Y^=E(Y|X=x)minimizesE{(Y^−Y)2|X=x}

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.**

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

## 3.1 Linear Regression for Predictions

## Linear Regression for Prediction

### 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

library(HistData)

set.seed(1983)

galton\_heights <- GaltonFamilies %>%

filter(gender == "male") %>%

group\_by(family) %>%

sample\_n(1) %>%

ungroup() %>%

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)

m <- mean(train\_set$son)

# squared loss

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

# fit linear regression model

fit <- lm(son ~ father, data = train\_set)

fit$coef

y\_hat <- fit$coef[1] + fit$coef[2]\*test\_set$father

mean((y\_hat - test\_set$son)^2)

## 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)

# read help files

?predict.lm

?predict.glm

## Regression for a Categorical Outcome

### Key points

* + The regression approach can be extended to categorical data. For example, we can try regression to estimate the conditional probability:

p(x)=Pr(Y=1|X=x)=β0+β1x

* + Once we have estimates β0 and  β1, we can obtain an actual prediction p(x). Then we can define a specific decision rule to form a prediction.

### Code

library(dslabs)

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

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"))

heights %>%

mutate(x = round(height)) %>%

group\_by(x) %>%

filter(n() >= 10) %>%

summarize(prop = mean(sex == "Female")) %>%

ggplot(aes(x, prop)) +

geom\_point()

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"]

## Logistic Regression

### Key points

* + **Logistic regression** is an extension of linear regression that assures that the estimate of conditional probability Pr(Y=1|X=x) is between 0 and 1. This approach makes use of the logistic transformation:

g(p)=logp1−p

* + With logistic regression, we model the conditional probability directly with:

g{Pr(Y=1|X=x)}=β0+β1x

* + 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])

range(p\_hat)

# 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")

y\_hat\_logit <- ifelse(p\_hat\_logit > 0.5, "Female", "Male") %>% factor

confusionMatrix(y\_hat\_logit, test\_set$sex)$overall[["Accuracy"]]

## Case Study: 2 or 7

### 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:

g{p(x1,x2}=g{Pr(Y=1|X1=x1,X2=x2)}=β0+β1x1+β2x2

* + 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 p(x1,x2). 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"]

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)

## 3.2 Smoothing

## Introduction to Smoothing

### 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

### Key points

* + The general idea of smoothing is to group data points into strata in which the value of f(x) can be assumed to be constant. We can make this assumption because we think f(x) changes slowly and, as a result, f(x) is almost constant in small windows of time.
  + This assumption implies that a good estimate for f(x) is the average of the Yi values in the window. The estimate is:

f^(x0)=1N0∑i∈A0Yi

* + In smoothing, we call the size of the interval |x−x0| 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)

### 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 f(x). **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.args = list(degree=1))

## 3.3 Working with Matrices

## Matrices

### 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

library(tidyverse)

library(dslabs)

if(!exists("mnist")) mnist <- read\_mnist()

class(mnist$train$images)

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

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

## Matrix Notation

### Key points

* + In matrix algebra, we have three main types of objects: **scalars**, **vectors**, and **matrices**.
    - **Scalar:**  α=1
    - **Vector:**  X1=(x 1,1 ….. x N,1)
    - **Matrix:** X=[X1X2]=((x 1,1 …x N,1)(x 1,2 …x N,2))
  + 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])

x\_1 <- 1:5

x\_2 <- 6:10

cbind(x\_1, x\_2)

dim(x)

dim(x\_1)

dim(as.matrix(x\_1))

dim(x)

## 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

# fill by row

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

mat\_t

identical(t(mat), mat\_t)

matrix(my\_vector, 5, 5)

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

### 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")

avgs <- apply(x, 1, mean)

sds <- apply(x, 2, sd)

## 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

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)]

x[c(2,3),]

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

dim(new\_x)

class(x[,1])

dim(x[1,])

#preserve the matrix class

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

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

## Indexing with Matrices and 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)

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

mat <- matrix(1:15, 5, 3)

mat[mat > 6 & mat < 12] <- 0

mat

#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

### 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 = "/")

## [Section 4: Distance, Knn, Cross-validation, and Generative Models](https://courses.edx.org/courses/course-v1:HarvardX+PH125.8x+1T2020/course/" \l "block-v1:HarvardX+PH125.8x+1T2020+type@chapter+block@7c50a2fc6d44433fa902be8ed097f301)

## 4.1 Nearest Neighbours

## 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 N is the number of observations, the distance between two predictors, say 1 and 2, is:

dist(1,2)=sqrt(∑i=1N((xi,1−xi,2)^2))

* + 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

library(tidyverse)

library(dslabs)

if(!exists("mnist")) mnist <- read\_mnist()

set.seed(1995) # if using R 3.5 or earlier

set.seed(1995) # if using R 3.6 or later

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]

x\_1 <- x[1,]

x\_2 <- x[2,]

x\_3 <- x[3,]

#distance between two numbers

sqrt(sum((x\_1 - x\_2)^2))

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

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

#compute distance using matrix algebra

sqrt(crossprod(x\_1 - x\_2))

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

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

#compute distance between each row

d <- dist(x)

class(d)

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

#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))

d\_492 <- as.matrix(d)[492,]

image(1:28, 1:28, matrix(d\_492, 28, 28))

## Knn

### 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 (x1,x2) for which we want an estimate of p(x1,x2), we look for the **k nearest points** to (x1,x2) 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: outcome∼predictor1+predictor2+predictor3. 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

#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]

#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"]

## Overtraining and Oversmoothing

### 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 k=1. With k=1, the estimate for each (x1,x2) in the training set is obtained with just the y corresponding to that point.
    - When we try a larger k, the k 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 k, 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 k 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"]

y\_hat\_knn <- predict(knn\_fit, mnist\_27$test, type = "class")

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

#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"]]

#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"]

#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)]

max(accuracy$test)

## 4.3 Generative Models

## 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 X (we model how the entire data, X and Y, are generated).

## Naive Bayes

### Key points

* + Bayes' rule:

p(x)=Pr(Y=1|X=x)=fX|Y=1(X)Pr(Y=1)/fX|Y=0(X)Pr(Y=0)+fX|Y=1(X)Pr(Y=1)

with fX|Y=1 and fX|Y=0 representing the distribution functions of the predictor X for the two classes Y=1 and Y=0.

* + 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

library("caret")

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))

params

# Estimating the prevalence

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

pi

# 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

### Key points

* + The Naive Bayes approach includes a **parameter to account for differences in prevalence** π=Pr(Y=1). If we use hats to denote the estimates, we can write p(x)^ as:

p^(x)=Pr(Y=1|X=x)=f^X|Y=1(x)π^/f^X|Y=0(x)(1−π^)+f^X|Y=1(x)Pr(Y=1)

* + The Naive Bayes approach gives us a direct way to correct the imbalance between sensitivity and specificity by simply forcing π^ 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))

# Computing specificity

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

# 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))

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

# 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

### Key points

* + **Quadratic discriminant analysis (QDA)** is a version of Naive Bayes in which we assume that the distributions pX|Y=1(x) and pX|Y=0(x) 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))

# 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"]

# 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))

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"]

## 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()

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

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

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

train\_lda <- train(y ~ ., method = "lda", data = train\_set)

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

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"]

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")

## [Section 5: Classification with More than Two Classes and the Caret Package](https://courses.edx.org/courses/course-v1:HarvardX+PH125.8x+1T2020/course/" \l "block-v1:HarvardX+PH125.8x+1T2020+type@chapter+block@3f70797a568946e195863415b94d25fd)

## [5.1: Classification with More than Two Classes](https://courses.edx.org/courses/course-v1:HarvardX+PH125.8x+1T2020/course/#block-v1:HarvardX+PH125.8x+1T2020+type@sequential+block@78cfd6af400d452c9aa9f8070403e55c)

## Trees Motivation

### Key points

* + LDA and QDA are **not meant to be used with many predictors p** 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 p predictors, the distance between two observations is computed in p-dimensional space.

## Classification and Regression Trees (CART)

### 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 Y by **partitioning the predictor**s.
  + The general idea here is to **build a decision tree** and, at end of each node, obtain a predictor y^. Mathematically, we are **partitioning the predictor space** into J non-overlapping regions, R1 , R2, ..., RJ and then for any predictor x that falls within region Rj, estimate f(x) with the average of the training observations yi for which the associated predictor xi in also in Rj.
  + To pick j and its value s, we find the pair that **minimizes the residual sum of squares (RSS)**:

∑i:xiR1(j,s)(yi−y^R1)2+∑i:xiR2(j,s)(yi−y^R2)2

* + 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

library(tidyverse)

library(dslabs)

data("olive")

olive %>% as\_tibble()

table(olive$region)

olive <- select(olive, -area)

# Predict region using KNN

library(caret)

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)

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

library(caret)

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

### 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**.

Gini(j)=∑k=1Kp^j,k(1−p^j,k)

entropy(j)=−∑k,k=1Kp^j,klog(p^j,k),with 0×log(0)defined as 0

* + 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"]

## 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

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")

library(randomForest)

train\_rf <- randomForest(y ~ ., data=mnist\_27$train)

confusionMatrix(predict(train\_rf, mnist\_27$test), mnist\_27$test$y)$overall["Accuracy"]

# 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"]

## 5.2 Caret Package

## Caret Package

### 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

library(tidyverse)

library(dslabs)

data("mnist\_27")

library(caret)

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"]]

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

## Tuning Parameters with Caret

### Caret package links

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

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

### 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")

modelLookup("knn")

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

train\_knn$finalModel

confusionMatrix(predict(train\_knn, mnist\_27$test, type = "raw"),

mnist\_27$test$y)$overall["Accuracy"]

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])

install.packages("gam")

modelLookup("gamLoess")

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

## 5.3 Titanic Exercise

--Solved—

## Section 6: Model Fitting and Recommendation System

## 6.1 Case Study: MNIST

## Case Study: MNIST

### 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

library(dslabs)

mnist <- read\_mnist()

names(mnist)

dim(mnist$train$images)

class(mnist$train$labels)

table(mnist$train$labels)

# 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

### 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

library(matrixStats)

sds <- colSds(x)

qplot(sds, bins = 256, color = I(“black”))

library(caret)

nzv <- nearZeroVar(x)

image(matrix(1:784 %in% nzv, 28, 28))

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

length(col\_index)

## Model Fitting for MNIST Data

### 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"]

cm$byClass[,1:2]

library(Rborist)

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

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"]

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

### 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

library(randomForest)

x <- mnist$train$images[index,]

y <- factor(mnist$train$labels[index])

rf <- randomForest(x, y, ntree = 50)

imp <- importance(rf)

imp

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

### 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)

## 6.2 Recommendation Systems

### Netflix Challenge links

For more information about the "Netflix Challenge," you can check out these sites:

* + <https://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest/>
  + <http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/>
  + <https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf>

### Key points

* + **Recommendation systems** are more complicated machine learning challenges because each outcome has a different set of predictors. For example, different users rate a different number of movies and rate different movies.
  + To compare different models or to see how well we’re doing compared to a baseline, we will use **root mean squared error (RMSE) as our loss function**. We can interpret RMSE similar to standard deviation.
  + If N is the number of user-movie combinations, yu,i is the rating for movie i by user u, and y^u,i is our prediction, then **RMSE is defined as follows**:

1N∑u,i(y^u,i−yu,i)2−−−−−−−−−−−−−−−√

### Code

**Please refer to the textbook for an updated version of the code that may contain some corrections.**

library(dslabs)

library(tidyverse)

data("movielens")

head(movielens)

movielens %>%

summarize(n\_users = n\_distinct(userId),

n\_movies = n\_distinct(movieId))

keep <- movielens %>%

dplyr::count(movieId) %>%

top\_n(5) %>%

pull(movieId)

tab <- movielens %>%

filter(userId %in% c(13:20)) %>%

filter(movieId %in% keep) %>%

select(userId, title, rating) %>%

spread(title, rating)

tab %>% knitr::kable()

users <- sample(unique(movielens$userId), 100)

rafalib::mypar()

movielens %>% filter(userId %in% users) %>%

select(userId, movieId, rating) %>%

mutate(rating = 1) %>%

spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%

as.matrix() %>% t(.) %>%

image(1:100, 1:100,. , xlab="Movies", ylab="Users")

abline(h=0:100+0.5, v=0:100+0.5, col = "grey")

movielens %>%

dplyr::count(movieId) %>%

ggplot(aes(n)) +

geom\_histogram(bins = 30, color = "black") +

scale\_x\_log10() +

ggtitle("Movies")

movielens %>%

dplyr::count(userId) %>%

ggplot(aes(n)) +

geom\_histogram(bins = 30, color = "black") +

scale\_x\_log10() +

ggtitle("Users")

library(caret)

set.seed(755)

test\_index <- createDataPartition(y = movielens$rating, times = 1,

p = 0.2, list = FALSE)

train\_set <- movielens[-test\_index,]

test\_set <- movielens[test\_index,]

test\_set <- test\_set %>%

semi\_join(train\_set, by = "movieId") %>%

semi\_join(train\_set, by = "userId")

RMSE <- function(true\_ratings, predicted\_ratings){

sqrt(mean((true\_ratings - predicted\_ratings)^2))

}

## Building the Recommendation System

### Key points

* + We start with a model that **assumes the same rating for all movies and all users**, with all the differences explained by random variation: If μ represents the true rating for all movies and users and ϵ represents independent errors sampled from the same distribution centered at zero, then:

Yu,i=μ+ϵu,i

* + In this case, the **least squares estimate of μ** — the estimate that minimizes the root mean squared error — is the average rating of all movies across all users.
  + We can improve our model by adding a term, bi, that represents the **average rating for movie i**:

Yu,i=μ+bi+ϵu,i

bi is the average of Yu,i minus the overall mean for each movie i.

* + We can further improve our model by adding bu, the **user-specific effect**:

Yu,i=μ+bi+bu+ϵu,i

* + Note that because there are thousands of b's, the lm() function will be very slow or cause R to crash, so **we don’t recommend using linear regression to calculate these effects**.

### Code

**Please refer to the textbook for an updated version of the code that may contain some corrections.**

mu\_hat <- mean(train\_set$rating)

mu\_hat

naive\_rmse <- RMSE(test\_set$rating, mu\_hat)

naive\_rmse

predictions <- rep(2.5, nrow(test\_set))

RMSE(test\_set$rating, predictions)

rmse\_results <- data\_frame(method = "Just the average", RMSE = naive\_rmse)

fit <- lm(rating ~ as.factor(userId), data = movielens)

mu <- mean(train\_set$rating)

movie\_avgs <- train\_set %>%

group\_by(movieId) %>%

summarize(b\_i = mean(rating - mu))

movie\_avgs %>% qplot(b\_i, geom ="histogram", bins = 10, data = ., color = I("black"))

predicted\_ratings <- mu + test\_set %>%

left\_join(movie\_avgs, by='movieId') %>%

.$b\_i

model\_1\_rmse <- RMSE(predicted\_ratings, test\_set$rating)

rmse\_results <- bind\_rows(rmse\_results,

data\_frame(method="Movie Effect Model",

RMSE = model\_1\_rmse ))

rmse\_results %>% knitr::kable()

train\_set %>%

group\_by(userId) %>%

summarize(b\_u = mean(rating)) %>%

filter(n()>=100) %>%

ggplot(aes(b\_u)) +

geom\_histogram(bins = 30, color = "black")

# lm(rating ~ as.factor(movieId) + as.factor(userId))

user\_avgs <- test\_set %>%

left\_join(movie\_avgs, by='movieId') %>%

group\_by(userId) %>%

summarize(b\_u = mean(rating - mu - b\_i))

predicted\_ratings <- test\_set %>%

left\_join(movie\_avgs, by='movieId') %>%

left\_join(user\_avgs, by='userId') %>%

mutate(pred = mu + b\_i + b\_u) %>%

.$pred

model\_2\_rmse <- RMSE(predicted\_ratings, test\_set$rating)

rmse\_results <- bind\_rows(rmse\_results,

data\_frame(method="Movie + User Effects Model",

RMSE = model\_2\_rmse ))

rmse\_results %>% knitr::kable()

## 6.3 Regularization

### Regularization

### Notes

* + To improve our results, we will use **regularization**. Regularization constrains the total variability of the effect sizes by penalizing large estimates that come from small sample sizes.
  + To estimate the b’s, we will now **minimize this equation**, which contains a penalty term:

1N∑u,i(yu,i−μ−bi)2+λ∑ib2i

The first term is the mean squared error and the second is a penalty term that gets larger when many b’s are large.

The values of b that minimize this equation are given by:

b^i(λ)=1λ+ni∑u=1ni(Yu,i−μ^),

where ni is a number of ratings b for movie i.

* + The **larger λ is, the more we shrink**. λ is a tuning parameter, so we can use cross-validation to choose it. We should be using full cross-validation on just the training set, without using the test set until the final assessment.
  + We can also use regularization to estimate the **user effect**. We will now minimize this equation:

1N∑u,i(yu,i−μ−bi−bu)2+λ(∑ib2i+∑ub2u)

### Code

**Please refer to the textbook for an updated version of the code that may contain some corrections.**

library(dslabs)

library(tidyverse)

library(caret)

data("movielens")

set.seed(755)

test\_index <- createDataPartition(y = movielens$rating, times = 1,

p = 0.2, list = FALSE)

train\_set <- movielens[-test\_index,]

test\_set <- movielens[test\_index,]

test\_set <- test\_set %>%

semi\_join(train\_set, by = "movieId") %>%

semi\_join(train\_set, by = "userId")

RMSE <- function(true\_ratings, predicted\_ratings){

sqrt(mean((true\_ratings - predicted\_ratings)^2))

}

mu\_hat <- mean(train\_set$rating)

naive\_rmse <- RMSE(test\_set$rating, mu\_hat)

rmse\_results <- data\_frame(method = "Just the average", RMSE = naive\_rmse)

mu <- mean(train\_set$rating)

movie\_avgs <- train\_set %>%

group\_by(movieId) %>%

summarize(b\_i = mean(rating - mu))

predicted\_ratings <- mu + test\_set %>%

left\_join(movie\_avgs, by='movieId') %>%

.$b\_i

model\_1\_rmse <- RMSE(predicted\_ratings, test\_set$rating)

rmse\_results <- bind\_rows(rmse\_results,

data\_frame(method="Movie Effect Model",

RMSE = model\_1\_rmse ))

user\_avgs <- test\_set %>%

left\_join(movie\_avgs, by='movieId') %>%

group\_by(userId) %>%

summarize(b\_u = mean(rating - mu - b\_i))

predicted\_ratings <- test\_set %>%

left\_join(movie\_avgs, by='movieId') %>%

> left\_join(user\_avgs, by='userId') %>%

mutate(pred = mu + b\_i + b\_u) %>%

.$pred

model\_2\_rmse <- RMSE(predicted\_ratings, test\_set$rating)

rmse\_results <- bind\_rows(rmse\_results,

data\_frame(method="Movie + User Effects Model",

RMSE = model\_2\_rmse ))

test\_set %>%

left\_join(movie\_avgs, by='movieId') %>%

mutate(residual = rating - (mu + b\_i)) %>%

arrange(desc(abs(residual))) %>%

select(title, residual) %>% slice(1:10) %>% knitr::kable()

movie\_titles <- movielens %>%

select(movieId, title) %>%

distinct()

movie\_avgs %>% left\_join(movie\_titles, by="movieId") %>%

arrange(desc(b\_i)) %>%

select(title, b\_i) %>%

slice(1:10) %>%

knitr::kable()

movie\_avgs %>% left\_join(movie\_titles, by="movieId") %>%

arrange(b\_i) %>%

select(title, b\_i) %>%

slice(1:10) %>%

knitr::kable()

train\_set %>% dplyr::count(movieId) %>%

left\_join(movie\_avgs) %>%

left\_join(movie\_titles, by="movieId") %>%

arrange(desc(b\_i)) %>%

select(title, b\_i, n) %>%

slice(1:10) %>%

knitr::kable()

train\_set %>% dplyr::count(movieId) %>%

left\_join(movie\_avgs) %>%

left\_join(movie\_titles, by="movieId") %>%

arrange(b\_i) %>%

select(title, b\_i, n) %>%

slice(1:10) %>%

knitr::kable()

lambda <- 3

mu <- mean(train\_set$rating)

movie\_reg\_avgs <- train\_set %>%

group\_by(movieId) %>%

summarize(b\_i = sum(rating - mu)/(n()+lambda), n\_i = n())

data\_frame(original = movie\_avgs$b\_i,

regularlized = movie\_reg\_avgs$b\_i,

n = movie\_reg\_avgs$n\_i) %>%

ggplot(aes(original, regularlized, size=sqrt(n))) +

geom\_point(shape=1, alpha=0.5)

train\_set %>%

dplyr::count(movieId) %>%

left\_join(movie\_reg\_avgs) %>%

left\_join(movie\_titles, by="movieId") %>%

arrange(desc(b\_i)) %>%

select(title, b\_i, n) %>%

slice(1:10) %>%

knitr::kable()

train\_set %>%

dplyr::count(movieId) %>%

left\_join(movie\_reg\_avgs) %>%

left\_join(movie\_titles, by="movieId") %>%

arrange(b\_i) %>%

select(title, b\_i, n) %>%

slice(1:10) %>%

knitr::kable()

predicted\_ratings <- test\_set %>%

left\_join(movie\_reg\_avgs, by='movieId') %>%

mutate(pred = mu + b\_i) %>%

.$pred

model\_3\_rmse <- RMSE(predicted\_ratings, test\_set$rating)

rmse\_results <- bind\_rows(rmse\_results,

data\_frame(method="Regularized Movie Effect Model",

RMSE = model\_3\_rmse ))

rmse\_results %>% knitr::kable()

lambdas <- seq(0, 10, 0.25)

mu <- mean(train\_set$rating)

just\_the\_sum <- train\_set %>%

group\_by(movieId) %>%

summarize(s = sum(rating - mu), n\_i = n())

rmses <- sapply(lambdas, function(l){

predicted\_ratings <- test\_set %>%

left\_join(just\_the\_sum, by='movieId') %>%

mutate(b\_i = s/(n\_i+l)) %>%

mutate(pred = mu + b\_i) %>%

.$pred

return(RMSE(predicted\_ratings, test\_set$rating))

})

qplot(lambdas, rmses)

lambdas[which.min(rmses)]

lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

mu <- mean(train\_set$rating)

b\_i <- train\_set %>%

group\_by(movieId) %>%

summarize(b\_i = sum(rating - mu)/(n()+l))

b\_u <- train\_set %>%

left\_join(b\_i, by="movieId") %>%

group\_by(userId) %>%

summarize(b\_u = sum(rating - b\_i - mu)/(n()+l))

predicted\_ratings <-

test\_set %>%

left\_join(b\_i, by = "movieId") %>%

left\_join(b\_u, by = "userId") %>%

mutate(pred = mu + b\_i + b\_u) %>%

.$pred

return(RMSE(predicted\_ratings, test\_set$rating))

})

qplot(lambdas, rmses)

lambda <- lambdas[which.min(rmses)]

lambda

rmse\_results <- bind\_rows(rmse\_results,

data\_frame(method="Regularized Movie + User Effect Model",

RMSE = min(rmses)))

rmse\_results %>% knitr::kable()

## Matrix Factorization

### Key points

* + Our earlier models fail to account for an important source of variation related to the fact that groups of movies and groups of users have similar rating patterns. We can observe these patterns by studying the residuals and **converting our data into a matrix where each user gets a row and each movie gets a column**:

ru,i=yu,i−b^i−b^u,

where yu,i is the entry in row u and column i.

* + We can **factorize the matrix of residuals r** into a vector p and vector q, ru,i≈puqi, allowing us to explain more of the variance using a model like this:

Yu,i=μ+bi+bu+puqi+ϵi,j

* + Because our example is more complicated, we can use **two factors to explain the structure and two sets of coefficients to describe users**:

Yu,i=μ+bi+bu+pu,1q1,i+pu,2q2,i+ϵi,j

* + To estimate factors using our data instead of constructing them ourselves, we can use **principal component analysis (PCA) or singular value decomposition (SVD)**.

### Code

train\_small <- movielens %>%

group\_by(movieId) %>%

filter(n() >= 50 | movieId == 3252) %>% ungroup() %>% #3252 is Scent of a Woman used in example

group\_by(userId) %>%

filter(n() >= 50) %>% ungroup()

y <- train\_small %>%

select(userId, movieId, rating) %>%

spread(movieId, rating) %>%

as.matrix()

rownames(y)<- y[,1]

y <- y[,-1]

colnames(y) <- with(movie\_titles, title[match(colnames(y), movieId)])

y <- sweep(y, 1, rowMeans(y, na.rm=TRUE))

y <- sweep(y, 2, colMeans(y, na.rm=TRUE))

m\_1 <- "Godfather, The"

m\_2 <- "Godfather: Part II, The"

qplot(y[ ,m\_1], y[,m\_2], xlab = m\_1, ylab = m\_2)

m\_1 <- "Godfather, The"

m\_3 <- "Goodfellas"

qplot(y[ ,m\_1], y[,m\_3], xlab = m\_1, ylab = m\_3)

m\_4 <- "You've Got Mail"

m\_5 <- "Sleepless in Seattle"

qplot(y[ ,m\_4], y[,m\_5], xlab = m\_4, ylab = m\_5)

cor(y[, c(m\_1, m\_2, m\_3, m\_4, m\_5)], use="pairwise.complete") %>%

knitr::kable()

set.seed(1)

options(digits = 2)

Q <- matrix(c(1 , 1, 1, -1, -1), ncol=1)

rownames(Q) <- c(m\_1, m\_2, m\_3, m\_4, m\_5)

P <- matrix(rep(c(2,0,-2), c(3,5,4)), ncol=1)

rownames(P) <- 1:nrow(P)

X <- jitter(P%\*%t(Q))

X %>% knitr::kable(align = "c")

cor(X)

t(Q) %>% knitr::kable(aling="c")

P

set.seed(1)

options(digits = 2)

m\_6 <- "Scent of a Woman"

Q <- cbind(c(1 , 1, 1, -1, -1, -1),

c(1 , 1, -1, -1, -1, 1))

rownames(Q) <- c(m\_1, m\_2, m\_3, m\_4, m\_5, m\_6)

P <- cbind(rep(c(2,0,-2), c(3,5,4)),

c(-1,1,1,0,0,1,1,1,0,-1,-1,-1))/2

rownames(P) <- 1:nrow(X)

X <- jitter(P%\*%t(Q), factor=1)

X %>% knitr::kable(align = "c")

cor(X)

t(Q) %>% knitr::kable(align="c")

P

six\_movies <- c(m\_1, m\_2, m\_3, m\_4, m\_5, m\_6)

tmp <- y[,six\_movies]

cor(tmp, use="pairwise.complete")

### SVD and PCA

### Key points

* + - You can think of **singular value decomposition (SVD)** as an algorithm that finds the vectors p and q that permit us to write the matrix of residuals r with m rows and n columns in the following way:

ru,i=pu,1q1,i+pu,2q2,i+...+pu,mqm,i,

with the variability of these terms decreasing and the p’s uncorrelated to each other.

* + - **SVD also computes the variabilities** so that we can know how much of the matrix’s total variability is explained as we add new terms.
    - The **vectors q are called the principal components** and the **vectors p are the user effects**. By using principal components analysis (PCA), matrix factorization can capture structure in the data determined by user opinions about movies.

### Code

y[is.na(y)] <- 0

y <- sweep(y, 1, rowMeans(y))

pca <- prcomp(y)

dim(pca$rotation)

dim(pca$x)

plot(pca$sdev)

var\_explained <- cumsum(pca$sdev^2/sum(pca$sdev^2))

plot(var\_explained)

library(ggrepel)

pcs <- data.frame(pca$rotation, name = colnames(y))

pcs %>% ggplot(aes(PC1, PC2)) + geom\_point() +

geom\_text\_repel(aes(PC1, PC2, label=name),

data = filter(pcs,

PC1 < -0.1 | PC1 > 0.1 | PC2 < -0.075 | PC2 > 0.1))

pcs %>% select(name, PC1) %>% arrange(PC1) %>% slice(1:10)

pcs %>% select(name, PC1) %>% arrange(desc(PC1)) %>% slice(1:10)

pcs %>% select(name, PC2) %>% arrange(PC2) %>% slice(1:10)

pcs %>% select(name, PC2) %>% arrange(desc(PC2)) %>% slice(1:10)

## Comprehensive test containing more notes than assignments

In this exercise set, we will be covering a topic useful for understanding matrix factorization: the singular value decomposition (SVD). SVD is a mathematical result that is widely used in machine learning, both in practice and to understand the mathematical properties of some algorithms. This is a rather advanced topic and to complete this exercise set you will have to be familiar with linear algebra concepts such as matrix multiplication, orthogonal matrices, and diagonal matrices.

The SVD tells us that we can **decompose** an N×p matrix Y with p<N as

Y=UDV⊤

with U and V **orthogonal** of dimensions N×p and p×p respectively and D a p×p **diagonal** matrix with the values of the diagonal decreasing:

d1,1≥d2,2≥…dp,p

In this exercise, we will see one of the ways that this decomposition can be useful. To do this, we will construct a dataset that represents grade scores for 100 students in 24 different subjects. The overall average has been removed so this data represents the percentage point each student received above or below the average test score. So a 0 represents an average grade (C), a 25 is a high grade (A+), and a -25 represents a low grade (F). You can simulate the data like this:

set.seed(1987)

#if using R 3.6 or later, use `set.seed(1987, sample.kind="Rounding")` instead

n <- 100

k <- 8

Sigma <- 64 \* matrix(c(1, .75, .5, .75, 1, .5, .5, .5, 1), 3, 3)

m <- MASS::mvrnorm(n, rep(0, 3), Sigma)

m <- m[order(rowMeans(m), decreasing = TRUE),]

y <- m %x% matrix(rep(1, k), nrow = 1) + matrix(rnorm(matrix(n\*k\*3)), n, k\*3)

colnames(y) <- c(paste(rep("Math",k), 1:k, sep="\_"),

paste(rep("Science",k), 1:k, sep="\_"),

paste(rep("Arts",k), 1:k, sep="\_"))

Our goal is to describe the student performances as succinctly as possible. For example, we want to know if these test results are all just a random independent numbers. Are all students just about as good? Does being good in one subject  imply you will be good in another? How does the SVD help with all this? We will go step by step to show that with just three relatively small pairs of vectors we can explain much of the variability in this 100×24 dataset.

### **Q1**

1/1 point (graded)

You can visualize the 24 test scores for the 100 students by plotting an image:

my\_image <- function(x, zlim = range(x), ...){

colors = rev(RColorBrewer::brewer.pal(9, "RdBu"))

cols <- 1:ncol(x)

rows <- 1:nrow(x)

image(cols, rows, t(x[rev(rows),,drop=FALSE]), xaxt = "n", yaxt = "n",

xlab="", ylab="", col = colors, zlim = zlim, ...)

abline(h=rows + 0.5, v = cols + 0.5)

axis(side = 1, cols, colnames(x), las = 2)

}

my\_image(y)

How would you describe the data based on this figure?

The test scores are all independent of each other.

The students that are good at math are not good at science.

The students that are good at math are not good at arts.

The students that test well are at the top of the image and there seem to be three groupings by subject.

The students that test well are at the bottom of the image and there seem to be three groupings by subject.

correct

Submit

You have used 2 of 2 attemptsSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

Show Answer

Correct (1/1 point)

Review

### **Q2**

1/1 point (graded)

You can examine the correlation between the test scores directly like this:

my\_image(cor(y), zlim = c(-1,1))

range(cor(y))

axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)

Which of the following best describes what you see?

The test scores are independent.

Test scores in math and science are highly correlated but scores in arts are not.

There is high correlation between tests in the same subject but no correlation across subjects.

There is correlation among all tests, but higher if the tests are in science and math and even higher within each subject.

correct

Submit

You have used 2 of 2 attemptsSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

Show Answer

Correct (1/1 point)

Review

### **Q3**

1/1 point (graded)

Remember that orthogonality means that U⊤U and V⊤V are equal to the identity matrix. This implies that we can also rewrite the decomposition as

YV=UD or U⊤Y=DV⊤

We can think of YV and U⊤V as two transformations of Y that preserve the total variability of Y since U and V are orthogonal.

Use the function svd() to compute the SVD of y. This function will return U, V, and the diagonal entries of D.

s <- svd(y)

names(s)

You can check that the SVD works by typing:

y\_svd <- s$u %\*% diag(s$d) %\*% t(s$v)

max(abs(y - y\_svd))

Compute the sum of squares of the columns of Y and store them in ss\_y. Then compute the sum of squares of columns of the transformed YV and store them in ss\_yv. Confirm that sum(ss\_y) is equal to sum(ss\_yv).

What is the value of sum(ss\_y) (and also the value of sum(ss\_yv))?  correct

175435

175434.6 Loading

**Explanation**

ss\_y <- apply(y^2, 2, sum)

ss\_yv <- apply((y%\*%s$v)^2, 2, sum)

sum(ss\_y)

sum(ss\_yv)

Submit

You have used 1 of 10 attemptsSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

SaveSave Your Answer Show Answer

Answers are displayed within the problem

Review

### **Q4**

1/1 point (graded)

We see that the total sum of squares is preserved. This is because V is orthogonal. Now to start understanding how YV is useful, plot ss\_y against the column number and then do the same for ss\_yv.

What do you observe?

ss\_y and ss\_yv are decreasing and close to 0 for the 4th column and beyond.

ss\_yv is decreasing and close to 0 for the 4th column and beyond.

ss\_y is decreasing and close to 0 for the 4th column and beyond.

There is no discernible pattern to either ss\_y or ss\_yv.

correct

Submit

You have used 1 of 2 attemptsSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

SaveSave Your Answer Show Answer

Correct (1/1 point)

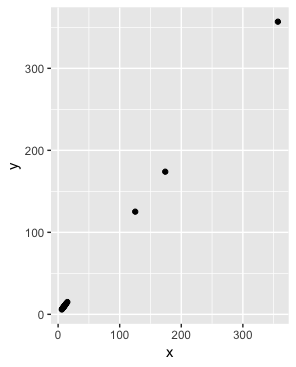
Review

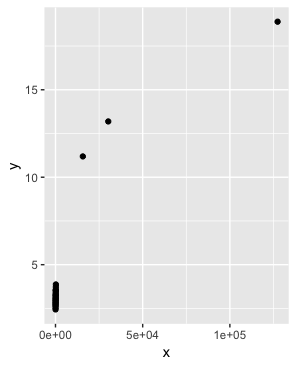
### **Q5**

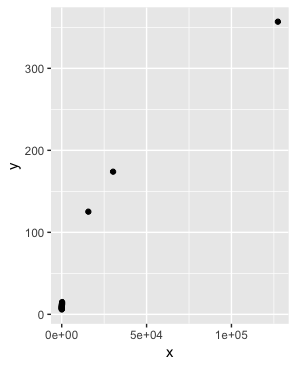
1/1 point (graded)

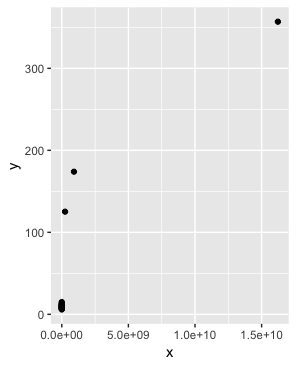
Now notice that we didn't have to compute ss\_yv because we already have the answer. How? Remember that YV=UD and because U is orthogonal, we know that the sum of squares of the columns of UD are the diagonal entries of D squared. Confirm this by plotting the square root of ss\_yv versus the diagonal entries of D.

Which of these plots is correct?









correct

Submit

You have used 1 of 2 attemptsSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

SaveSave Your Answer Show Answer

Correct (1/1 point)

Review

### **Q6**

1/1 point (graded)

So from the above we know that the sum of squares of the columns of Y (the total sum of squares) adds up to the sum of s$d^2 and that the transformation YV gives us columns with sums of squares equal to s$d^2. Now compute the percent of the total variability that is explained by just the first three columns of YV.

What proportion of the total variability is explained by the first three columns of YV?

Enter a decimal, **not** the percentage.

  correct

0.99 Loading

Submit

You have used 1 of 10 attemptsSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

SaveSave Your Answer Show Answer

Correct (1/1 point)

Review

### **Q7**

1/1 point (graded)

Before we continue, let's show a useful computational trick to avoid creating the matrix diag(s$d). To motivate this, we note that if we write U out in its columns [U1,U2,…,Up] then UD is equal to

UD=[U1d1,1,U2d2,2,…,Updp,p]

Use the sweep function to compute UD without constructing diag(s$d) or using matrix multiplication.

Which code is correct?



identical(t(s$u %\*% diag(s$d)), sweep(s$u, 2, s$d, FUN = "\*"))



identical(s$u %\*% diag(s$d), sweep(s$u, 2, s$d, FUN = "\*"))



identical(s$u %\*% t(diag(s$d)), sweep(s$u, 2, s$d, FUN = "\*"))



identical(s$u %\*% diag(s$d), sweep(s$u, 2, s, FUN = "\*"))

correct

Submit

You have used 1 of 2 attemptsSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

SaveSave Your Answer Show Answer

Correct (1/1 point)

Review

### **Q8**

0/1 point (graded)

We know that U1d1,1, the first column of UD, has the most variability of all the columns of UD. Earlier we looked at an image of Y using my\_image(y), in which we saw that the student to student variability is quite large and that students that are good in one subject tend to be good in all. This implies that the average (across all subjects) for each student should explain a lot of the variability. Compute the average score for each student, plot it against U1d1,1, and describe what you find.

What do you observe?

There is no relationship between the average score for each student and U1d1,1.

There is a linearly decreasing relationship between the average score for each student and U1d1,1.

There is a linearly increasing relationship between the average score for each student and U1d1,1. correct

There is an exponentially increasing relationship between the average score for each student and U1d1,1.

There is an exponentially decreasing relationship between the average score for each student and U1d1,1.

incorrect

**Explanation**

You can generate the plot using plot(-s$u[,1]\*s$d[1], rowMeans(y)).

Submit

You have used 2 of 2 attemptsSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

Show Answer

Answers are displayed within the problem

Review

### **Q9**

1/1 point (graded)

We note that the signs in SVD are arbitrary because:

UDV⊤=(−U)D(−V)⊤

With this in mind we see that the first column of UD is almost identical to the average score for each student except for the sign.

This implies that multiplying Y by the first column of V must be performing a similar operation to taking the average. Make an image plot of V and describe the first column relative to others and how this relates to taking an average.

How does the first column relate to the others, and how does this relate to taking an average?

The first column is very variable, which implies that the first column of YV is the sum of the rows of Y multiplied by some non-constant function, and is thus not proportional to an average.

The first column is very variable, which implies that the first column of YV is the sum of the rows of Y multiplied by some non-constant function, and is thus proportional to an average.

The first column is very close to being a constant, which implies that the first column of YV is the sum of the rows of Y multiplied by some constant, and is thus proportional to an average.

The first three columns are all very close to being a constant, which implies that these columns are the sum of the rows of Y multiplied by some constant, and are thus proportional to an average.

correct

Submit

You have used 2 of 2 attemptsSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

Show Answer

Correct (1/1 point)

Review

The following four exercises are all **ungraded** and are provided to give you an additional opportunity to practice working with matrices in a continuation of the exercises with this dataset.

We recommend that you attempt to write the code on your own **before** hitting "submit" and viewing the answers.

### **Q10 - UNGRADED**

0 points possible (ungraded)

We already saw that we can rewrite UD as

U1d1,1+U2d2,2+⋯+Updp,p

with Uj the j-th column of U. This implies that we can rewrite the entire SVD as:

Y=U1d1,1V⊤1+U2d2,2V⊤2+⋯+Updp,pV⊤p

with Vj the jth column of V. Plot U1, then plot V⊤1 using the same range for the y-axis limits, then make an image of U1d1,1V⊤1 and compare it to the image of Y. Hint: use the my\_image() function defined above. Use the drop=FALSE argument to assure the subsets of matrices are matrices.

**Explanation**

The plot can be made using the following code:

plot(s$u[,1], ylim = c(-0.25, 0.25))

plot(s$v[,1], ylim = c(-0.25, 0.25))

with(s, my\_image((u[, 1, drop=FALSE]\*d[1]) %\*% t(v[, 1, drop=FALSE])))

my\_image(y)

Submit

You have used 1 of 1 attemptSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

Show Answer

Answers are displayed within the problem

Review

### **Q11 - UNGRADED**

0 points possible (ungraded)

We see that with just a vector of length 100, a scalar, and a vector of length 24, we can actually come close to reconstructing the a 100×24 matrix. This is our first matrix factorization:

Y≈d1,1U1V⊤1

In the exercise in Q6, we saw how to calculate the percent of total variability explained. However, our approximation only explains the observation that good students tend to be good in all subjects. Another aspect of the original data that our approximation does not explain was the higher similarity we observed within subjects. We can see this by computing the difference between our approximation and original data and then computing the correlations. You can see this by running this code:

resid <- y - with(s,(u[, 1, drop=FALSE]\*d[1]) %\*% t(v[, 1, drop=FALSE]))

my\_image(cor(resid), zlim = c(-1,1))

axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)

Now that we have removed the overall student effect, the correlation plot reveals that we have not yet explained the within subject correlation nor the fact that math and science are closer to each other than to the arts. So let's explore the second column of the SVD.

Repeat the previous exercise (Q10) but for the second column: Plot U2, then plot V⊤2 using the same range for the y-axis limits, then make an image of U2d2,2V⊤2 and compare it to the image of resid.

Submit

You have used 0 of 1 attemptSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

SaveSave Your Answer Show Answer

### **Q12 - UNGRADED**

0 points possible (ungraded)

The second column clearly relates to a student's difference in ability in math/science versus the arts. We can see this most clearly from the plot of s$v[,2]. Adding the matrix we obtain with these two columns will help with our approximation:

Y≈d1,1U1V⊤1+d2,2U2V⊤2

We know it will explain sum(s$d[1:2]^2)/sum(s$d^2) \* 100 percent of the total variability. We can compute new residuals like this:

resid <- y - with(s,sweep(u[, 1:2], 2, d[1:2], FUN="\*") %\*% t(v[, 1:2]))

my\_image(cor(resid), zlim = c(-1,1))

axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)

and see that the structure that is left is driven by the differences between math and science. Confirm this by first plotting U3, then plotting V⊤3 using the same range for the y-axis limits, then making an image of U3d3,3V⊤3 and comparing it to the image of resid.

Submit

You have used 0 of 1 attemptSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

SaveSave Your Answer Show Answer

### **Q13 - UNGRADED**

0 points possible (ungraded)

The third column clearly relates to a student's difference in ability in math and science. We can see this most clearly from the plot of s$v[,3]. Adding the matrix we obtain with these two columns will help with our approximation:

Y≈d1,1U1V⊤1+d2,2U2V⊤2+d3,3U3V⊤3

We know it will explain: sum(s$d[1:3]^2)/sum(s$d^2) \* 100 percent of the total variability. We can compute new residuals like this:

resid <- y - with(s,sweep(u[, 1:3], 2, d[1:3], FUN="\*") %\*% t(v[, 1:3]))

my\_image(cor(resid), zlim = c(-1,1))

axis(side = 2, 1:ncol(y), rev(colnames(y)), las = 2)

We no longer see structure in the residuals: they seem to be independent of each other. This implies that we can describe the data with the following model:

Y=d1,1U1V⊤1+d2,2U2V⊤2+d3,3U3V⊤3+ε

with ε a matrix of independent identically distributed errors. This model is useful because we summarize of 100×24 observations with 3×(100+24+1)=375 numbers.

Furthermore, the three components of the model have useful interpretations:

1 - the overall ability of a student

2 - the difference in ability between the math/sciences and arts

3 - the remaining differences between the three subjects.

The sizes d1,1,d2,2 and d3,3 tell us the variability explained by each component. Finally, note that the components dj,jUjV⊤j are equivalent to the jth principal component.

Finish the exercise by plotting an image of Y, an image of d1,1U1V⊤1+d2,2U2V⊤2+d3,3U3V⊤3 and an image of the residuals, all with the same zlim.

Submit

You have used 0 of 1 attemptSome problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

------------------------------------------------------------------------------------------------------------------