Section 1: Classification: Filtering spam

```
# The emails dataset is already loaded into your workspace

# Show the dimensions of emails
dim(emails)

# Inspect definition of spam_classifier()
spam_classifier <- function(x){
  prediction <- rep(NA, length(x)) # initialize prediction vector
  prediction[x > 4] <- 1
  prediction[x >= 3 & x <= 4] <- 0
  prediction[x >= 2.2 & x < 3] <- 1
  prediction[x >= 1.4 & x < 2.2] <- 0
  prediction[x > 1.25 & x < 1.4] <- 1
  prediction[x <= 1.25] <- 0
  return(prediction) # prediction is either 0 or 1
}

# Apply the classifier to the avg_capital_seq column: spam_pred
spam_pred <- sapply(emails$avg_capital_seq,spam_classifier)

# Compare spam_pred to emails$spam. Use ==
spam_pred == emails$spam
```

Regression: LinkedIn views for the next 3 days

```
# linkedin is already available in your workspace
# Create the days vector
days <- seq(1,21,1)

# Fit a linear model called on the linkedin views per day: linkedin_lm
linkedin_lm <- lm(linkedin ~ days)

# Predict the number of views for the next three days: linkedin_pred
future_days <- data.frame(days = 22:24)
linkedin_pred <- predict(linkedin_lm,future_days)

# Plot historical data and predictions
plot(linkedin ~ days, xlim = c(1, 24))
points(22:24, linkedin_pred, col = "green")
```

Clustering: Separating the iris species

```
# Set random seed. Don't remove this line.
set.seed(1)

# Chop up iris in my_iris and species
my_iris <- iris[-5]
species <- iris$Species

# Perform k-means clustering on my_iris: kmeans_iris
kmeans_iris <- kmeans(my_iris,3)

# Compare the actual Species to the clustering using table()
table(species,kmeans_iris$cluster)
```

```
# Plot Petal.Width against Petal.Length, coloring by cluster

plot(Petal.Length ~ Petal.Width, data = my_iris, col = kmeans_iris$cluster)
```

Getting practical with supervised learning

```
# Set random seed. Don't remove this line.

set.seed(1)


# Take a look at the iris dataset

str(iris)

summary(iris)



# A decision tree model has been built for you

tree <- rpart(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,

            data = iris, method = "class")


# A dataframe containing unseen observations

unseen <- data.frame(Sepal.Length = c(5.3, 7.2),

                  Sepal.Width = c(2.9, 3.9),

                  Petal.Length = c(1.7, 5.4),

                  Petal.Width = c(0.8, 2.3))


# Predict the label of the unseen observations. Print out the result.


predict(tree,unseen,type="class")
```

How to do unsupervised learning (1)

```
# The cars data frame is pre-loaded


# Set random seed. Don't remove this line.

set.seed(1)


# Explore the cars dataset

str(cars)

summary(cars)



# Group the dataset into two clusters: km_cars

km_cars <- kmeans(cars,2)


# Print out the contents of each cluster

km_cars$cluster
```

How to do unsupervised learning (2)

```
# The cars data frame is pre-loaded


# Set random seed. Don't remove this line

set.seed(1)


# Group the dataset into two clusters: km_cars

km_cars <- kmeans(cars, 2)


# Add code: color the points in the plot based on the clusters
```

```
plot(cars,col=km_cars$cluster)


# Print out the cluster centroids

km_cars$centers


# Replace the ___ part: add the centroids to the plot

points(2.692000,99.47368, pch = 22, bg = c(1, 2), cex = 2)


points(3.984923,215.69231, pch = 22, bg = c(1, 2), cex = 2)
```

(1) Identify a face on a list of Facebook photos. You can train your system on tagged Facebook pictures. (2) Given some features, predict whether a fruit has gone bad or not. Several supermarkets provided you with their previous observations and results. (3) Group DataCamp students into three groups. Students within the same group should be similar, while those in different groups must be dissimilar.

Section 2: Measuring model performance or error

The Confusion Matrix

```
# The titanic dataset is already loaded into your workspace

# Set random seed. Don't remove this line

set.seed(1)


# Have a look at the structure of titanic

str(titanic)


# A decision tree classification model is built on the data

tree <- rpart(Survived ~ ., data = titanic, method = "class")

# Use the predict() method to make predictions, assign to pred
```

```
pred <- predict(tree,titanic,type="class")


# Use the table() method to make the confusion matrix
table(titanic$Survived,pred)
```

Deriving ratios from the Confusion Matrix

```
# The confusion matrix is available in your workspace as conf


# Assign TP, FN, FP and TN using conf
TP <- conf[1, 1] # this will be 212
FN <- conf[1, 2] # this will be 78
FP <- conf[2,1] # fill in
TN <- conf[2,2] # fill in


# Calculate and print the accuracy: acc
acc <- (TP+TN)/(TP+FN+FP+TN)
acc


# Calculate and print out the precision: prec
prec <- TP/(TP+FP)
prec



# Calculate and print out the recall: rec
rec <- TP/(TP+FN)
rec
```

The quality of a regression

```
# The air dataset is already loaded into your workspace
# Take a look at the structure of air
str(air)

# Inspect your colleague's code to build the model
fit <- lm(dec ~ freq + angle + ch_length, data = air)

# Use the model to predict for all values: pred
pred <- predict(fit,air)

# Use air$dec and pred to calculate the RMSE
rmse <- sqrt((1/nrow(air)) * sum( (air$dec - pred) ^ 2))

# Print out rmse
Rmse
```

Adding complexity to increase quality

```
# The air dataset is already loaded into your workspace

# Previous model
fit <- lm(dec ~ freq + angle + ch_length, data = air)
pred <- predict(fit)
rmse <- sqrt(sum( (air$dec - pred) ^ 2) / nrow(air))
rmse

# Your colleague's more complex model
fit2 <- lm(dec ~ freq + angle + ch_length + velocity + thickness, data = air)
```

# Use the model to predict for all values: pred2

pred2 <- predict(fit2)

# Calculate rmse2

rmse2 <- sqrt(sum((air$dec - pred2) ^ 2) / nrow(air))

# Print out rmse2

rmse2

Split the sets

# The titanic dataset is already loaded into your workspace

# Set random seed. Don't remove this line.

set.seed(1)

# Shuffle the dataset, call the result shuffled

n <- nrow(titanic)

shuffled <- titanic[sample(n),]

train_indices <- 1:round(0.7 * n)

train <- shuffled[train_indices,]

test_indices <- (round(0.7 * n) + 1):n

test <- shuffled[test_indices,]

# Split the data in train and test

# Print the structure of train and test

str(train)

```
str(test)
```

First you train, then you test

```
# The titanic dataset is already loaded into your workspace


# Set random seed. Don't remove this line.

set.seed(1)


# Shuffle the dataset; build train and test

n <- nrow(titanic)

shuffled <- titanic[sample(n),]

train <- shuffled[1:round(0.7 * n),]

test <- shuffled[(round(0.7 * n) + 1):n,]


# Fill in the model that has been learned.

tree <- rpart(Survived ~ ., train, method = "class")


# Predict the outcome on the test set with tree: pred

pred <- predict(tree,test,type = "class")


# Calculate the confusion matrix: conf

conf <- table(test$Survived,pred)


# Print this confusion matrix

Conf
```

Using Cross Validation

```
# The shuffled dataset is already loaded into your workspace
# Set random seed. Don't remove this line.
set.seed(1)

# Initialize the accs vector
accs <- rep(0,6)

for (i in 1:6) {
  # These indices indicate the interval of the test set
  indices <- (((i-1) * round((1/6)*nrow(shuffled))) + 1):((i*round((1/6) * nrow(shuffled))))

  # Exclude them from the train set
  train <- shuffled[-indices,]

  # Include them in the test set
  test <- shuffled[indices,]

  # A model is learned using each training set
  tree <- rpart(Survived ~ ., train, method = "class")

  # Make a prediction on the test set using tree
pred <- predict(tree,test,type="class")

  # Assign the confusion matrix to conf
conf <- table(test$Survived,pred)

  # Assign the accuracy of this model to the ith index in accs
  accs[i] <- sum(diag(conf))/sum(conf)
}
```

```
# Print out the mean of accs

mean_accs <- mean(accs)

mean_accs
```

Overfitting the spam!

```
# The spam filter that has been 'learned' for you

spam_classifier <- function(x){

  prediction <- rep(NA, length(x)) # initialize prediction vector

  prediction[x > 4] <- 1

  prediction[x >= 3 & x <= 4] <- 0

  prediction[x >= 2.2 & x < 3] <- 1

  prediction[x >= 1.4 & x < 2.2] <- 0

  prediction[x > 1.25 & x < 1.4] <- 1

  prediction[x <= 1.25] <- 0

  return(factor(prediction, levels = c("1", "0"))) # prediction is either 0 or 1

}


# Apply spam_classifier to emails_full: pred_full

pred_full <-sapply(emails_full$avg_capital_seq,spam_classifier)


# Build confusion matrix for emails_full: conf_full

conf_full <- table(emails_full$spam,pred_full)


# Calculate the accuracy with conf_full: acc_full

acc_full <- sum(diag(conf_full))/sum(conf_full)


# Print acc_full

acc_full
```

Increasing the bias

```
# The all-knowing classifier that has been learned for you

# You should change the code of the classifier, simplifying it

spam_classifier <- function(x){

  prediction <- rep(NA, length(x))

  prediction[x > 4] <- 1

  prediction[x <= 4] <- 0

  return(factor(prediction, levels = c("1","0")))

}


# conf_small and acc_small have been calculated for you

conf_small <- table(emails_small$spam, spam_classifier(emails_small$avg_capital_seq))

acc_small <- sum(diag(conf_small)) / sum(conf_small)

acc_small


# Apply spam_classifier to emails_full and calculate the confusion matrix: conf_full

conf_full <- table(emails_full$spam, spam_classifier(emails_full$avg_capital_seq))


# Calculate acc_full

acc_full <- sum(diag(conf_full)) / sum(conf_full)


# Print acc_full

acc_full
```

Section 3: Classification

Learn a decision tree

```
# The train and test set are loaded into your workspace.

# Set random seed. Don't remove this line
set.seed(1)

# Load the rpart, rattle, rpart.plot and RColorBrewer package
library(rpart)
library(rattle)
library(rpart.plot)
library(RColorBrewer)


# Fill in the ___, build a tree model: tree
tree <- rpart(Survived ~ ., train, method="class")

# Draw the decision tree
fancyRpartPlot(tree)
```

Classify with the decision tree

```
# The train and test set are loaded into your workspace.

# Code from previous exercise
set.seed(1)
library(rpart)
tree <- rpart(Survived ~ ., train, method = "class")
```

```
# Predict the values of the test set: pred
pred <- predict(tree,test,type="class")

# Construct the confusion matrix: conf
conf <- table(test$Survived,pred)

# Print out the accuracy
acc <- sum(diag(conf))/sum(conf)
acc
```

Pruning the tree

```
# All packages are pre-loaded, as is the data

# Calculation of a complex tree
set.seed(1)
tree <- rpart(Survived ~ ., train, method = "class", control = rpart.control(cp=0.00001))



# Draw the complex tree
fancyRpartPlot(tree)

# Prune the tree: pruned
pruned <- prune(tree,cp=0.01)

# Draw pruned
fancyRpartPlot(pruned)
```

Splitting criterion

```
# All packages, emails, train, and test have been pre-loaded
# Set random seed. Don't remove this line.
set.seed(1)

# Train and test tree with gini criterion
tree_g <- rpart(spam ~ ., train, method = "class")
pred_g <- predict(tree_g, test, type = "class")
conf_g <- table(test$spam, pred_g)
acc_g <- sum(diag(conf_g)) / sum(conf_g)

# Change the first line of code to use information gain as splitting criterion
tree_i <- rpart(spam ~ ., train, method = "class", parms = list(split = "information"))
pred_i <- predict(tree_i, test, type = "class")
conf_i <- table(test$spam, pred_i)
acc_i <- sum(diag(conf_i)) / sum(conf_i)

# Draw a fancy plot of both tree_g and tree_i
fancyRpartPlot(tree_g)
fancyRpartPlot(tree_i)



# Print out acc_g and acc_i
acc_g
acc_i
```

Learn a KNN

Preprocess the data

```
# train and test are pre-loaded
# Store the Survived column of train and test in train_labels and test_labels

train_labels <- train$Survived

test_labels <- test$Survived


# Copy train and test to knn_train and knn_test

knn_train <- train

knn_test <- test


# Drop Survived column for knn_train and knn_test

knn_train$Survived <- NULL

knn_test$Survived <- NULL


# normalize Pclass

min_class <- min(knn_train$Pclass)

max_class <- max(knn_train$Pclass)

knn_train$Pclass <- (knn_train$Pclass - min_class) / (max_class - min_class)

knn_test$Pclass <- (knn_test$Pclass - min_class) / (max_class - min_class)


# normalize Age

min_age <- min(knn_train$Age)

max_age <- max(knn_train$Age)

knn_train$Age <- (knn_train$Age - min_age) / (max_age - min_age)

knn_test$Age <- (knn_test$Age - min_age) / (max_age - min_age)
```

K's choice

```
# knn_train, knn_test, train_labels and test_labels are pre-loaded


# Set random seed. Don't remove this line.
```

```
set.seed(1)

# Load the class package, define range and accs
library(class)
range <- 1:round(0.2 * nrow(knn_train))
accs <- rep(0, length(range))

for (k in range) {

  # Fill in the ___, make predictions using knn: pred
  pred <- knn(train = knn_train, test = knn_test, cl = train_labels, k = k)

  # Fill in the ___, construct the confusion matrix: conf
  conf <- table(test_labels, pred)

  # Fill in the ___, calculate the accuracy and store it in accs[k]
  accs[k] <- sum(diag(conf))/sum(conf)
}

# Plot the accuracies. Title of x-axis is "k".
plot(range, accs, xlab = "k")

# Calculate the best k
which.max(accs)
```

The ROC Curve

Creating the ROC curve (1)

```
# train and test are pre-loaded
# Set random seed. Don't remove this line

set.seed(1)


# Build a tree on the training set: tree

tree <- rpart(income ~ ., train, method = "class")


# Predict probability values using the model: all_probs

all_probs <- predict(tree,test,type="prob")


# Print out all_probs

all_probs


# Select second column of all_probs: probs

probs <- all_probs[,2]
```

Creating the ROC curve (2)

```
# train and test are pre-loaded


# Code of previous exercise

set.seed(1)

tree <- rpart(income ~ ., train, method = "class")

probs <- predict(tree, test, type = "prob")[,2]


# Load the ROCR library

library(ROCR)


# Make a prediction object: pred
```

```
pred <- prediction(probs,test$income)

# Make a performance object: perf
perf <- performance(pred,"tpr","fpr")

# Plot this curve
plot(perf)
```

The area under the curve

```
# test and train are loaded into your workspace

# Build tree and predict probability values for the test set
set.seed(1)
tree <- rpart(income ~ ., train, method = "class")
probs <- predict(tree, test, type = "prob")[,2]

# Load the ROCR library
library(ROCR)

# Make a prediction object: pred
pred <- prediction(probs,test$income)

# Make a performance object: perf
perf <- performance(pred,"tpr","fpr")

# Print out the AUC
perf@y.values[[1]]
```

Comparing the methods

# Load the ROCR library

library(ROCR)

# Make the prediction objects for both models: pred_t, pred_k

pred_t <- prediction(probs_t,test$spam)

pred_k <- prediction(probs_k,test$spam)

# Make the performance objects for both models: perf_t, perf_k

perf_t <- performance(pred_t,"tpr","fpr")

perf_k <- performance(pred_k,"tpr","fpr")

# Draw the ROC lines using draw_roc_lines()

draw_roc_lines(perf_t,perf_k)

Section 4: Regression

Simple linear regression: your first step!

# The kang_nose dataset and nose_width_new are already loaded in your workspace.

# Plot nose length as function of nose width.

plot(kang_nose, xlab = "nose width", ylab = "nose length")

# Fill in the ___, describe the linear relationship between the two variables: lm_kang

lm_kang <- lm(nose_length ~ nose_width, data = kang_nose)

# Print the coefficients of lm_kang

lm_kang$coefficients

# Predict and print the nose length of the escaped kangoroo

```
pred <- predict(lm_kang,nose_width_new)
pred
```

Performance measure: RMSE

```
# kang_nose is pre-loaded in your workspace

# Build model and make plot
lm_kang <- lm(nose_length ~ nose_width, data=kang_nose)
plot(kang_nose, xlab = "nose width", ylab = "nose length")
abline(lm_kang$coefficients, col = "red")

# Apply predict() to lm_kang: nose_length_est
nose_length_est <- predict(lm_kang)

# Calculate difference between the predicted and the true values: res
res <- kang_nose$nose_length - nose_length_est

# Calculate RMSE, assign it to rmse and print it
rmse <- sqrt((1/nrow(kang_nose)) * sum( (kang_nose$nose_length - nose_length_est) ^ 2))
rmse
```

Performance measures: R-squared

```
# kang_nose, lm_kang and res are already loaded in your workspace

# Calculate the residual sum of squares: ss_res
ss_res <- sum( (res) ^ 2)

# Determine the total sum of squares: ss_tot
mn <- mean(kang_nose$nose_length)
```

```
ss_tot <- sum((kang_nose$nose_length - mn)^2)

# Calculate R-squared and assign it to r_sq. Also print it.
r_sq <- 1 - (ss_res/ss_tot)
r_sq

# Apply summary() to lm_kang
summary(lm_kang)
```

Another take at regression: be critical

```
# world_bank_train and cgdp_afg is available for you to work with

# Plot urb_pop as function of cgdp
plot(world_bank_train, xlab = "cgdp", ylab = "urb_pop")

# Set up a linear model between the two variables: lm_wb
lm_wb <- lm(urb_pop ~ cgdp, data = world_bank_train)

# Add a red regression line to your scatter plot
abline(lm_wb$coefficients, col = "red")

# Summarize lm_wb and select R-squared
summary(lm_wb)$r.squared

# Predict the urban population of afghanistan based on cgdp_afg
predict(lm_wb,cgdp_afg)
```

Non-linear, but still linear?

```
# world_bank_train and cgdp_afg is available for you to work with
# Plot: change the formula and xlab
plot(urb_pop ~ log(cgdp), data = world_bank_train,
    xlab = "log(GDP per Capita)",
    ylab = "Percentage of urban population")

# Linear model: change the formula
lm_wb <- lm(urb_pop ~ log(cgdp), data = world_bank_train)

# Add a red regression line to your scatter plot
abline(lm_wb$coefficients, col = "red")

# Summarize lm_wb and select R-squared
summary(lm_wb)$r.squared

# Predict the urban population of afghanistan based on cgdp_afg
predict(lm_wb,cgdp_afg)
```

Going all-in with predictors!

```
        # shop_data has been loaded in your workspace

# Add a plot: sales as a function of inventory. Is linearity plausible?
plot(sales ~ sq_ft, shop_data)
plot(sales ~ size_dist, shop_data)
plot(sales ~ inv, shop_data)

# Build a linear model for net sales based on all other variables: lm_shop
lm_shop <- lm(sales ~ . , data=shop_data)
```

# Summarize lm_shop

summary(lm_shop)

Are all predictors relevant?

# shop_data, shop_new and lm_shop have been loaded in your workspace

# Plot the residuals in function of your fitted observations

plot(lm_shop$fitted.values,lm_shop$residuals)

# Make a Q-Q plot of your residual quantiles

qqnorm(lm_shop$residuals,ylab = "Residual Quantiles")

# Summarize your model, are there any irrelevant predictors?

summary(lm_shop)

# Predict the net sales based on shop_new.

predict(lm_shop,shop_new)

Are all predictors relevant? Take 2!

# choco_data has been loaded in your workspace

# Add a plot: energy/100g as function of total size. Linearity plausible?

plot(energy ~ protein, choco_data)

plot(energy ~ fat, choco_data)

plot(energy ~ size, choco_data)

```
# Build a linear model for the energy based on all other variables: lm_choco

lm_choco <- lm(energy ~ ., data=choco_data)


# Plot the residuals in function of your fitted observations

plot(lm_choco$fitted.values,lm_choco$residuals)


# Make a Q-Q plot of your residual quantiles

qqnorm(lm_choco$residuals,ylab = "Residual Quantiles")


# Summarize lm_choco

summary(lm_choco)
```

K-Nearest Neighbors and Generalization

Does your model generalize?

```
# world_bank_train, world_bank_test and lm_wb_log are pre-loaded


# Build the log-linear model

lm_wb_log <- lm(urb_pop ~ log(cgdp), data = world_bank_train)


# Calculate rmse_train

rmse_train <- sqrt(mean(lm_wb_log$residuals ^ 2))


# The real percentage of urban population in the test set, the ground truth

world_bank_test_truth <- world_bank_test$urb_pop


# The predictions of the percentage of urban population in the test set

world_bank_test_input <- data.frame(cgdp = world_bank_test$cgdp)
```

```
world_bank_test_output <- predict(lm_wb_log, world_bank_test_input)

# The residuals: the difference between the ground truth and the predictions
res_test <- world_bank_test_output - world_bank_test_truth


# Use res_test to calculate rmse_test
rmse_test <- sqrt(mean(res_test ^ 2))

# Print the ratio of the test RMSE over the training RMSE
rmse_test/rmse_train
```

Your own k-NN algorithm!

```
###
# You don't have to change this!
# The algorithm is already coded for you;
# inspect it and try to understand how it works!
my_knn <- function(x_pred, x, y, k){
  m <- length(x_pred)
  predict_knn <- rep(0, m)
  for (i in 1:m) {

    # Calculate the absolute distance between x_pred[i] and x
    dist <- abs(x_pred[i] - x)

    # Apply order() to dist, sort_index will contain
    # the indices of elements in the dist vector, in
    # ascending order. This means sort_index[1:k] will
    # return the indices of the k-nearest neighbors.
    sort_index <- order(dist)

    # Apply mean() to the responses of the k-nearest neighbors
    predict_knn[i] <- mean(y[sort_index[1:k]])

  }
  return(predict_knn)
}
###

# world_bank_train and world_bank_test are pre-loaded

# Apply your algorithm on the test set: test_output
test_output <- my_knn(world_bank_test$cgdp,world_bank_train$cgdp,world_bank_train$urb_pop,30)

# Have a look at the plot of the output
plot(world_bank_train,
     xlab = "GDP per Capita",
     ylab = "Percentage Urban Population")
points(world_bank_test$cgdp, test_output, col = "green")
```

Parametric vs non-parametric!

```
# world_bank_train, world_bank_test is pre-loaded
# lm_wb and lm_wb_log have been trained on world_bank_train
# The my_knn() function is available

# Define ranks to order the predictor variables in the test set
```

```
ranks <- order(world_bank_test$cgdp)

# Scatter plot of test set
plot(world_bank_test,
     xlab = "GDP per Capita", ylab = "Percentage Urban Population")

# Predict with simple linear model and add line
test_output_lm <- predict(lm_wb, data.frame(cgdp = world_bank_test$cgdp))
lines(world_bank_test$cgdp[ranks], test_output_lm[ranks], lwd = 2, col = "blue")

# Predict with log-linear model and add line
test_output_lm_log <- predict(lm_wb_log, data.frame(cgdp = world_bank_test$cgdp))
lines(world_bank_test$cgdp[ranks], test_output_lm_log[ranks], lwd = 2, col = "red")

# Predict with k-NN and add line
test_output_knn <- my_knn(world_bank_test$cgdp, world_bank_train$cgdp,
world_bank_train$urb_pop, 30)
lines(world_bank_test$cgdp[ranks], test_output_knn[ranks], lwd = 2, col = "green")

# Calculate RMSE for simple linear model
sqrt(mean( (test_output_lm - world_bank_test$urb_pop) ^ 2))

# Calculate RMSE for log-linear model
sqrt(mean( (test_output_lm_log - world_bank_test$urb_pop) ^ 2))

# Calculate RMSE for k-NN technique
sqrt(mean( (test_output_knn - world_bank_test$urb_pop) ^ 2))
```

Section 5: Clustering

K-means: how well did you do earlier?

```
# seeds and seeds_type are pre-loaded in your workspace


# Set random seed. Don't remove this line.

set.seed(100)


# Do k-means clustering with three clusters, repeat 20 times: seeds_km

seeds_km <- kmeans(seeds, centers = 3, nstart = 20)


# Print out seeds_km

seeds_km


# Compare clusters with actual seed types. Set k-means clusters as rows

table(seeds_km$cluster, seeds_type)


# Plot the length as function of width. Color by cluster

plot(seeds$width, seeds$length, xlab = "Length", ylab = "Width", col = seeds_km$cluster)
```

The influence of starting centroids

```
# seeds is pre-loaded in your workspace


# Set random seed. Don't remove this line.

set.seed(100)


# Apply kmeans to seeds twice: seeds_km_1 and seeds_km_2

seeds_km_1 <- kmeans(seeds, centers = 5, nstart = 1)

seeds_km_2 <- kmeans(seeds, centers = 5, nstart = 1)
```

```
# Return the ratio of the within cluster sum of squares

seeds_km_1$tot.withinss/seeds_km_2$tot.withinss


# Compare the resulting clusters

table(seeds_km_1$cluster, seeds_km_2$cluster)
```

Making a scree plot!

```
# The dataset school_result is pre-loaded


# Set random seed. Don't remove this line.

set.seed(100)


# Explore the structure of your data

str(school_result)


# Initialise ratio_ss

ratio_ss <- rep(0, 7)


# Finish the for-loop

for (k in 1:7) {


  # Apply k-means to school_result: school_km

  school_km <- kmeans(school_result, k, nstart = 20)


  # Save the ratio between of WSS to TSS in kth element of ratio_ss

  ratio_ss[k] <- school_km$tot.withinss / school_km$totss


}
```

```
# Make a scree plot with type "b" and xlab "k"

plot(ratio_ss, type = "b", xlab = "k")
```

What is your optimal k?

You want to choose `k` such that your clusters are compact and well separated. However, the`ratio_ss` keeps decreasing as `k` increases. Hence, if you were to minimize this ratio as function of `k`, you'd end up with a cluster for every school, which is a meaningless result. You should choose `k` such that when increasing it, the impact on `ratio_ss` is not significant. The elbow in the scree plot will help you identify this turning point.
Can you tell which of the following values of `k` will provide a meaningful clustering with overall compact and well separated clusters? The ratio, `ratio_ss`, is still in your workspace.

Performance and scaling issues
Standardized vs non-standardized clustering (1)

```
# The dataset run_record has been loaded in your workspace


# Set random seed. Don't remove this line.

set.seed(1)


# Explore your data with str() and summary()

str(run_record)

summary(run_record)


# Cluster run_record using k-means: run_km. 5 clusters, repeat 20 times

run_km <- kmeans(run_record, 5, nstart = 20)


# Plot the 100m as function of the marathon. Color using clusters
```

```
plot(run_record$marathon, run_record$X100m, col = run_km$cluster,
     xlab = "marathon", ylab ="100m", main = "Run Records")

# Calculate Dunn's index: dunn_km. Print it.
dunn_km <- dunn(clusters = run_km$cluster, Data = run_record)
print(dunn_km)
```

Standardized vs non-standardized clustering (2)

```
# The dataset run_record as well as run_km are available

# Set random seed. Don't remove this line.
set.seed(1)

# Standardize run_record, transform to a dataframe: run_record_sc
run_record_sc <- as.data.frame(scale(run_record))

# Cluster run_record_sc using k-means: run_km_sc. 5 groups, let R start over 20 times
run_km_sc <- kmeans(run_record_sc, 5, nstart = 20)

# Plot records on 100m as function of the marathon. Color using the clusters in run_km_sc
plot(run_record$marathon, run_record$X100m, col = run_km_sc$cluster,
     xlab = "marathon", ylab ="100m", main = "Run Records")


# Compare the resulting clusters in a nice table
table(run_km$cluster,run_km_sc$cluster)

# Calculate Dunn's index: dunn_km_sc. Print it.
```

```
dunn_km_sc <- dunn(clusters = run_km_sc$cluster, Data = run_record_sc)

dunn_km_sc
```

Hierarchical Clustering

Single Hierarchical Clustering

```
# The dataset run_record_sc has been loaded in your workspace


# Apply dist() to run_record_sc: run_dist

run_dist <- dist(run_record_sc)


# Apply hclust() to run_dist: run_single

run_single <- hclust(run_dist,method="single")


# Apply cutree() to run_single: memb_single

memb_single <- cutree(run_single,k=5)


# Apply plot() on run_single to draw the dendrogram

plot(run_single)


# Apply rect.hclust() on run_single to draw the boxes

rect.hclust(run_single,k=5,border=2:6)
```

Complete Hierarchical Clustering

```
# run_record_sc is pre-loaded


# Code for single-linkage

run_dist <- dist(run_record_sc, method = "euclidean")

run_single <- hclust(run_dist, method = "single")
```

```
memb_single <- cutree(run_single, 5)

plot(run_single)

rect.hclust(run_single, k = 5, border = 2:6)


# Apply hclust() to run_dist: run_complete

run_complete <- hclust(run_dist, method = "complete")


# Apply cutree() to run_complete: memb_complete

memb_complete <- cutree(run_complete, 5)


# Apply plot() on run_complete to draw the dendrogram

plot(run_complete)


# Apply rect.hclust() on run_complete to draw the boxes

rect.hclust(run_complete, k = 5, border = 2:6)


# table() the clusters memb_single and memb_complete. Put memb_single in the rows

table(memb_single,memb_complete)
```

Hierarchical vs k-means

```
# run_record_sc, run_km_sc, memb_single and memb_complete are pre-calculated


# Set random seed. Don't remove this line.

set.seed(100)


# Dunn's index for k-means: dunn_km

dunn_km <- dunn(cluster=run_km_sc$cluster,Data=run_record_sc)


# Dunn's index for single-linkage: dunn_single
```

```
dunn_single <- dunn(cluster=memb_single,Data=run_record_sc)

# Dunn's index for complete-linkage: dunn_complete
dunn_complete <- dunn(cluster=memb_complete,Data=run_record_sc)

# Compare k-means with single-linkage
table(run_km_sc$cluster,memb_single)

# Compare k-means with complete-linkage
table(run_km_sc$cluster,memb_complete)
```

Interpreting Dunn's Index

Clustering US states based on criminal activity

```
# Set random seed. Don't remove this line.
set.seed(1)

# Scale the dataset: crime_data_sc
 crime_data_sc <- scale(crime_data)

# Perform k-means clustering: crime_km
crime_km <- kmeans(crime_data_sc,4,nstart=20)

# Perform single-linkage hierarchical clustering
## Calculate the distance matrix: dist_matrix
dist_matrix <- dist(crime_data_sc,method="euclidean")

## Calculate the clusters using hclust(): crime_single
crime_single <- hclust(dist_matrix, method = "single")
```

```
## Cut the clusters using cutree: memb_single
memb_single <- cutree(crime_single, 4)

# Calculate the Dunn's index for both clusterings: dunn_km, dunn_single
dunn_km <- dunn(cluster=crime_km$cluster,Data=crime_data_sc)
dunn_single <- dunn(cluster=memb_single,Data=crime_data_sc)


# Print out the results
dunn_km
dunn_single
```