Week 10 Exercise Solutions

Question 1

(a)

Q: Load **StocksCluster.csv** into a data frame called **stocks**. How many observations are in the dataset?

A:

```r
stocks <- read.csv("StocksCluster.csv")
nrow(stocks)
```

```
## [1] 11580
```

There are 11580 observations in the dataset.

(b)

Q: What proportion of the observations have positive returns in December?

A:

```r
(table(stocks$PositiveDec)/nrow(stocks))["1"]
```

```
##        1
## 0.546114
```

0.546114 of the observations have positive returns in December.

(c)

Q: What is the maximum correlation between any two return variables in the dataset? You should look at the pairwise correlations between `ReturnJan`, `ReturnFeb`, `ReturnMar`, `ReturnApr`, `ReturnMay`, `ReturnJune`, `ReturnJuly`, `ReturnAug`, `ReturnSep`, `ReturnOct`, and `ReturnNov`.

A: Assuming we are looking for the most positive correlation,

```
max(cor(stocks[,1:11]) - diag(rep(1, 11)))
```

```
## [1] 0.1916728
```

The maximum correlation between any two return variables in the dataset is 0.1916728.

Note that, were two variables perfectly negatively correlated, they would not have positive correlation. It is important to see if one is looking for the most positive correlation or greatest absolute correlation.

(d)

Q: Which month (from January through November) has the largest mean return across all observations in the dataset? Which month (from January through November) has the smallest mean return across all observations in the dataset?

A:

```
names(sort(colMeans(stocks[,1:11]), decreasing = TRUE)[1])  # largest
```

```
## [1] "ReturnApr"
```

```
names(sort(colMeans(stocks[,1:11]), decreasing = FALSE)[1])  # smallest
```

```
## [1] "ReturnSep"
```

April has the largest average return while September has the smallest (most negative) average return.

(e)

Q: Run the following commands to split the data into a training set and testing set, putting 70% of the data in the training set and 30% of the data in the testing set:

```
> set.seed(144)
> spl <- sample.split(stocks$PositiveDec, SplitRatio = 0.7)
> stocksTrain <- subset(stocks, spl == TRUE)
> stocksTest <- subset(stocks, spl == FALSE)
```

Then, use the `stocksTrain` data frame to train a logistic regression model (name it `StocksModel`) to predict `PositiveDec` using all the other variables as independent variables. What is the overall accuracy on the training set, using a threshold of 0.5?

A:

```r
set.seed(144)
library(caTools)
spl <- sample.split(stocks$PositiveDec, SplitRatio = 0.7)
stocksTrain <- subset(stocks, spl == TRUE)
stocksTest <- subset(stocks, spl == FALSE)
stocksModel <- glm(PositiveDec ~ ., data = stocksTrain,
                   family = "binomial")
stocksPredict <- predict(stocksModel, newdata = stocksTrain,
                         type = "response")
```

Given that we will be asked to get the accuracy a lot, it is easier to define a function now,

```r
accuracy <- function(predict_object, data, threshold=0.5) {
  return(sum(diag(table(predict_object >= threshold, data))) / length(data))
}
accuracy(stocksPredict, stocksTrain$PositiveDec)
```

```
## [1] 0.5711818
```

The overall accuracy of the model on the training set is 0.5711818.

(f)

Q: Now obtain test set predictions from `StocksModel`. What is the overall accuracy of the model on the test set, again using a threshold of 0.5?

A:

```r
stocksPredicttest <- predict(stocksModel, newdata = stocksTest,
                             type = "response")
accuracy(stocksPredicttest, stocksTest$PositiveDec)
```

```
## [1] 0.5670697
```

The overall accuracy of the model on the test set is 0.5670697.

(g)

Q: What is the accuracy on the test set of a baseline model that always predicts the most common outcome in the training set?

A:

```
most_common <- names(sort(table(stocksTrain$PositiveDec),
                          decreasing = TRUE)[1])
table(stocksTest$PositiveDec)[most_common]/nrow(stocksTest)
```

```
##         1
## 0.5460564
```

The accuracy on the test set of a baseline model that always predicts the most common outcome in the training set is 0.5460564.

(h)

Q: Now, let's cluster the stocks. The first step in this process is to remove the dependent variable using the following commands:

```
> limitedTrain <- stocksTrain
> limitedTrain$PositiveDec <- NULL
> limitedTest <- stocksTest
> limitedTest$PositiveDec <- NULL
```

Why do we need to remove the dependent variable in the clustering phase of the cluster-then-predict methodology?

i. Leaving in the dependent variable might lead to unbalanced clusters
ii. Removing the dependent variable decreases the computational effort needed to cluster
iii. Needing to know the dependent variable value to assign an observation to a cluster defeats the purpose of the methodology

A:

```
limitedTrain <- stocksTrain
limitedTrain$PositiveDec <- NULL
limitedTest <- stocksTest
limitedTest$PositiveDec <- NULL
```

We remove the dependent variable because needing to know the dependent variable value to assign an observation to a cluster defeats the purpose of the methodology.

(i)

Q: In some cases where we have a training and testing set, we might want to normalize by the mean and standard deviation of the variables in the training set. We can do this by using the `caret` package passing just the training set to the preProcess function, which normalizes variables by subtracting by the mean and dividing by the standard deviation.

```
> library(caret)
> preproc <- preProcess(limitedTrain)
> normTrain <- predict(preproc, limitedTrain)
> normTest <- predict(preproc, limitedTest)
```

What is the mean of the `ReturnJan` variable in `normTrain`?

What is the mean of the `ReturnJan` variable in `normTest`?

A:

```r
library(caret)
preproc <- caret::preProcess(limitedTrain)
normTrain <- predict(preproc, limitedTrain)
normTest <- predict(preproc, limitedTest)
mean(normTrain$ReturnJan)
```

```
## [1] 2.100586e-17
```

```r
mean(normTest$ReturnJan)
```

```
## [1] -0.0004185886
```

The mean of the `ReturnJan` variable in `normTrain` is 0.00000000000000002100586[1]. The mean of the `ReturnJan` variable in `normTest` is -0.0004185886.

(j)

Q: Why is the mean `ReturnJan` variable much closer to 0 in `normTrain` than in `normTest`?

i. Small rounding errors exist in the normalization procedure
ii. The distribution of the `ReturnJan` variable is different in the training and testing set
iii. The distribution of the dependent variable is different in the training and testing set

A: The distribution of the ReturnJan variable is different in the training and testing set.

[1] Note that comparison of this value has revealed differences as compared to running `colMeans(normTrain)["ReturnJan]`. This is likely due to floating point error. It is difficult to say without further evidence which function has greater precision, although `mean` should have greater precision.

(k)

Q: Set the random seed to 144 (it is important to do this again, even though we did it earlier). Run k-means clustering with 3 clusters on `normTrain`, storing the result in an object called `km`. Which cluster has the largest number of observations?

i. Cluster 1
ii. Cluster 2
iii. Cluster 3

A:

```
set.seed(144)
km <- kmeans(normTrain, centers = 3)
names(sort(table(km$cluster), decreasing = TRUE)[1])
```

```
## [1] "2"
```

Cluster 2 has the largest number of observations.

(l)

Q: In this question, we use the `flexclust` package to obtain training set and testing set cluster assignments for our observations and to do the predictions. Use the following commands:

```
> library(flexclust)
> km.kcca <- as.kcca(km, normTrain)
> clusterTrain <- predict(km.kcca)
> clusterTest <- predict(km.kcca, newdata=normTest)
```

How many test-set observations were assigned to Cluster 2?

A:

```
library(flexclust)
km.kcca <- as.kcca(km, normTrain)
clusterTrain <- predict(km.kcca)
clusterTest <- predict(km.kcca, newdata = normTest)
unname(table(clusterTest)["2"])
```

```
## [1] 2029
```

2029 test-set observations were assigned to Cluster 2.

(m)

Q: Using the subset function, build data frames `stocksTrain1`, `stocksTrain2`, and `stocksTrain3`, containing the elements in the `stocksTrain` data frame assigned to clusters 1, 2, and 3, respectively (be careful to take subsets of `stocksTrain`, not of `normTrain`). Similarly build `stocksTest1`, `stocksTest2`, and `stocksTest3` from the `stocksTest` data frame. Which training set data frame has the highest average value of the dependent variable?

A:

```r
# not recommended
# recommended way is to use a list to contain all models
stocksTrain1 <- subset(stocksTrain, clusterTrain == 1)
stocksTrain2 <- subset(stocksTrain, clusterTrain == 2)
stocksTrain3 <- subset(stocksTrain, clusterTrain == 3)
stocksTest1 <- subset(stocksTest, clusterTest == 1)
stocksTest2 <- subset(stocksTest, clusterTest == 2)
stocksTest3 <- subset(stocksTest, clusterTest == 3)
mean(stocksTrain1$PositiveDec)
```

```
## [1] 0.6103267
```

```r
mean(stocksTrain2$PositiveDec)
```

```
## [1] 0.5250476
```

```r
mean(stocksTrain3$PositiveDec)
```

```
## [1] 0.4799107
```

`stocksTrain1` has the observations with highest average value of the dependant variable.

(n)

Q: Build logistic regression models `StocksModel1`, `StocksModel2`, and `StocksModel3`, which predict `PositiveDec` using all the other variables as independent variables. `StocksModel1` should be trained on `stocksTrain1`, `StocksModel2` should be trained on `stocksTrain2`, and `StocksModel3` should be trained on `stocksTrain3`. Which variables have a positive sign for the coefficient in at least one of `StocksModel1`, `StocksModel2`, and `StocksModel3` and a negative sign for the coefficient in at least one of `StocksModel1`, `StocksModel2`, and `StocksModel3`?

A:

```r
StocksModel1 <- glm(PositiveDec ~ ., data = stocksTrain1, family = "binomial")
StocksModel2 <- glm(PositiveDec ~ ., data = stocksTrain2, family = "binomial")
StocksModel3 <- glm(PositiveDec ~ ., data = stocksTrain3, family = "binomial")
stock_returns <- colnames(stocksTrain)[1:11]
coef_mat <- cbind(coef(StocksModel1)[stock_returns],
                  coef(StocksModel2)[stock_returns],
                  coef(StocksModel3)[stock_returns])
names(which(!(apply(coef_mat > 0, 1, all) | apply(coef_mat < 0, 1, all))))
```

```
## [1] "ReturnFeb"  "ReturnMar"  "ReturnJuly" "ReturnSep"
```

The variables `ReturnFeb`, `ReturnMar`, `ReturnJuly` and `ReturnSep` have coefficients of at least one positive sign and a negative signs in the models.

(o)

Q: Using `StocksModel1`, make test-set predictions called `PredictTest1` on the data frame `stocksTest1`. Using `StocksModel2`, make test-set predictions called `PredictTest2` on the data frame `stocksTest2`. Using `StocksModel3`, make test-set predictions called `PredictTest3` on the data frame `stocksTest3`.

What is the overall accuracy of `StocksModel1` on the test set `stocksTest1`, using a threshold of 0.5?

What is the overall accuracy of `StocksModel2` on the test set `stocksTest2`, using a threshold of 0.5?

What is the overall accuracy of `StocksModel3` on the test set `stocksTest3`, using a threshold of 0.5?

A:

```r
PredictTest1 <- predict(StocksModel1, newdata = stocksTest1, type = "response")
PredictTest2 <- predict(StocksModel2, newdata = stocksTest2, type = "response")
PredictTest3 <- predict(StocksModel3, newdata = stocksTest3, type = "response")
accuracy(PredictTest1, stocksTest1$PositiveDec)
```

```
## [1] 0.6446125
```

```r
accuracy(PredictTest2, stocksTest2$PositiveDec)
```

```
## [1] 0.5367176
```

```r
accuracy(PredictTest3, stocksTest3$PositiveDec)
```

```
## [1] 0.625323
```

The overall accuracy of `StocksModel1` on the test set `stocksTest1`, is 0.6446125. The overall accuracy of `StocksModel2` on the test set `stocksTest2`, is 0.5367176. The overall accuracy of `StocksModel3` on the test set `stocksTest3`, is 0.625323.

(p)

Q: To compute the overall test-set accuracy of the cluster-then-predict approach, we can combine all the test-set predictions into a single vector and all the true outcomes into a single vector:

```
> AllPredictions <- c(PredictTest1, PredictTest2, PredictTest3)
> AllOutcomes <- c(stocksTest1$PositiveDec, stocksTest2$PositiveDec, stocksTest3$Positiv
```

What is the overall test-set accuracy of the cluster-then-predict approach, again using a threshold of 0.5?

A:

```r
AllPredictions <- c(PredictTest1, PredictTest2, PredictTest3)
AllOutcomes <- c(stocksTest1$PositiveDec, stocksTest2$PositiveDec,
                 stocksTest3$PositiveDec)
accuracy(AllPredictions, AllOutcomes)
```

```
## [1] 0.5794473
```

The overall test-set accuracy of the cluster-then-predict approach is 0.5794473.

Question 2

(a)

Q: Read the dataset into the dataframe **citi**. How many bike stations are there in this dataset?

A:

```r
citi <- read.csv("citibike.csv", stringsAsFactors = FALSE)
length(unique(c(citi$startstation, citi$endstation)))
```

```
## [1] 329
```

There are 329 bike stations in this dataset.

(b)

Q: On which day of the week, is the average duration of the trips taken by bikers the maximum?

A:

```r
names(sort(tapply(citi$tripduration, citi$day, mean), decreasing = TRUE)[1])
```

```
## [1] "Sat"
```

The average duration of the trips taken by bikers is the maximum on Saturday.

(c)

Q: What is the start hour when the maximum number of bikes are rented? What is the start hour when the minimum number of bikes are rented?

A:

```r
names(which.max(table(citi$starttime)))
```

```
## [1] "18"
```

```r
names(which.min(table(citi$starttime)))
```

```
## [1] "4"
```

The start hour when the maximum number of bikes are rented is 6 pm. The start hour when the minimum number of bikes are rented is 4 am.

(d)

Q: In this dataset, what proportion of the bikes are rented by female users?

A:

```r
unname(table(citi$gender)["2"]/nrow(citi))
```

```
## [1] 0.2349904
```

In this dataset, 0.2349904 of the bikes are rented by female users.

(e)

Q: One of the challenges to do clustering with this data is the presence of the categorical variable for the **day** variable. To tackle this, we will define seven new binary variables (**Mon** to **Sun**), each of which takes a value of 1 if the corresponding trip is taken on that day and 0 otherwise. Write down one such sample R command(s) that you use to do this for a given day of the week.

A:

```r
days <- unique(citi$day)
for (day in days) {
  citi[day] <- as.integer(citi$day == day)
}
```

We iterate through all the days and call `as.integer()` as required.

(f)

Q: In clustering data, it is often important to normalize the variables so that they are all on the same scale. If you clustered this dataset without normalizing, which variable would you expect to dominate in the distance calculations?

- **tripduration**
- **gender**
- **age**
- **starttime**
- **Mon**

A: It is relatively easy to check the scale of each column, we do so by checking the max of the relevant columns.

```
apply(citi[c("tripduration", "gender", "age", "starttime", "Mon")], 2, max)
```

```
## tripduration       gender          age    starttime          Mon
##       730955            2           75           23            1
```

The variable `tripduration` is likely to dominate in the distance calculations.

(g)

Q: Normalize the variables **tripduration, gender, age, starttime, Mon, ..., Sun** by using the **scale()** function in R. We normalize such that each variable has mean 0 and standard deviation 1. What is the maximum value of **tripduration** in the normalized dataset?

A:

```
citi_norm <- citi
vars_to_norm <- setdiff(names(citi), c("startstation", "endstation", "day"))
citi_norm[vars_to_norm] <- apply(citi_norm[vars_to_norm], 2, scale)
max(citi_norm$tripduration)
```

```
## [1] 402.9514
```

The maximum value of `tripduration` in the normalized dataset is 402.9513973.

(h)

Q: We will not use hierarchical clustering for this dataset. Why do you think hierarchical clustering might have a problem with this dataset?

- We have categorical variables in this dataset, so we cannot use hierarchical clustering.
- We might have too many variables in the dataset for hierarchical clustering to handle.
- We might have too many observations in the dataset for hierarchical clustering to handle.
- We are sure of the number of clusters in this application, so using hierarchical clustering does not make sense.

A: We might have too many observations in the dataset for hierarchical clustering to handle.

(i)

Q: Run the k-means algorithm on the normalized dataset with 10 clusters. Use `set.seed(100)` in R just before running the code. You only want to use the variables `tripduration`, `gender`, `age`, `starttime`, `Mon`, ..., `Sun` in building your clusters. Use the default settings to build your model. How many trips are there in the largest and smallest clusters respectively?

A:

```r
set.seed(100)
km_i <- kmeans(citi_norm[,vars_to_norm], centers = 10)
min(km_i$size)
```

```
## [1] 415
```

```r
max(km_i$size)
```

```
## [1] 148501
```

There are 148501 and 415 trips in the largest and smallest clusters respectively.

(j)

Q: Which cluster best fits the description “trips taken primarily by older users on Saturdays”? You can use the centers of the clusters to answer this question.

A: Unfortunately, this question refers to the older system of seeding, so to reproduce that, we will use the old system of seeding. **This will not happen in the exam**.

```
RNGversion("3.5.3")
```

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-uniform
## 'Rounding' sampler used
```

```
set.seed(100)
km_j <- kmeans(citi_norm[,vars_to_norm], centers = 10)
var_j <- c("age", "Sat")
km_j$centers[,var_j]
```

```
##            age        Sat
## 1  -0.10540533 -0.3391057
## 2  -0.64091446  2.9489285
## 3   0.02627901 -0.3391057
## 4   0.07826060 -0.3391057
## 5   1.11985924  2.9489285
## 6   0.03803946 -0.3391057
## 7   0.07311514 -0.3391057
## 8  -0.12675059 -0.3391057
## 9   0.02757155 -0.3391057
## 10 -0.09463120 -0.3391057
```

By manual inspection of the clusters, it seems that cluster 5 is the best fit for the statement. There is no ‘easy’ way to computationally do this, since it is not a priori clear which cluster is considered old and if a cluster is considered active at a particular day.

Still, the following code can calculate for the sums of euclidean distance between different quantiles of the normalised values and the cluster centers,

```r
print_closest_by_quantiles <- function(km_obj, input_variables) {
  for (p in seq(0.8, 1, 0.05)) {
  print(paste(sprintf("%-6s", p),
              which.min(apply(
                (apply(citi_norm[input_variables],
                       2, quantile, probs = p) -
                       t(km_obj$centers[,input_variables])) ^ 2,
                2, sum))
              )
        )
  }
}
```

And we can just use it,

```r
print_closest_by_quantiles(km_j, var_j)
```

```
## [1] "0.8    4"
## [1] "0.85   4"
## [1] "0.9    5"
## [1] "0.95   5"
## [1] "1      5"
```

This shows that the center of cluster 5 is the closest to the 90-th percentile (and above) of the normalised values in `age` and `Sat`.

(k)

Q: Which cluster best fits the description "longer trips taken primarily by female users either on Tuesdays or Wednesdays"? You can use the centers of the clusters to answer this question.

A:

```r
var_k <- c("gender", "Tue", "Wed")
km_j$centers[,var_k]
```

```
##         gender        Tue        Wed
## 1   1.80299766 -0.4855556 -0.4882786
## 2   0.18761389 -0.4855556 -0.4882786
## 3  -0.01315714 -0.4855556 -0.4882786
## 4  -0.55423157 -0.4855556  2.0480080
## 5  -0.13967955 -0.4855556 -0.4882786
## 6  -0.55423157 -0.4855556 -0.4882786
## 7  -0.55423157  2.0594932 -0.4882786
## 8   0.08515270 -0.4855556 -0.4882786
## 9  -0.02634887 -0.4855556 -0.4882786
## 10  1.80380832  0.7792304  0.7874450
```

By manual inspection of the clusters, it seems that cluster 10 is the best fit for the statement. We follow (j),

```
print_closest_by_quantiles(km_j, var_k)
```

```
## [1] "0.8    1"
## [1] "0.85   10"
## [1] "0.9    10"
## [1] "0.95   10"
## [1] "1      10"
```

This shows that the center of cluster 10 is the closest to the 85-th percentile (and above) of the normalised values in `gender`, `Tue` and `Wed`.

(l)

Q: If we ran k-means clustering a second time without making any additional calls to `set.seed`, we would expect

- Different results from the first k-means clustering
- Identical results to the first k-means clustering

A: Different results - we can only replicate results if we use the same seed.

(m)

Q: If we ran k-means clustering a second time, again running the command `set.seed(100)` right before doing the clustering, we would expect

- Different results from the first k-means clustering
- Identical results to the first k-means clustering

A: Identical results to the first k-means clustering.

(n)

Q: Suppose the marketing department at Citi Bike decided that instead of using the days of the week for clustering, they would like to use a single variable weekday which took a value of 1 if the trip started on Monday, Tuesday, Wednesday, Thursday, or Friday and 0 if it started on a Saturday or Sunday. Redo the clustering. As before, remember to normalize the weekday variable and run the k-means algorithm on the normalized dataset with 10 clusters. Use set.seed(100) in R before running the code. Which cluster best fits the description "longer trips taken by older female users on weekdays"? You can use the centers of the clusters to answer this question.

A:

```
weekend_days <- c("Sat", "Sun")
citi_norm$weekday <- as.integer(!(citi$day %in% weekend_days))
citi_norm$weekday <- scale(citi_norm$weekday)
set.seed(100)
cluster_var <- c("tripduration", "gender", "age", "starttime", "weekday")
km_l <- kmeans(citi_norm[,cluster_var], centers = 10)
var_n <- c("tripduration", "gender", "age", "weekday")
km_l$centers[,var_n]
```

```
##     tripduration     gender        age    weekday
## 1    0.409925533  1.7800715  1.1147478  0.5041004
## 2    0.003522478  0.2187535 -0.6427132 -1.9590838
## 3   -0.047614897 -0.5542316  1.0398396  0.5104419
## 4   -0.078466769 -0.5208387 -0.5914811  0.3764170
## 5    0.009448121 -0.1418490  1.1043608 -1.9590838
## 6   -0.015523192 -0.5542316  0.4787926  0.5104419
## 7   -0.018969283 -0.5542316 -0.7725312  0.5104419
## 8   -0.010347262 -0.5542316  1.8753805  0.5104419
## 9   -0.096648552 -0.5542316 -0.5627644  0.5104419
## 10   0.019099791  1.8042973 -0.6737577  0.5104419
```

By manual inspection of the clusters, it seems that cluster 1 is the best fit for the statement.

```
print_closest_by_quantiles(km_l, var_n)
```

```
## [1] "0.8     1"
## [1] "0.85    1"
## [1] "0.9     1"
## [1] "0.95    1"
## [1] "1       1"
```

This shows that the center of cluster 1 is the closest to the 80-th percentile (and above) of the normalised values in `tripduration`, `gender`, `age` and `weekday`.

(o)

Q: Which cluster best fits the description "short trips taken by younger male users early on weekdays"? You can use the centers of the clusters to answer this question.

A:

```r
var_o <- c("tripduration", "age", "gender", "starttime", "weekday")
km_l$centers[,var_o]
```

```
##    tripduration        age     gender  starttime    weekday
## 1   0.409925533  1.1147478  1.7800715 -0.1593482  0.5041004
## 2   0.003522478 -0.6427132  0.2187535  0.1959095 -1.9590838
## 3  -0.047614897  1.0398396 -0.5542316 -1.0983165  0.5104419
## 4  -0.078466769 -0.5914811 -0.5208387 -1.4382010  0.3764170
## 5   0.009448121  1.1043608 -0.1418490 -0.0903787 -1.9590838
## 6  -0.015523192  0.4787926 -0.5542316  0.7174943  0.5104419
## 7  -0.018969283 -0.7725312 -0.5542316  0.9527464  0.5104419
## 8  -0.010347262  1.8753805 -0.5542316  0.3917602  0.5104419
## 9  -0.096648552 -0.5627644 -0.5542316 -0.1497968  0.5104419
## 10  0.019099791 -0.6737577  1.8042973  0.1652332  0.5104419
```

By manual inspection of the clusters, it seems that cluster 9 is the best fit for the statement.

For this part, we note that while we need `tripduration`, `age`, `gender` and `starttime` to be as negative as possible, we want `weekday` to be as positive as possible. In fact, in all of the previous parts, all of the variables should be as positive as possible so that the function would work.

Since this is the last part, we will opt to just transform the data,

```r
neg_to_pos <- c("tripduration", "age", "gender", "starttime")
km_l$centers[,neg_to_pos] <- -km_l$centers[,neg_to_pos]
```

And print the function.

```r
print_closest_by_quantiles(km_l, var_o)
```

```
## [1] "0.8    4"
## [1] "0.85   4"
## [1] "0.9    4"
## [1] "0.95   4"
## [1] "1      9"
```

This shows that the center of cluster 4 is also a very close cluster to the statement while the center of cluster 9 is the closest to the maximum values of the transformed normalised values in `tripduration`, `age`, `gender`, `starttime` and `weekday`.

Finally, we change back the RNG,

```r
RNGversion(paste0(version[c("major","minor")], collapse = "."))
```

Question 3

Preamble

Q: In this question, you will extend the collaborative filtering model from the class. In the class we developed three models for collaborative filtering—using average item rating, using average user rating and by taking the average of the k nearest users using the Pearson correlation metric. In this question, you will extend the last model by using a weighted average where the weights are the similarity metric defined by the Pearson correlation. Develop an R code to do this and verify the quality of the fit by changing the number of neighbors in the set {10,50,100,150,200,250}. Compute the root mean squared error in each of these cases.

A: We adopt the code as used in the lesson,

```r
# Load the ratings dataset
ratings <- read.csv("ratings.csv")

# Create an empty matrix with 706 rows (users) and 8552 (movies)
Data <- matrix(nrow = length(unique(ratings$userId)),
               ncol = length(unique(ratings$movieId)))

# We name the rows and columns with the
# unique users and movieid in the dataset
rownames(Data) <- unique(ratings$userId)
colnames(Data) <- unique(ratings$movieId)

# Fill in the matrix
for (i in 1:nrow(ratings)) {
  Data[as.character(ratings$userId[i]),
       as.character(ratings$movieId[i])] <- ratings$rating[i]
}
```

Splitting the data,

```r
set.seed(1)
spl1 <- sample(1:nrow(Data), 0.98*nrow(Data)) # spl1 has 98% of the rows
spl1c <- setdiff(1:nrow(Data), spl1)          # spl1c has the remaining ones
set.seed(2)
spl2 <- sample(1:ncol(Data), 0.8*ncol(Data))  # spl2 has 80% of the columns
spl2c <- setdiff(1:ncol(Data), spl2)          # spl2c has the rest
UserPred <- matrix(nrow = length(spl1c), ncol = length(spl2c))
```

Dealing with Correlation

We will need to modify the code from this point on for convenience because we do need the correlation data as the code in class simply replaces the correlation data again and again so only the final correlation data is kept, (note that warnings are suppressed)

```r
# Keep track of the correlation between users
Cor <- matrix(nrow = length(spl1c), ncol =  length(spl1))
# Sort users in term of decreasing correlations
Order <- matrix(nrow = length(spl1c), ncol = length(spl1))

# Absolute correlations
AbsCor <- matrix(nrow = length(spl1c), ncol =  length(spl1))
AbsOrder <- matrix(nrow = length(spl1c), ncol = length(spl1))

# For distance correlation
for (i in 1:length(spl1c)) {
  for (j in 1:length(spl1)) {
    Cor[i, j] <- cor(Data[spl1c[i],spl2],
                     Data[spl1[j], spl2],
                     use = "pairwise.complete.obs")
  }

  V <- order(Cor[i,], decreasing = TRUE, na.last = NA)
  Order[i,] <- c(V, rep(NA, times = length(spl1) - length(V)))
  AbsCor[i,] <- abs(Cor[i,])
  absV <- order(AbsCor[i,], decreasing = TRUE, na.last = NA)
  AbsOrder[i,] <- c(absV, rep(NA, times = length(spl1) - length(V)))
}
```

First of all, we are limited by the correlations within the data,

```r
num_correlated <- apply(Order, 1, function(x){sum(!is.na(x))})
num_correlated
```

```
##  [1] 366 393 486 209 416 419 420 365 239 526 373 331 557 386 424
```

We note that there is one user for which no correlation is defined for up to 250 other users.

Secondly, we need to find out exactly how we can weight negative correlations so we can predict properly. For example, when the neighbour set size is 250, what should be done with the negatively weighted correlations?

```
format(round(sort(Cor[14,], decreasing = TRUE)[200:250], 2), nsmall = 2)
```

```
##  [1] " 0.00" " 0.00" " 0.00" " 0.00" " 0.00" " 0.00" " 0.00" " 0.00" " 0.00"
## [10] " 0.00" "-0.01" "-0.01" "-0.01" "-0.02" "-0.02" "-0.03" "-0.04" "-0.06"
## [19] "-0.06" "-0.06" "-0.08" "-0.08" "-0.08" "-0.08" "-0.08" "-0.09" "-0.09"
## [28] "-0.09" "-0.11" "-0.11" "-0.11" "-0.13" "-0.14" "-0.14" "-0.14" "-0.15"
## [37] "-0.15" "-0.16" "-0.16" "-0.16" "-0.16" "-0.17" "-0.17" "-0.18" "-0.19"
## [46] "-0.19" "-0.19" "-0.20" "-0.20" "-0.20" "-0.21"
```

Ignore Negative Correlations

Our first method is to ignore all negative correlations and set them to 0, as a weight of 0 is essentially ignored in weighted averages.

```
num_neighbors_set <- c(10, 50, 100, 150, 200, 250)
RMSE_table <- matrix(nrow = 5, ncol = length(num_neighbors_set))
for (k in 1:length(num_neighbors_set)) {
  for (i in 1:length(spl1c)) {
    non_neg_cor <- replace(Cor[i,][Order[i,]],
                           which(Cor[i,][Order[i,]] < 0),
                           0)[1:num_neighbors_set[k]]
    UserPred[i,] <- apply(Data[spl1[Order[i, 1:num_neighbors_set[k]]], spl2c],
                          2, weighted.mean, w = non_neg_cor, na.rm = TRUE
                          )
  }
  RMSE_table[2, k] <- sqrt(mean((Data[spl1c, spl2c] - UserPred)^2, na.rm = TRUE))
}
```

Note that we can also reproduce what the original model gave using equal weights,

```
for (k in 1:length(num_neighbors_set)) {
  for (i in 1:length(spl1c)) {
    UserPred[i,] <- apply(Data[spl1[Order[i, 1:num_neighbors_set[k]]], spl2c],
                          2, weighted.mean,
                          w = rep(1, times = num_neighbors_set[k]),
                          na.rm = TRUE
                          )
  }
  RMSE_table[1, k] <- sqrt(mean((Data[spl1c, spl2c] - UserPred)^2, na.rm = TRUE))
}
```

Rescaling Correlations

We can try to rescale the correlations to within the interval $[0, 1]$. This is not ideal in the sense that 0 would map to 0.5 and -1 would map to 0. Essentially, this gives greater weight to positively correlated users, and positive weight to uncorrelated users. But we need some helper functions,

```r
all_equal_within_vector <- function(x, tol=10^-6) {
  if (!anyNA(x)) {
    return(all(abs(x - mean(x)) < tol))
  }
  else {
    return(FALSE)
  }
}
rescale <- function(x) {
  if (all_equal_within_vector(x)) {
    return(x)
  }
  else {
    return(x - min(x))/(max(x) - min(x))
  }
}
```

Predicting,

```r
for (k in 1:length(num_neighbors_set)) {
  for (i in 1:length(spl1c)) {
    UserPred[i,] <- apply(Data[spl1[Order[i, 1:num_neighbors_set[k]]], spl2c],
                          2, weighted.mean,
                          w = rescale(Cor[i,][Order[i,]][1:num_neighbors_set[k]]),
                          na.rm = TRUE
                          )
  }
  RMSE_table[3, k] <- sqrt(mean((Data[spl1c, spl2c] - UserPred)^2, na.rm = TRUE))
}
```

Remapping Negative Correlations

Next, we will map the negative correlations to their 'opposite' value. For example, for two perfectly correlated users, a rating of 5 means a rating of 0.5 for the other. A rating of 4.5 means a rating of 1.0 for the other. For correlations of absolute values between 0 and 1, they will be scaled against the mean of the whole rating scale, so if two users have correlation -0.5, a rating of 5 indicates a rating of 1.625 for the other.

Further, we will use the absolute correlations to order the correlations[2]. This means more highly-correlated (positively or negatively) users will be more important in prediction,

```r
f <- function(val, cor) {
  return(2.75 + (val - 2.75)*cor)
}
for (k in 1:length(num_neighbors_set)) {
  for (i in 1:length(spl1c)) {
    data_i <- Data[spl1[AbsOrder[i, 1:num_neighbors_set[k]]], spl2c]
    x_i <- f(data_i, Cor[i,][AbsOrder[i,]][1:num_neighbors_set[k]])
    UserPred[i,] <- apply(x_i, 2, weighted.mean,
                          w = (AbsCor[i,][AbsOrder[i,]][1:num_neighbors_set[k]]),
                          na.rm = TRUE
                          )
  }
  RMSE_table[4, k] <- sqrt(mean((Data[spl1c,spl2c] - UserPred)^2,na.rm = TRUE))
}
```

Distance Correlation

(Note: this method does not answer the question as the question requires use of the Pearson correlation coefficient)

Our next method utilises distance correlation, which by definition lies within $[0, 1]$. The function we will use does not work very well with missing values and vectors of non-equal length. Hence, we will first remove all missing values and then we will take a sample of size equal to the smaller vector and take their distance correlation,

```r
library(energy)
dCor <- matrix(nrow = length(spl1c), ncol =  length(spl1))
dOrder <- matrix(nrow = length(spl1c), ncol = length(spl1))
for (i in 1:length(spl1c)) {
  for (j in 1:length(spl1)) {
    x_1 <- Data[spl1c[i],spl2]
    x_1 <- x_1[!is.na(x_1)]
    x_2 <- Data[spl1[j], spl2]
    x_2 <- x_2[!is.na(x_2)]
    if (length(x_1) <= length(x_2)) {
      x_2 <- sample(x_2, length(x_1))
    } else {
      x_1 <- sample(x_1, length(x_2))
```

[2] This would be the same as using r^2 to order them, since both x^2 and $|x|$ behave identically in terms of monotonicity in disjoint regions of positive and negative numbers

```r
    }
    dCor[i, j] <- dcor(x_1, x_2)
  }
  V <- order(dCor[i,], decreasing = TRUE)
  dOrder[i,] <- c(V, rep(NA, times = length(spl1) - length(V)))
}
```

```r
for (k in 1:length(num_neighbors_set)) {
  for (i in 1:length(spl1c)) {
    UserPred[i,] <- apply(Data[spl1[dOrder[i, 1:num_neighbors_set[k]]], spl2c],
                          2, weighted.mean,
                          w = dCor[i,][dOrder[i,]][1:num_neighbors_set[k]],
                          na.rm = TRUE
                          )
  }
  RMSE_table[5, k] <- sqrt(mean((Data[spl1c, spl2c] - UserPred)^2, na.rm = TRUE))
}
```

Results

```r
colnames(RMSE_table) <- num_neighbors_set
rownames(RMSE_table) <- c("Original", "Ignore", "Rescaling", "Remapping",
                          "Distance")
RMSE_table
```

```
##                  10       50      100      150       200       250
## Original  0.9864062 1.064186 1.034837 1.077052 0.9955846 0.9893346
## Ignore    0.9864062 1.062456 1.024786 1.062634 0.9898706 0.9824682
## Rescaling 0.9864062 1.083734 1.027508 1.060833 1.0059590 0.9775067
## Remapping 1.7899086 1.453799 1.233978 1.187514 1.1760125 1.1755686
## Distance  1.2985708 1.130295 1.056750 1.071925 1.0748844 1.1035064
```

From the table, it seems that simpler methods like using the correlation-ordering with equal weights and ignoring negative values by setting them to 0 is more accurate on the test set than any of the alternatives.