Homework 2

This week was devoted to validation methods and clustering. The following document provides answers to all questions including optional ones.

Task 3.1

a)

QUESTION:
Using the same data set (credit_card_data.txt or credit_card_data-headers.txt) as in Question 2.2, use the ksvm or kknn function to find a good classifier: (a) using cross-validation (do this for the k-nearest-neighbors model; SVM is optional);

SOLUTION:

```
#loading the libraries
library(kernlab)
library(kknn)
library(readr)
library(tidyverse)
library(plyr)
library(dplyr)
library(epiDisplay)
library(caret)
library(splitTools)
library(stats)
library(factoextra)
```

```
#Loading the data
data <- read.table("C:/Users/1/Documents/data 2.2/credit_card_data.txt")
```

kNN model using caret library

```
#Using cross validation on kNN
set.seed(2) #for the model to not change everytime
control <- trainControl(method = 'repeatedcv',#k- fold cross validation
                        number = 10, #number of folds
                        repeats = 10)

knn_fit <- train(as.factor(V11)~., #dot means all other columns
                 data = data,
                 method = "knn",
                 trControl = control, #controls how model acts
                 preProcess = c("center", "scale"), #used for scaling and centering the data
                 tuneLength = 20) #how many k values to test
knn_fit
```

```
## k-Nearest Neighbors
##
## 654 samples
##  10 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 588, 588, 588, 589, 588, 588, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.8456910  0.6890714
##    7  0.8439915  0.6867816
##    9  0.8360259  0.6708482
##   11  0.8308205  0.6600053
##   13  0.8332421  0.6646937
##   15  0.8291087  0.6560113
##   17  0.8331045  0.6639952
##   19  0.8379974  0.6738025
##   21  0.8395267  0.6763205
##   23  0.8363287  0.6692706
##   25  0.8366340  0.6693244
##   27  0.8381914  0.6719285
##   29  0.8392896  0.6737346
##   31  0.8403453  0.6755939
##   33  0.8401892  0.6751860
##   35  0.8414108  0.6774814
##   37  0.8421801  0.6790847
##   39  0.8444692  0.6836954
##   41  0.8432523  0.6811509
##   43  0.8424879  0.6793777
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

ANSWER:
k = 5 is found to be the most efficient and shows 84.57% of accuracy among all when using k-fold cross validation. Before while training and testing the model on the same dataset, k = 15 seemed to be more efficient, but now using different datasets help us to see clear image of how model acts.

SVM model using caret library

```
#Using cross validation on SVM
set.seed(2) #for the model to not change everytime
control <- trainControl(method = 'repeatedcv',#k- fold cross validation
                        number = 10, #number of folds
                        repeats = 10)

svm_fit <- train(as.factor(V11)~., #dot means all other columns
                 data = data,
                 method = "svmLinear",
                 trControl = control, #controls how model acts
                 preProcess = c("center", "scale")) #used for scaling and centering the data
svm_fit
```

```
## Support Vector Machines with Linear Kernel
##
## 654 samples
##  10 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 588, 588, 588, 589, 588, 588, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.862337  0.7267625
##
## Tuning parameter 'C' was held constant at a value of 1
```

ANSWER:
In the first homework the model predicted 86.39% of data points correctly at C = 0.0018. This time it is 86.23% at **C = 1** which is a little smaller and less optimistic but does not make huge difference.

b)

QUESTION:
Using the same data set (credit_card_data.txt or credit_card_data-headers.txt) as in Question 2.2, use the ksvm or kknn function to find a good classifier: (b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).

SOLUTION:

Splitting the dataset

```
#Splitting the data into train, test and validation set using splitTools library
set.seed(123)
splitting  = partition(data$V11, p = c(train_set = 0.6, validation_set = 0.2, test_set = 0.2))
train_set = data[splitting$train_set, ]
test_set = data[splitting$test_set, ]
validation_set = data[splitting$validation_set, ]
```

SVM model

The following code sections use the same principals as in Homework 1, using ksvm function.

```
#Creating the indicators to assess the model
c_values <- 10^seq(-6, 6, 1) #Range of C coefficient is set from 10^-6 to 10^6. It progresses geometrically by 10
margin <- numeric(13) #We already know how many values are there going to be (13 C-values), so it is set to lengt
h 13
total_error <- numeric(13) #Training error is equal to 1 - prediction accuracy, but it was still set for illustra
tion purposes
prediction_accuracy <- numeric(13)
```

```
#Creating the loop to iterate every C value and storing the indicators
for (i in seq(c_values)) {
  set.seed(1)
  svm_fit <- ksvm(as.matrix(train_set[,1:10]),
                  as.factor(train_set[,11]),
                  type = 'C-svc',
                  kernel = 'vanilladot', #linear kernel model
                  C = c_values[i], #takes C coef from vector c_values
                  scaled=TRUE
                  )
  a <- colSums(svm_fit@xmatrix[[1]] * svm_fit@coef[[1]])
  margin[i] <- 2/(sqrt(sum(a^2))) #the formulae for the distance between two lines
  prediction_accuracy[i] <- sum(predict(svm_fit,validation_set[,1:10]) == validation_set[,11]) /nrow(validation_s
et)
}
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
#Showcasing the results
results <- data.frame(c_values, margin, prediction_accuracy)
results
```

```
##    c_values       margin prediction_accuracy
## 1     1e-06 6274.0848379           0.5454545
## 2     1e-05  629.1371882           0.5454545
## 3     1e-04   63.0415195           0.5454545
## 4     1e-03    6.6323367           0.6136364
## 5     1e-02    2.0734027           0.8863636
## 6     1e-01    1.9893204           0.8863636
## 7     1e+00    1.9823373           0.8863636
## 8     1e+01    1.9813315           0.8863636
## 9     1e+02    1.9812225           0.8863636
## 10    1e+03    2.0036233           0.8863636
## 11    1e+04    1.9999160           0.8863636
## 12    1e+05    1.9320408           0.8863636
## 13    1e+06    0.7466815           0.7272727
```

C = 0.01 seems to perform the best on the validation set among all other constraints having prediction accuracy of 88.64%. Let's check how this model performs on test dataset.

```
#Running test dataset to check the best model
best_fit <- ksvm(as.matrix(train_set[,1:10]),
                  as.factor(train_set[,11]),
                  type = 'C-svc',
                  kernel = 'vanilladot', #linear kernel model
                  C = 0.01, #takes C coef from vector c_values
                  scaled=TRUE
                  )
```

```
##  Setting default kernel parameters
```

```
prediction_accuracy <- sum(predict(svm_fit,test_set[,1:10]) == test_set[,11]) /nrow(test_set)
best_fit
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 0.01
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 222
##
## Objective Function Value : -1.6995
## Training error : 0.153453
```

```
paste0('Prediction accuracy on the test set: ',prediction_accuracy)
```

```
## [1] "Prediction accuracy on the test set: 0.740458015267176"
```

It performs worse on the test set than on the validation set.

kNN model

The following code sections use the same principals as in Homework 1, using kkNN function.

```
#Setting the k between 1 and 40 (bigger range requires too much time to process)
k_values <- seq(1,40)
knn_accuracy_total <- numeric(40) #a vector to store results of the model
knn_prediction <- rep(0,nrow(train_set)) #repeats 0 on all data points
for (k in k_values) {
  set.seed(1)
  knn_model <- kknn(train_set[,11]~.,
                      train_set[,1:10],
                      validation_set[,1:10],
                      k = k,
                      kernel = 'optimal',
                      scale = TRUE
                      )
  knn_accuracy_total[k] <- sum(round(fitted(knn_model)) == validation_set[,11]) /nrow(validation_set) #stores the
accuracy percentage for every k value
}
```

```
#Illustrating the results
knn_results <- data.frame(k_values, knn_accuracy_total)
knn_results
```

```
##    k_values knn_accuracy_total
## 1         1          0.7500000
## 2         2          0.7500000
## 3         3          0.7500000
## 4         4          0.7500000
## 5         5          0.8030303
## 6         6          0.8106061
## 7         7          0.8181818
## 8         8          0.8257576
## 9         9          0.8333333
## 10       10          0.8409091
## 11       11          0.8409091
## 12       12          0.8409091
## 13       13          0.8409091
## 14       14          0.8409091
## 15       15          0.8409091
## 16       16          0.8409091
## 17       17          0.8484848
## 18       18          0.8484848
## 19       19          0.8560606
## 20       20          0.8560606
## 21       21          0.8560606
## 22       22          0.8560606
## 23       23          0.8560606
## 24       24          0.8560606
## 25       25          0.8560606
## 26       26          0.8560606
## 27       27          0.8560606
## 28       28          0.8560606
## 29       29          0.8560606
## 30       30          0.8560606
## 31       31          0.8560606
## 32       32          0.8560606
## 33       33          0.8560606
## 34       34          0.8636364
## 35       35          0.8636364
## 36       36          0.8712121
## 37       37          0.8787879
## 38       38          0.8787879
## 39       39          0.8787879
## 40       40          0.8787879
```

```
paste0('Best k: ', knn_results[which(knn_results$knn_accuracy_total == max(knn_results$knn_accuracy_total)),1], '
with an accuracy of ' ,round(max(knn_results$knn_accuracy_total)*100,2))
```

```
## [1] "Best k: 37 with an accuracy of 87.88"
## [2] "Best k: 38 with an accuracy of 87.88"
## [3] "Best k: 39 with an accuracy of 87.88"
## [4] "Best k: 40 with an accuracy of 87.88"
```

There are four k values that perform equally in the range of 40. We will choose the most cost efficient one k = 37.

```
#Let's check how k = 37 performs on test dataset
best_model <- kknn(train_set[,11]~.,
                      train_set[,1:10],
                      test_set[,1:10],
                      k = k,
                      kernel = 'optimal',
                      scale = TRUE
                      )
paste0('Accuracy of performance on the test set: ',round(sum(round(fitted(best_model)) == test_set[,11]) /nrow(te
st_set)*100,2)) #accuracy of perfomance on the test dataset
```

```
## [1] "Accuracy of performance on the test set: 84.73"
```

ANSWER:
Model performs for 3% worse on the test dataset than on the validation set. All these analysis show how svm model predicts better on the validation set in comparison with kNN model, while kNN model performs better on the test dataset by 10% comparing with the svm model. Despite this ambigious result we need to assess which model is the best looking at how it performs on the validation set. Therefore, SVM model works better than kNN model.

Task 4.1

QUESTION:
Describe a situation or problem from your job, everyday life, current events, etc., for which a clustering model would be appropriate. List some (up to 5) predictors that you might use.

ANSWER: As we understood from the lecture notes, clustering is similar to classification but in this case we do not know the response to the data points and the number of levels in the response. Thus, as an example, let's say we have found a whole new recipe book that we want to clusterize according to similarities in the recipes (bakery, vegan, pescetarianism, fastfood and etc.). How these recipes should be grouped definitely would depend on the following predictors:
* How it is cooked
* Meat consistence
* Time of cooking
* Flour consistence
* Amount of oil

Task 4.2

QUESTION:
The iris data set iris.txt contains 150 data points, each with four predictor variables and one categorical response. The predictors are the width and length of the sepal and petal of flowers and the response is the type of flower. The data is available from the R library datasets and can be accessed with iris once the library is loaded. It is also available at the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Iris). The response values are only given to see how well a specific method performed and should not be used to build the model.

Use the R function kmeans to cluster the points as well as possible. Report the best combination of predictors, your suggested value of k, and how well your best clustering predicts flower type.

SOLUTION:

```
#Let's see what the iris.txt dataset is:
my_iris <- iris
head(my_iris,5)
```

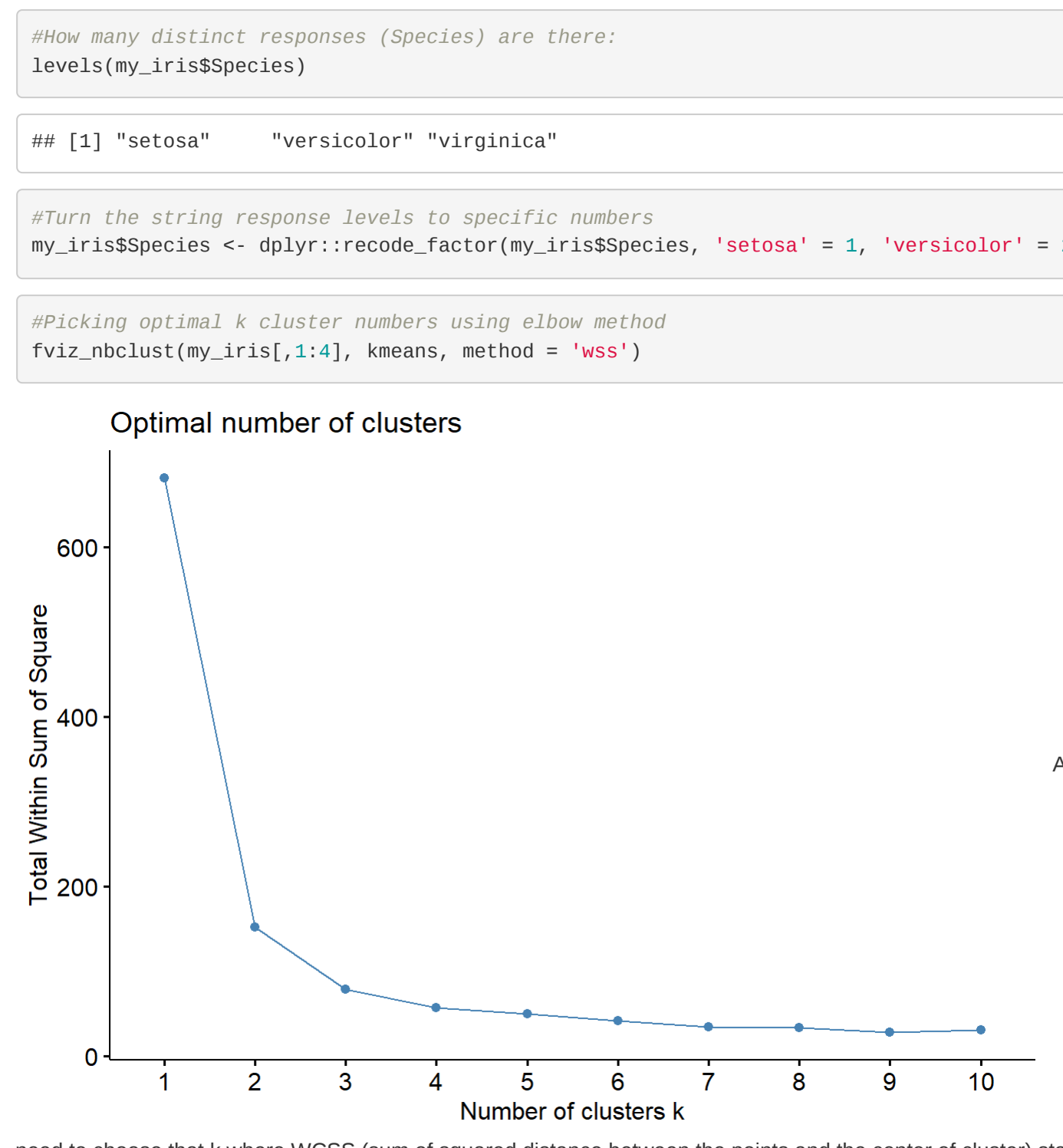
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

```
#How many distinct responses (Species) are there:
levels(my_iris$Species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

```
#Turn the string response levels to specific numbers
my_iris$Species <- dplyr::recode_factor(my_iris$Species, 'setosa' = 1, 'versicolor' = 2, 'virginica' = 3)
```

```
#Picking optimal k cluster numbers using elbow method
fviz_nbclust(my_iris[,1:4], kmeans, method = 'wss')
```



According to the elbow method we need to choose that k where WCSS (sum of squared distance between the points and the center of cluster) starts decreasing. In this case it is k = 3.

```
#Performing the clustering using the kmeans algorithm
set.seed(123)
clustering <- kmeans(my_iris[,1:4],
                       centers = 3,
                       nstart = 30)
clustering
```

```
## K-means clustering with 3 clusters of sizes 50, 62, 38
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     5.006000    3.428000     1.462000    0.246000
## 2     5.901613    2.748387     4.393548    1.433871
## 3     6.850000    3.073684     5.742105    2.071053
##
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3
## [112] 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3
## [149] 3 2
##
## Within cluster sum of squares by cluster:
## [1] 15.15100 39.82097 23.87947
##  (between_SS / total_SS =  88.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
#Testing the prediction with the original responses
accuracy <- sum(clustering$cluster == my_iris$Species) / nrow(my_iris)
accuracy
```

```
## [1] 0.8933333
```

ANSWER:
The results show that optimal **k = 3** found from the elbow method predicts clusters with **89.33 %** of accuracy.

```
#Let's plot our clusters
fviz_cluster(clustering, my_iris[,-5])
```