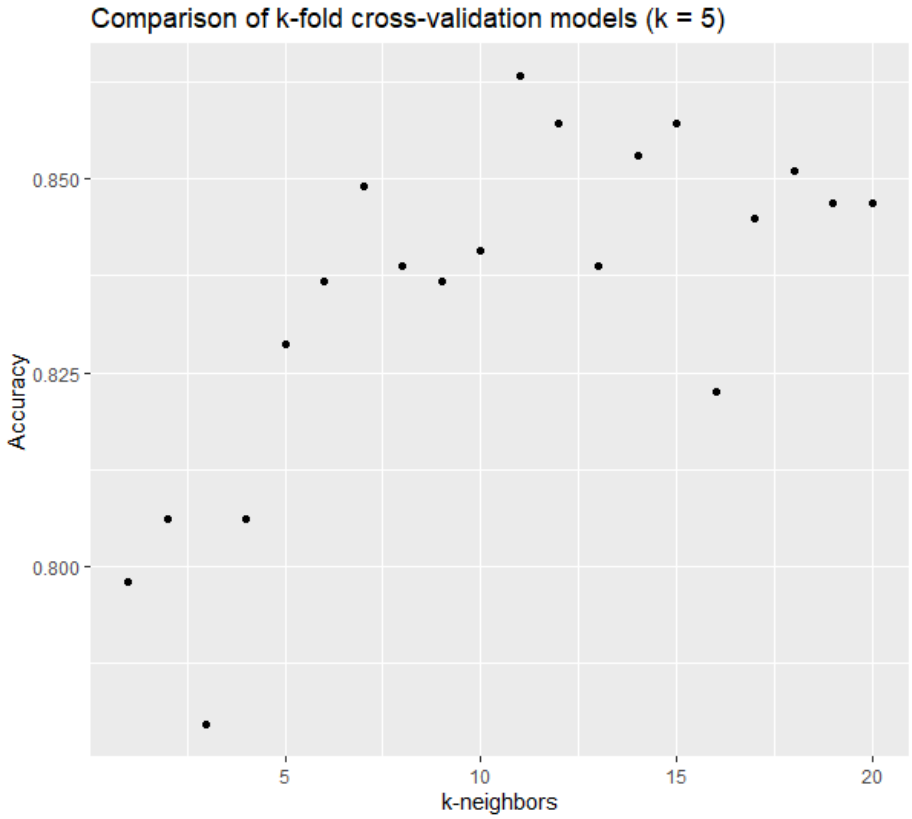
Question 3.1

For part (a), I implemented k-fold cross-validation to find a good value of k-nearest neighbors for the credit card dataset. I decided to use the "cv.kknn" function, which runs similarly to the "kknn" function, although you can specify the number of partitions to use for the cross-validation. I chose to use a k-value of 5, split the dataset into training and test sets, and then ran the cv.kknn function multiple times on the training set for different values of k-nearest neighbors to find which one has the best accuracy.

Attached is a plot of the 5-fold model accuracy versus the number of k-neighbors. In general, it seems as if k-values lower than 5 had a consistently lower accuracy, while k-values between 10-15 seemed to have the highest accuracy. These results are similar to last week's homework, in which k = 12 or 15 were the most optimal k-values. However, it's important to keep in mind that without setting a specific seed, the randomness that occurs each time you run the script means that the same k-value may not be the most accurate every time. For this case, I will be choosing k = 11 as the "optimal" k-value, with an accuracy of about 86%.

After choosing the optimal k-value, I did a final run of the kknn function on the test set to confirm the results; the accuracy for the test set was 82%, lower than the model accuracy.



CODE

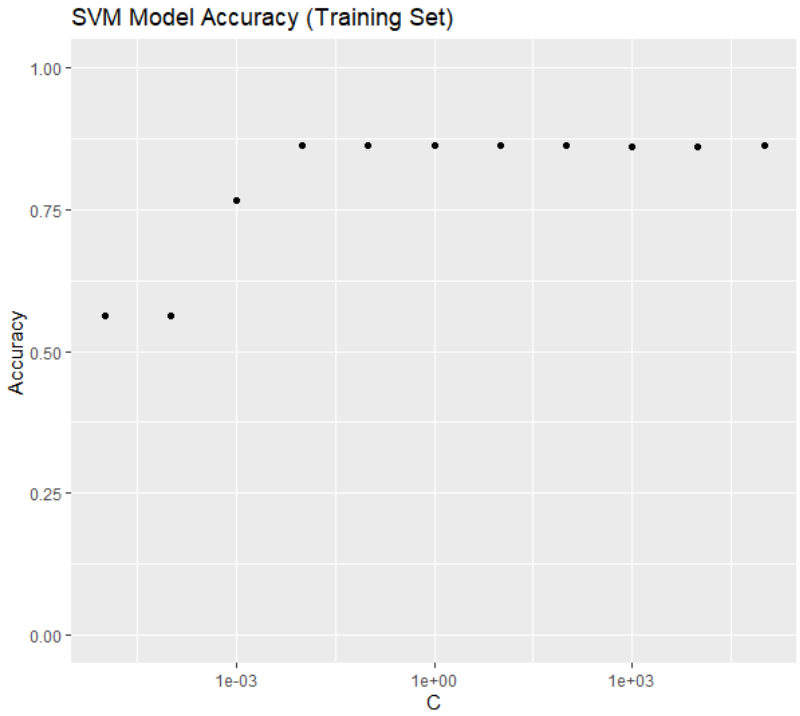
```
# load packages
library(kknn)
library(ggplot2)
library(caret)

# clear environment
rm(list = ls())

# load dataset
path <- "/Users/Dylan Rivera/Desktop/OMSA/Spring 2025/ISYE 6501/HW2/"
ccData <- read.csv(file.path(path,"credit_card_data-headers.txt"), sep="")

# initialize empty data frame to store accuracy
comparison <- data.frame(k = numeric(), Accuracy = numeric())

#set seed
set.seed(1)

# create training and test sets
index <- sample(seq_len(nrow(ccData)), size = floor(0.75*nrow(ccData)))
data_train <- ccData[index,]
data_test <- ccData[-index,]

# for the training set, loop through several values of k
for (j in 1:20) {
 # run cv.kknn
 results <- cv.kknn(R1~A1+A2+A3+A8+A9+A10+A11+A12+A14+A15, data_train, k = j, kcv = 5, distance = 2, kernel =
"optimal", scale = TRUE)

 # round predictions to 0 or 1
 for (i in 1:length(results[[1]][,2])) {
  if (results[[1]][i,2] < 0.5) {
   results[[1]][i,2] = 0
  } else {
   results[[1]][i,2] = 1
  }
 }

 # calculate accuracy
 accuracy <- sum(results[[1]][,1] == results[[1]][,2]) / nrow(results[[1]])

 # add accuracy into table
 row <- c(j, accuracy)
 comparison <- rbind(comparison, row)
 colnames(comparison) <- c("k", "Accuracy")
}

# plot accuracy vs k
plot <- ggplot(comparison, aes(k, Accuracy))
plot + geom_point() + labs(title = "Comparison of k-fold cross-validation models (k = 5)") + xlab("k-neighbors")
```

```
# re-run on the test dataset
results_test <- data.frame(Prediction = numeric(), Actual = numeric())
for (x in 1:654){
  test_kknn <- kknn(R1~A1+A2+A3+A8+A9+A10+A11+A12+A14+A15, data_test[-x,], data_test[x,], k = 11, distance =
2, kernel = "optimal", scale = TRUE)

  # compare prediction with actual value
  prediction <- fitted.values(test_kknn)
  actual <- data_test[x,11]

  if (prediction < 0.5){
    prediction = 0
  } else {
    prediction = 1
  }
  # add results into table
  row_test <- c(prediction, actual)
  results_test <- rbind(results_test, row_test)
  colnames(results_test) <- c("Prediction", "Actual")
}

# calculate accuracy
accuracy_test <- sum(results_test[,1]==results_test[,2]) / nrow(results_test)
```

For part (b), I chose to compare various SVM models, ranging from C = 1e-5 to C = 1e5. I split the credit card dataset according to the following proportions:

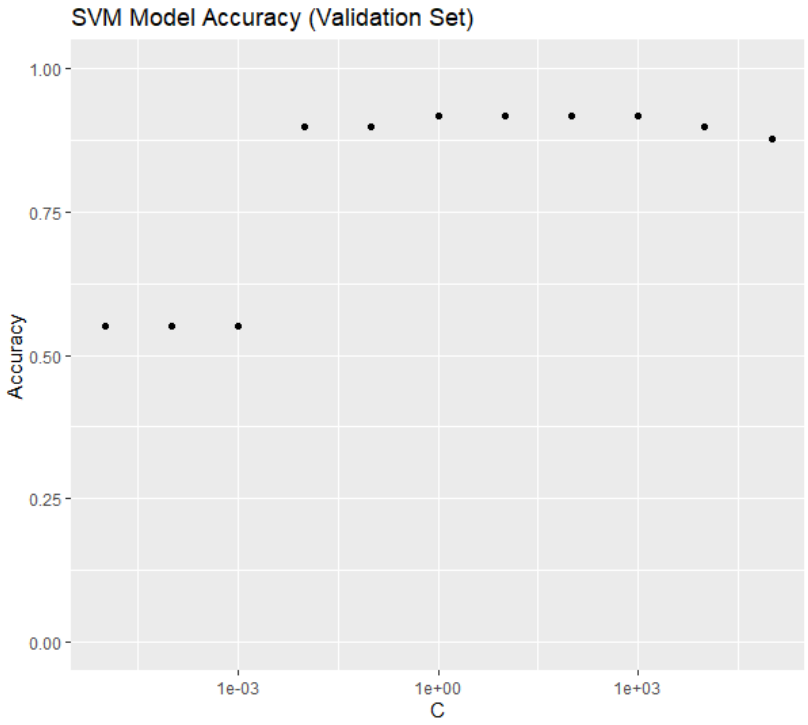
- 70% training
- 15% validation
- 15% test

I first ran the ksvm function on the training set and got similar results as last week's homework, with C values ranging from 1e-2 to 1e5 having the highest accuracy.



C	Accuracy
1e-05	0.5623632
1e-04	0.5623632
1e-03	0.7658643
1e-02	0.8621444
1e-01	0.8621444
1e+00	0.8621444
1e+01	0.8621444
1e+02	0.8621444
1e+03	0.8599562
1e+04	0.8599562
1e+05	0.8621444

I then ran the ksvm function again on the validation set to choose the optimal value of C. In this case, C values ranging from 1 to 1000 had the highest accuracy. It was interesting to see that the accuracies on the validation set were generally higher than those on the training set. I interpret this to simply be the result of the random sampling of rows that were placed in the set. I chose C = 10 as the "optimal" hyperparameter.



C	Accuracy
1e-05	0.5510204
1e-04	0.5510204
1e-03	0.5510204
1e-02	0.8979592
1e-01	0.8979592
1e+00	0.9183673
1e+01	0.9183673
1e+02	0.9183673
1e+03	0.9183673
1e+04	0.8979592
1e+05	0.8775510

Lastly, I ran ksvm a final time on the test set (with C = 10) to confirm the chosen model's quality. This resulted in an accuracy of 87%, which seems like a reasonable value in line with the findings from last week. Below is the list of coefficients for the classifier:

A1	double [1]	0.1600948
A2	double [1]	-0.3939615
A3	double [1]	0.3202281
A8	double [1]	0.4193147
A9	double [1]	1.206844
A10	double [1]	-0.4106693
A11	double [1]	0.05820159
A12	double [1]	0.2230713
A14	double [1]	-0.1989596
A15	double [1]	0.6247378

CODE

```
# load packages
library(kernlab)
library(ggplot2)
library(caret)

# clear environment
rm(list = ls())

# load dataset
path <- "/Users/Dylan Rivera/Desktop/OMSA/Spring 2025/ISYE 6501/HW2/"
ccData <- read.csv(file.path(path,"credit_card_data-headers.txt"), sep="")

# set seed
set.seed(1)

# split data into training, validation, and test sets
spec <- c(train = .7, validate = .15, test = .15)
g <- sample(cut(seq(nrow(ccData)), nrow(ccData)*cumsum(c(0,spec)), labels = names(spec)))
splits <- split(ccData, g)

# call ksvm for training data, loop through different values of C
results_train <- data.frame(C = numeric(), Accuracy = numeric())
C_train <- 1e-5
while (C_train <= 1e5) {
  # call ksvm function
  model <- ksvm(as.matrix(splits$train[,1:10]), as.factor(splits$train[,11]), type="C-svc", kernel="vanilladot", C =
C_train,scaled=TRUE)

  # calculate a1…am
  a <- colSums(model@xmatrix[[1]] * model@coef[[1]])

  # calculate a0
  a0 <- -(model@b)

  # see what the model predicts
  pred <- predict(model,splits$train[,1:10])
```

```
  # see what fraction of the model's predictions match the actual classification
  accuracy <- sum(pred == splits$train[,11]) / nrow(splits$train)

  # add value of C and model's accuracy into table
  row <- c(C_train, accuracy)
  results_train <- rbind(results_train, row)
  colnames(results_train) <- c("C", "Accuracy")

  # iterate to next C
  C_train = C_train*10
}

# plot graph of accuracy vs C value
plot_train <- ggplot(results_train, aes(C, Accuracy)) + geom_point() + scale_x_log10() + ylim(0,1) + labs(title = "SVM Model
Accuracy (Training Set)")

# run ksvm on validation data
results_val <- data.frame(C = numeric(), Accuracy = numeric())
C_val <- 1e-5
while (C_val <= 1e5) {
  model <- ksvm(as.matrix(splits$validate[,1:10]), as.factor(splits$validate[,11]), type="C-svc", kernel="vanilladot", C =
C_val, scaled=TRUE)

  a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
  a0 <- -(model@b)
  pred <- predict(model,splits$validate[,1:10])
  accuracy <- sum(pred == splits$validate[,11]) / nrow(splits$validate)

  row <- c(C_val, accuracy)
  results_val <- rbind(results_val, row)
  colnames(results_val) <- c("C", "Accuracy")

  C_val = C_val * 10
}

plot_val <- ggplot(results_val, aes(C, Accuracy)) + geom_point() + scale_x_log10() + ylim(0,1) + labs(title = "SVM Model
Accuracy (Validation Set)")

# after choosing best C, run ksvm on the test set
model <- ksvm(as.matrix(splits$test[,1:10]), as.factor(splits$test[,11]), type="C-svc", kernel="vanilladot", C = 10,
scaled=TRUE)
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
a0 <- -(model@b)
pred <- predict(model,splits$test[,1:10])
accuracy_test <- sum(pred == splits$test[,11]) / nrow(splits$test)
```

Question 4.1

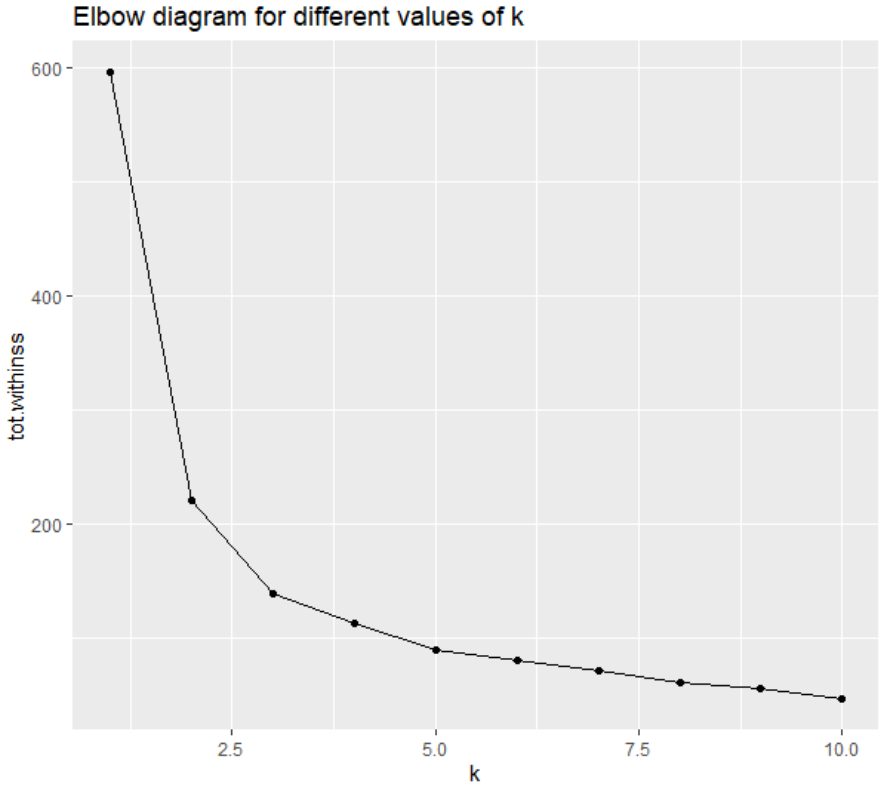
I'm an avid soccer fan, and with how lucrative the prize money is for certain competitions, top teams want to optimize their recruitment efforts in order to obtain the very best players to fit the team's style of play. For example, teams that are more possession-based (in other words, they try to keep the ball for long periods of time) will look at statistics such as:

- Pass accuracy
- Times dispossessed per game
- Successful dribbles per game
- Assists per game
- Tackles per game

The intent is to cluster players with similar stats and possibly assign certain transfer values to players to identify which players the team can purchase within a given budget.

Question 4.2

First, I scaled the dataset, as the "Pedal Width" column seemed to not be on a similar scale as the other columns. Then, I looped the kmeans function through k values of 1 through 10 to choose an optimal number of clusters. The elbow diagram is shown below; through simple visual examination, I chose to proceed with a k value of 4.



However, when I then ran kmeans with k = 4 and compared it to the actual correct classification of irises, I discovered that the 4-cluster model was consistently splitting the “setosa” data points, so I reduced the number of clusters to 3.

I then played around with selecting different combinations of predictors to actually use as the input to the kmeans, and ignoring some. For example, only passing in the “Petal Length” and “Petal Width” columns, or only the “Sepal Length” and “Petal Width” columns, etc. I found that the combination of Petal Width and Petal Length separated the data into defined clusters pretty well, with only a few data points falling into “incorrect” categories, as shown below:

```
> test <- kmeans(iris_scaled[,3:4], centers = 3, nstart = 10)
> table(test$cluster, iris$Species)

   setosa versicolor virginica
1       0         48         4
2      50          0         0
3       0          2        46
```

I understand that technically this is supposed to be an unsupervised learning algorithm, and since we were given access to the actual classifications, it was slightly easier to estimate the necessary number of clusters and what each cluster should look like. If I were to experiment further, I would see if it were possible to fit the data into a different number of clusters (like 4 or 5, for example), while having similar success in having well-defined cluster boundaries.

CODE

```
# load packages
library(ggplot2)

# clear environment
rm(list = ls())

# load dataset
path <- "/Users/Dylan Rivera/Desktop/OMSA/Spring 2025/ISYE 6501/HW2/"
iris <- read.csv(file.path(path,"iris.txt"), sep="")

# initialize table to store results of the kmeans function
tot_sum <- data.frame(k = numeric(), totss = numeric())

#scale data
iris_scaled <- scale(iris[,1:4], center = TRUE, scale = TRUE)
iris_scaled <- as.data.frame(iris_scaled)
iris_scaled['Species'] <- iris[,5]

# run kmeans algorithm, looping through for a number of clusters
for (i in 1:10){
 set.seed(1)
 cluster <- kmeans(iris_scaled[,1:4], centers = i, nstart = 10)

 row <- c(i, cluster$tot.withinss)
 tot_sum <- rbind(tot_sum, row)
 colnames(tot_sum) <- c("k", "tot.withinss")
}

plot <- ggplot(tot_sum, aes(k, tot.withinss)) + geom_point() + geom_line() + xlim(1,10) + labs(title = "Elbow diagram for
different values of k")
plot

# run kmeans for k = 4
set.seed(1)
cluster_4 <- kmeans(iris_scaled[,1:4], centers = 4, nstart = 10)
table(cluster_4$cluster, iris$Species)


cluster_3 <- kmeans(iris_scaled[,1:4], centers = 3, nstart = 10)
table(cluster_3$cluster, iris$Species)

test <- kmeans(iris_scaled[,3:4], centers = 3, nstart = 10)
table(test$cluster, iris$Species)
```