Q2.1

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

One situation might be predicting how voters vote. Hence political parties might be interested to gather data and predict whether an individual is likely to vote for or against their candidate. Possible predictors might be

- Education level
- Income level
- Gender
- Race
- City they live in

Q2.2 part 1

The model used here is the vanilladot model, which is basically a linear kernel. The kernel generating function provided in kernlab for this linear kernel is: k(x, x') =< x, x' >. In simpler terms, the classifier is a straight line with the equation of this form: $\sum_{i=1}^{m} a_i x_i + a_0 = 0$

I wrote a loop to loop vary C from 1 to 1000. The model's accuracy is next stored in a vector which I plotted a graph with. Results suggest that accuracy improved as C increased from 1 to 1000.

I processed the text file to obtain a matrix suitable for ksvm with the following code:

```
================================================================================
library(kernlab)
ccdata = read.table("2.2credit_card_dataSummer2018.txt",
header=FALSE)
ccmatrix = data.matrix(ccdata)
================================================================================
```

The above code imports the kernlab library, reads the textfile and converts the dataframe into a matrix.

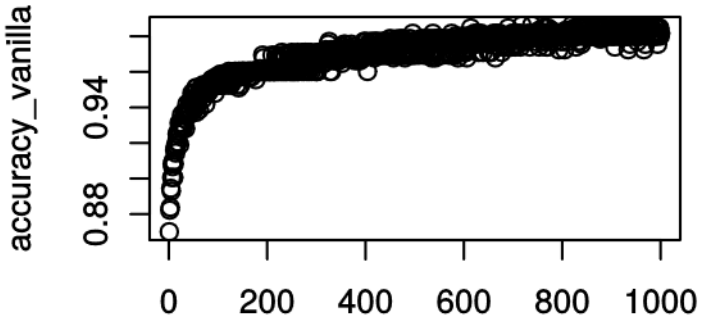
The following is the code I used to obtain the predictions and model's accuracy:

```
================================================================================
accuracy_vanilla <- c() #empty array to store the accuracies
for (i in 1:1000){
ccmodel <- ksvm(ccmatrix[,1:10], ccmatrix[,11], type = "C-svc",
kernal = "vanilladot", C=i, scaled=TRUE)
a <- colSums(ccmodel@xmatrix[[1]] * ccmodel@coef[[1]])
a0 <- -ccmodel@b
pred <- predict(ccmodel,ccmatrix[,1:10])
accuracy_vanilla[i] <- sum(pred == ccdata[,11]) / nrow(ccdata)
```

```
}
```

==========

The resultant graph (plotted in R) investigating how accuracy varies with C is shown below



It can be seen from the graph above that accuracy improves as C is increased.

The maximum accuracy occurred at C=782, with an accuracy of 0.986. My classifier is:

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 782

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0956274980931966

Number of Support Vectors : 227

Objective Function Value : -36774.56
Training error : 0.019878
```

$$\sum_{i=1}^{m} a_i x_i + a_0 = 0$$

Where:

```
a1=-51.06 a2=-12.71 a3=-34.35 a4=123.67 a5=77.50 a6=-85.92 a7=102.53
a8=-60.22 a9=-72.06 a10=98.13 a0=0.61
```

Q2.2 part 2

I modified the kernel to use besseladot instead. The kernel generating function provided in kernlab for this Bessel kernel is: $k(x, x') = (-\text{Bessel}^{n}_{(v+1)}\sigma||x - x'||^2)$

My code to generate the predictions and resulting accuracy is as shown below:

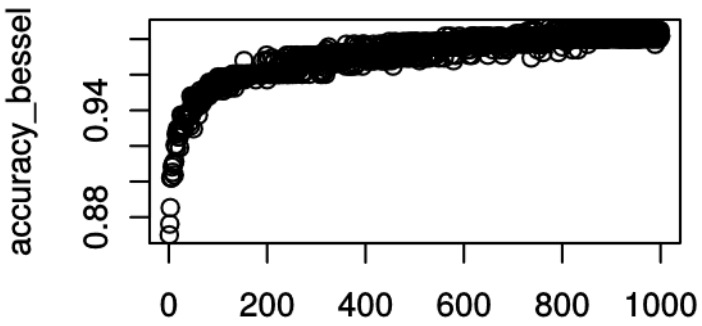
==========

```
accuracy_bessel <- c() #empty array to store the accuracies

for (i in 1:1000){

ccmodel <- ksvm(ccmatrix[,1:10], ccmatrix[,11], type = "C-svc",
kernal = "besseladot", C=i, scaled=TRUE)
```

```
a <- colSums(ccmodel@xmatrix[[1]] * ccmodel@coef[[1]])

a0 <- -ccmodel@b

pred <- predict(ccmodel,ccmatrix[,1:10])

accuracy_bessel[i] <- sum(pred == ccdata[,11]) / nrow(ccdata)

  }
```

==========

Just as before, I plotted a graph to visualize how accuracy varies with C for the Bessel kernel.



It appears the Bessel model has a similar trend as the vanilla model, where accuracy improves as C is increased. Both models achieve a similar level of accuracy as well.

Q2.2 part 3

For this portion, I used the kknn function in R to make predictions. My code is as follow:

==========

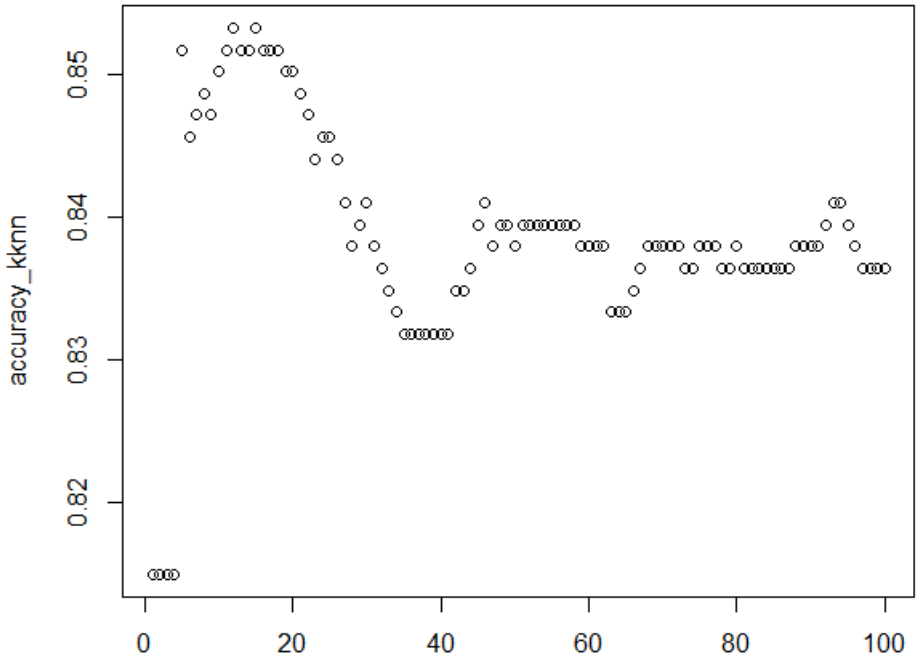
```
accuracy_kknn <- c()

for (num in 1:10) {

for (i in 1:nrow(ccdata)) {

    CC_knn <- kknn(V11~V1+V2+V3+V4+V5+V6+V7+V8+V9+V10, ccdata[-i,], ccdata[i,], k=num, scale = TRUE)

    pred_knn[i] <- as.integer(fitted(CC_knn)+0.5) #this code ensures the prediction is an integer 0 or 1

}

accuracy_kknn[num] <- sum(pred_knn == ccdata$V11)/nrow(ccdata)

}

accuracy_kknn
```

==========

The outer loop changes the value of k from 1 to 100.

The inner loop loops through every row of the dataframe. In every iteration, I call the kknn function to make a prediction about the outcome of the current row, using all other rows as training data.

Similar to the other questions, I did a plot analysing how the model's accuracy varies with k.



It can be observed that accuracy is relatively flat 1<=k<=4, but takes a distinct improvement when k >=5, and fluctuates up and down. The highest accuracy occurs at k=12, with an accuracy of 0.853.

Q3.1 (a)

I'm making use of train.kknn function for this question. This model makes use of leave-one-out cross validation to train the kknn model. My code is as follows:

==========

```
library(kknn)

ccdata = read.table("2.2credit_card_dataSummer2018.txt",
header=FALSE)

ccmatrix = data.matrix(ccdata)


training_points = sample(1:nrow(ccdata),
as.integer(0.8*nrow(ccdata)), replace=FALSE)

all_points = 1:nrow(ccdata)

testing_points = all_points[!all_points %in% training_points]

testing_data = ccdata[testing_points,]

training_data = ccdata[training_points,]
```

```
cc_train <- train.kknn(V11~., data = training_data, kmax=200, scale
= TRUE)

train_pred <- predict(cc_train, testing_data)

train_pred <-as.integer(train_pred +0.5)

sum(train_pred == testing_data$V11)/nrow(testing_data)
```

==========

My code splits the data into two sets, with 80% of the points used for training, and the remaining 20% used for testing. The selection is done randomly. The training reveals the following:

```
Call: train.kknn(formula = V11 ~ ., data = training_data, kmax = 200,
scale = TRUE)

Type of response variable: continuous

minimal mean absolute error: 0.1835564

Minimal mean squared error: 0.1062502

Best kernel: optimal

Best k: 44
```

The accuracy of this model when used on the testing data reveals an accuracy of 0.847, which was decent.

Q3.1 (b)

I'm using ksvm for this question.

==========

```
library(kernlab)

ccdata = read.table("2.2credit_card_dataSummer2018.txt",
header=FALSE)

ccmatrix = data.matrix(ccdata)

#creating 3 sets of data through random selection

training_points = sample(1:nrow(ccdata),
as.integer(0.7*nrow(ccdata)), replace=FALSE)

all_points = 1:nrow(ccdata)

remaining_points = all_points[!all_points %in% training_points]

validation_points = sample(remaining_points,
length(remaining_points)/2, replace=FALSE)

testing_points = remaining_points[!remaining_points %in%
validation_points]
```

```
training_matrix = data.matrix(ccdata[training_points,])

validation_matrix = data.matrix(ccdata[validation_points,])

testing_matrix = data.matrix(ccdata[testing_points,])


# finding out the best C value using the validation data

accuracy_vanilla <- c()

for (i in seq(100,500,100)){

ccmodel <- ksvm(training_matrix[,1:10], training_matrix[,11], type =
"C-svc", kernal = "vanilladot", C=i, scaled=TRUE)

a <- colSums(ccmodel@xmatrix[[1]] * ccmodel@coef[[1]])

a0 <- -ccmodel@b

pred <- predict(ccmodel,validation_matrix[,1:10])

accuracy_vanilla[i] <- sum(pred == validation_matrix[,11]) /
nrow(validation_matrix)

    }

#performing final testing using the chosen C value

ccmodel_final <- ksvm(training_matrix[,1:10], training_matrix[,11],
type = "C-svc", kernal = "vanilladot", C=200, scaled=TRUE)

a_final <- colSums(ccmodel_final@xmatrix[[1]] *
ccmodel_final@coef[[1]])

a0_final <- -ccmodel_final@b

pred_final <- predict(ccmodel_final,testing_matrix[,1:10])

sum(pred_final == testing_matrix[,11]) / nrow(testing_matrix)
```

==========

The data was divided into 3 sets, with 70% of the data assigned to the training set, 15% to validation set and 15% to testing set. The division was done randomly.

A loop was used to loop through various values of C, and 200 was found to give the best outcome. Hence for the final testing, C =200. The model used is vanilladot.

This final model had an accuracy of 0.798. This is of course lower than what was obtained when the entire data set was used for training, but this outcome was well within expectations.

The model is as follows:

```
Support Vector Machine object of class "ksvm"

SV type: C-svc (classification)

parameter : cost C = 200
```

Gaussian Radial Basis kernel function.

Hyperparameter : sigma = 0.114239126680849

Number of Support Vectors : 152

Objective Function Value : -5855.806

Training error : 0.015317

The set of coefficients are as follow:

a1=-54.80 a2=-18.35 a3=-50.39 a4=126.36 a5=85.52 a6=-94.41 a7=103.10 a8=-68.22 a9=-77.88 a10=100.11 a0=0.71