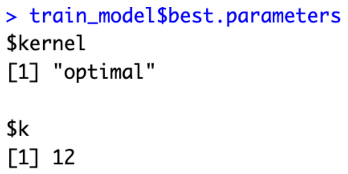
```
**************************************QUESTION 3.1a- using kknn*****************************
#Set working directory to load the dataset
setwd("~/Downloads/IYSE6501/hw2-SP22/data 3.1")

ccdata <- read.table("credit_card_data-headers.txt",header=TRUE)
View(ccdata)

#Set the seed for reproducibility
set.seed(1)

#Install kknn package to use kknn function:
install.packages("kknn")
library(kknn)


set.seed(2)

#Train the KNN model using cross-validation

#This following code is reference from office hour on Jan 20:
train_model <- train.kknn(as.factor(R1)~., ccdata, kmax=22, scale=TRUE) #I chose kmax=22 because I simply want to test k from 1 to 22

#this line of code  is from office hour on Jan 20 to predict the train model:
fitted(train_model)[[4]][1:nrow(ccdata)]

#Check the best parameters
train_model$best.parameters
```



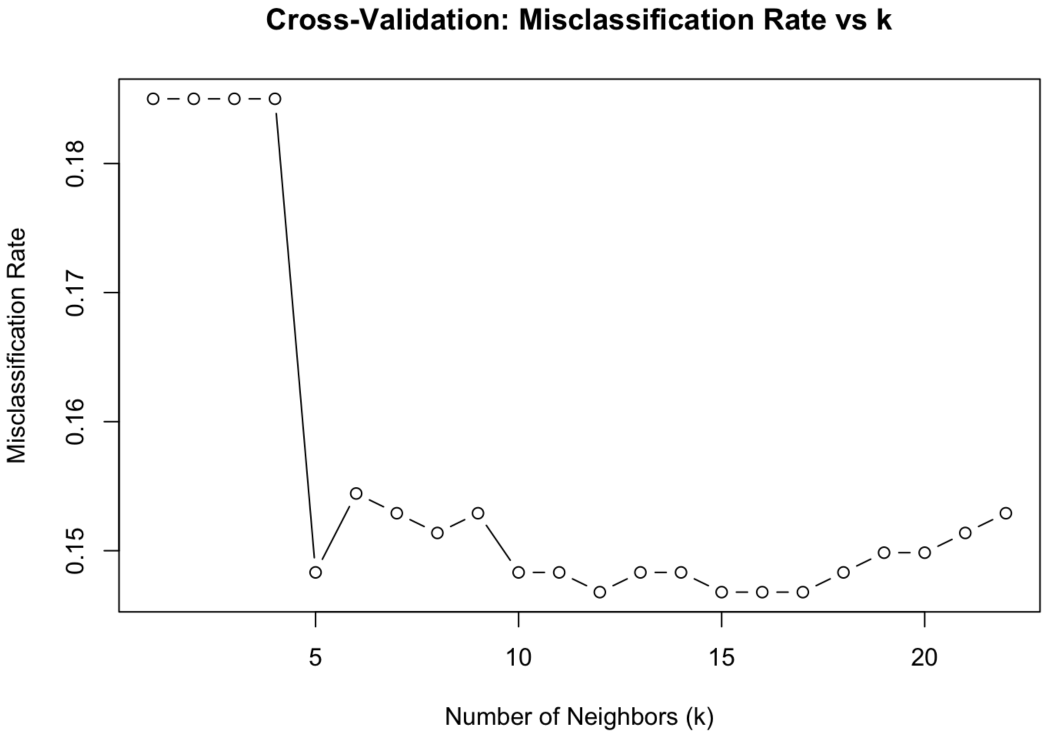
```
> train_model$best.parameters
$kernel
[1] "optimal"

$k
[1] 12
```

```
#Split the data to do kknn model based on the k and kernel we found above, 70% for training and 30% for testing:
train_set <- sample(1:nrow(ccdata), size=0.7*nrow(ccdata), replace=FALSE)
train_data <- ccdata[train_set,]
test_data <- ccdata[-train_set,]

#Train the model based on the best parameters found above, the train set and the test set that are split above:

model_k12 <- kknn(as.factor(R1)~.,train=train_data,test=test_data, k=12, kernel="optimal", scale=TRUE)
```

```
#get prediction from the model with k=12:
predic_model_k12 <- fitted(model_k12)

#Calculate test accuracy:
#This code is referenced from HW1:

test_accuracy = sum(predic_model_k12==test_data$R1)/nrow(test_data)
```

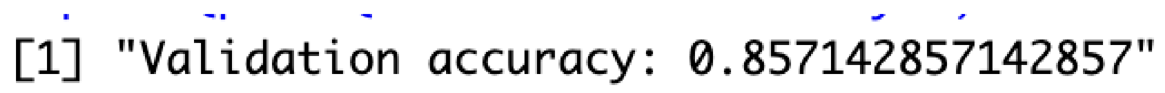
```
test_accuracy                  0.842639593908629
```

```
# Create a plot for misclassification error vs k

# Misclassification rates for different k values

misclass_errors <- train_model$MISCLASS

# this code is referenced from ChatGPT:
plot(1:22, misclass_errors, type='b', xlab="Number of Neighbors (k)", ylab="Misclassification
Rate",
    main="Cross-Validation: Misclassification Rate vs k")
```



As you can see, misclassification is lowest at k=12, so this model works for k=12, kernel=”optimal”, and the accuracy of 84.26%

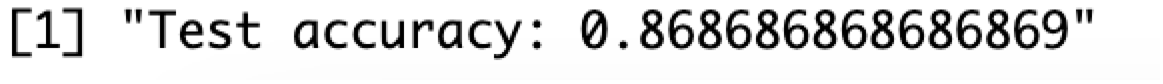
```
**************************************QUESTION 3.1b**************************************
#Load data
ccdata1 <- read.table("credit_card_data-headers.txt", header=TRUE)

#Define factors and repsonse in the dataset
response <- as.factor(ccdata1$R1)
predictors <- ccdata1[,1:10]

#install the package kknn
install.packages("kknn")
library(kknn)

#Set the seed for the reproduce:
set.seed(1)

#Split the data, 70% for training:
train_data <- sample(1:nrow(ccdata1), size=0.7*nrow(ccdata1),replace=FALSE)

training_data <- ccdata1[train_data,]
remaining_data <- ccdata1[-train_data,]

#The remaining 30% of data, use half of it for validating and half of it for testing:
validate_set <- sample(1:nrow(remaining_data), size=0.5*nrow(remaining_data),
replace=FALSE)
validation_data <- remaining_data[validate_set,]
testing_data <- remaining_data[-validate_set,]

#Those loop codes are referenced from HW1, 2-2-3 solution:
#Finding the Best k Using Cross-Validation on the Training Set:

check_accuracy = function(X){  #define a function check_accuracy to compute the accuracy for
different values of k in KNN by iterating over all training data and making predictions.
  predicted <- rep(0,(nrow(training_data))) # predictions: start with a vector of all zeros

  # for each row, estimate its response based on the other rows

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

    # data[-i] means we remove row i of the data when finding nearest neighbors...
    #...otherwise, it'll be its own nearest neighbor!

    model=kknn(R1~.,training_data[-i,],training_data[i,],k=X, scale = TRUE) # use scaled data
    # record whether the prediction is at least 0.5 (round to one) or less than 0.5 (round to zero)

    predicted[i] <- as.integer(fitted(model)+0.5) # round off to 0 or 1
  }

  # calculate fraction of correct predictions

  accuracy = sum(predicted == training_data[,11]) / nrow(training_data)
  return(accuracy)
}


# Now call the function for values of k from 1 to 20


acc <- rep(0,20) # set up a vector of 20 zeros to start
for (X in 1:20){
  acc[X] = check_accuracy(X) # test kknn with X neighbors
}
acc


# Find the best k based on validation data.
best_k <- which.max(acc)

# Train kknn model with the best k on the training data
best_model <- kknn(R1 ~ ., training_data, validation_data, k = best_k, scale = TRUE)

#Evaluating the Model on the Validation Set:
# Predict on validation data, this code is referenced from HW1 solution:
predicted_validation <- as.integer(fitted(best_model) + 0.5)

# Calculate accuracy in validation data:
validation_accuracy <- sum(predicted_validation == validation_data$R1) / nrow(validation_data)
print(paste("Validation accuracy:", validation_accuracy))
```



```
[1] "Validation accuracy: 0.857142857142857"
```

```
# Train final kknn model with best k on the combined training and validation data
#This code is referenced from ChatGPT:
```

```
final_model <- kknn(R1 ~ ., rbind(training_data, validation_data), testing_data, k = best_k, scale = TRUE)

#Evaluating the Model on the Test Set
# Predict on test data, this code is referenced from HW1 solution:
predicted_test <- as.integer(fitted(final_model) + 0.5)

# Calculate test accuracy
test_accuracy <- sum(predicted_test == testing_data$R1) / nrow(testing_data)
```



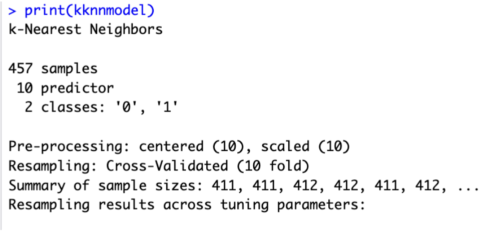
```
[1] "Test accuracy: 0.868686868686869"
```

So the accuracy for this model with 70% training data, 15% validation data, 15% test data, without overlapping each other, is 86.86%.

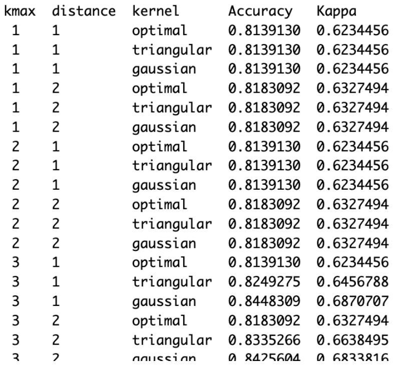
************************************Question 3.1a using caret package****************************

```
#Set file as working directory:
setwd("~/Downloads/IYSE6501/hw2-SP22/data 3.1")

#Load the data in R and call it cc_data:
cc_data <- read.table("credit_card_data-headers.txt", header=TRUE)

#Install the kernlab package to use kknn function:
install.packages("kknn")    #for k-nearest-neighbor
library("kknn")
install.packages("caret")   #for cross validation
library("caret")

#Set the seed for reproduce:
set.seed(1)

#Indicate the response value is at column R1:
cc_data$R1 <- factor(cc_data$R1)
predictors <- cc_data[, -which(names(cc_data) == "R1")]  # Exclude the response variable from the predictors

#Start separating data into a training set, and a test set:

#Start on a training set, take 70% the cc_data:
training_set <- sample(1:nrow(cc_data), size=0.7*nrow(cc_data), replace=FALSE)
#replace=FALSE make sure there is no overlapping data

#To avoid overlapping, I do and extra step to avoid overlapping, and indicate the train set and the remaining set:

train_set <- cc_data[training_set,]
test_set <- cc_data[-training_set,]

#Code reference: https://www.geeksforgeeks.org/svm-with-cross-validation-in-r/

#Set up cross validation control using 10-fold cross validation. The model will be trained 10 times, each time using a different subset as a test set.

train_control <- trainControl(
  method ="cv", #choose the cross validation method
  number=10, #choose 10 folds,
  savePredictions = "final" #parameter ensures that all the predictions are saved
)
```

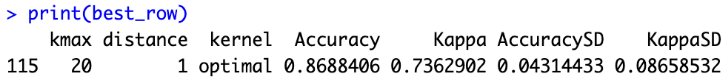
```
#I want to explore all the possible combinations that are made from all the k-nearest-neighbors, the kernels, the distance (Manhattan and Eucdilean), I used function expand.grid():

#This idea is referenced from chatGPT:

tune_grid <- expand.grid(
  kmax= 1:22, #all values from 1 to 22 for the number of neighbor
  distance=c(1,2), #testing Manhattan distance at 1 and Euclidean distance at 2
  kernel=c("optimal", "triangular", "gaussian") #testing different kernel type in kknn
)

train_set$R1 <- factor(train_set$R1)

#train the k-nearest-neighbor using cross validation:

kknnmodel <- train(
  R1~., #response
  data = train_set, #training data
  method = "kknn",
  trControl = train_control,
  tuneGrid = tune_grid, #test the combinations in the grid
  preProcess = c("center", "scale") #This code is referenced from chatGPT, to make sure the data is centered (subtracted by the mean), and is scaled (divided by standard deviation) before training.
)

print(kknnmodel)

# Extract the results from the train object
results <- kknnmodel$results

# Find the row with the highest accuracy
best_row <- results[which.max(results$Accuracy), ]

# Print the best hyperparameters and accuracy
print(best_row)
```



```
> print(kknnmodel)
k-Nearest Neighbors

457 samples
 10 predictor
  2 classes: '0', '1'

Pre-processing: centered (10), scaled (10)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 411, 411, 412, 412, 411, 412, ...
Resampling results across tuning parameters:
```



```
kmax  distance  kernel      Accuracy   Kappa
 1    1         optimal     0.8139130  0.6234456
 1    1         triangular  0.8139130  0.6234456
 1    1         gaussian    0.8139130  0.6234456
 1    2         optimal     0.8183092  0.6327494
 1    2         triangular  0.8183092  0.6327494
 1    2         gaussian    0.8183092  0.6327494
 2    1         optimal     0.8139130  0.6234456
 2    1         triangular  0.8139130  0.6234456
 2    1         gaussian    0.8139130  0.6234456
 2    2         optimal     0.8183092  0.6327494
 2    2         triangular  0.8183092  0.6327494
 2    2         gaussian    0.8183092  0.6327494
 3    1         optimal     0.8139130  0.6234456
 3    1         triangular  0.8249275  0.6456788
 3    1         gaussian    0.8448309  0.6870707
 3    2         optimal     0.8183092  0.6327494
 3    2         triangular  0.8335266  0.6638495
 3    2         gaussian    0.8425604  0.6833816
```



```
> print(best_row)
    kmax distance  kernel  Accuracy     Kappa AccuracySD    KappaSD
115   20        1 optimal 0.8688406 0.7362902 0.04314433 0.08658532
```

So the best k nearest number is 20 with the accuracy of 86.88%