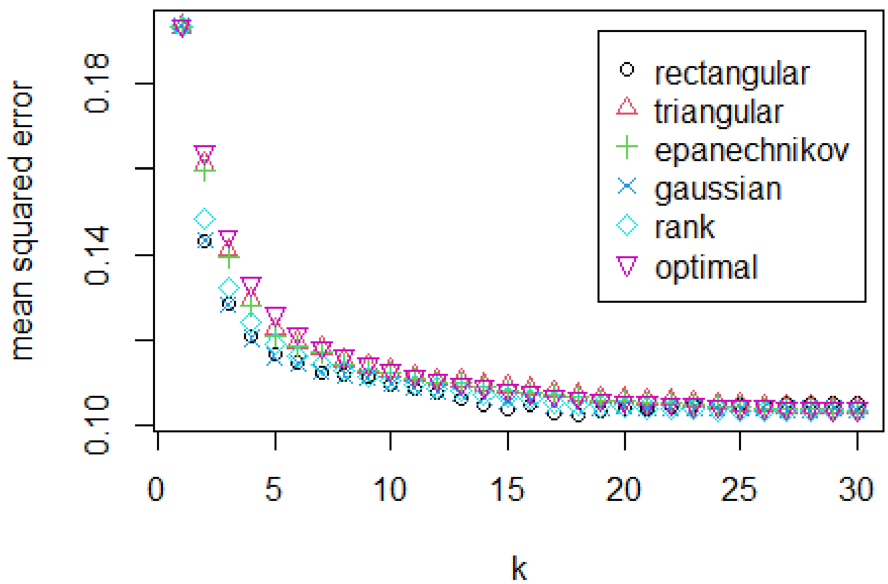
Homework2Solution.R

2020-09-02

```r
#
#             HomeWork 2 Submission by Haritha Pulletikurti
#
#  Question 3.1
#  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); and
#



# (a) Using Cross- Validation

#Step 1: Split the whole dataset into 2 distinct sets: Train and Test
#Step 2: we have to split the  Training data into k different pieces and
making k different models
#        using the same hyperparameters (e.g,. C or k),but different subsets
of training
#        data giving us different model parameters
#Step 3: Trained the chosen model on ALL of Training data to find model
parameters
#Step 4: Reported the picked model's accuracy as its performance on TestData

# Start with a clear environment
rm(list = ls())

setwd("C:\\Users\\harit\\OneDrive\\Documents\\GA-Tech Courses\\Sem 1 - ISYE
6501 - Intro to Analytics Modeling\\Homeworks\\Homework 2 Question\\Homework 2
Solution")

# Load the libraries
library(kknn)
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:kknn':
##
##     contr.dummy

#load the credit card data with headers
credit_card_data <- read.table("credit_card_data-headers.txt",
stringsAsFactors = FALSE, header = TRUE)

#display the first and last few rows of the credit card data
head(credit_card_data)

##   A1    A2    A3   A8 A9 A10 A11 A12 A14 A15 R1
## 1  1 30.83 0.000 1.25  1   0   1   1 202   0  1
## 2  0 58.67 4.460 3.04  1   0   6   1  43 560  1
## 3  0 24.50 0.500 1.50  1   1   0   1 280 824  1
## 4  1 27.83 1.540 3.75  1   0   5   0 100   3  1
## 5  1 20.17 5.625 1.71  1   1   0   1 120   0  1
## 6  1 32.08 4.000 2.50  1   1   0   0 360   0  1

tail(credit_card_data)

##     A1    A2     A3   A8 A9 A10 A11 A12 A14 A15 R1
## 649  1 40.58  3.290 3.50  0   1   0   0 400   0  0
## 650  1 21.08 10.085 1.25  0   1   0   1 260   0  0
## 651  0 22.67  0.750 2.00  0   0   2   0 200 394  0
## 652  0 25.25 13.500 2.00  0   0   1   0 200   1  0
## 653  1 17.92  0.205 0.04  0   1   0   1 280 750  0
## 654  1 35.00  3.375 8.29  0   1   0   0   0   0  0

set.seed(5)


#Step 1 : Split the whole data set into 2 distinct sets: Train and Test

#Generate a random sample of 75% of the rows to Training data

random_rows_for_traindata<- createDataPartition(y =
1:nrow(credit_card_data),p=0.75,list = FALSE)

#The Test Data set is now 60% of the original Credit Card data.
TrainingData = credit_card_data[random_rows_for_traindata,]

TestData= credit_card_data[-random_rows_for_traindata,]

dim(credit_card_data)

## [1] 654  11

dim(TrainingData)

## [1] 492  11
```

```
dim(TestData)

## [1] 162  11

# Step 2 : Leave One Out fold Cross Validation
#Perform Cross-Validation with Kmax = 30  for all of the models on Train data
and pick the best one.
# Using the train.kknn() function to get the best K and the best performing
Kernel.These are called hyper parameters



set.seed(88800)
Best_K_And_KernelModel <- train.kknn(R1~A1+A2+A3+A8+A9+A10+A11+A12+A14+A15,
                         data=TrainingData, kernel = c("rectangular",
"triangular", "epanechnikov", "gaussian",
                                                       "rank",
"optimal"),kmax = 30, scale = TRUE)



Best_K_And_KernelModel

##
## Call:
## train.kknn(formula = R1 ~ A1 + A2 + A3 + A8 + A9 + A10 + A11 +     A12 +
A14 + A15, data = TrainingData, kmax = 30, kernel = c("rectangular",
"triangular", "epanechnikov", "gaussian", "rank", "optimal"),     scale =
TRUE)
##
## Type of response variable: continuous
## minimal mean absolute error: 0.1930894
## Minimal mean squared error: 0.1025858
## Best kernel: rectangular
## Best k: 18

plot(Best_K_And_KernelModel)
```




```
#Analysis of the fitted values of Leave - one -out Cross Validation #

#Now Let us consider the accuracies of different K values returned by
train.kknn

# Create and Initialize  30 zero vectors to hold prediction accuracies of
different k values ( 1: 30)
CV_accuracy=rep(0,30)

# calculate prediction qualities

for (k in 1:30) {
  CVModel_Prediction <-
as.integer(fitted(Best_K_And_KernelModel)[[k]][1:nrow(TrainingData)] + 0.5)
  CV_accuracy[k] <-(sum(CVModel_Prediction == TrainingData[,11])/
nrow(TrainingData))*100

}

CV_accuracy

##  [1] 80.69106 81.91057 83.13008 84.34959 83.94309 85.16260 83.94309
83.94309
##  [9] 83.33333 83.94309 83.13008 83.13008 82.92683 83.73984 84.34959
83.33333
## [17] 84.34959 84.14634 84.14634 83.94309 83.53659 83.73984 83.53659
```

```
83.73984
## [25] 84.14634 83.94309 84.55285 84.55285 84.34959 84.55285

#Step3:Training the kknn model with optimal hyper parameters
# As Train.kknn predicted the best accuracy is at k=17.
# Use these hyperparameters to find the accuracy of the kknn model.

#Create n zero predictions for training data
  predictions_for_training_data<- rep(0,(nrow(TrainingData)))

  for (i in 1:nrow(TrainingData)){
    #Run the kknn function and ensure it doesn't use i itself

KNNModel_for_TrainingData=kknn(R1~A1+A2+A3+A8+A9+A10+A11+A12+A14+A15,Training
Data[-i,],
                  TrainingData[i,],k=18,kernel = "rectangular",scale = TRUE)
    predictions_for_training_data[i] <-
as.integer(fitted(KNNModel_for_TrainingData)+0.5)
  }

#Check the accuracy of the prediction on the Tranining Data.

  Training_Data_Accuracy_by_KKNN = sum(predictions_for_training_data ==
TrainingData[,11]) / nrow(TrainingData)

  Training_Data_Accuracy_by_KKNN

## [1] 0.8414634

#Step 4: Use hyper parameters on Test data to get analyze the Accuracy rate
obtained from the Training data

# First Validate the KKNN Model

# Use these hyperparameters k = 18 and Kernel = Rectangular to find the
accuracy of the kknn model using Test Data


#Create n zero predictions for Test data

  predictions_for_Test_data<- rep(0,(nrow(TestData)))

  for (i in 1:nrow(TestData)){
    #Run the kknn function and ensure it doesn't use i itself

KNNModel_for_TestData=kknn(R1~A1+A2+A3+A8+A9+A10+A11+A12+A14+A15,TestData[-
i,],
                              TestData[i,],k=18,kernel =
"rectangular",distance = 2,scale = TRUE)
```

```
    predictions_for_Test_data[i] <-
as.integer(fitted(KNNModel_for_TestData)+0.5)
  }

  #Check the accuracy of the prediction on the Test Data.

  Test_Data_Accuracy_by_KKNN = (sum(predictions_for_Test_data ==
TestData[,11]) / nrow(TestData))*100

  Test_Data_Accuracy_by_KKNN

## [1] 82.71605

# Inference:
#             The KKNN Model: Best K = 18 , Best Distance d = 2 , Best Kernel
= Rectangular
#             Percentage Accuracy for Training Data ( 75% of the
credit_card_data) is 84.146341%
#             Percentage Accuracy for Test Data ( 25% of the remaining
credit_card_data) is 82.71605%
# So, the Trained Model performs better on the Training Data than on the Test
Data.

#Question 3.1:
#  (b)  splitting the data into training, validation, and test data sets
(pick either KNN or SVM;the other is optional).
# Solution:

# Splitting the data:
# As I am comparing results of the two models
# I will need to split the data into three parts - Training data, Validation
Data , Test Data.
# 20% for testing and 60% for Training  and 20% Validation.



  set.seed(1)

  #Generate a random sample of 60% of the rows to Training data
  random_rows_for_traindata<-createDataPartition(y =
1:nrow(credit_card_data),p=0.6,list = FALSE)
  #The Test Data set is now 60% of the original Credit Card data.
  TrainData = credit_card_data[random_rows_for_traindata,]


  #The remaining 40% of data can be assigned to a TrainAndValidate Data set.
  Test_And_Validating_Data = credit_card_data[-random_rows_for_traindata,]
```

```
  #Split the remaining data equally between Test and Validate data

  #Generate a random sample of 20% of the rows fo Train_And_Validate_Data

  random_rows_for_Test_And_Validating_Data<- createDataPartition(y =
1:nrow(Test_And_Validating_Data),p=0.5,list = FALSE)

  #The Test Data set is now 20% of the Training and Validation data.
  ValidatingData =
Test_And_Validating_Data[random_rows_for_Test_And_Validating_Data,]


  #The remaining 20% of Test and Validation data  can be assigned to a Test
Data set.
  TestingData = Test_And_Validating_Data[-
random_rows_for_Test_And_Validating_Data,]


  #Now We have three Data Sets,  60% TrainingData, 20% ValidationData, 20%
TestData.

  #Test the nrows(Original Creditcard data) = nrows(training data + Test Data
+ Validation Data.)


  writeLines(sprintf("Number of Rows:\n CreditCardData - %d\n Training data -
%d\n Validation data = %d\n Test Data = %d\n Total Split data = %d\n",
                     nrow(credit_card_data),
nrow(TrainData),nrow(ValidatingData),nrow(TestingData),
                     nrow(TrainData) + nrow(ValidatingData) +
nrow(TestingData)))

## Number of Rows:
##  CreditCardData - 654
##  Training data - 394
##  Validation data = 132
##  Test Data = 128
##  Total Split data = 654

  # Model 1 : KSVM Model using C = 100 and Vanilladot Kernel on Training Data


  # Load Library
  library(kernlab)

##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```r
  VanillaDotModelForTrainingData <- ksvm(as.matrix(TrainData[,1:10]),
as.factor(TrainData[,11]), type="C-svc", kernel="vanilladot", C=100,
scaled=TRUE)
```

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

```r
  VanillaDotModelForTrainingData
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 192
##
## Objective Function Value : -10200.52
## Training error : 0.129442
```

```r
  VanillaDotModelpredictionForTrainingData <-
predict(VanillaDotModelForTrainingData,TrainData[,1:10])

  # see what percentage of the model’s predictions match the actual
classification
  AccuracyResultsforVanillaDotKernelOnTrainDataSet=
sum(VanillaDotModelpredictionForTrainingData == TrainData[,11]) /
nrow(TrainData)*100

  AccuracyResultsforVanillaDotKernelOnTrainDataSet
```

```
## [1] 87.05584
```

```r
  #* Model 2 : KSVM Model using C = 100 and Polydot Kernel on Training Data

  PolyDotKernelSVMmodelForTrainingData <-
ksvm(as.matrix(TrainData[,1:10]),as.factor(TrainData[,11]),type="C-
svc",kernel="polydot",C=100,scaled=TRUE)
```

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

```r
  PolyDotKernelSVMmodelForTrainingData
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
```

```
## Polynomial kernel function.
##  Hyperparameters : degree =  1  scale =  1  offset =  1
##
## Number of Support Vectors : 185
##
## Objective Function Value : -10200.46
## Training error : 0.129442

  PolyDotKernelsvmmodelpredictionForTrainingData <-
predict(PolyDotKernelSVMmodelForTrainingData,TrainData[,1:10])

  # see what percentage of the model's predictions match the actual
classification
  AccuracyResultsforPolyDotKernelOnTrainDataSet=
sum(PolyDotKernelsvmmodelpredictionForTrainingData == TrainData[,11]) /
nrow(TrainData)*100

  AccuracyResultsforPolyDotKernelOnTrainDataSet

## [1] 87.05584

  #  Models Inference based on the Training Data Set
  #The Polydot Kernel and Vanilladot Kernel gives 87.05584% of accuracy rate
with 0.12944 Training error,
  # Vanilla dot : Number of Support Vectors = 192 for C=100.
  # Poly dot :    Number of Support Vectors = 185 for C=100.



  #* Model 1 : KSVM Model using C = 100 and Vanilladot Kernel on Validation
Data


  VanillaDotModelForValidationData <- ksvm(as.matrix(ValidatingData[,1:10]),
as.factor(ValidatingData[,11]), type="C-svc", kernel="vanilladot", C=100,
scaled=TRUE)

##  Setting default kernel parameters

  VanillaDotModelForValidationData

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 34
##
```

```
## Objective Function Value : -2830.456
## Training error : 0.090909

  VanillaDotModelpredictionForValidationData <-
predict(VanillaDotModelForValidationData,ValidatingData[,1:10])

  # see what percentage of the model's predictions match the actual
classification
  AccuracyResultsforVanillaDotKernelForValidationData=
sum(VanillaDotModelpredictionForValidationData == ValidatingData[,11]) /
nrow(ValidatingData)*100

  AccuracyResultsforVanillaDotKernelForValidationData

## [1] 90.90909

  #* Model 2 : KSVM Model using C = 100 and Polydot Kernel on Validation Data

  PolyDotKernelSVMmodelForValidationData <-
ksvm(as.matrix(ValidatingData[,1:10]),as.factor(ValidatingData[,11]),type="C-
svc",kernel="polydot",C=100,scaled=TRUE)

##  Setting default kernel parameters

  PolyDotKernelSVMmodelForValidationData

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
## Polynomial kernel function.
##  Hyperparameters : degree =  1  scale =  1  offset =  1
##
## Number of Support Vectors : 34
##
## Objective Function Value : -2830.476
## Training error : 0.090909

  PolyDotKernelsvmmodelpredictionForValidationData <-
predict(PolyDotKernelSVMmodelForValidationData,ValidatingData[,1:10])

  # see what percentage of the model's predictions match the actual
classification
  AccuracyResultsforPolyDotKernelForValidationData=
sum(PolyDotKernelsvmmodelpredictionForValidationData == ValidatingData[,11])
/ nrow(ValidatingData)*100

  AccuracyResultsforPolyDotKernelForValidationData

## [1] 90.90909
```

```
  # Models' Inference based on the Validation Data Set
  #The Polydot Kernel and Vanilladot Kernel gives 90.90909 % of accuracy rate
with 0.090909 training error,
  # Vanilla dot : Number of Support Vectors = 34 for C=100.
  # Poly dot :    Number of Support Vectors = 34 for C=100.




  #* Model 1 : KSVM Model using C = 100 and Vanilladot Kernel on Test Data


  VanillaDotModelForTestData <- ksvm(as.matrix(TestingData[,1:10]),
as.factor(TestingData[,11]), type="C-svc", kernel="vanilladot", C=100,
scaled=TRUE)

##  Setting default kernel parameters

  VanillaDotModelForTestData

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 51
##
## Objective Function Value : -4614.018
## Training error : 0.179688

  VanillaDotModelpredictionForTestData <-
predict(VanillaDotModelForTestData,TestingData[,1:10])

  # see what percentage of the model’s predictions match the actual
classification
  AccuracyResultsforVanillaDotKernelForTestData=
sum(VanillaDotModelpredictionForTestData == TestingData[,11]) /
nrow(TestingData)*100

  AccuracyResultsforVanillaDotKernelForTestData

## [1] 82.03125

  #* Model 2 : KSVM Model using C = 100 and Polydot Kernel on Test Data

  PolyDotKernelSVMmodelForTestData <-
ksvm(as.matrix(TestingData[,1:10]),as.factor(TestingData[,11]),type="C-
svc",kernel="polydot",C=100,scaled=TRUE)
```

```
##  Setting default kernel parameters

  PolyDotKernelSVMmodelForTestData

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 100
##
## Polynomial kernel function.
##  Hyperparameters : degree =  1  scale =  1  offset =  1
##
## Number of Support Vectors : 52
##
## Objective Function Value : -4613.977
## Training error : 0.179688

  PolyDotKernelsvmmodelpredictionForTestData <-
predict(PolyDotKernelSVMmodelForTestData,TestingData[,1:10])

  # see what percentage of the model's predictions match the actual
classification
  AccuracyResultsforPolyDotKernelForTestData=
sum(PolyDotKernelsvmmodelpredictionForTestData == TestingData[,11]) /
nrow(TestingData)*100

  AccuracyResultsforPolyDotKernelForTestData

## [1] 82.03125

  #  Models Inference based on the Training Data Set
  #The Polydot Kernel and Vanilladot Kernel gives 87.05584% of accuracy rate
with 0.12944 Training error,
  # Vanilla dot : Number of Support Vectors = 192 for C=100.
  # Poly dot :    Number of Support Vectors = 185 for C=100.

  #  Models Inference based on the Validation Data Set
  #The Polydot Kernel and Vanilladot Kernel gives 90.90909 % of accuracy rate
with 0.0909 training error,
  # Vanilla dot : Number of Support Vectors = 34 for C=100.
  # Poly dot :    Number of Support Vectors = 34 for C=100.

  #  Models Inference based on the Test Data Set
  #The Polydot Kernel and Vanilladot Kernel gives  82.03125% of accuracy rate
with 0.179688 training error,
  # Vanilla dot : Number of Support Vectors = 51 for C=100.
  # Poly dot :    Number of Support Vectors = 52 for C=100.

  #  Models Inference based Overall Training, Validation and Testing :
  #The Polydot Kernel and Vanilladot Kernel gives highest 90.90909% of
```

```
accuracy rate with 0.0909 training error for
  # Validation Data Set and then on Training Data Set 87.05584% of accuracy
rate with 0.12944 training error
  # which are both higher than the Test data set results.
  #
  # Based on the Test Data Set results "Vanilla Dot" Model Yeilds the best
among the two models as the number of support
  # vectors is less than the Polydot Model with an accuracy of 82.03125%

  #Question 4.1
  #
  #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:
  # I currently work in a company which develops solutions for Change
Managemnent for Robotics.
  # Over time we have developed solutions using different languages and
different libraries and many such solutions
  # involve legacy code. We can develop a clustering model which can
categorize the developed solutions based on the
  # language type, used version of the libraries etc. This will help the
Management to learn about which solutions use
  # legacy code and need to be rewritten using the latest technologies or
which can be of highest customer value.
  #
  #The Predictors that can be used here can be "Application/Module Name",
"Language Used for development","Inbuilt Library
  #"Version", "Version of the Operating System the application is compatible
with" and
  # "Number of Customers for that Application".



  #Question 4.2
  #
  #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.


  # Start with a clear environment
  rm(list=ls())

   #load libraries

  library(kknn)
  library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

  library(factoextra)

## Welcome! Want to learn more? See two factoextra-related books at
https://goo.gl/ve3WBa

  library(ggplot2)

  # set the working directory
  setwd("C:\\Users\\harit\\OneDrive\\Documents\\GA-Tech Courses\\Sem 1 - ISYE
6501 - Intro to Analytics Modeling\\Homeworks\\Homework 2 Question\\Homework 2
Solution")
  # Load the data from iris.txt
  set.seed(1)
  FlowerData <- read.table("iris.txt", stringsAsFactors = FALSE , header =
TRUE)

  head(FlowerData)

##   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
## 6          5.4         3.9          1.7         0.4  setosa

  tail(FlowerData)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 145          6.7         3.3          5.7         2.5 virginica
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

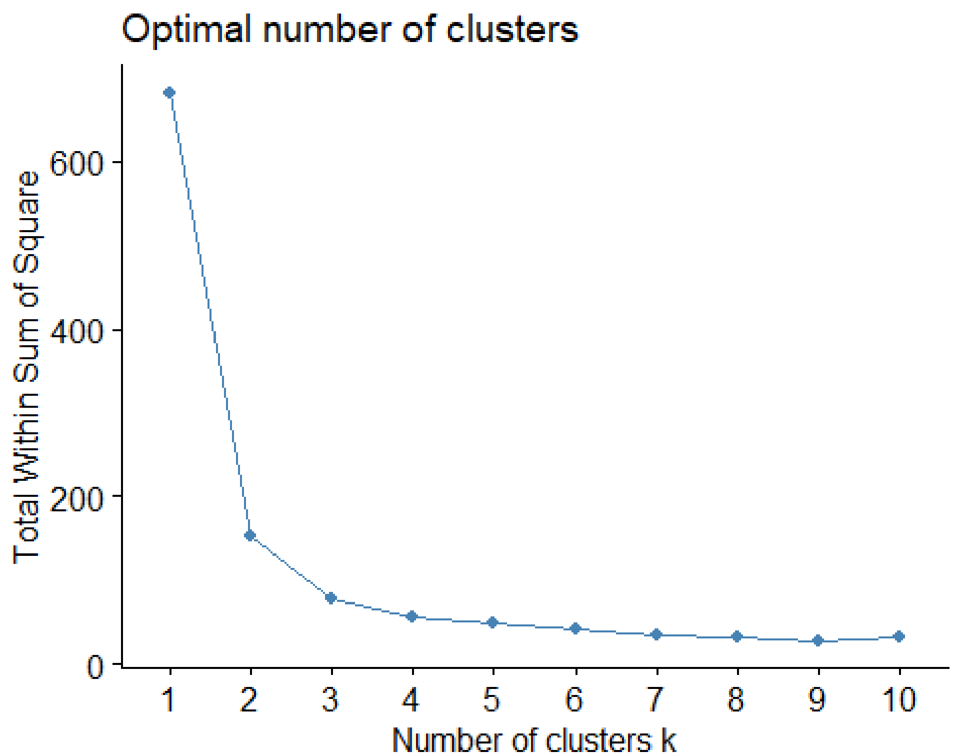
```
  # The iris.txt data contains there different responses "setosa" ,
"versicolor" and "virginica"
  # which indicate the 3 clusters the data need to be separated to.
  # For computational ease, lets assign numbers to each of these categories
  # Let "Setosa" = 1 , "versicolor" = 2 and "virginica" = 3

  Clusteres_Name_Number_Mapping <- c("setosa" = 1, "versicolor" = 2 ,
"virginica" = 3)
  FlowerData$Species <- Clusteres_Name_Number_Mapping[FlowerData$Species]

  head(FlowerData)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2       1
## 2          4.9         3.0          1.4         0.2       1
## 3          4.7         3.2          1.3         0.2       1
## 4          4.6         3.1          1.5         0.2       1
## 5          5.0         3.6          1.4         0.2       1
## 6          5.4         3.9          1.7         0.4       1
```

```
  tail(FlowerData)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 145          6.7         3.3          5.7         2.5       3
## 146          6.7         3.0          5.2         2.3       3
## 147          6.3         2.5          5.0         1.9       3
## 148          6.5         3.0          5.2         2.0       3
## 149          6.2         3.4          5.4         2.3       3
## 150          5.9         3.0          5.1         1.8       3
```

```
  # The Function fviz_nbclust() outpots the plot to evaluate the number of
clusters.
  # We need to input the FlowerData without the reponse variable to this
function.

  # We have 5 columns in the data given - 4 predictors (Sepal-Length, Sepal-
Width, Petal-Length,Petal-Width)
  # and one response variable (FlowerType)
  # Consider only the predictor columns to input in the fviz_nbclus()
function

  fviz_nbclust(FlowerData[,1:4],kmeans,method = "wss")
```



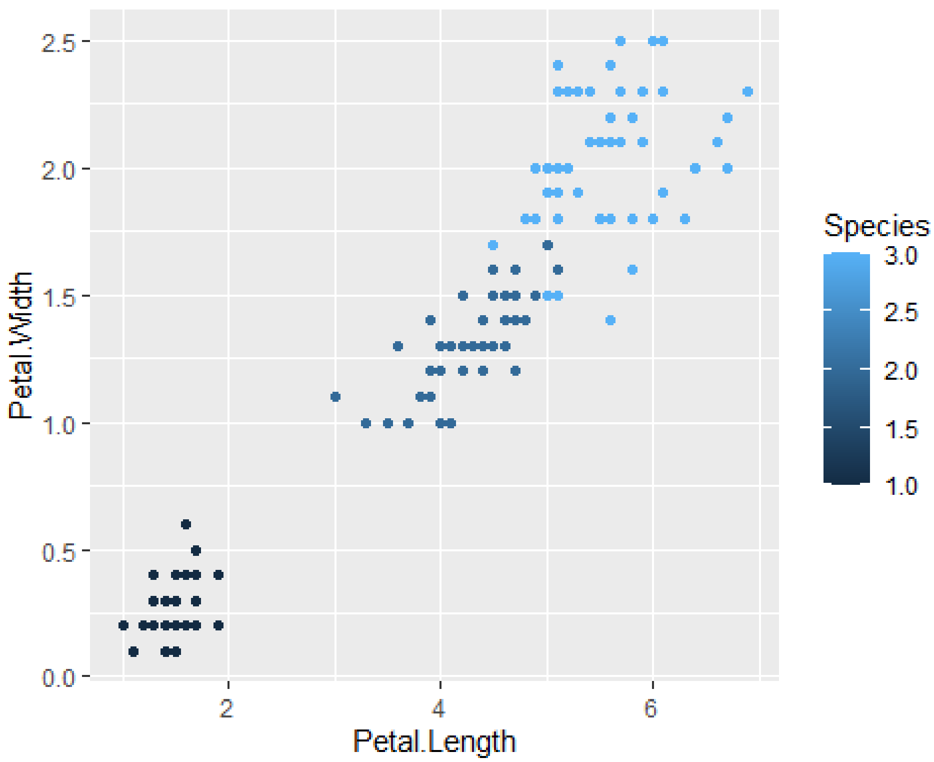

```
  #Based on the elbow method we can determine that k = 3 is the optimal value
of  K centered clusters.

  # Using the Kmeans()  function we can perform clustering

  clustering_result = kmeans(FlowerData[,1:4] ,centers = 3 , nstart =25)
  clustering_result

## 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   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
19  20
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
1   1
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38
39  40
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
1   1
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58
59  60
```

```
##   1   1   1   1   1   1   1   1   1   1   2   2   3   2   2   2   2   2
2   2
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78
79  80
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   3
2   2
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98
99 100
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
119 120
##   3   2   3   3   3   3   2   3   3   3   3   3   3   2   2   3   3   3
3   2
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138
139 140
##   3   2   3   2   3   3   2   2   3   3   3   3   3   2   3   3   3   3
2   3
## 141 142 143 144 145 146 147 148 149 150
##   3   3   2   3   3   3   2   3   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"
```

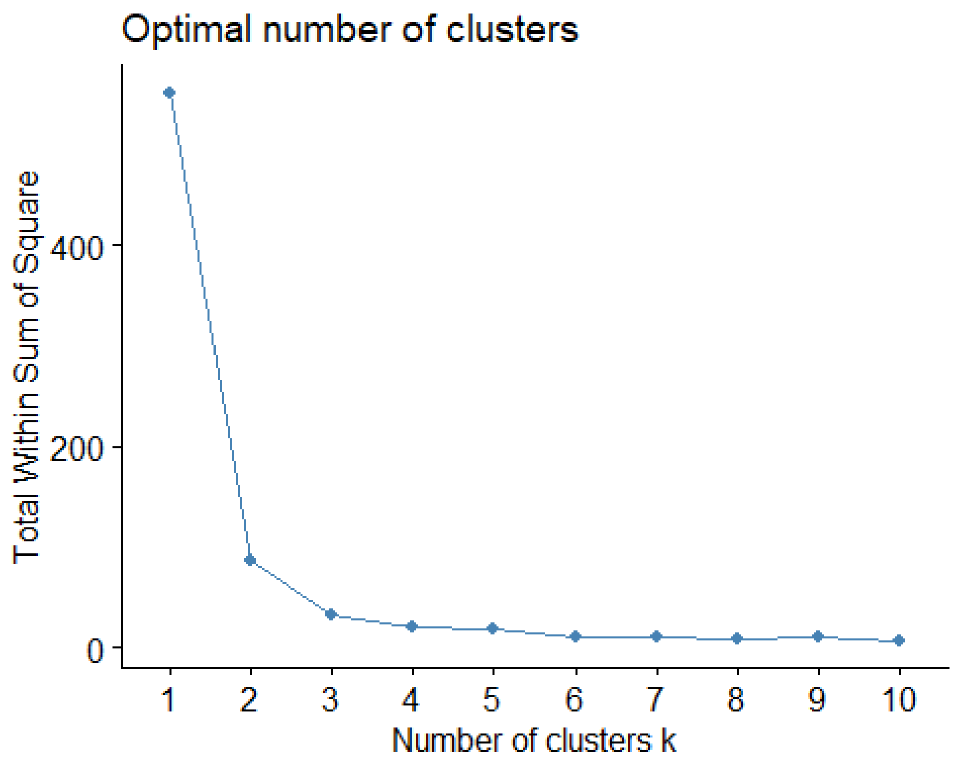
```
  predicted_cluster <-clustering_result$cluster

  Prediction_Accuracy_Percentage = (sum(predicted_cluster ==
FlowerData[,5])/nrow(FlowerData) )* 100
  Prediction_Accuracy_Percentage
```

```
## [1] 89.33333
```

```
  # Inference for Training Data: Number of Optimal Centers k = 3, Prediction
Accuracy = 89.52318



  ggplot(FlowerData,aes(Petal.Length,Petal.Width, color = Species)) +
geom_point()
```

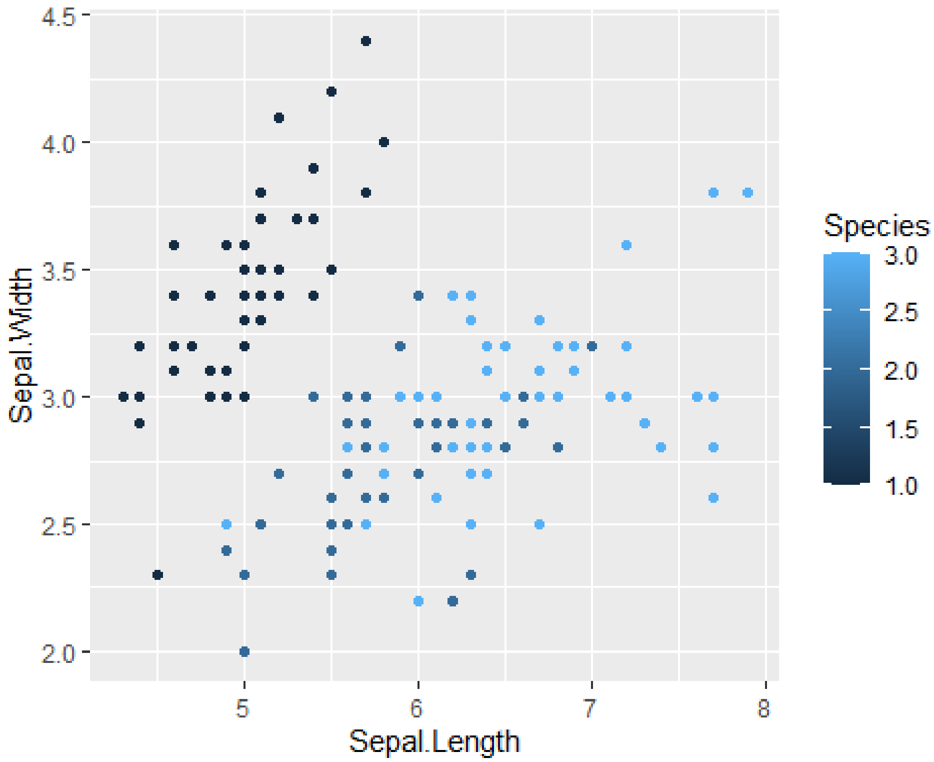



```
  #The Plot of Petal length Vs Petal Width given a better clustering with less number of outliers.
  # Consider only the predictors Petal Length and Petal Width columns to input in the fviz_nbclus() function

  fviz_nbclust(FlowerData[,3:4],kmeans,method = "wss")
```

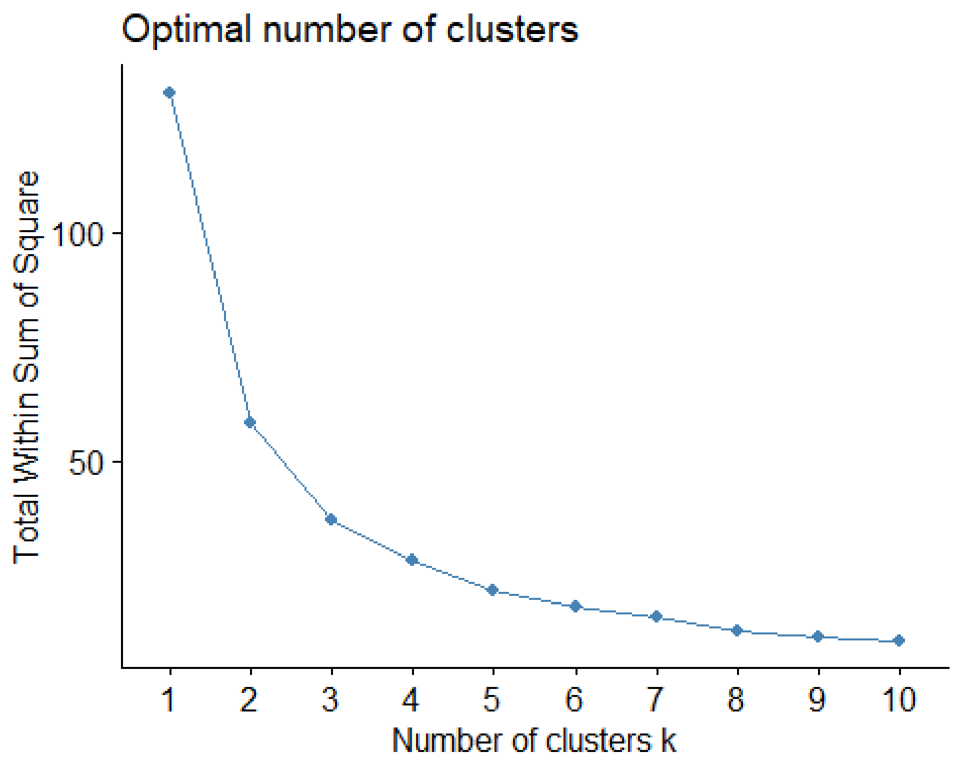



```
  #Based on the elbow method we can determine that k = 3 is the optimal value
of  K centered clusters.

  # Using the Kmeans()  function we can perform clustering

  clustering_result_forPetalPredictors = kmeans(FlowerData[,3:4] ,centers = 3
, nstart =25)
  clustering_result_forPetalPredictors

## K-means clustering with 3 clusters of sizes 48, 52, 50
##
## Cluster means:
##   Petal.Length Petal.Width
## 1     5.595833    2.037500
## 2     4.269231    1.342308
## 3     1.462000    0.246000
##
## Clustering vector:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
19  20
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
3   3
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38
39  40
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
3   3
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58
```

```
59  60
##   3   3   3   3   3   3   3   3   3   3   2   2   2   2   2   2   2   2
2   2
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78
79  80
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   1
2   2
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98
99 100
##   2   2   2   1   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
119 120
##   1   1   1   1   1   1   2   1   1   1   1   1   1   1   1   1   1   1
1   2
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138
139 140
##   1   1   1   1   1   1   2   1   1   1   1   1   1   1   1   1   1   1
2   1
## 141 142 143 144 145 146 147 148 149 150
##   1   1   1   1   1   1   1   1   1   1
##
## Within cluster sum of squares by cluster:
## [1] 16.29167 13.05769  2.02200
##  (between_SS / total_SS =  94.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
"tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"

  predicted_clusterforPetalPredictors <-
clustering_result_forPetalPredictors$cluster

  Prediction_Accuracy_PercentageforPetalPredictors =
(sum(predicted_clusterforPetalPredictors == FlowerData[,5])/nrow(FlowerData)
)* 100
  Prediction_Accuracy_PercentageforPetalPredictors

## [1] 32

  # Inference for Petal Length Vs Width on  Training Data: Number of Optimal
Centers k = 3, Prediction Accuracy = 32%


  ggplot(FlowerData,aes(Sepal.Length,Sepal.Width, color = Species)) +
geom_point()
```

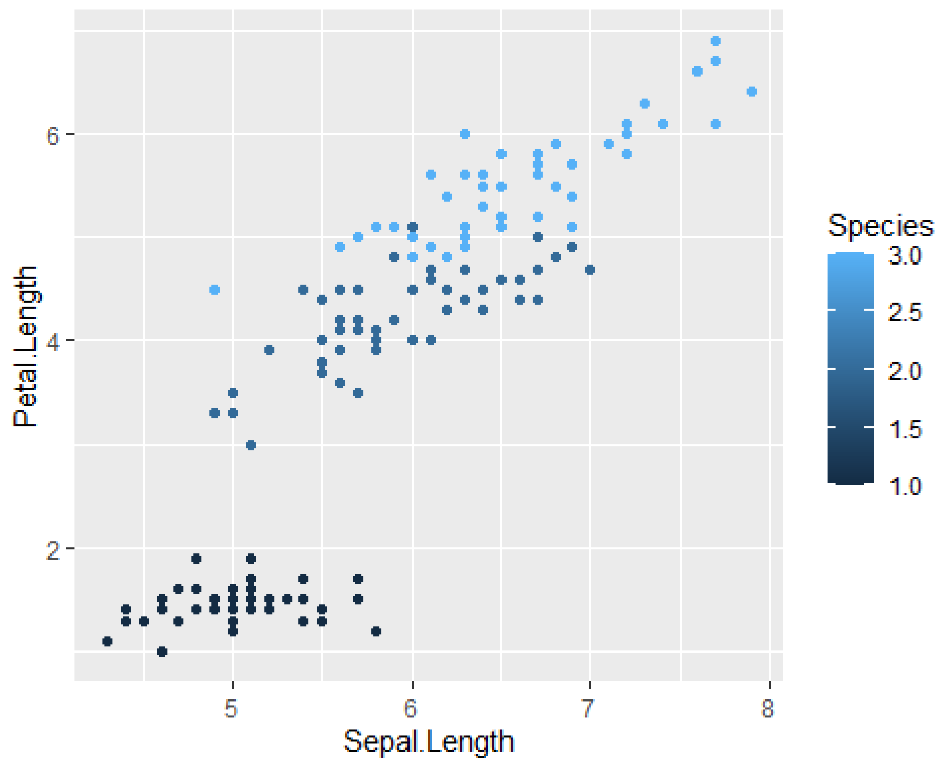



```
  #The Plot of Sepal length Vs Sepal Width
  # Consider only the predictors Petal Length and Petal Width columns to
input in the fviz_nbclus() function

  fviz_nbclust(FlowerData[,1:2],kmeans,method = "wss")
```

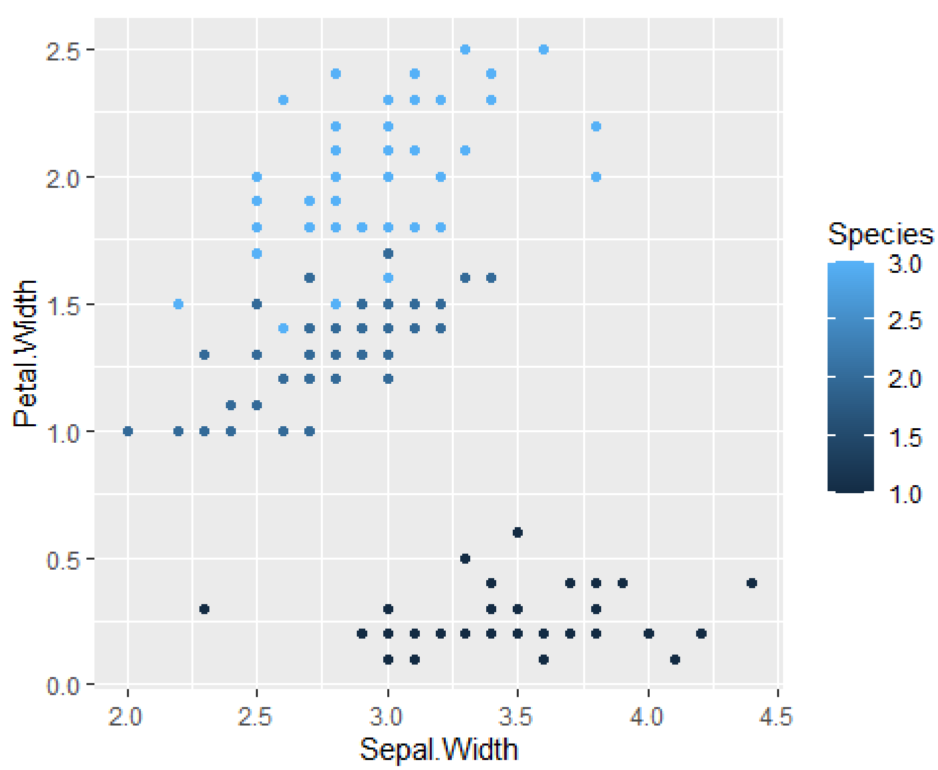



```
  #Based on the elbow method we can determine that k = 3 is the optimal value
of  K centered clusters.

  # Using the Kmeans()  function we can perform clustering

  clustering_result_forSepalPredictors = kmeans(FlowerData[,1:2] ,centers =
3, nstart =25)
  clustering_result_forSepalPredictors

## K-means clustering with 3 clusters of sizes 53, 50, 47
##
## Cluster means:
##   Sepal.Length Sepal.Width
## 1     5.773585    2.692453
## 2     5.006000    3.428000
## 3     6.812766    3.074468
##
## Clustering vector:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
19  20
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38
39  40
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
2   2
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58
```

```
59  60
##   2   2   2   2   2   2   2   2   2   2   3   3   3   1   3   1   3   1
3   1
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78
79  80
##   1   1   1   1   1   3   1   1   1   1   1   1   1   1   3   3   3   3
1   1
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98
99 100
##   1   1   1   1   1   1   3   1   1   1   1   1   1   1   1   1   1   1
1   1
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
119 120
##   3   1   3   3   3   3   1   3   3   3   3   3   3   1   1   3   3   3
3   1
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138
139 140
##   3   1   3   1   3   3   1   1   3   3   3   3   3   1   1   3   3   3
1   3
## 141 142 143 144 145 146 147 148 149 150
##   3   3   1   3   3   3   1   3   3   1
##
## Within cluster sum of squares by cluster:
## [1] 11.3000 13.1290 12.6217
##  (between_SS / total_SS =  71.6 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
"tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"

  predicted_clusterforSepalPredictors <-
clustering_result_forSepalPredictors$cluster

  Prediction_Accuracy_PercentageforSepalPredictors =
(sum(predicted_clusterforSepalPredictors == FlowerData[,5])/nrow(FlowerData)
)* 100
  Prediction_Accuracy_PercentageforSepalPredictors

## [1] 23.33333

  # Inference for Training Data: Number of Optimal Centers k = 3, Prediction
Accuracy = 23.3333%

  # Similarly consider other combinations of predictors for ploting and
analysing

  ggplot(FlowerData,aes(Sepal.Length,Petal.Length, color = Species)) +
geom_point()
```




```r
  ggplot(FlowerData,aes(Sepal.Width,Petal.Width, color = Species)) +
geom_point()
```



```
  #*Inference of Question 4.2
  #* When the Clustereing is done for all the Flower Data set,
  #* Elbow Diagram predicted that the Optimal Centers k=3 and the kmeans 
perdicted the Accuracy as 89.52318%
  #* Clustering for only two predictors - Petal Length and Petal Width are 
taken
  #*Elbow Diagram predicted that the Optimal Centers k=3 and the kmeans 
perdicted the Accuracy as 32%
  #* Clustering for only two predictors - Sepal Length and Sepal Width are 
taken
  #*Elbow Diagram predicted that the Optimal Centers k=3 and the kmeans 
perdicted the Accuracy as 23%
  #*So the Best Clustering Model is done when we consider all the four 
predictors - Petal.Length,Petal.Width
  #*Sepal.length and Sepal.Width with K =3 and Accuracy = 89.52318%
```