ISyE 6501-HOMEWORK 2

Group Members:
Jinyu, Huang | jhuang472@gatech.edu | GTID: 903522245
Chengqi, Huang | chengqihuang@gatech.edu | GTID: 903534690
Yuefan, Hu | yuefanhu@gatech.edu | GTID:903543027
Jingyu, Li | alanli@gatech.edu | GTID: 903520148

Qusetion 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)
Answer

1. Cross Validation for KNN

We use 10-fold cross validation to explore the KNN model for the "Credit Approval Data Set". We randomly choose 80% of the data set for cross validation(training and validation), and the remaining 20% for testing.

```
> library("kknn")
> set.seed(1234)
> data = read.table("credit_card_data-headers.txt", header=TRUE)  # import data
> data[,11] = as.factor(data[,11])  # change the dependent variable into norminal
> index = sample(x=2, size=nrow(data), replace=TRUE, prob=c(0.8,0.2))
>                                   # split data for cross validation and testing
> data_cross_validation = data[index==1,]
> data_testing = data[index==2,]
```

Based on our initial exploration in Homework 1, We apply cross validation to the following different KNN model settings(*distance=1 or 2, kernel="rectangular" or "triangular" or "optimal", k=5 to 15*), so that we can select the best model among them. We use kknn library's cross validation function *cv.kknn* to analyze the data.

```
> cross_validation_knn = function(dis_type, kernel_type){
+   # output 10-fold cross validation results from k=5 to k=15,
+   # in which distance and kernel as parameters
+   set.seed(1234)
+   results = vector()
+   for (i in seq(5,15)){
+     cvknn_model = cv.kknn(R1~., data_cross_validation, kcv=10,
+                           distance=dis_type, kernel=kernel_type,
+                           k=i, scale=TRUE)
+             # use kknn library's cross validation function cv.kknn
+     results[i-4] = round(cvknn_model[[2]]*100,3)
+             # average accuracy(%) in 10-fold cross validation for a certain model
+   }
+   return (results)
+ }
>
> setting_combination = matrix(nrow=6,ncol=2)  # store the setting parameters
> setting_combination[,1] = c(1,2,1,2,1,2)  # distance parameter
> setting_combination[,2] = c("rectangular","rectangular",
+                             "triangular","triangular",
+                             "optimal","optimal")  # kernel parameter
>
```

```
> compare_table = matrix(nrow=11, ncol=7)  # store the accuracy value of different models
> compare_table[,1] = c(5,6,7,8,9,10,11,12,13,14,15)  # k values for knn model
> colnames(compare_table) = c("k","Euc.dis_rec","Min.dis_rec",
+                             "Euc.dis_tri","Min.dis_tri",
+                             "Euc.dis_opt","Min.dis_opt")
>
> for (i in seq(1,nrow(setting_combination))){
+           # traverse different distance and kernel settings
+   dis_type = setting_combination[i,1]
+   kernel_type = setting_combination[i,2]
+   compare_table[,i+1] = cross_validation_knn(dis_type, kernel_type)
+ }
>
> cat("Accuracy for Different Models\n"); compare_table
```

```
Accuracy for Different Models
```

```
       k Euc.dis_rec Min.dis_rec Euc.dis_tri Min.dis_tri Euc.dis_opt
 [1,]  5      91.137      92.293      89.210      89.595      88.632
 [2,]  6      91.908      91.908      90.944      91.522      90.944
 [3,]  7      91.908      91.908      89.403      90.751      89.981
 [4,]  8      91.908      93.064      91.908      92.486      91.522
 [5,]  9      91.329      92.100      92.293      91.715      92.678
 [6,] 10      90.751      91.137      91.137      91.329      91.137
 [7,] 11      91.137      90.944      89.595      91.137      90.751
 [8,] 12      91.908      92.100      92.293      92.486      92.486
 [9,] 13      91.522      90.751      92.293      92.293      92.293
[10,] 14      90.173      90.559      91.329      91.908      92.100
[11,] 15      90.751      91.329      91.908      92.293      92.100
      Min.dis_opt
 [1,]      89.017
 [2,]      91.715
 [3,]      90.173
 [4,]      91.522
 [5,]      91.329
 [6,]      91.137
 [7,]      90.944
 [8,]      92.293
 [9,]      92.100
[10,]      91.329
[11,]      91.522
```

```
> max_location = which(compare_table==max(compare_table), arr.ind=TRUE)
>         # find the row number and column num of maximum accuracy
> cat("The maximum accuracy is at: row",max_location[1,1],", col",max_location[1,2])
```

```
The maximum accuracy is at: row 4 , col 3
```

In general, the accuracies vary not much among different models. Based on the cross validation results we get, the KNN model with *k=8, distance=2, kernel="rectangular"* has the best fitting performance(93.06%). Then we use testing data set to evaluate the quality of this model. We found that the accuracy performance for this model is 86.67%. This result also illustrates that generally the observed best model's quality in validation tend to be optmistic.

```
> selected_model = kknn(R1~.,data_cross_validation, data_testing,
+                       distance=2, kernel="rectangular", k=8,scale=TRUE)
```

```
> fitting_result = table(data_testing$R1, fitted(selected_model))
>         # counting table for predicting category against real category
> matching_percentage = round(sum(fitting_result[1,1],fitting_result[2,2])
+                             /sum(fitting_result)*100,3)
> matching_percentage
```

```
[1] 86.667
```

2. Cross Validation for SVM

Then we use 10-fold cross validation to explore the SVM model for the "Credit Approval Data Set". Same as what we do for KNN model, we randomly choose 80% of the data set for cross validation(training and validation), and the remaining 20% for testing.

```
> library("kernlab")
> set.seed(1234)
> data = data.matrix(read.table("credit_card_data-headers.txt", header=TRUE))
>                               # import data and change to matrix
> index = sample(x=2, size=nrow(data), replace=TRUE, prob=c(0.8,0.2))
>                               # split data for cross validation and testing
> data_cross_validation = data[index==1,]
> data_testing = data[index==2,]
```

We set *cross=10* in *ksvm* model to run 10-fold cross validation. In the output of the model, *model@cross* is the average proprtion of misclassifcation of the ten rounds of validating training models. So the average accuracy is one minus *model@cross*. We build a function to explore the SVM model with different parameter settings(kernels and C values), and the function will return the accuracy of each model.

```
> cross_validation_svm = function(kernel_type, C_values){
+           # output 10-fold cross validation results for each SVM model
+   set.seed(1234)
+   cvsvm_model = ksvm(data_cross_validation[,1:10],data_cross_validation[,11],
+                      type="C-svc", kernel=kernel_type, C=C_values,
+                      cross=10, scaled=TRUE)
+   return (round((1-cvsvm_model@cross)*100,3))  # accuracy in percentage
+ }
```

We choose among models with different kernels and C values. Considering the computatioanl efficiency, we just choose several values as examples. According to the results, several models' shows the same best performance, whose accuracy equals to 85.94%. We choose the relatively simple one, linear classifier, with *kernel="vanilladot"* and *C=100*. Then we use testing data to evaluate its quality.

```
> C_set = c(1,100,10000)  # C values
> kernel_set = c("rbfdot", "polydot", "vanilladot", "tanhdot", "anovadot")  # kernels
>
> compare_table = matrix(nrow=3, ncol=6)  # store the accuracy value of different models
> compare_table[,1] = C_set  # C values for ksvm model
> colnames(compare_table) = c('C',"rbfdot","polydot","vanilladot","tanhdot","anovadot")
>
> for (i in seq(1,length(C_set))){
+   for (j in seq(1,length(kernel_set))){
+     compare_table[i,j+1] = cross_validation_svm(kernel_set[j], C_set[i])
+   }
+ }
```

```
> compare_table
```

```
         C rbfdot polydot vanilladot tanhdot anovadot
```

```
[1,]     1 85.928  85.943     85.943  74.389   85.943
[2,]   100 79.955  85.943     85.943  73.235   82.466
[3,] 10000 77.655  85.943     85.943  74.005   82.274
```

We use the whole cross validation data set to run the SVM model with *kernel="vanilladot"* and *C=100*. Based on the testing result, the fitting accuracy of the model is 87.41%. And the function of the classifier is: $-0.0000519X_1 - 0.0000618X_2 + 0.0000228X_3 + 0.0001391X_4 + 0.9986232X_5 + 0.0000238X_6 - 0.00006X_7 - 0.0000657X_8 + 0.0000324X_9 + 0.0000032X_{10} + 0.0674232 = 0$

```
> selected_model = ksvm(data_cross_validation[,1:10],data_cross_validation[,11],
+                       type="C-svc", kernel="vanilladot", C=100, scaled=TRUE)
```

```
> pred = predict(selected_model, data_testing[,1:10])
> matching_percentage = round(sum(pred==data_testing[,11])/nrow(data_testing)*100,3)
> x <- selected_model@xmatrix[[1]]
> coe <- selected_model@coef[[1]]
> a <- colSums(x*coe)
> a0 <- -selected_model@b
> cat("Accuracy:",matching_percentage,"\n")
```

```
Accuracy: 87.407
```

```
> cat("classifier coefficients:\n");a
```

```
classifier coefficients:
```

```
           A1            A2            A3            A8            A9
-5.189643e-05 -6.182213e-05  2.277850e-05  1.391046e-04  9.986232e-01
          A10           A11           A12           A14           A15
 2.383889e-05 -6.004201e-05 -6.570307e-05  3.235040e-05  3.178269e-06
```

```
> cat("classifier intercept:\n");a0
```

```
classifier intercept:
```

```
[1] 0.06742318
```

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

Answer

1. KNN Model

We randomly split the data into 60%, 20% and 20% for training, validation and testing. We use KNN model to explore the classification based on "Credit Approval Data Set".

```
> library("kknn")
> set.seed(1234)
> data = read.table("credit_card_data-headers.txt", header=TRUE)  # import data
> data[,11] = as.factor(data[,11])  # change the dependent variable into norminal
> index = sample(x=3, size=nrow(data), replace=TRUE, prob=c(0.6,0.2,0.2))
>                                   # split data for training, validation and testing
> data_training = data[index==1,]
> data_validation = data[index==2,]
> data_testing = data[index==3,]
```

Similar to the process above, We use different KNN model settings(*distance=1 or 2, kernel="rectangular" or "triangular" or "optimal", k=5 to 15*), so that we can select the best model among them. The table below shows the model with *distance=1*, *kernel="optimal"* and *k=10* performs the best, which validating accuracy is 90.37%. The best model we get here is different with the one we get through cross validation in question a. We think it reflects the different random effect when we do validation with different data set, and the results

is not comparable.

```
> validation_knn = function(dis_type, kernel_type){
+   # output KNN model validation results, in which kernel and C as parameters
+   results = vector()
+   for (i in seq(5,15)){
+     knn_model = kknn(R1~., data_training, data_validation,
+                      distance=dis_type, kernel=kernel_type, k=i, scale=TRUE)
+     fitting_result = table(data_validation$R1, fitted(knn_model))
+             # counting table for predicting category against real category
+     results[i-4] = round(sum(fitting_result[1,1],fitting_result[2,2])/
+                            sum(fitting_result)*100,3)
+                          # accuracy(%) for validation for different k values
+   }
+   return (results)
+ }
>
> setting_combination = matrix(nrow=6,ncol=2)  # store the setting parameters
> setting_combination[,1] = c(1,2,1,2,1,2)  # distance parameter
> setting_combination[,2] = c("rectangular","rectangular",
+                             "triangular","triangular",
+                             "optimal","optimal")  # kernel parameter
>
> compare_table = matrix(nrow=11, ncol=7)  # store the accuracy value of different models
> compare_table[,1] = c(5,6,7,8,9,10,11,12,13,14,15)  # k values for knn model
> colnames(compare_table) = c("k","Euc.dis_rec","Min.dis_rec",
+                             "Euc.dis_tri","Min.dis_tri",
+                             "Euc.dis_opt","Min.dis_opt")
>
> for (i in seq(1,nrow(setting_combination))){
+           # traverse different distance and kernel settings
+   dis_type = setting_combination[i,1]
+   kernel_type = setting_combination[i,2]
+   compare_table[,i+1] = validation_knn(dis_type, kernel_type)
+ }
>
> cat("Accuracy for Different Models\n");compare_table
```

```
Accuracy for Different Models
```

	k	Euc.dis_rec	Min.dis_rec	Euc.dis_tri	Min.dis_tri	Euc.dis_opt	Min.dis_opt
[1,]	5	88.889	86.667	87.407	86.667	89.630	86.667
[2,]	6	88.889	86.667	88.148	86.667	88.889	88.148
[3,]	7	85.926	86.667	87.407	85.926	88.889	
[4,]	8	85.926	86.667	85.926	85.185	88.889	
[5,]	9	88.148	85.926	88.148	86.667	88.889	
[6,]	10	88.148	85.926	86.667	87.407	90.370	
[7,]	11	87.407	85.926	87.407	87.407	89.630	
[8,]	12	87.407	85.926	87.407	87.407	89.630	
[9,]	13	88.148	85.926	87.407	86.667	89.630	
[10,]	14	88.148	85.926	86.667	85.926	89.630	
[11,]	15	88.148	87.407	86.667	86.667	88.889	

```
 [3,]      88.148
 [4,]      89.630
 [5,]      88.889
 [6,]      88.889
 [7,]      88.889
 [8,]      88.889
 [9,]      88.148
[10,]      87.407
[11,]      87.407
```

```
> max_location = which(compare_table==max(compare_table), arr.ind=TRUE)
>         # find the row number and column num of maximum accuracy
> cat("The maximum accuracy is at: row",max_location[1,1],", col",max_location[1,2])
```

```
The maximum accuracy is at: row 6 , col 6
```

Then we use testing data set to evaluate the quality of the model. We found that the accuracy performance for this model is 82.48%. Again, this result indicates that generally the observed model quality for the best one in validation tend to be optmistic.

```
> selected_model = kknn(R1~.,data_training, data_testing,
+                       distance=1, kernel="optimal", k=10,scale=TRUE)
> fitting_result = table(data_testing$R1, fitted(selected_model))
>         # counting table for predicting category against real category
> matching_percentage = round(sum(fitting_result[1,1],fitting_result[2,2])/
+                             sum(fitting_result)*100,3)
> matching_percentage
```

```
[1] 82.482
```

2. SVM Model

Similarlly, we do 60-20-20 data splitting and then use SVM model to explore the classification based on "Credit Approval Data Set".

```
> library("kernlab")
> set.seed(1234)
> data = data.matrix(read.table("credit_card_data-headers.txt", header=TRUE))
>                             # import data and change to matrix
> index = sample(x=3, size=nrow(data), replace=TRUE, prob=c(0.6,0.2,0.2))
>         # split data for training, validation and testing
> data_training = data[index==1,]
> data_validation = data[index==2,]
> data_testing = data[index==3,]
```

We explore and compare different SVM models(different kernels and C values) by applying ksvm function.

```
> validation_svm = function(kernel_type, C_values){
+   # output SVM model validation results, in which kernel and C as parameters
+   svm_model = ksvm(data_training[,1:10], data_training[,11],
+                    type="C-svc", kernel=kernel_type, C=C_values, scaled=TRUE)
+   pred = predict(svm_model, data_validation[,1:10])
+   return (round(sum(pred==data_validation[,11])/nrow(data_validation)*100,3))
+           # return the validating results
+ }
>
> C_set = c(1,100,10000)  # C values
> kernel_set = c("rbfdot","polydot","vanilladot","tanhdot","anovadot")  # kernels
```

```
>
> compare_table = matrix(nrow=3, ncol=6)  # store the accuracy value of different models
> compare_table[,1] = C_set  # C values for ksvm model
> colnames(compare_table) = c('C',"rbfdot", "polydot",
+                             "vanilladot", "tanhdot", "anovadot")
>
> for (i in seq(1,length(C_set))){
+   for (j in seq(1,length(kernel_set))){
+     compare_table[i,j+1] = validation_svm(kernel_set[j], C_set[i])
+   }
+ }
```

Based on the results below, there are several models' performance are nearly the same. The maximum fitting accuracy is 88.9% with *kernel="anovadot"* and *C=100*.

```
> compare_table
```

```
         C rbfdot polydot vanilladot tanhdot anovadot
[1,]     1 84.444  87.407     87.407  74.815   87.407
[2,]   100 79.259  87.407     87.407  75.556   88.889
[3,] 10000 79.259  87.407     87.407  75.556   87.407
```

Then we use testing data set to evaluate the model we select via validation. The accuracy of the model is 72.99%, which is lower than the validation results. This result also illustrates that generally the observed model quality in validation tend to be optmistic. And the function for classifier is:
$-432.30X_1 - 314.35X_2 + 451.84X_3 + 185.99X_4 + 2601.73X_5 - 1413.44X_6 + 1441.54X_7 - 400.71X_8 - 16.04X_9 + 748.95X_{10} - 17.32 = 0$

```
> selected_model = ksvm(data_training[,1:10], data_training[,11],
+                       type="C-svc", kernel="tanhdot", C=100, scaled=TRUE)
> pred = predict(selected_model, data_testing[,1:10])
> matching_percentage = round(sum(pred==data_testing[,11])/nrow(data_testing)*100,3)
> x <- selected_model@xmatrix[[1]]
> coe <- selected_model@coef[[1]]
> a <- colSums(x*coe)
> a0 <- -selected_model@b
```

```
> cat("Accuracy:",matching_percentage,"\n")
```

```
Accuracy: 72.993
```

```
> cat("classifier coefficients:\n");a
```

```
classifier coefficients:
```

```
         A1          A2          A3          A8          A9         A10
 -432.29598  -314.34970   451.84256   185.98624  2601.73457 -1413.43665
        A11         A12         A14         A15
 1441.54102  -400.71373   -16.03503   748.95013
```

```
> cat("classifier intercept:\n");a0
```

```
classifier intercept:
```

```
[1] -17.31583
```

Qusetion 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

Supplier clustering:

Suppose we are running a plant, we need to divide our current suppliers for a single type of material into clusters in order to better integrate resources to optimize our put-away and manufacture process, as well as reduce cost. The features we select are as following:

1)Delivery-time: ratio variable, length of time from the date the order is made to the time of receiving material(often in days);

Reason: The suppliers whose delivery time is relatively longer will have a negative effect on the production.

2)On Time Delivery Rate: ratio variable, $\frac{actual\ material\ delivered\ on\ time\ by\ quantity}{total\ material\ delivered\ confirmed\ by\ order} \times 100\%$;

Reason: Suppliers with lower on-time delivery rate will increase uncertainty.

3)Qualification Rate: ratio variable, $\frac{qualified\ incoming\ batch}{total\ incoming\ batch} \times 100\%$;

Reason: If the supplier has relatively low qualification rate, which means the materials are damaged and can't be used, it will add extra costs.

4)Price: ratio variable, price offered by different suppliers for the same type of material;

Reason: Suppliers who offers high price material will endanger our own actual profit rate.

Qusetion 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.

Answer

After inputing the raw data, we scale the features and use the whole data set for clustering.

```
> data = iris
> data[,1:4] = scale(data[,1:4], center=TRUE, scale=TRUE) # scale the features
```

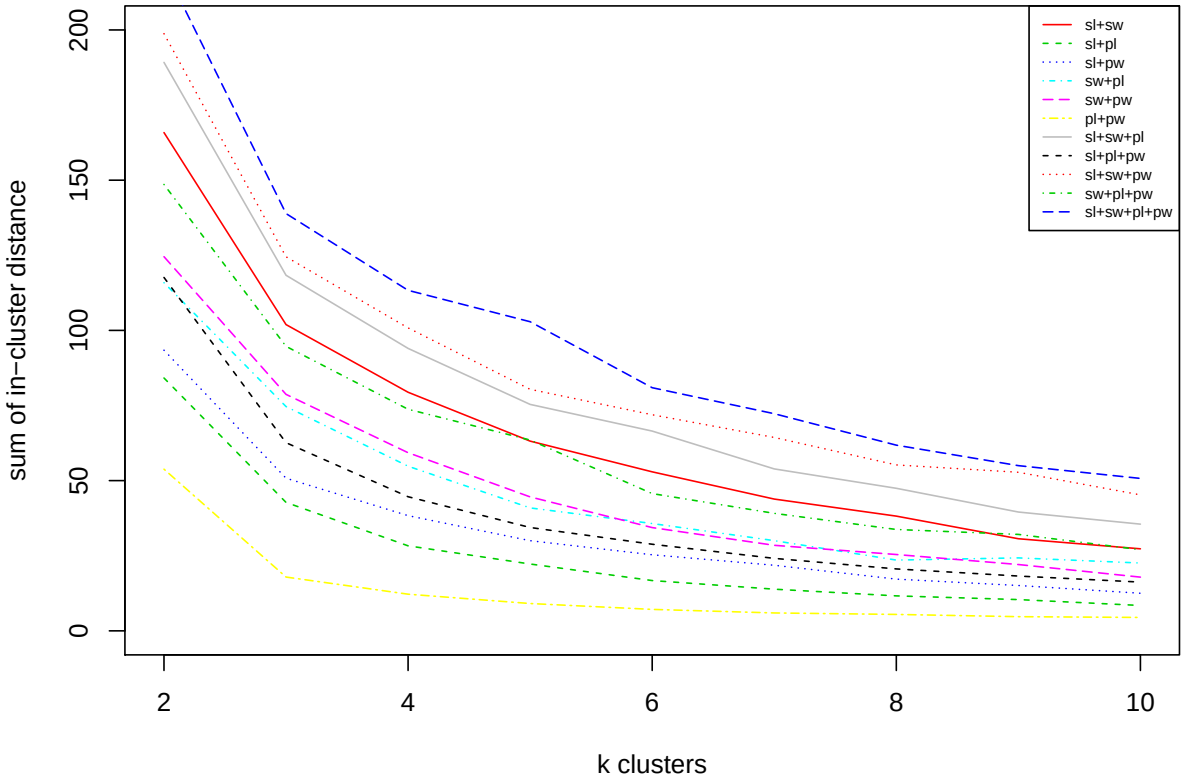
We use the enumeration approach to explore the best combination of predictors(*Sepal.Length*, *Sepal.Width*, *Petal.Length* and *Petal.Width*) and number of clusters. Specificlly, the range of k in our trails is 2 to 10, and all combinations of the four predictors are included in our trails. We store the total within-cluster sum of squares (*model[["tot.withinss"]]*) for comparison and finding the best k.

```
> k_set = c(2,3,4,5,6,7,8,9,10)  # k values
>
> compare_table = matrix(nrow=9, ncol=12)
>         # store the total distance from each point
>         # to the cluster center of different models
> compare_table[,1] = k_set  # k values for kmeans model
> colnames(compare_table) = c('k',"sl+sw","sl+pl","sl+pw","sw+pl","sw+pw","pl+pw",
+                             "sl+sw+pl","sl+pl+pw","sl+sw+pw","sw+pl+pw","sl+sw+pl+pw")
> for (i in k_set){
+   kmeans_model = kmeans(data[,c(1,2)], centers=i, iter.max=1000,
+                         nstart=5, algorithm="Hartigan-Wong")
+   compare_table[i-1,2] = round(kmeans_model[["tot.withinss"]],3)
```

```
+   kmeans_model = kmeans(data[,c(1,3)], centers=i, iter.max=1000,
+                         nstart=5, algorithm="Hartigan-Wong")
+   compare_table[i-1,3] = round(kmeans_model[["tot.withinss"]],3)
+   kmeans_model = kmeans(data[,c(1,4)], centers=i, iter.max=1000,
+                         nstart=5, algorithm="Hartigan-Wong")
+   compare_table[i-1,4] = round(kmeans_model[["tot.withinss"]],3)
+   kmeans_model = kmeans(data[,c(2,3)], centers=i, iter.max=1000,
+                         nstart=5, algorithm="Hartigan-Wong")
+   compare_table[i-1,5] = round(kmeans_model[["tot.withinss"]],3)
+   kmeans_model = kmeans(data[,c(2,4)], centers=i, iter.max=1000,
+                         nstart=5, algorithm="Hartigan-Wong")
+   compare_table[i-1,6] = round(kmeans_model[["tot.withinss"]],3)
+   kmeans_model = kmeans(data[,c(3,4)], centers=i, iter.max=1000,
+                         nstart=5, algorithm="Hartigan-Wong")
+   compare_table[i-1,7] = round(kmeans_model[["tot.withinss"]],3)
+   kmeans_model = kmeans(data[,c(1,2,3)], centers=i, iter.max=1000,
+                         nstart=5, algorithm="Hartigan-Wong")
+   compare_table[i-1,8] = round(kmeans_model[["tot.withinss"]],3)
+   kmeans_model = kmeans(data[,c(1,3,4)], centers=i, iter.max=1000,
+                         nstart=5, algorithm="Hartigan-Wong")
+   compare_table[i-1,9] = round(kmeans_model[["tot.withinss"]],3)
+   kmeans_model = kmeans(data[,c(1,2,4)], centers=i, iter.max=1000,
+                         nstart=5, algorithm="Hartigan-Wong")
+   compare_table[i-1,10] = round(kmeans_model[["tot.withinss"]],3)
+   kmeans_model = kmeans(data[,c(2,3,4)], centers=i, iter.max=1000,
+                         nstart=5, algorithm="Hartigan-Wong")
+   compare_table[i-1,11] = round(kmeans_model[["tot.withinss"]],3)
+   kmeans_model = kmeans(data[,c(1,2,3,4)], centers=i, iter.max=1000,
+                         nstart=5, algorithm="Hartigan-Wong")
+   compare_table[i-1,12] = round(kmeans_model[["tot.withinss"]],3)
+ }
```

We draw the line plot, whose horizontal axis is k values and vertical axis is the total within-cluster sum of squares. This plot shows the difference of in-group sum of squares among various feature combinations and also it shows how in_group sum of squares change along with number of cluster increasing. In general, the breaking points of each line are all at k=3. And the model with features *petal.length* and *petal.width* has the least in-cluster total distance comparing to other models with different feature combinations.

```
> plot(compare_table[,1], compare_table[,2],
+      type="n",
+      xlab="k clusters",
+      ylab="sum of in-cluster distance",
+      ylim=c(0,200))# build the plot
> for (i in seq(2,12)){  # draw lines
+   lines(compare_table[,1], compare_table[,i], col=i, lty=i-1)
+ }
> legend('topright',colnames(compare_table)[2:12],lty=1:11,col=2:12,cex=0.6)
```



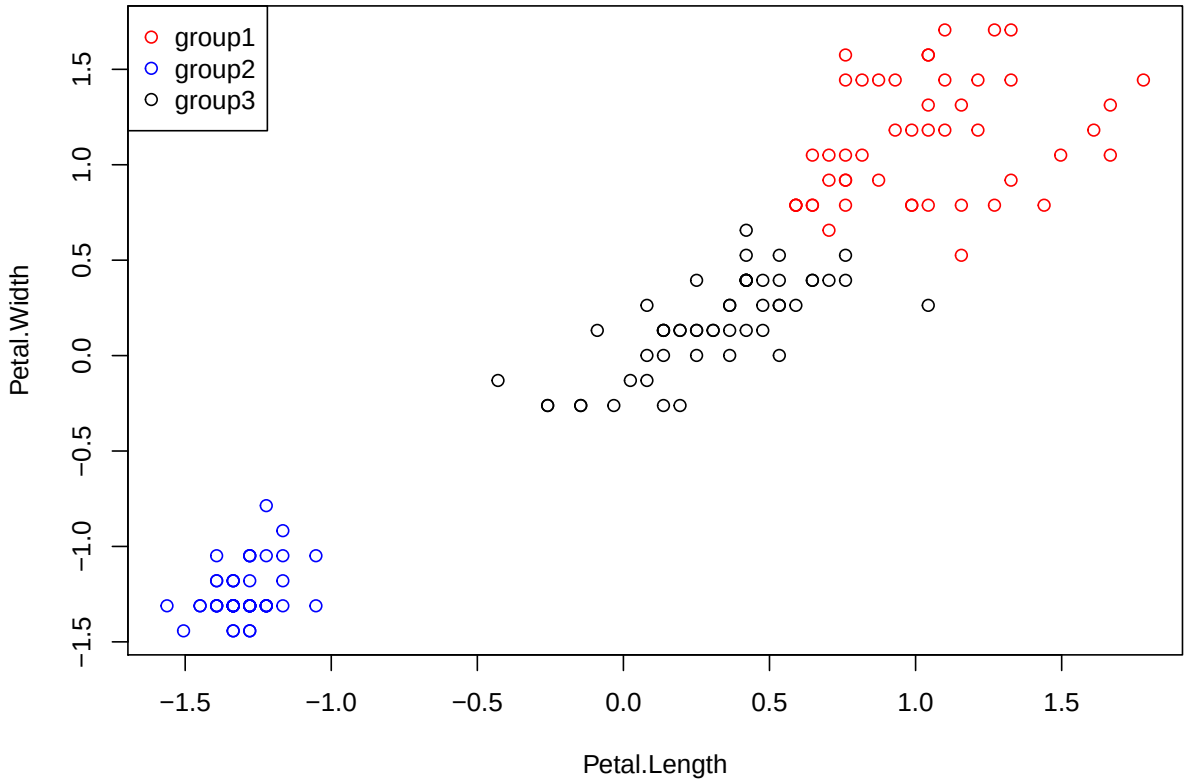
We run the clustering model with k=3 basing on *petal.length* and *petal.width* again. The group size of three clusters are 50, 58 and 52. The scatter plot for clustering is as following.

```
> kmeans_model = kmeans(data[,c(3,4)], centers=3, iter.max=1000,
+                       nstart=5, algorithm="Hartigan-Wong")
> cat("group size:\n");kmeans_model["size"]
```

```
group size:

$size
[1] 48 50 52
```

```
> data[,6] = kmeans_model["cluster"]
> plot(data[,3], data[,4],
+      type="n",
+      xlab="Petal.Length",
+      ylab="Petal.Width")# build the plot
> points(data[data["cluster"]==1,3],data[data["cluster"]==1,4],col="red")
> points(data[data["cluster"]==2,3],data[data["cluster"]==2,4],col="blue")
> points(data[data["cluster"]==3,3],data[data["cluster"]==3,4],col="black")
> legend("topleft",c("group1","group2","group3"),pch=1,col=c("red","blue","black"))
```



According to the scatter plot, the cluster at the bottomleft clearly separetes with the other two. But as for the points in topright, they spread continuously in the plane. So what if we just merge these two clusters into one, which means we set k equal to 2? We try another widely used indicator to compare how well each data point has been clustered between k=2 and k=3, which is called *Silhouette*. The concept of silhouette is to measure how close(or similar) one object is to its own group compared to other groups. And the value of silhouette is from -1 to 1, and higher value indicates that the object is well matched to its group and is away from other groups. As the results below, silhouette value is 0.74 when k=2, which is higher than the value when k=3. So based on this indicator, two cluster appear to be a good choice.

```
> library(cluster)
> library(fpc)
> kmeans_k_3 = kmeans(data[,c(3,4)], centers=3, iter.max=1000,
+                       nstart=5, algorithm="Hartigan-Wong")
>                       # kmeans model for k=3
> km_stats1 = cluster.stats(d=dist(data[,3:4],method="minkowski"),
+                           kmeans_k_3[["cluster"]],silhouette=TRUE)
>                       # compute "silhouette"
> cat("slihouette for k=3:\n")
```

```
slihouette for k=3:
```

```
> km_stats1$avg.silwidth  # average slihouette value for all points
```

```
[1] 0.6741313
```

```
> km_stats1$clus.avg.silwidths  # average slihouette value for different clusters
```

```
        1         2         3
0.5103231 0.6038153 0.9045159
```

```
> kmeans_k_2 = kmeans(data[,c(3,4)], centers=2, iter.max=1000,
+                       nstart=5, algorithm="Hartigan-Wong")
>                       # kmeans model for k=2
> km_stats2 = cluster.stats(d=dist(data[,3:4],method="minkowski"),
+                           kmeans_k_2[["cluster"]],silhouette=TRUE)
>                       # compute "silhouette"
> cat("slihouette for k=2:\n")
```

```
slihouette for k=2:
```

```
> km_stats2$avg.silwidth  # average slihouette value for all points
```

```
[1] 0.743372
```

```
> km_stats2$clus.avg.silwidths  # average slihouette value for different clusters
```

```
        1         2
0.6526172 0.9248815
```

We propose that because clustering is an exploratory analytic method and we don't know the true results, there is no way to choose an absolutely "right" model. We can use some indicators to evaluate different models. But in fact, different indicators have different characters and emphases, so the results may vary. As what we see in this case, if we use the elbow method and decide by the line plot, k=3 is a better choice than k=2. It's not hard to imagine that although the points at topright are continuous, they spread in a wide range. So if we cut them into two groups, the in-group sum of squares will have a significant decrease for the whole data set. On the other hand, silhouette considers not only in-group distance, but also out-group distance. Conceptually, if we cut a bunch of nearly continuous points into two parts, compared with the "gain" for in-group distance, "loss" in out-group distance will be larger. So clustering with k=2 will perform better on this indicator. To summarize, we believe that it's better if we can have some theory base or hypothesis for clustering, acting as a top-down support to build the model and to interpret the results we get, which may be better than a pure data-driven clustering.