HW #2

Code

Question 1 * SVM and KNN models with cross validation and train/test/validation splits*

Loading the data and required packages

Hide

Hide

```
require("kernlab")
```

```
Loading required package: kernlab
```

Hide

Hide

```
require("kknn")
```

```
Loading required package: kknn
```

Hide

Hide

```
df <- read.table("https://d37djvu3ytnwxt.cloudfront.net/assets/course
ware/v1/39b78ff5c5c28981f009b54831d81649/asset-v1:GTx+ISYE6501x+2T201
7+type@asset+block/credit_card_data-headers.txt", header = TRUE)
print(dim(df))
```

```
[1] 654  11
```

Hide

Hide

```
head(df)
```

	A1	A2	A3	A8	A9	A10	A11	A12	A14
	<int>	<dbl>	<dbl>	<dbl>	<int>	<int>	<int>	<int>	<int>
1	1	30.83	0.000	1.25	1	0	1	1	202
2	0	58.67	4.460	3.04	1	0	6	1	43

3	0	24.50	0.500	1.50	1	1	0	1	280
4	1	27.83	1.540	3.75	1	0	5	0	100
5	1	20.17	5.625	1.71	1	1	0	1	120
6	1	32.08	4.000	2.50	1	1	0	0	360

6 rows | 1-10 of 11 columns

To compare multiple models, the data will be split into 3 sets: training (to fit the model), validation (to compare different models) and test (to compare between the best models evulated from the validation set).

The dataset will be split into these parts as such:

Training - *60%*

Validation - *20%*

Test - *20*

Hide

Hide

```
# Setting a seed for reproducible results
set.seed(123)
# A function for splitting a dataframe into splits
train_test_val_split <- function(df, train_split = .7, val_split = 0,
test_split = .3){
  # Split a dataframe/matrix into training, validation and test split
s

  # Args
  # df - dataframe/matrix - the orginal dataset to split
  # train_split - float - the percentage (as a decimal) of the df to
retain as a training set.
  #                       Defaults to 70%
  # val_split - float - The percentage (as a decimal) of the df to re
tain as a validation set.
  #                     Defaults to 0%
  # test_split - float - The percentage (as a decimal) of the df to r
etain as a test set.
  #                     Defaults to 30%



  # Shuffling the rows of the dataframe to randomize before splitting
  df <- df[sample(nrow(df)),]
  # Forking into an if/else depending on whether a validation set is 
desired
  if (val_split != 0){
    # Finding the indexes to split our df at
    train_split_end_index <- round(nrow(df)*train_split)

    test_split_start_index <- train_split_end_index + 1

    test_split_end_index <- test_split_start_index + round(nrow(df)*t
est_split)

    validation_split_start_index <- test_split_end_index + 1

    validation_split_end_index <- nrow(df)

    # Creating our df splits
    train <- df[0:train_split_end_index,]

    test <- df[test_split_start_index:test_split_end_index,]

    validation <- df[validation_split_start_index:validation_split_en
d_index,]

    # Must wrap our df in a list as R doesn't allow multiple values t
o be returned
    output <- list(train, test, validation)
  } else {
    # Finding the indexes to split our df at
    train_split_end_index <- round(nrow(df)*train_split)

    test_split_start_index <- train_split_end_index + 1

    test_split_end_index <- nrow(df)

    # Creating our df splits
    train <- df[0:train_split_end_index,]

    test <- df[test_split_start_index:test_split_end_index,]

    # Must wrap our df in a list as R doesn't allow multiple values t
o be returned
    output <- list(train, test)
  }

  return(output)
```

```
}
```

Testing the function

Hide

Hide

```
splits = train_test_val_split(df, train_split = .6, val_split = .2, t
est_split = .2)
print(class(splits[[1]]) )
```

```
[1] "data.frame"
```

Hide

Hide

```
print(dim(splits[[1]]))
```

```
[1] 392  11
```

Hide

Hide

```
print(class(splits[[2]]) )
```

```
[1] "data.frame"
```

Hide

Hide

```
print(dim(splits[[2]]))
```

```
[1] 132  11
```

Hide

Hide

```
print(class(splits[[3]]) )
```

```
[1] "data.frame"
```

Hide

Hide

```
print(dim(splits[[3]]))
```

```
[1] 130  11
```

Hide

Hide

```
splits = train_test_val_split(df, train_split = .6, test_split = .2)
print(class(splits[[1]]) )
```

```
[1] "data.frame"
```

Hide

Hide

```
print(dim(splits[[1]]))
```

```
[1] 392  11
```

Hide

Hide

```
print(class(splits[[2]]) )
```

```
[1] "data.frame"
```

Hide

Hide

```
print(dim(splits[[2]]))
```

```
[1] 262  11
```

Hide

Hide

```
# There shouldn't be a validation split in this instance
#print(class(splits[[3]]) )
#print(dim(splits[[3]]))
```

Creating dataframes for training, validation and test sets **Note to grader** Because my train/test/validation function shuffles the df prior to splitting, the exact accuracy and potentially parameters of the following models may not exactly match when you run this code. I set a seed for reproducible results but results may differ if that doesn’t work.

Hide

Hide

```
splits = train_test_val_split(df, train_split = .6, val_split = .2, t
est_split = .2)
# Our Splits from indexing the returned list
train <- splits[[1]]
test <- splits[[2]]
validation <- splits[[3]]
# Sanity Check
print(class(splits[[1]]) )
```

```
[1] "data.frame"
```

Hide

Hide

```
print(dim(splits[[1]]))
```

```
[1] 392  11
```

Hide

Hide

```
print(class(splits[[2]]) )
```

```
[1] "data.frame"
```

Hide

Hide

```
print(dim(splits[[2]]))
```

```
[1] 132  11
```

Hide

Hide

```
print(class(splits[[3]]) )
```

```
[1] "data.frame"
```

Hide

Hide

```
print(dim(splits[[3]]))
```

```
[1] 130  11
```

Now to train and test various models on the validation set

Hide

Hide

```
# One potential model
# This is looking through many different kernels and has a max k valu
e of 250
# train.knn utilizes cross validation as long as the parameter kcv is
set to a value
# Hypothetically, this should come up with our best parameters and ac
curacy on the train and test set
# However, this could be a recipe for overfitting
knn_model <- train.kknn(R1 ~ .,
                        # Train on training set
                        data = train,
                        # Test on validation set
                        test= validation,
                        # Use 5 fold cross validation
                        kcv = 5,
                        # Scale our data
                        scaled=TRUE,
                        # Try multiple k values with the max being 10
0

                        kmax = 100,
                        # Try multiple kernels
                        kernel = c("optimal", "triangular", "rectangu
lar",
           "epanechnikov","cos","inv", "gaussian","triweight","biweig
ht"))
# Our model summary
summary(knn_model)
```

```


Call:
train.kknn(formula = R1 ~ ., data = train, kmax = 100, kernel = c("op
timal",     "triangular", "rectangular", "epanechnikov", "cos", "inv"
,     "gaussian", "triweight", "biweight"), test = validation,     kc
v = 5, scaled = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1964286
Minimal mean squared error: 0.1073078
Best kernel: inv
Best k: 14
```

The best model (on my splits) uses a k of 14 and the ‘inv’ kernel.

Testing on the validation set

Hide

Hide

```
# Predicting on our validation set
pred_knn <- round(predict(knn_model, validation[,1:10]))
print("Prediction % =")
```

```
[1] "Prediction % ="
```

Hide

Hide

```
# Getting our simple accuracy - correct predictions/total predictions
print(sum(pred_knn == validation[,11]) / nrow(validation))
```

```
[1] 0.8692308
```

Let’s build an SVM model to compare

Hide

Hide

```
# ksvm requires matrices as inputs
# Transforming our data splits into matrices
ksvm_train <- as.matrix(train)
ksvm_validation <- as.matrix(validation)
ksvm_test <- as.matrix(test)
```

Training a model

Hide

Hide

```
# Training a ksvm model with cross-validation enabled
ksvm_model <- ksvm(
                  # Our data to train on (all variables)
                  ksvm_train[,1:10],
                  # Our target variable
                  ksvm_train[,11],
                  # 5 fold cross validation
                  cross=5,
                  # A classification SVM
                  type="C-svc",
                  # Linear kernel (the best one I found after trying
many)
                  kernel= "vanilladot",
                  # Default C value
                  C=1,
                  # Scaled data
                  scaled=TRUE)
```

```
 Setting default kernel parameters
```

Hide

Hide

```
# predicitions using the svm model on the dataset
pred_svm <- predict(ksvm_model, ksvm_validation[,1:10])
accuracy <- sum(pred_svm == validation[,11])/nrow(ksvm_validation)
print("Accuracy:")
```

```
[1] "Accuracy:"
```

Hide

Hide

```
print(round(accuracy, 6))
```

```
[1] 0.861538
```

After trying multiple models the vanilladot, polydot, and laplace all returned an accuracy of roughly 87% (on my splits). For simplicity sake, I'll stick with the vanilladot linear kernel.

Both models scored similar accuracy. Let's see which one performs better on the test (hold out) set

Hide

Hide

```
# Predictions with our SVM model on the test set
pred_svm_test <- predict(ksvm_model, ksvm_test[,1:10])
accuracy <- sum(pred_svm_test == test[,11])/nrow(ksvm_validation)
print("Accuracy for SVM:")
```

```
[1] "Accuracy for SVM:"
```

Hide

Hide

```
print(round(accuracy, 6))
```

```
[1] 0.838462
```

Hide

Hide

```
# Predictions with our knn model on the test set
pred_knn_test <- round(predict(knn_model, test[,1:10]))
print("Accuracy for KNN:")
```

```
[1] "Accuracy for KNN:"
```

Hide

Hide

```
# Getting our simple accuracy - correct predictions/total predictions
print(sum(pred_knn_test == test[,11]) / nrow(test))
```

```
[1] 0.8106061
```

SVM outperformed the KNN model with a 83% accuracy vs. 81%. 83% is great and 81% is not bad either. In this case it would be worth digging a bit deeper into other metrics (confusion matrix, precision, recall, etc) and determine which model is better for the bank's goals. All things held equal and looking solely at accuracy of predictions, the SVM model outperforms the KNN model.

Question 2 *Real Life Clustering Situation*

In my job I've used different clustering algorithms on text documents to try and find latent topics within them. A common example of this is the Reuters news dataset, which you can cluster into seperate topics like 'news', 'international', 'financial' clusters among others.

In my own life I could cluster on something like drives from my home. Some of the predictors could be travel time, day of the week, time of day, direction, and average speed. The day could be clustered around these variables and seperate into clusters like 'daily commute', 'errands', 'weekend trip' etc.

Question 3 *K-Means on the Iris dataset*

Loading the dataset

Hide

Hide

```
url <- "https://d37djvu3ytnwxt.cloudfront.net/assets/courseware/v1/26
886db51f665dbde534f8c6326694b5/asset-v1:GTx+ISYE6501x+3T2017+type@ass
et+block/iris.txt"
df <- read.table(url, header = TRUE)
head(df)
```

	Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	Species <fctr>
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

6 rows

The K-means model is generally used for unsupervised learning and is not as common for prediction. For this problem I will create 3 clusters using K-means and compare how well those clusters predict the 3 flower species.

To validate such a low number of clusters, I will plot an elbow graph showing the within groups sum of squares by the number of clusters. As the wss(within groups sum of squares) decreases, the clusters are more tightly compacted.

Elbow Diagram

Hide

Hide

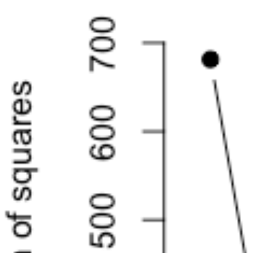
```
#Elbow Method for finding the optimal number of clusters
# Compute and plot wss for k = 2 to k = 15.
k.max <- 15
# Using the four descriptive variables
data <- df[1:4]
# For each value of k, computing the within groups sum of squares
wss <- sapply(1:k.max,
              function(k){kmeans(data, k, nstart=50,iter.max = 15 )$t
ot.withinss})
wss
```

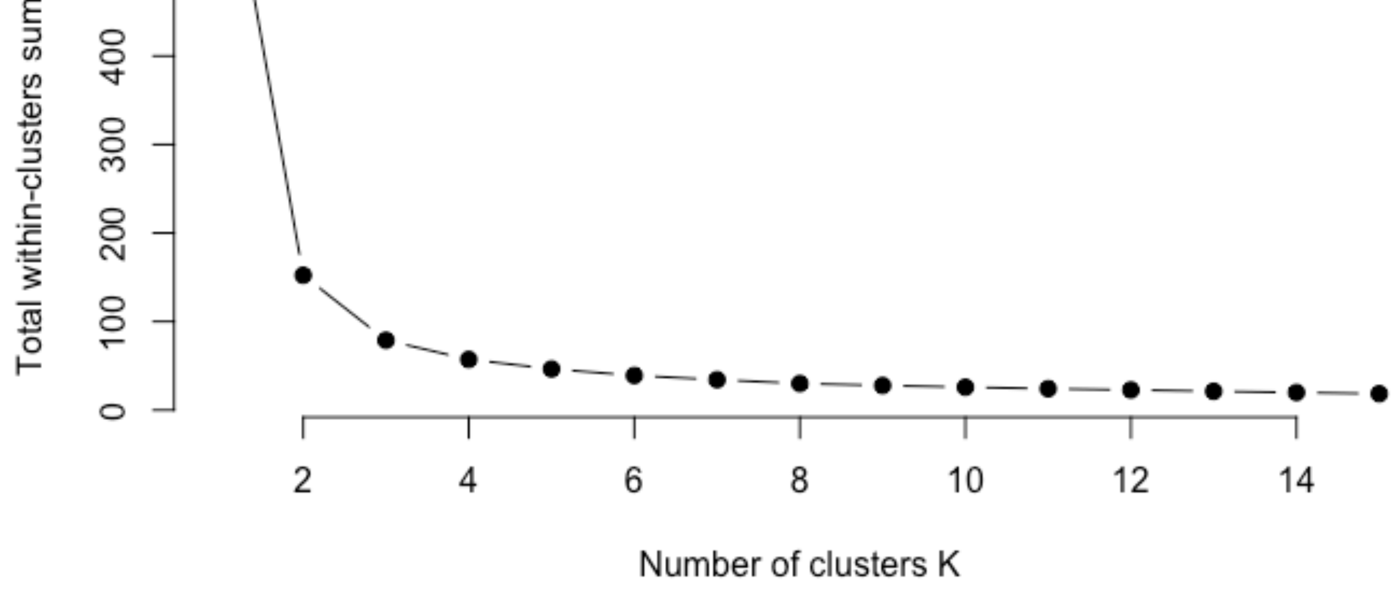
```
 [1] 681.37060 152.34795  78.85144  57.22847  46.44618  39.03999  34.
29823  29.98894  27.78609
[10]  25.83405  24.14778  22.74465  21.22433  19.85299  18.49200
```

Hide

Hide

```
# Plotting this
plot(1:k.max, wss,
     type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters K",
     ylab="Total within-clusters sum of squares")
```





From the diagram 2-3 clusters is sufficient. Now to find which combination of variables creates clusters that predict flower species type.

Hide

Hide

```
kmeans_cls <- kmeans(
                    # Variables to consider
                    df[,1:4],
                    # 3 clusters to match number of species
                    centers = 3,
                    # To ensure convergence
                    iter.max = 100,
                    # Number of random samples to use as starting poi
nts
                    nstart = 50)
print("Sepal Length, Sepal Width, Petal Length, Petal Width:")
```

```
[1] "Sepal Length, Sepal Width, Petal Length, Petal Width:"
```

Hide

Hide

```
# A table showing our clusters and how the species fall into them
table(kmeans_cls$cluster, df$Species)
```

```
    setosa versicolor virginica
  1     50          0         0
  2      0         48        14
  3      0          2        36
```

Hide

Hide

```
kmeans_cls <- kmeans(
                    # Variables to consider
                    df[,2:4],
                    # 3 clusters to match number of species
                    centers = 3,
                    # To ensure convergence
                    iter.max = 100,
                    # Number of random samples to use as starting poi
nts
                    nstart = 50)
print("Sepal Width, Petal Length, Petal Width:")
```

```
[1] "Sepal Width, Petal Length, Petal Width:"
```

Hide

Hide

```
# A table showing our clusters and how the species fall into them
table(kmeans_cls$cluster, df$Species)
```

```
    setosa versicolor virginica
  1      0         48         5
  2     50          0         0
  3      0          2        45
```

Hide

Hide

```
kmeans_cls <- kmeans(
                    # Variables to consider
                    df[,2:3],
                    # 3 clusters to match number of species
                    centers = 3,
                    # To ensure convergence
                    iter.max = 100,
                    # Number of random samples to use as starting poi
nts
                    nstart = 50)
```

```
print("Sepal Width, Petal Length:")
```

```
[1] "Sepal Width, Petal Length:"
```

Hide

Hide

```
# A table showing our clusters and how the species fall into them
table(kmeans_cls$cluster, df$Species)
```

```
   
    setosa versicolor virginica
  1      0         48         9
  2      0          2        41
  3     50          0         0
```

Hide

Hide

```
kmeans_cls <- kmeans(
                    # Variables to consider
                    df[,1:3],
                    # 3 clusters to match number of species
                    centers = 3,
                    # To ensure convergence
                    iter.max = 100,
                    # Number of random samples to use as starting poi
nts
                    nstart = 50)
print("Sepal Length, Sepal Width, Petal Length:")
```

```
[1] "Sepal Length, Sepal Width, Petal Length:"
```

Hide

Hide

```
# A table showing our clusters and how the species fall into them
table(kmeans_cls$cluster, df$Species)
```

```
   
    setosa versicolor virginica
  1      0          5        37
  2      0            45          13
  3     50             0           0
```

Hide

Hide

```
kmeans_cls <- kmeans(
                    # Variables to consider
                    df[,1:2],
                    # 3 clusters to match number of species
                    centers = 3,
                    # To ensure convergence
                    iter.max = 100,
                    # Number of random samples to use as starting poi
nts
                    nstart = 50)
print("Sepal Length, Sepal Width:")
```

```
[1] "Sepal Length, Sepal Width:"
```

Hide

Hide

```
# A table showing our clusters and how the species fall into them
table(kmeans_cls$cluster, df$Species)
```

```


    setosa versicolor virginica
  1     50          0         0
  2      0         38        15
  3      0         12        35
```

Hide

Hide

```
kmeans_cls <- kmeans(
                    # Only using Petal Length and Width Variables
                    df[,3:4],
                    # 3 clusters to match number of species
                    centers = 3,
                    # To ensure convergence
                    iter.max = 100,
                    # Number of random samples to use as starting poi
nts
```

```
                    nstart = 50)
print("Petal Length and Width:")
```

```
[1] "Petal Length and Width:"
```

Hide

Hide

```
table(kmeans_cls$cluster, df$Species)
```

```

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

From these tables it appears that data can be clustered quite effectively using a few different combinations of variables.

However, for clustering in a manner that predicts flower species the best variables to use K-means with are petal length and width. When these are used with the K-means algorithm all but 6 data observations are assigned a cluster that matches their species type.

Hide

Hide

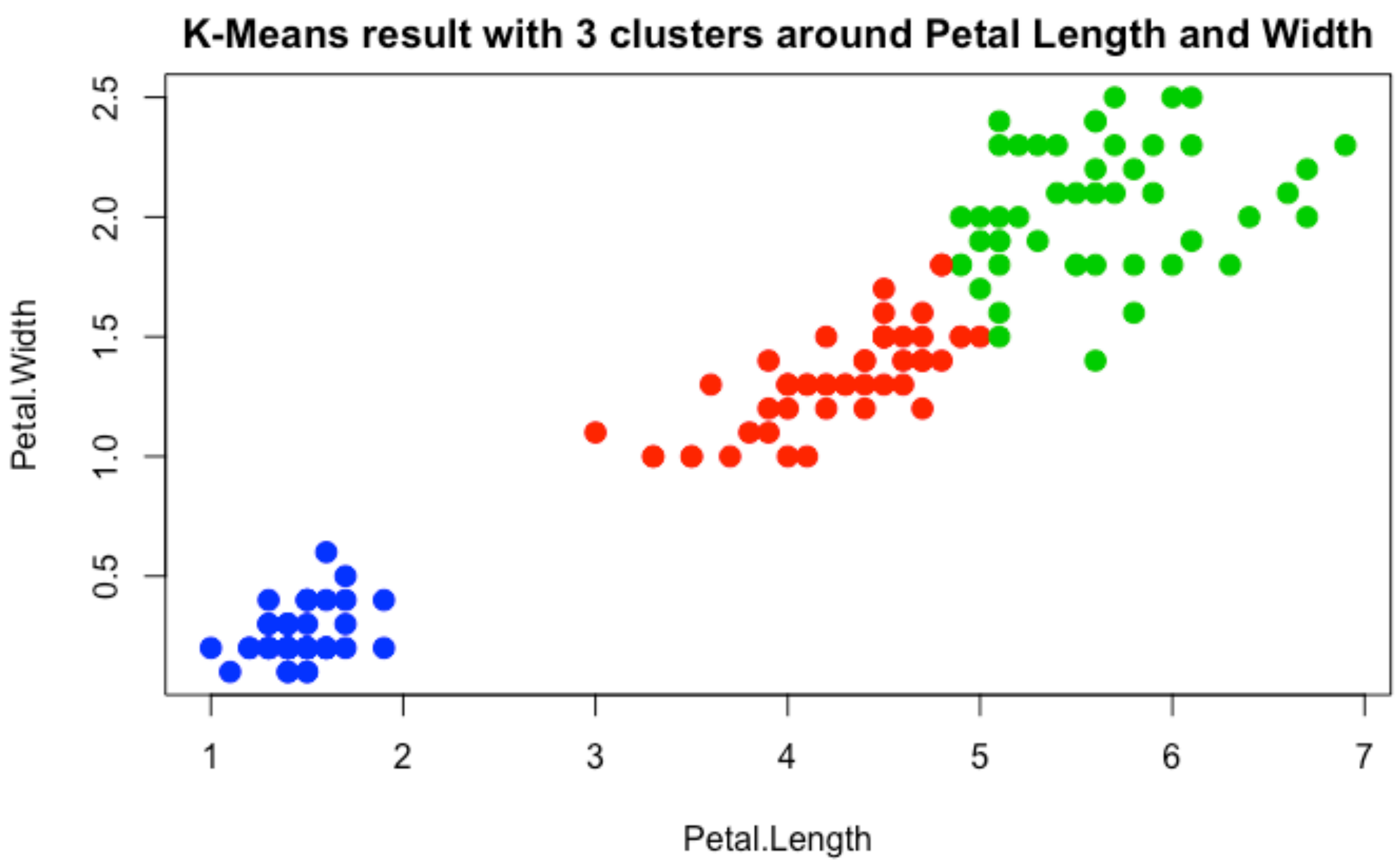
```
# The clusters correctly "predicted" all but 6 data points
correct_predictions <- nrow(df) - 6
accuracy <- correct_predictions/nrow(df)
accuracy
```

```
[1] 0.96
```

Hide

Hide

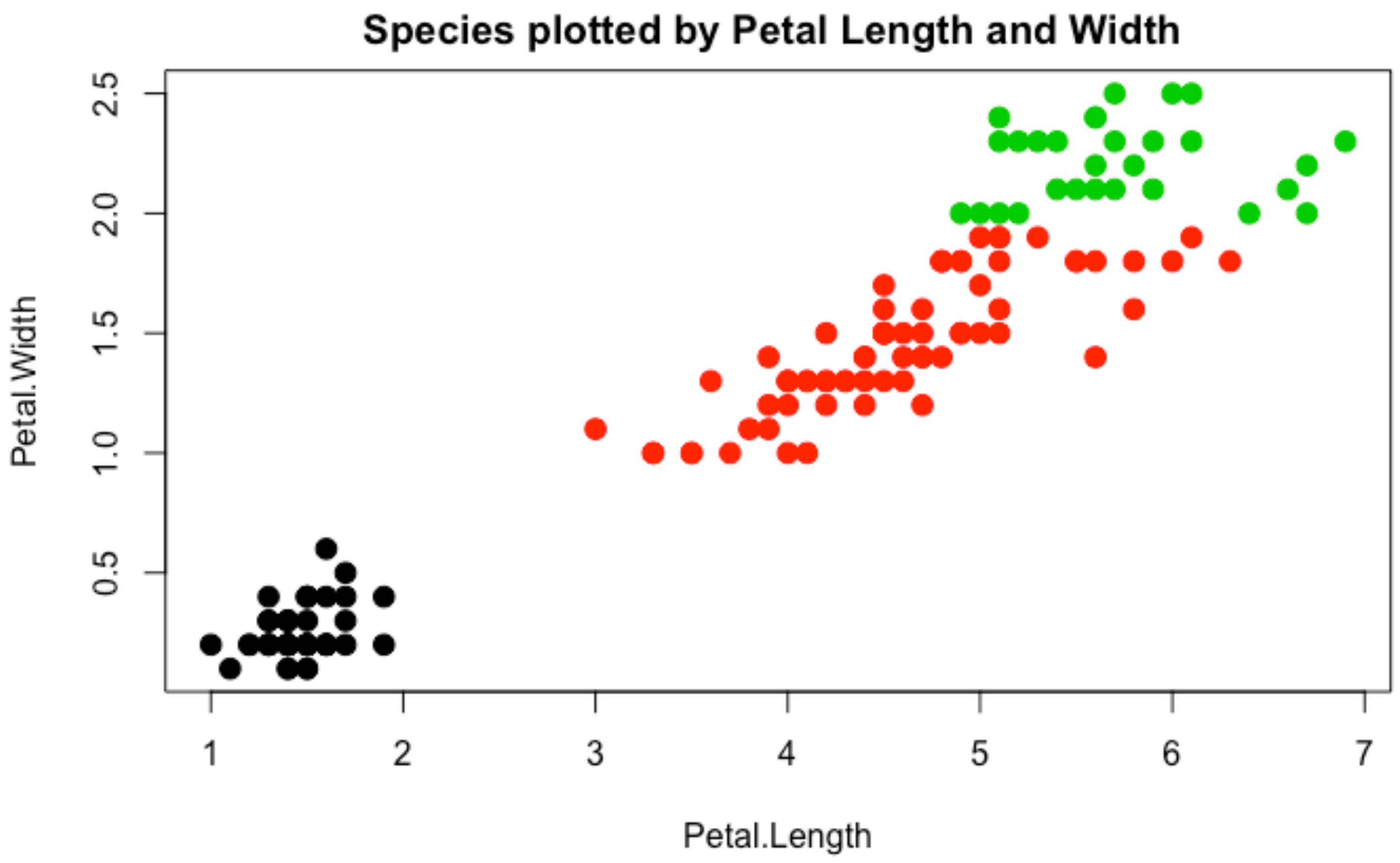
```
# If we create plot petal length and width we see that the clusters a
nd species flowers are very similar
# Plotting our data and clusters
plot(df[,3:4], col =(kmeans_cls$cluster +1) , main="K-Means result wi
th 3 clusters around Petal Length and Width", pch=20, cex=2)
```



Hide

Hide

```
plot(df[,3:4], col =(df[,4] + 1) , main="Species plotted by Petal Len
gth and Width ", pch=20, cex=2)
```



Accuracy of these clusters for predicting flower species: 96%

Best variables: Petal length and width

Number of clusters: 3