HOMEWORK 2. ISYE 6501

Guillermo de la Hera Casado

September 3rd, 2019

Model Validation.

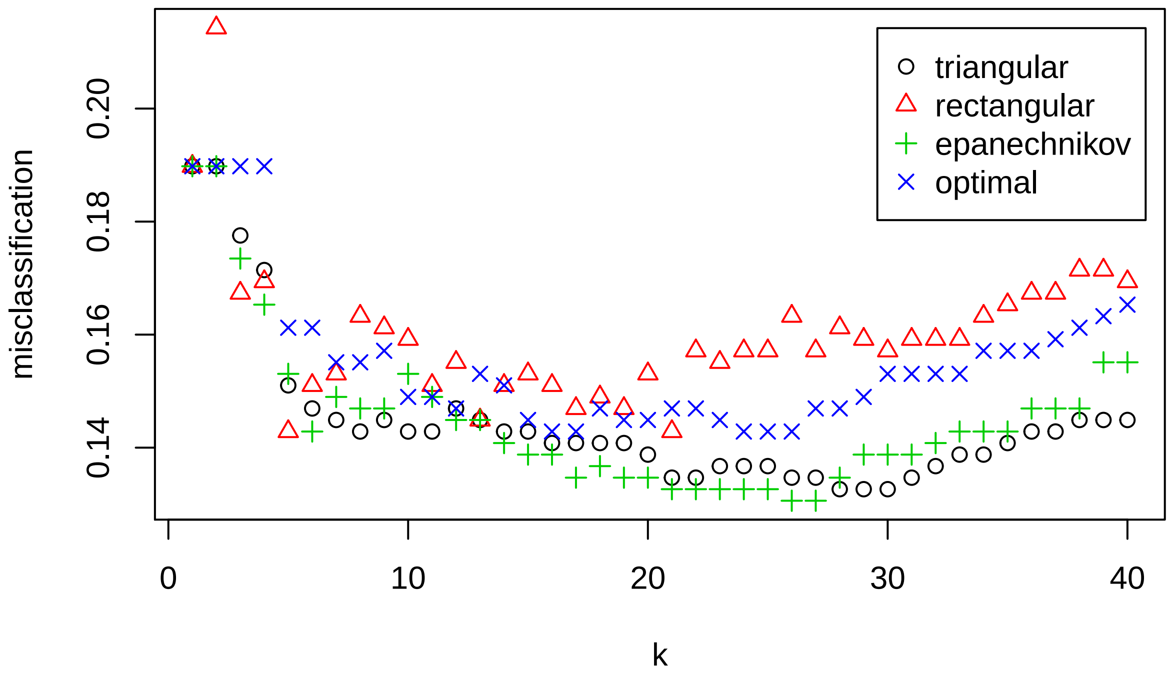
Question 3.1.a Cross-validation for k-nearest-neighbors model.

After reading the data and storing it in a variable, the next step performed was to split the data into training (75%) and test set (25%) by using the library *caTools*

Looking at KKNN package, there is a bult-in function to perform cross-validation: *train.kknn*. By default it performs leave-one out cross-validation on the training set very efficiently. That's the reason why it has been selected vs either *cv.kknn* or manually implemented loops.

I started with **Minkowski distance = 2 (Euclidean).** The objective was to take the best configuration of parameters based on accuracy maximization. Just one line of code allowed for checking different k values (from 1 to 40 in this case) and different kernels. It was very important to explicitely mention *R1* as a factor in order to optimize nominal responses, not continuous:

```r
valModel<- train.kknn(as.factor(R1)~., trainSet, kmax=40,
                      kernel = c("triangular", "rectangular","epanechnikov", "optimal"),
                      scale = TRUE, distance=2)
plot(valModel)
```



```
print(valModel$best.parameters$kernel)
```

```
## [1] "epanechnikov"
```

```
print(valModel$best.parameters$k)
```

```
## [1] 26
```

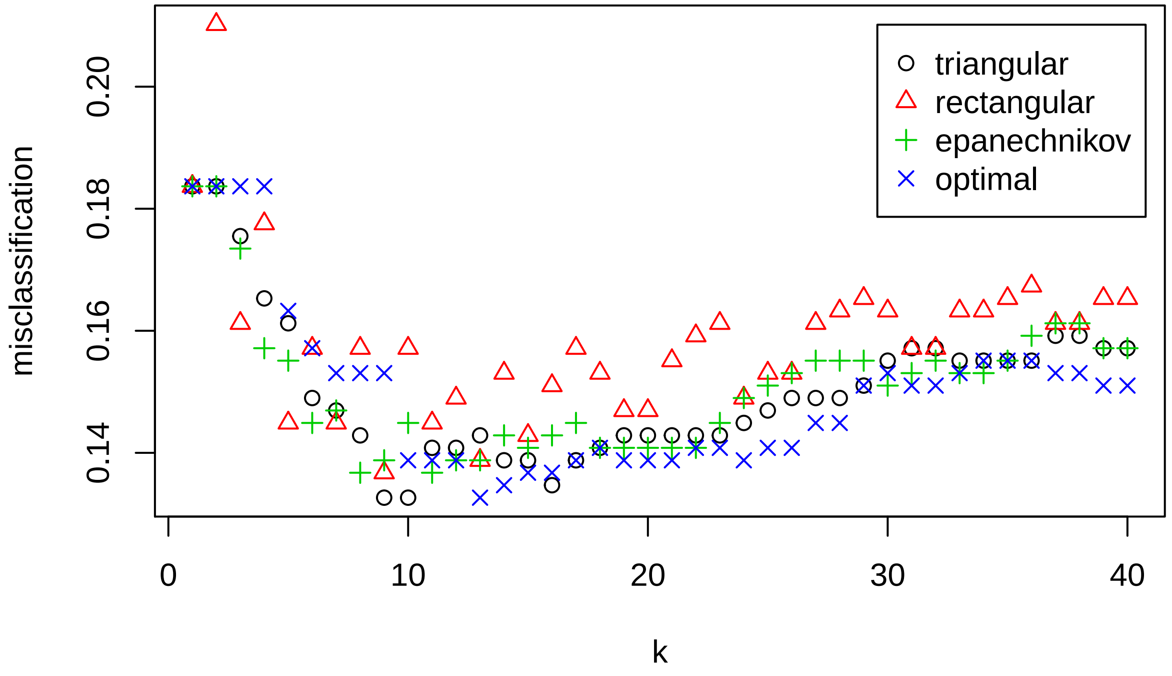
```
print(paste(round(min(valModel$MISCLASS)*100,1),"%"))
```

```
## [1] "13.1 %"
```

As it can be seen from the plot, *Epanechnikov* kernel is chosen, with a recommended k value = 26, minimizing the misclassification error as low as: **13.1%**

What if Minkowski distance = 1 (Manhattan) was used?

```
valModel<- train.kknn(as.factor(R1)~., trainSet, kmax=40,
                      kernel = c("triangular", "rectangular","epanechnikov", "optimal"),
                      scale = TRUE, distance=1)
plot(valModel)
```



```
print(valModel$best.parameters$kernel)
```

```
## [1] "triangular"
```

```
print(valModel$best.parameters$k)
```

```
## [1] 9
```

```
print(paste(round(min(valModel$MISCLASS)*100,1),"%"))
```

```
## [1] "13.3 %"
```

As it can be inferred from the plot, *Triangular* kernel is chosen, with a recommended k value = 9, minimizing the misclassification error as low as: **13.3%**

It seems that distance = 2 wins marginally on reducing misclassification error, so the below will be taken to train the model:

- distance=2
- Kernel = Epanechnikov
- K value = 26

and after that the test set will be used to estimate the final unbiased performance of the model. The test set has still not been used at any point during the cross validation stage, so it will help to estimate how well the model fits **the real relationship between predictors and response**

```
finalModel <- kknn(as.factor(R1)~., train = trainSet, test = testSet,  k = 26,
                   kernel = "epanechnikov", scale = TRUE,distance=2)
acc <- sum(finalModel$fit == testSet$R1) / nrow(testSet)
print(paste(round((1-acc)*100,2),"%"))
```

```
## [1] "17.07 %"
```

The final misclasification error was: **17%**

Question 3.1.b Creating manually training, validation and test sets for SVM.

The library: *caTools* has been used again to split the data. This time two random splits have been performed:

- Full dataset –> training set (60%) and other sets (40%)
- Other sets –> validation set (50%) and test set (50%)

So overall it looked as follows: training set (60%), validation set (20%) and test set (20%). The next step was to use the training data to build several models testing different C values. Every time a model was built, the validation data was considered in order to calculate predictions and capture accuracy:

```
C_values <- c(0.0001, 0.0003, 0.0005, 0.0007, 0.001, 0.003, 0.005, 0.01, 0.1, 1,100)
AccVanilla <- rep(-1, length(C_values))
counter <- 1
for(cVar in C_values){
invisible(capture.output(model <- ksvm(as.factor(R1)~., trainSet, type = "C-svc",
              kernel = "vanilladot", C = cVar, scaled = TRUE)))
pred <- predict(model, validationSet[,1:10])
AccVanilla[counter] <- paste(round((sum(pred == validationSet$R1) /
                                      nrow(validationSet))*100,1),"%",sep="")
counter <- counter + 1
}
```

Table 1: Vanilladot. Accuracy on validation set per C value

C Value	Overall Accuracy
1e-04	55%
3e-04	55%
5e-04	55%
7e-04	57.3%
1e-03	70.2%

C Value	Overall Accuracy
3e-03	88.5%
5e-03	88.5%
1e-02	88.5%
1e-01	88.5%
1e+00	88.5%
1e+02	88.5%

From the outcome, it was pretty clear that C value: **3e-03** provided a boost in classification accuracy **(up to 88.5%)**. Higher C Values didn't provide further improvement and brought up risk of overfitting.

The last step was to build the training set with the optimized C parameter and use the test set to calculate predictions, capturing the final unbiased accuracy of the model:

```
invisible(capture.output(model <- ksvm(as.factor(R1)~., trainSet, type = "C-svc",
              kernel = "vanilladot", C = 0.003, scaled = TRUE)))
pred <- predict(model, testSet[,1:10])
acc <- paste(round((sum(pred == testSet$R1) / nrow(testSet))*100,1),"%",sep="")
print(acc)
```

```
## [1] "84.6%"
```

The accuracy on the test set was: **84.6%**

Q4.1. Use cases for clustering

I work for a Pay-TV company with headquarters in Africa. Customers pre-pay monthly subscriptions and get channels specialized in different content e.g. football, movies, etc.

Gathering customer data and grouping them into clusters would be a great exercise for:

- understanding group characteristics of various valuable customers
- analyzing group characteristics of loss customers
- drafting consumption characteristics of customers

Several predictors could be used, few of them as follows:

- Demographics: country, age, family structure.
- Consumption patterns: monthly ARPU, currently active/inactive in the platform, total disconnection events, average disconnection length in days…

Q4.2. K-Means on Iris dataset

Finding the right predictors

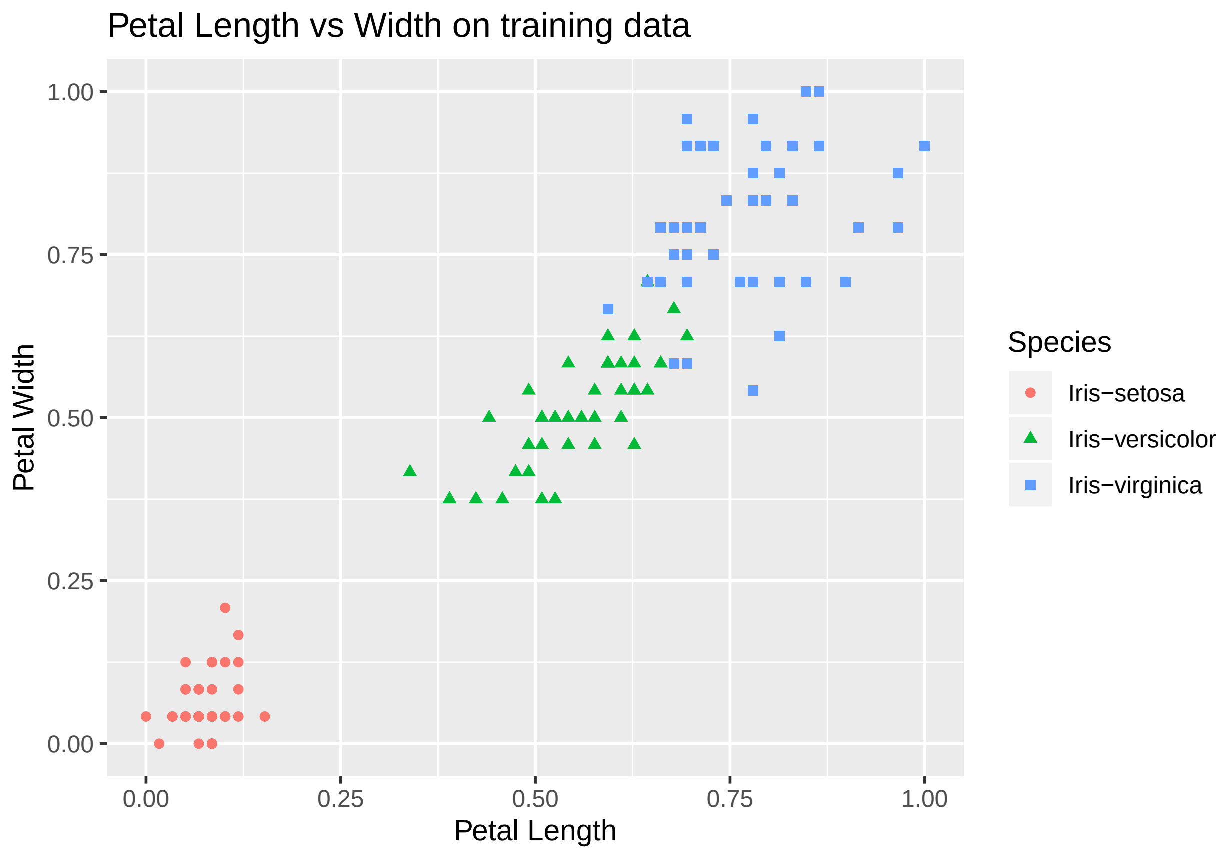
The first thing to do was to look at the data and see if there is a need for scaling. Normalization is relevant for K-Means, as it seems that it helps the K-Means algorithm to converge faster, avoiding derivatives to align along directions with higher variance. More info here, from StackExchange.com

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.054   Mean   :3.759   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##              Species
##  Iris-setosa    :50
##  Iris-versicolor:50
##  Iris-virginica :50
##
##
##
```

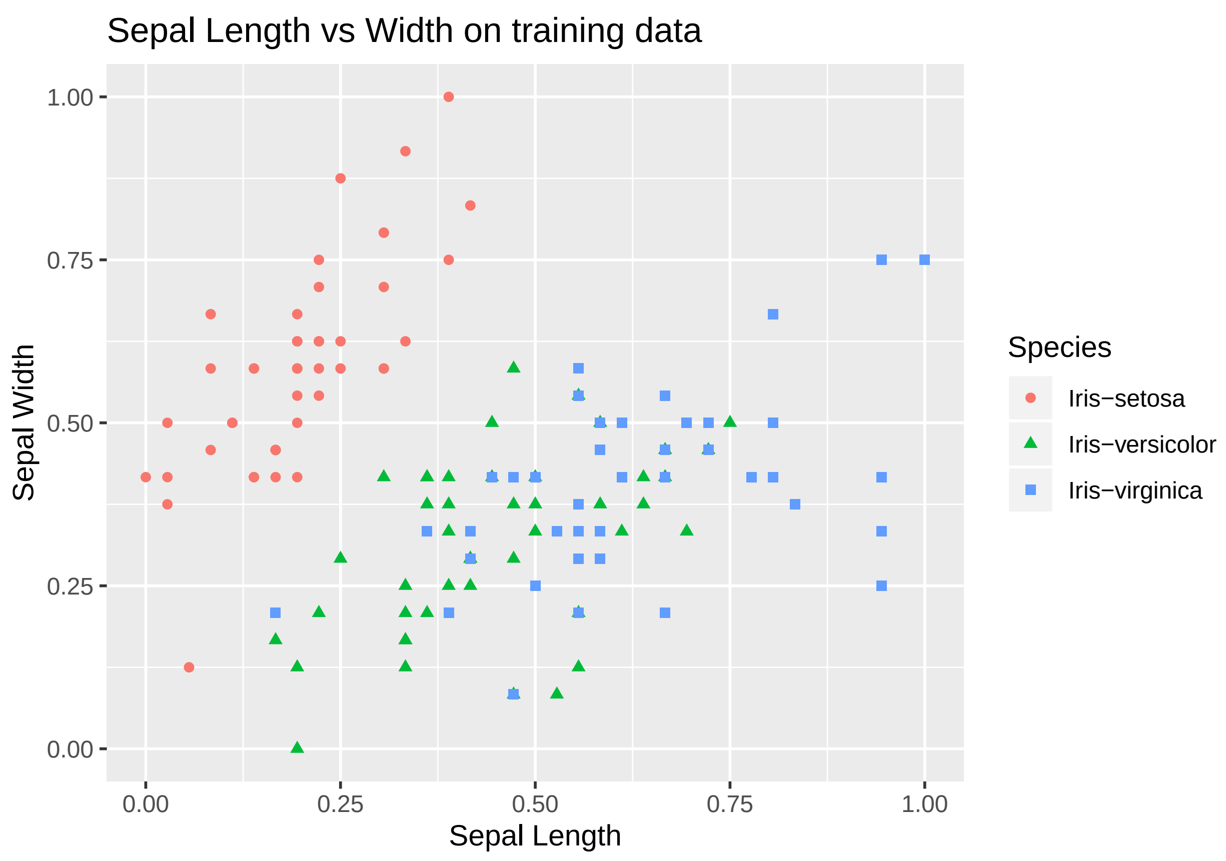
As it can be seen, each variable has a different scale, so we proceed to normalize the data for each predictor, according to MinMax algorithm.

The second task was to split the normalized data into Training (90%) and Test sets (10%). The training set will be used to build the K-Means clustering and the test set will be used to see how well our clustering differentiate species on unseen data.

The third task was to analyze, out of the four predictors, which combination may work better for clustering the data in a way that the species are clearly differentiated. For that, two scatterplots were obtained:



The combination of both Petal Length and Width seemed to do a good job on differentiating classes clearly. As showed by the plot below, **Sepal length and Width are not that effective**:



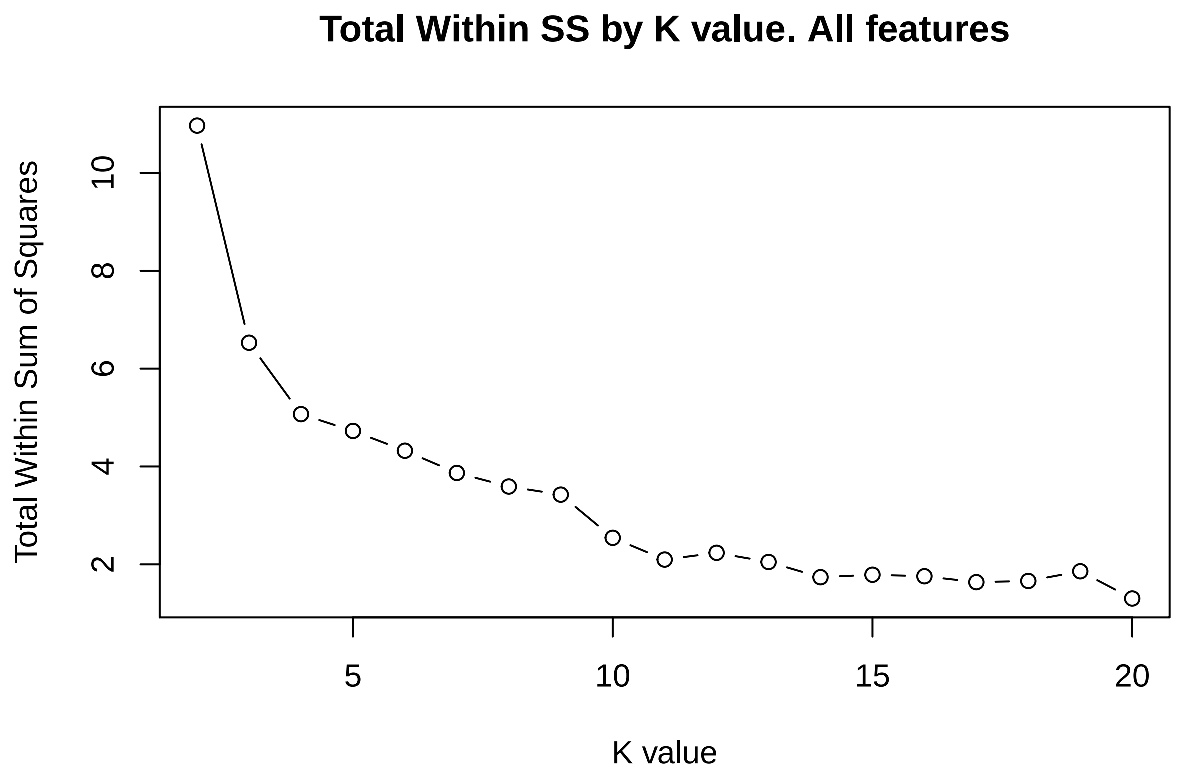
Optimizing K value

It would be interesting to research on the following question: **is the clustering model better when we use the four predictors, or only when we use the Petal attributes (so saying that Sepal attributes add no more than noise. . .)?**

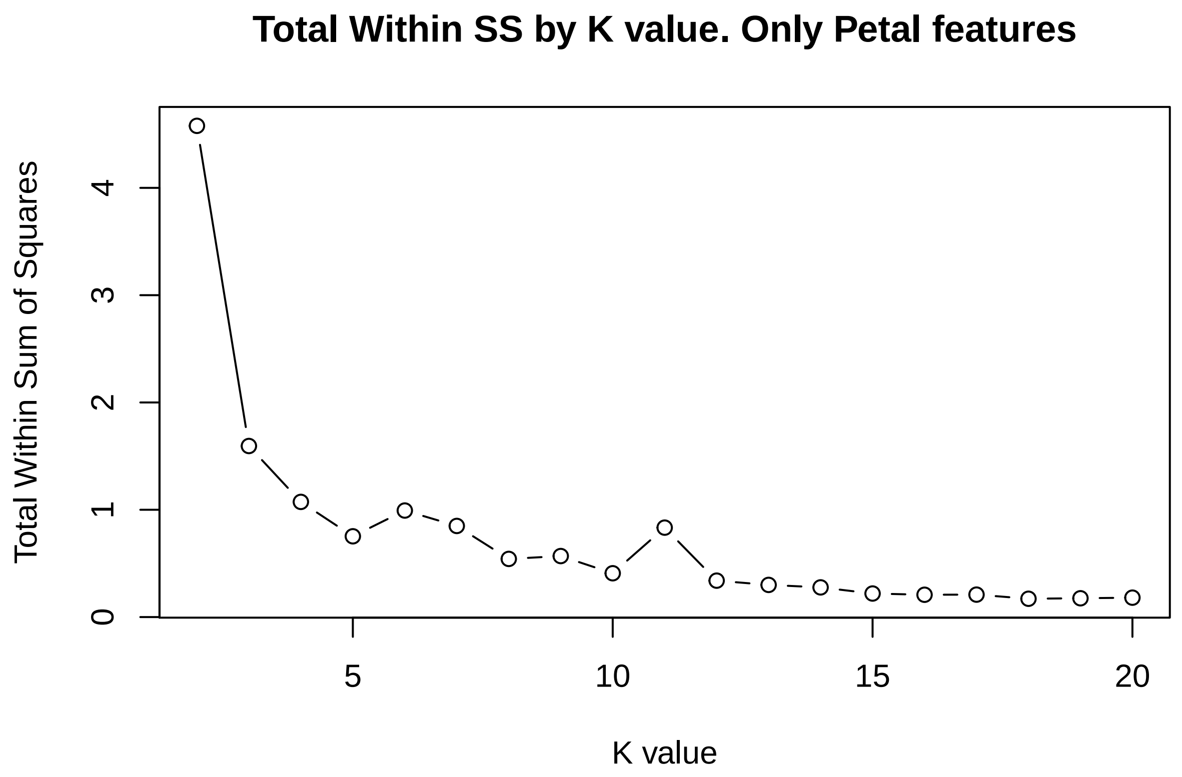
To answer this question, for each model:

- different k values were checked (value range: 2 to 20),
- the parameter *nstart* was set up to 100, to reduce the risk of the model being biased due to the initial position of centroids.
- the elbow chart was analyzed.

Starting with a model that includes the four predictors:



The best K value was k=3, making the Within sum of squares = 5.06 Looking now at the model with only Petal attributes:



The best K value was also K=3, but the Within sum of squares being much smaller: 1.07

Therefore, **we will stick going forward with only two predictors (Petal length and Width) in order to make the final K-Means model, with K=3**

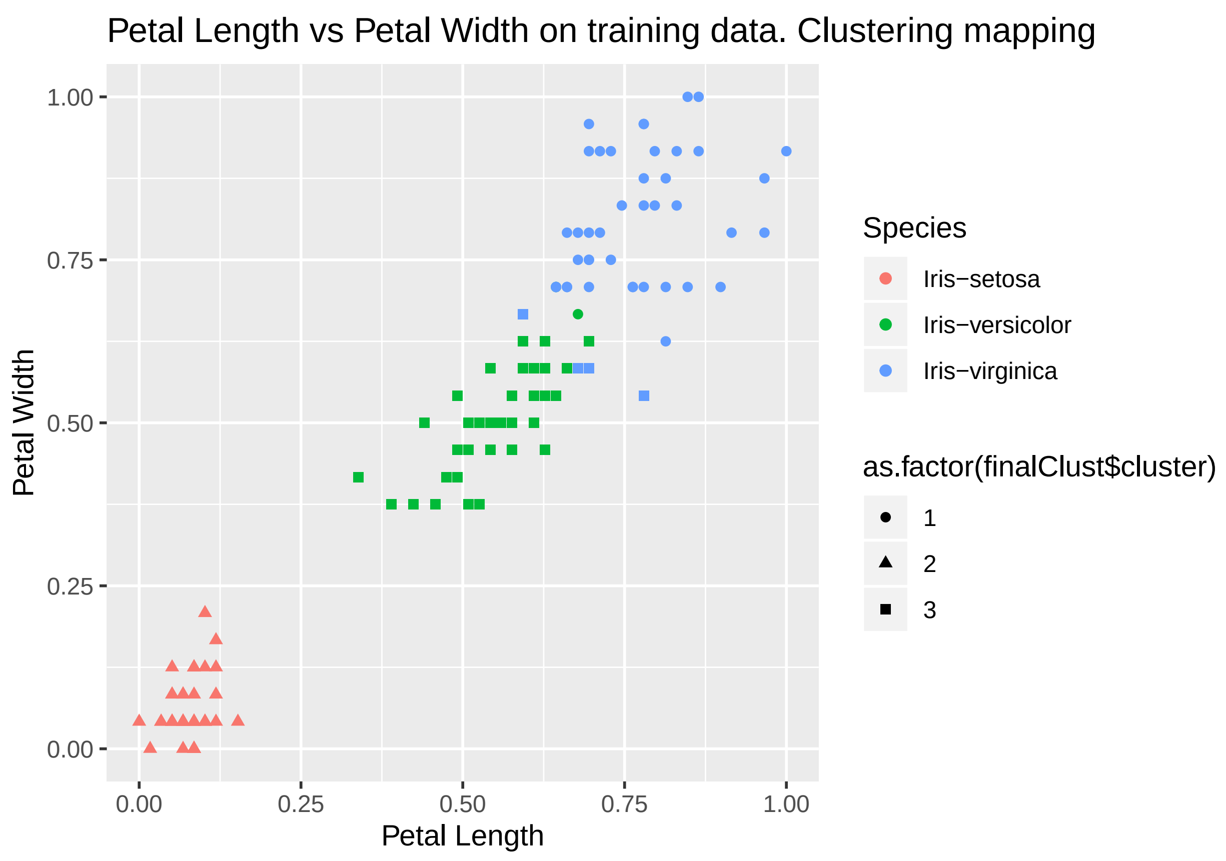
Representation of the final model

The final model was built, providing the following centroids for each cluster:

```r
finalClust <- kmeans(x = trainSet[,3:4], centers = 3, nstart=100)
print(finalClust$centers)
```

```
##   Petal.Length Petal.Width
## 1   0.76823019  0.80910853
## 2   0.07608286  0.06018519
## 3   0.55896141  0.51063830
```

The relationship between clusters and classes on the training data was represented visually:

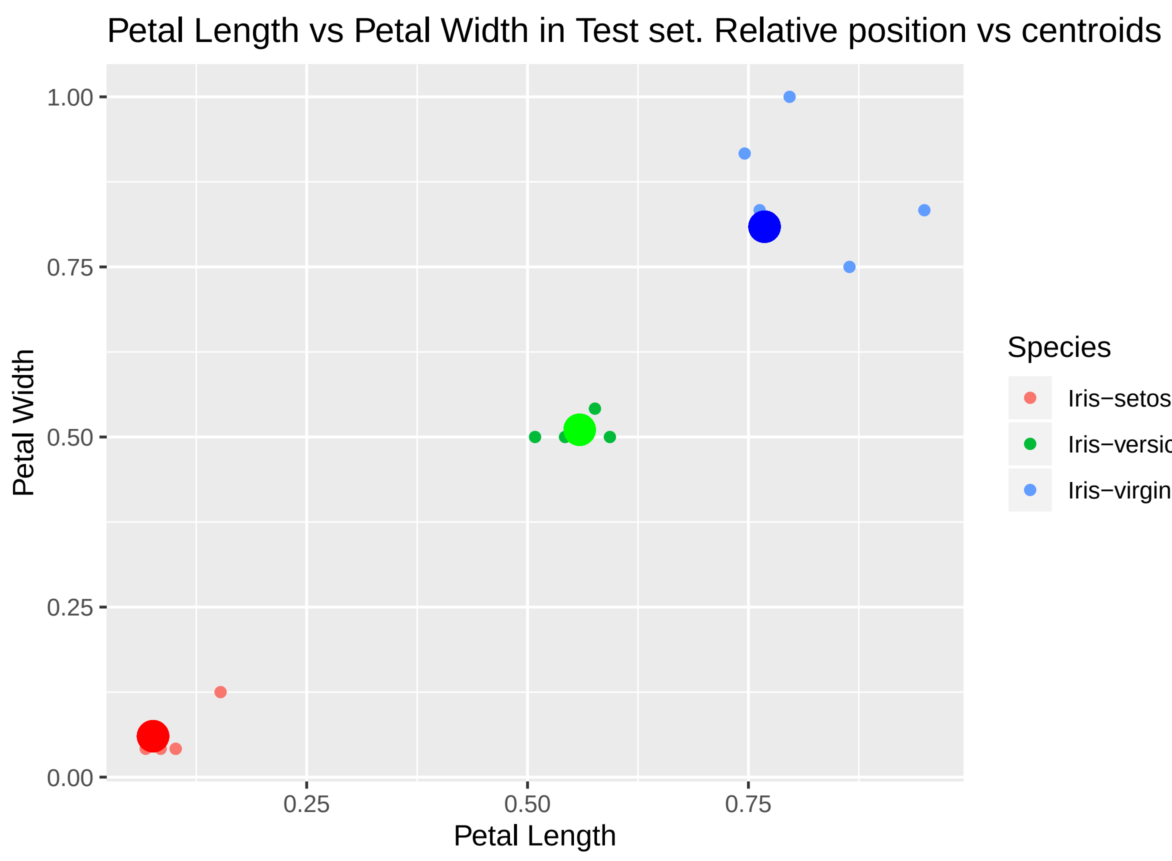


From the picture it looked evident that **the clusters assigned correlate almost perfectly with the Species distribution:**

- Cluster=1 fits the class: *iris-virginica.* (high measures of Petal length and width, blue color)
- Cluster=2 fits the class: *iris-setosa* (low measures of Petal length and width, red color)
- Cluster=3 fits the class: *iris-versicolor* (average measures of Petal length and width, green color)

Prediction accuracy on test data

The last task was to check on the Test set how well the clustering can provide inference on the class for each point.



- The points with small size represented the Test set instances, according to their Petal attributes.
- The points with big size represented the centroids given by the K-means algorithm on the training data.

From the view, it was clear that **centroids could be used to infer the Species of new instances even if it was unknown, as the mapping looks very accurate.**