k Nearest Neighbor Exercise

Question

Using the data set (credit_card_data.txt or credit_card_data-headers.txt), 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

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

Solution

First, install the libraries I will be using and remove all data in my environment, as is best practices

a)

First, load in the data

To do cross validation, it is a similar approach to last week. I will approach it with nested for-loops.

In the innermost for-loop, I will use k = 10 as an example. The optimal k will be found in the outer-most loop. I will start by creating the empty data frame df that will be populated later

I used cv.kknn to create my cross validation model.

The for-loop cycles through values 2:10, which will be the input for the number of pratitions for k-fold cross validation, or kcv. Scale is set to True to make the model more accurate.

The prediction values are generated by cv.kknn, which we then match to the original data kknn to get the accuracy. This is then loaded into a table.

```
##   no.partitions  k  accuracy
## 1             2 10 0.8425076
## 2             3 10 0.8409786
## 3             4 10 0.8318043
## 4             5 10 0.8425076
## 5             6 10 0.8440367
## 6             7 10 0.8547401
## 7             8 10 0.8562691
## 8             9 10 0.8532110
## 9            10 10 0.8532110
```

From the above result, we see the different accuracies for the number of partitions given a constant k-value of 10.

However, we also want to check the accuracies across multiple k values, or the number of clusters. To do this, we put the above code inside a for-loop that cycles through values 1:20

```
## [1] "data.frame size: 180 rows"
```

The data frame that runs through all possibilities of k and kcv has 180 rows, so we will extract the maximum.

```
paste("Optimal # Partitions (kcv): ", (df[which.max(df$accuracy),1]))
```

```
## [1] "Optimal # Partitions (kcv):  2"
```

```
paste("Optimal # Neighbors Considered (k): ", (df[which.max(df$accuracy),2]))
```

```
## [1] "Optimal # Neighbors Considered (k):  12"
```

```
paste("Optimal Accuract: ", (df[which.max(df$accuracy),3]))
```

```
## [1] "Optimal Accuract:  0.863914373088685"
```

b)

First, set up the good practices like we had in the beginning

```
## GOOD PRACTICES ##
library(kernlab)
library(kknn)
library(caTools)
rm(list = ls())
```

Next, we load in the data, create a data frame to be filled in later with our test set.

The structure is nexted loops. To test combinations of training sets, validation sets, and testing sets, the kknn data set needs to be shuffled. Shuffling the data set and then extracting the values of selection is simpler than shuffling the values of selection themselves.

In order to create random shuffles of the data, I set different seeds for each model, each giving a distinct random generation. The seeds come from the random generator which gives 10 differnet random values for the seed.

The first for-loop begins with cycling through the randomly generated seeds. The first code inside this for-loop is setting the seed. In the below example, to explain one piece at a time, I will set the seed to be 42. Later, we will use the for-loop

Next, we shuffle the kknndata set based on the seed and create the training, validation, and testing sets from this newly rearranged data.

In the next for-loop, we will need a data frame to hold the data, so I create an empty data frame.

Inside the next for-loop, we cycle through different k-values, or the number of neighbors.

Since we area going to be taking the sum of the predictor values, we set the total_sum to zero at the beginning of each loop.

Then, generator the kknn model as before, this time using the training and validation sets generated in the outer loop. Use this model to match against the actual values of the data, and find the accuracy by dividing the correct predictions over the total values.

Lastly, I created a data frame to hold all the possible combinations of seeds and k–values.

```
##     k  accuracy
## 1   1 0.8167939
## 2   2 0.8167939
## 3   3 0.8167939
## 4   4 0.8167939
## 5   5 0.8167939
## 6   6 0.8244275
## 7   7 0.8091603
## 8   8 0.8167939
## 9   9 0.8167939
## 10 10 0.8244275
## 11 11 0.8244275
## 12 12 0.8320611
## 13 13 0.8244275
## 14 14 0.8244275
## 15 15 0.8167939
## 16 16 0.8167939
## 17 17 0.8167939
## 18 18 0.8167939
## 19 19 0.8167939
## 20 20 0.8167939
```

Above is the data frame for all accuracies when the seeds is 42

Next, we loop through all the seeds, thus different sets of the training, validation, and testing sets.

With each loop, we add to a data frame that holds all the k values and accuraces. From this, we can extract the optimal k-value.

Once we have a completed data frame, we can extract the highest k value and accuracy.

With this k-value, we then insert it into the testing set to reveal it’s accuracy.

```
## [1] "Optimal K classifier:  18"
```

```
## [1] "Accuracy with Classifier:  0.900763358778626"
```

```
## [1] "Accuracy of model:  0.900763358778626"
```

Question

As always, begin with good practices of loading in the librarys, removing existing variables from the environment, and setting the seed for reproducible results.

Solution

We start by creating a data frame from the iris data set and scaling the data.

I will cycle through all the combinations of predictors to find the best clustering. To do this, I create a data frame containing the first and last numbers of the range. Creating the data frame makes calling on these ranges later.

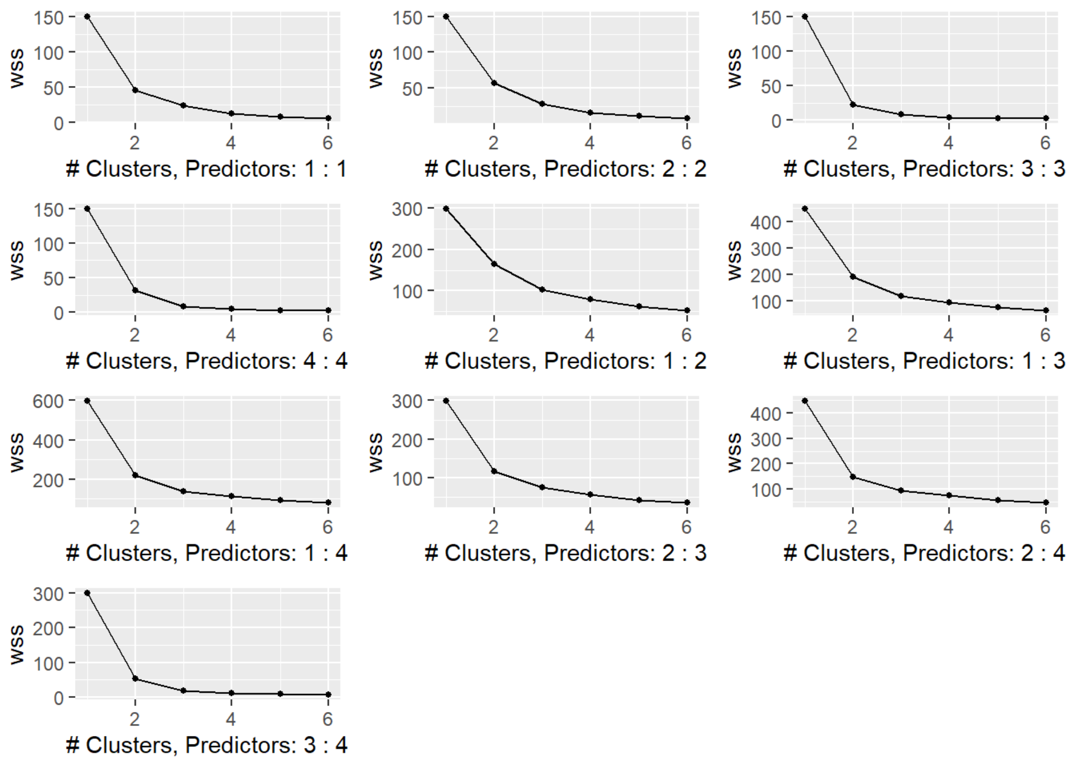
First creating an empty data frame to be populated later, I created a for-loop to cycle through the different combinations of predictors.

Inside the first loop is a nested loop that cycles through the number of centers we can use. In this model, I chose 2:6 different centers.

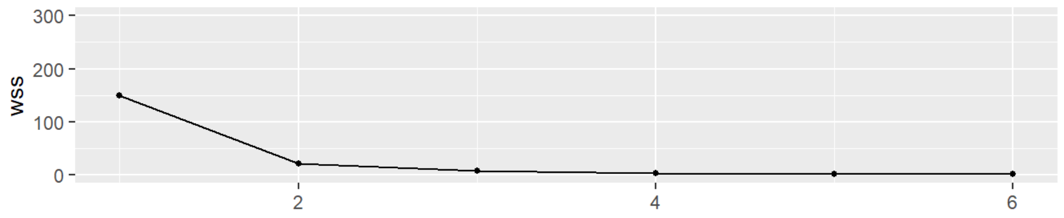
The cluster model is created from kmeans. The iris data is subset by the predictors indicated in the outer for-loop.

After this, we have a data frame that holds the WSS for the 10 combinations of predictors along with the number of clusters, resulting in 60 values.

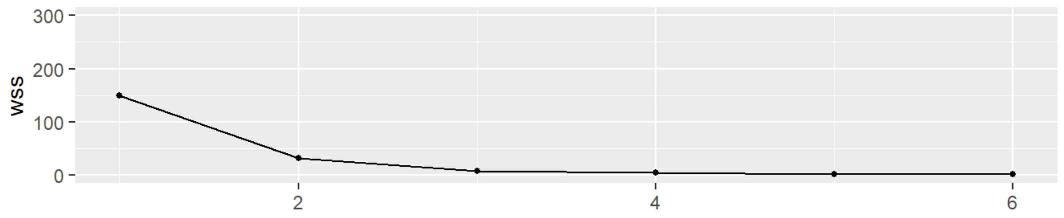
From here, I clustered the values by the predictors they used. Using function to create a plot and the lapply to print the plots, I ended with 10 elbow graphs.



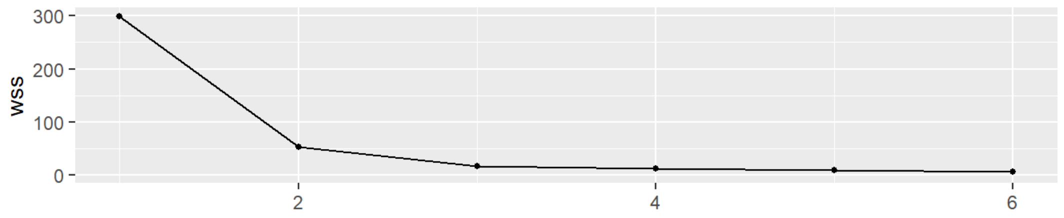
I then wanted to see the three best predictors, Pedal Length, Pedal Width, and both Pedal Length and Pedal Width, zoomed in to get a better idea.



Clusters, Predictors: 3 : 3 , # Clusters = 3: 7.8672160832692



Clusters, Predictors: 4 : 4 , # Clusters = 3: 8.45631908561609



Clusters, Predictors: 3 : 4 , # Clusters = 3: 17.9067828617938

While it does seem that the elbow point for the number of clusters is 2, the confusion matrix below says otherwise. I create a three confusion matrices against the real data. The first confusion matrix is using predictor Pedal Length, the second is using predictor Pedal Width, and the third is using both.

Predictor: Pedal Length, # Clusters: 3

setosa	versicolor	virginica
0	48	6
50	0	0
0	2	44

Predictor: Pedal Length and Pedal Width, # Clusters: 3

setosa	versicolor	virginica
0	2	46
50	0	0
0	48	4

Predictor: Pedal Width, # Clusters: 3

setosa	versicolor	virginica
0	2	46
50	0	0
0	48	4

The Confusion Matrix indicates that the predictors for just Pedal Width and Pedal Width and Length have the same accuracy. *Because the MSE was less according to the elbow points, however, I would use the sole predictor Pedal Width for clustering in this data set.*