R Notebook

Akhila Saineni

11/29/2020

```r
data = read.csv("Wholesale_customers_data.csv")
summary(data)
```

```
##     Channel          Region          Fresh             Milk      
##  Min.   :1.000   Min.   :1.000   Min.   :     3   Min.   :   55  
##  1st Qu.:1.000   1st Qu.:2.000   1st Qu.:  3128   1st Qu.: 1533  
##  Median :1.000   Median :3.000   Median :  8504   Median : 3627  
##  Mean   :1.323   Mean   :2.543   Mean   : 12000   Mean   : 5796  
##  3rd Qu.:2.000   3rd Qu.:3.000   3rd Qu.: 16934   3rd Qu.: 7190  
##  Max.   :2.000   Max.   :3.000   Max.   :112151   Max.   :73498  
##     Grocery          Frozen         Detergents_Paper    Delicassen     
##  Min.   :    3   Min.   :   25.0   Min.   :    3.0   Min.   :    3.0  
##  1st Qu.: 2153   1st Qu.:  742.2   1st Qu.:  256.8   1st Qu.:  408.2  
##  Median : 4756   Median : 1526.0   Median :  816.5   Median :  965.5  
##  Mean   : 7951   Mean   : 3071.9   Mean   : 2881.5   Mean   : 1524.9  
##  3rd Qu.:10656   3rd Qu.: 3554.2   3rd Qu.: 3922.0   3rd Qu.: 1820.2  
##  Max.   :92780   Max.   :60869.0   Max.   :40827.0   Max.   :47943.0
```

```r
top.n.custs = function (data,cols,n=5) { #Requires some data frame and the top N to remove
idx.to.remove =integer(0) #Initialize a vector to hold customers being removed
for (c in cols){ # For every column in the data we passed to this function
col.order =order(data[,c],decreasing=T) #Sort column "c" in descending order (bigger on top)
#Order returns the sorted index (e.g. row 15, 3, 7, 1, ...) rather than the actual values sorted.
idx =head(col.order, n) #Take the first n of the sorted column C to
idx.to.remove =union(idx.to.remove,idx) #Combine and de-duplicate the row ids that need to be removed
}
return(idx.to.remove) #Return the indexes of customers to be removed
}
```

```r
top.custs =top.n.custs(data,cols=3:8,n=5)
length(top.custs)
```

```
## [1] 19
```

```r
top.custs =top.n.custs(data, cols = 1:5,n=5)
length(top.custs)
```

```
## [1] 18
```

```r
data[top.custs,]
```

```
##     Channel Region  Fresh  Milk Grocery Frozen Detergents_Paper Delicassen
## 1         2      3  12669  9656    7561    214             2674       1338
## 2         2      3   7057  9810    9568   1762             3293       1776
## 3         2      3   6353  8808    7684   2405             3516       7844
## 5         2      3  22615  5410    7198   3915             1777       5185
## 6         2      3   9413  8259    5126    666             1795       1451
## 4         1      3  13265  1196    4221   6404              507       1788
## 182       1      3 112151 29627   18148  16745             4948       8550
## 126       1      3  76237  3473    7102  16538              778        918
## 285       1      3  68951  4411   12609   8692              751       2406
## 40        1      3  56159   555     902  10002              212       2916
## 259       1      1  56083  4563    2124   6422              730       3321
## 87        2      3  22925 73498   32114    987            20070        903
## 48        2      3  44466 54259   55571   7782            24171       6465
## 86        2      3  16117 46197   92780   1026            40827       2944
## 184       1      3  36847 43950   20170  36534              239      47943
## 62        2      3  35942 38369   59598   3254            26701       2017
## 334       2      2   8565  4980   67298    131            38102       1215
## 66        2      3     85 20959   45828     36            24231       1423
```

```r
data.rm.top=data[-c(top.custs),] #Remove the Customers
set.seed(76964057) #Set the seed for reproducibility
k =kmeans(data.rm.top[,-c(1,2)], centers=9) #Create 9 clusters, Remove columns 1 and 2
k$centers #Display cluster centers
```

```
##       Fresh      Milk   Grocery    Frozen Detergents_Paper Delicassen
## 1 16284.959  2014.797  2479.824  3569.365         469.5000  1006.4459
## 2  9387.688 21493.375 28239.938  2130.438       12863.3750  3733.8125
## 3  4245.147 11529.324 19961.676  1822.382        8295.7941  1342.0000
## 4 12770.682  5615.068  8925.636  1715.909        2991.9773  1831.2955
## 5 43030.118  3247.882  4356.824  4194.412         790.6471  2046.1765
## 6  4930.730  2231.241  2664.000  2723.131         577.1533   841.2336
## 7 22015.500  9937.000  7844.000 47939.000         671.5000  4153.5000
## 8  2615.443  7438.623 10379.705  1157.328        4604.9016  1248.2951
## 9 27083.811  5832.541  6710.946  4908.243        1176.2432  2044.0541
```

```r
k$size
```

```
## [1]  74  16  34  44  17 137   2  61  37
```

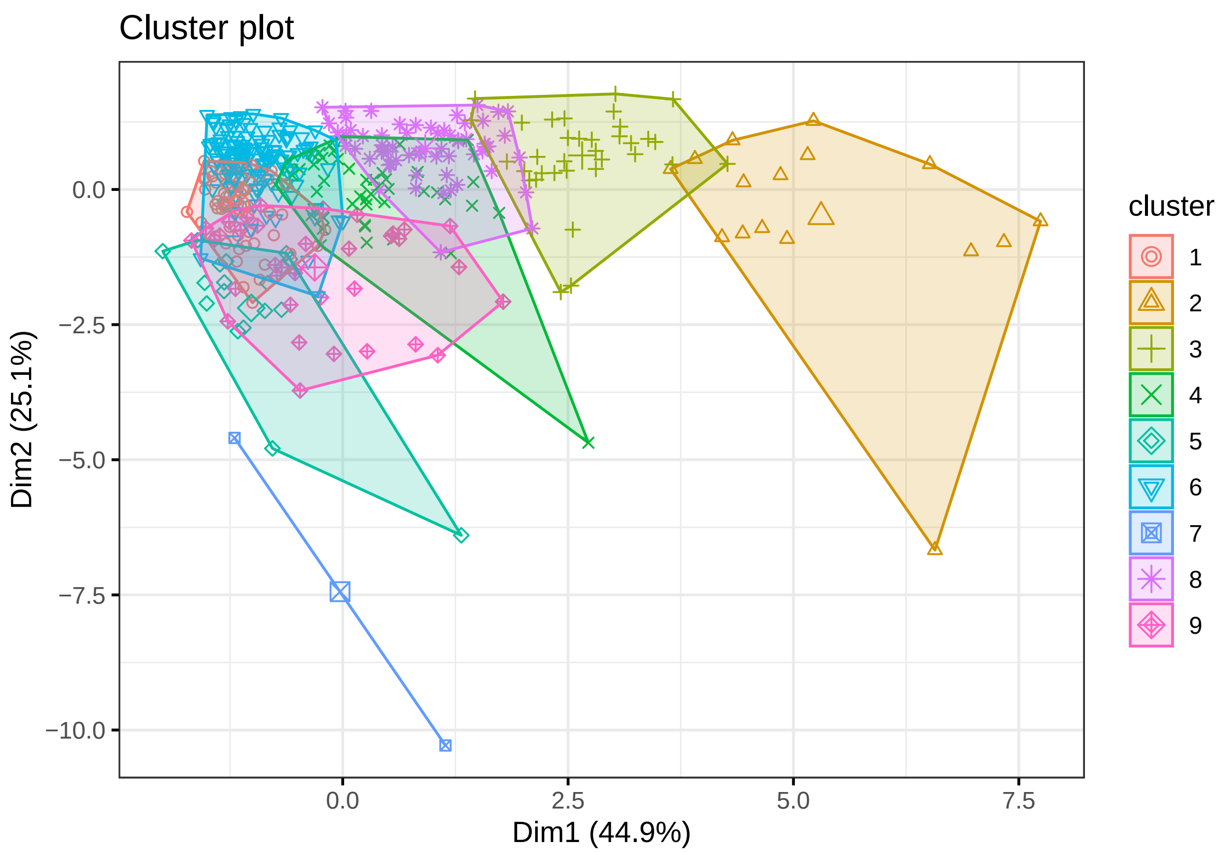
```r
#install.packages("ggpubr")
library("ggpubr")
```

```
## Loading required package: ggplot2
```

```r
library("factoextra")
```

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

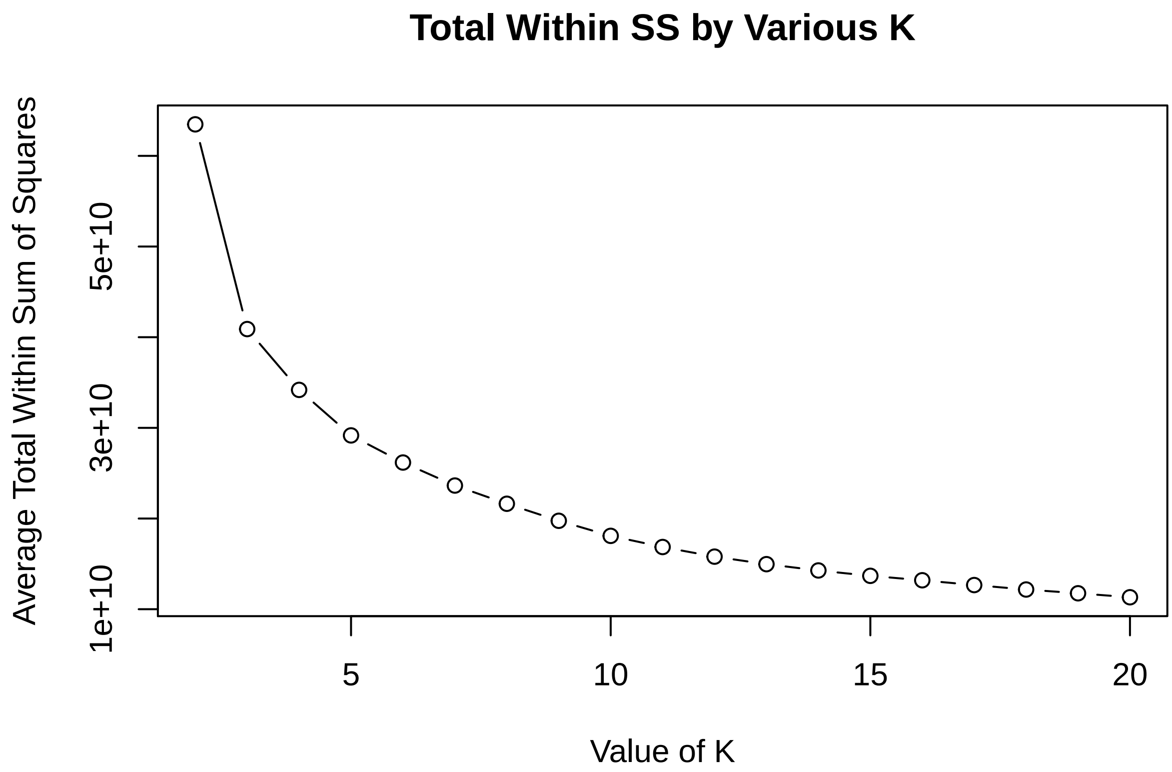
```r
fviz_cluster(k, data = data.rm.top[,-c(1,2)],
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_bw()
             )
```




```r
rng=2:20 #K from 2 to 20
tries =100 #Run the K Means algorithm 100 times
avg.totw.ss =integer(length(rng)) #Set up an empty vector to hold all of points
for(v in rng){ # For each value of the range variable
v.totw.ss =integer(tries) #Set up an empty vector to hold the 100 tries
for(i in 1:tries){
k.temp =kmeans(data.rm.top,centers=v) #Run kmeans
v.totw.ss[i] =k.temp$tot.withinss#Store the total withinss
}
avg.totw.ss[v-1] =mean(v.totw.ss) #Average the 100 total withinss
}
```

```
## Warning: did not converge in 10 iterations
```

```r
plot(rng,avg.totw.ss,type="b", main="Total Within SS by Various K",
ylab="Average Total Within Sum of Squares",
xlab="Value of K")
```




```r
sqrt(422)/2
```

```
## [1] 10.27132
```

Q1- Given this is an imperfect real-world, you need to determine what you believe is the best value for "k" and write-up this portion of your lab report. You should include a brief discussion of your k-Means analysis as well as the best value of "k" that you determine. You should include what mixture of variables within the clusters that this value of "k" results in. That is, you need to interpret your k-Means analysis and discuss what it means.

Answer:

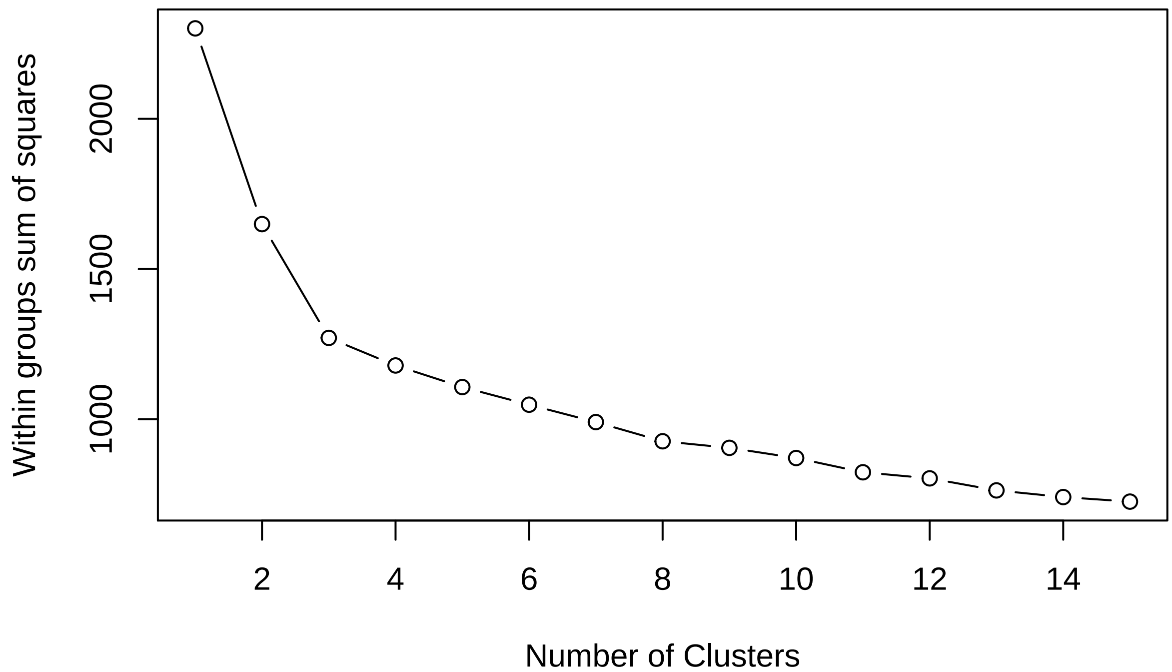
As for the identification of the best value of k in the kMeans algorithm, 2 methods are used Empirical and Elbow method. The empirical method recommends that 10 clusters are required whereas the elbow method analysis, considering 19 different clusterings that are ranging from 2 to 20 clusters and comparing the respective within sum of the squares, in other words the similarity of the points within the cluster. However the higher the number, the lower is the similarity. Each K means algorithm is set to run 100 times in order to achieve the centroids of each cluster. Upon looking at the elbow curve, the within sum is gradually decreasing and the 20 cluster model has the least average total within the sum. In this case 9 clusters are chosen based on the fact that the within sum is low as well as the fact that there won't be much difference in the average within sum after 9 clusters. The centers of the 9 clusters along with the size of each of the cluster is available above. The above plot depicts the points in each cluster.

Q2- How many points do you see in each cluster? 74 16 34 44 17 137 2 61 37 are the points within each of the 9 cluster. All these clusters are nominal but not ordinal.

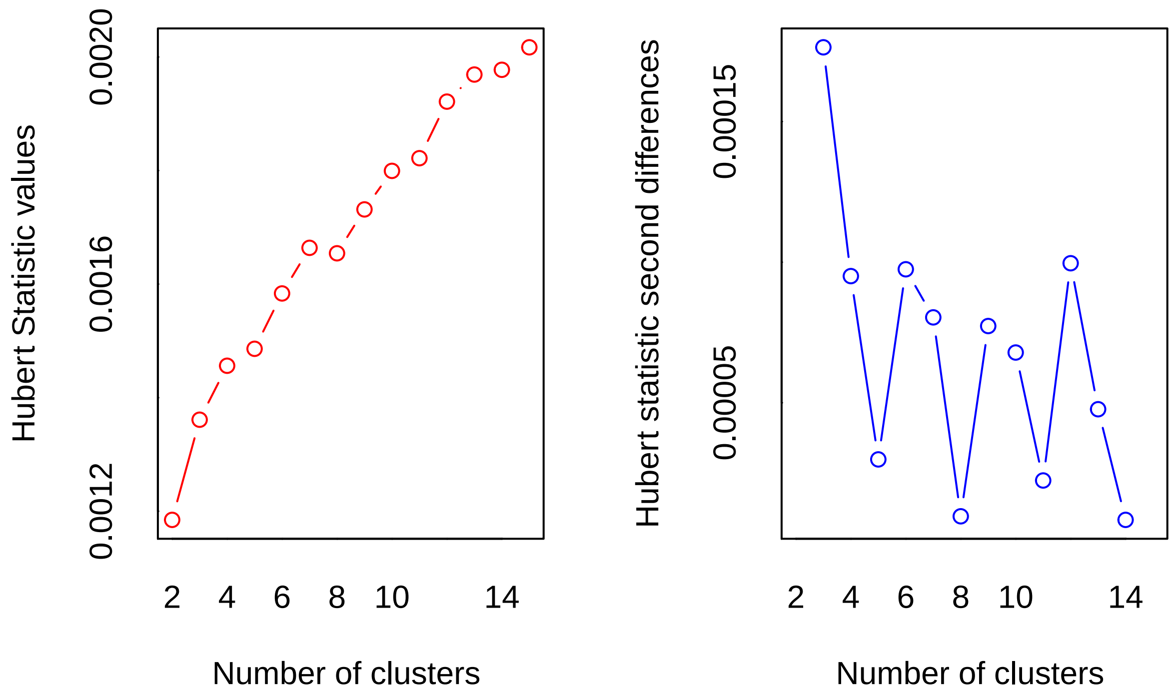
```r
wssplot = function(data, nc=15, seed=1234){
wss =(nrow(data)-1)*sum(apply(data,2,var))
for (i in 2:nc){
set.seed(seed)
wss[i] = sum(kmeans(data, centers=i)$withinss)}
plot(1:nc, wss, type="b", xlab="Number of Clusters", ylab="Within groups sum of squares")}




#Load data into R/RStudio and view it
wine = read.csv("wine.csv")
df = scale(wine[-1])
#Examine the data frame and plot the within sum of squares
head(df)
```

```
##        Alcohol  Malic.acid        Ash        Acl          Mg   Phenols
## [1,] 1.5143408 -0.56066822  0.2313998 -1.1663032 1.90852151 0.8067217
## [2,] 0.2455968 -0.49800856 -0.8256672 -2.4838405 0.01809398 0.5670481
## [3,] 0.1963252  0.02117152  1.1062139 -0.2679823 0.08810981 0.8067217
## [4,] 1.6867914 -0.34583508  0.4865539 -0.8069748 0.92829983 2.4844372
## [5,] 0.2948684  0.22705328  1.8352256  0.4506745 1.27837900 0.8067217
## [6,] 1.4773871 -0.51591132  0.3043010 -1.2860793 0.85828399 1.5576991
##      Flavanoids Nonflavanoid.phenols    Proanth  Color.int        Hue        OD
## [1,]  1.0319081           -0.6577078  1.2214385  0.2510088  0.3611585 1.8427215
## [2,]  0.7315653           -0.8184106 -0.5431887 -0.2924962  0.4049085 1.1103172
## [3,]  1.2121137           -0.4970050  2.1299594  0.2682629  0.3174085 0.7863692
## [4,]  1.4623994           -0.9791134  1.0292513  1.1827317 -0.4263410 1.1807407
## [5,]  0.6614853            0.2261576  0.4002753 -0.3183774  0.3611585 0.4483365
## [6,]  1.3622851           -0.1755994  0.6623487  0.7298108  0.4049085 0.3356589
##          Proline
## [1,]  1.01015939
## [2,]  0.96252635
## [3,]  1.39122370
## [4,]  2.32800680
## [5,] -0.03776747
## [6,]  2.23274072
```

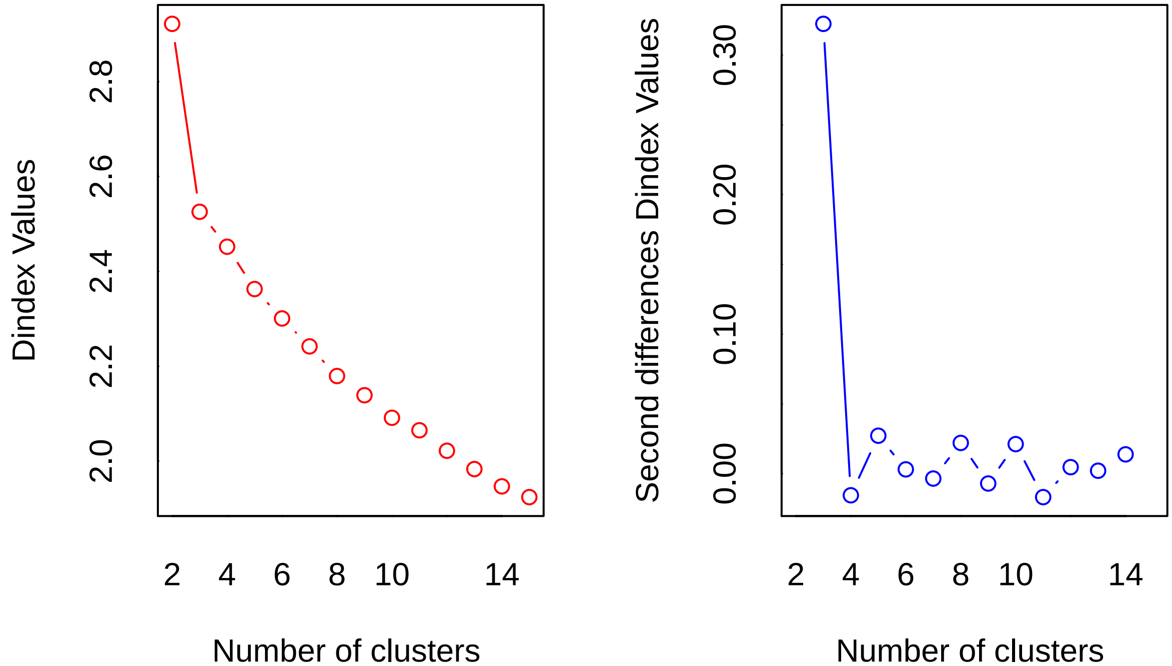
```r
wssplot(df)
```




```
#Start the k-Means analysis using the variable "nc" for the number of clusters
library("NbClust")
set.seed(1234)
nc = NbClust(df, min.nc=2, max.nc = 15, method = "kmeans")
```




```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresponds to a
##                 significant increase of the value of the measure i.e the significant peak in Hubert
##                 index second differences plot.
##
```

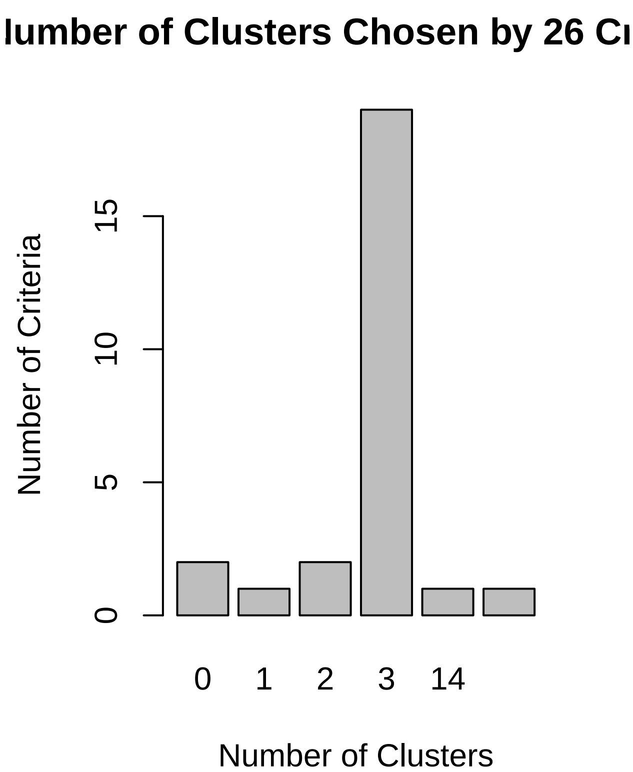



```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Dindex
##                 second differences plot) that corresponds to a significant increase of the value of
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 2 proposed 2 as the best number of clusters
## * 19 proposed 3 as the best number of clusters
## * 1 proposed 14 as the best number of clusters
## * 1 proposed 15 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  3
##
##
## *******************************************************************
```

```r
print(table(nc$Best.n[1,]))
```

```
##
##  0  1  2  3 14 15
##  2  1  2 19  1  1
```

```
barplot(table(nc$Best.n[1,]), xlab = "Number of Clusters", ylab = "Number of Criteria", main = "Number
```




```
#Enter the best number of clusters based on the information in the table and barplot
n = readline(prompt = "Enter the best number of clusters: ")
```

```
## Enter the best number of clusters:
```

```
n = as.integer(n)
n
```

```
## [1] NA
```

```
#Conduct the k-Means analysis using the best number of clusters
set.seed(1234)
fit.km = kmeans(df, 3, nstart=25)
print(fit.km$size)
```

```
## [1] 62 65 51
```

```
print(fit.km$centers)
```

```
##      Alcohol Malic.acid        Ash        Acl          Mg     Phenols
```

```
## 1  0.8328826 -0.3029551  0.3636801 -0.6084749  0.57596208  0.88274724
## 2 -0.9234669 -0.3929331 -0.4931257  0.1701220 -0.49032869 -0.07576891
## 3  0.1644436  0.8690954  0.1863726  0.5228924 -0.07526047 -0.97657548
##    Flavanoids Nonflavanoid.phenols     Proanth  Color.int        Hue         OD
## 1  0.97506900          -0.56050853  0.57865427  0.1705823  0.4726504  0.7770551
## 2  0.02075402          -0.03343924  0.05810161 -0.8993770  0.4605046  0.2700025
## 3 -1.21182921           0.72402116 -0.77751312  0.9388902 -1.1615122 -1.2887761
##      Proline
## 1  1.1220202
## 2 -0.7517257
## 3 -0.4059428
```

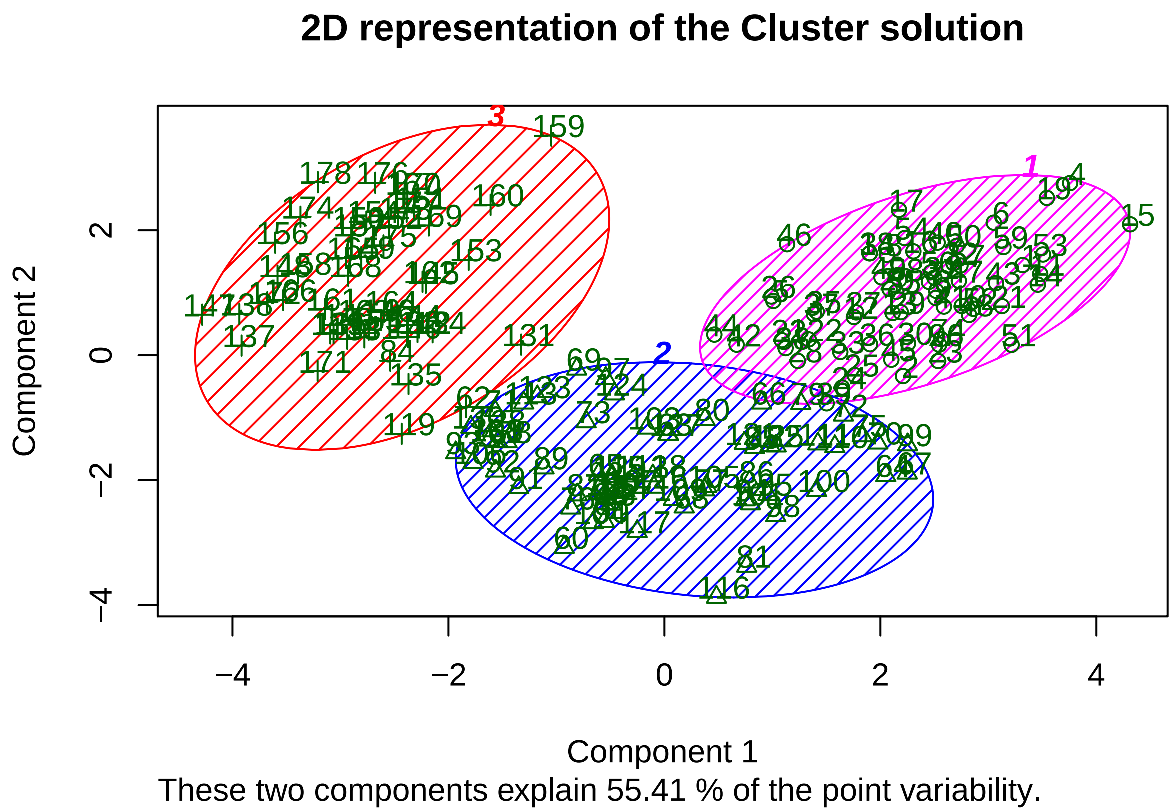
```r
print(aggregate(wine[-1], by=list(cluster=fit.km$cluster), mean))
```

```
##   cluster  Alcohol Malic.acid      Ash      Acl        Mg  Phenols Flavanoids
## 1       1 13.67677   1.997903 2.466290 17.46290 107.96774 2.847581  3.0032258
## 2       2 12.25092   1.897385 2.231231 20.06308  92.73846 2.247692  2.0500000
## 3       3 13.13412   3.307255 2.417647 21.24118  98.66667 1.683922  0.8188235
##   Nonflavanoid.phenols  Proanth Color.int       Hue       OD   Proline
## 1            0.2920968 1.922097  5.453548 1.0654839 3.163387 1100.2258
## 2            0.3576923 1.624154  2.973077 1.0627077 2.803385  510.1692
## 3            0.4519608 1.145882  7.234706 0.6919608 1.696667  619.0588
```

```r
ct.km = table(wine$Wine, fit.km$cluster)
print(ct.km)
```

```
##
##      1  2  3
##   1 59  0  0
##   2  3 65  3
##   3  0  0 48
```

```r
#Generate a plot of the clusters
library(cluster)
clusplot(df, fit.km$cluster, main='2D representation of the Cluster solution',
color=TRUE, shade=TRUE,
labels=2, lines=0)
```



Part 2 Write Up: In the above implementation/analysis, the wine dataset is scaled and later K-means algorithm is performed using the wssplot function for different k values from 1 to 15. After that based on the Hubert and D index, I have come to a understanding that 3 clusters is the best value of K. The size of the 3 clusters are 62, 65 and 51. The centers of the clusters as mentioned above. The 2D representation of the cluster is also mentioned above with the 2 components that explains around 55% of the variability. Later based on the confusion matrix we can determine that only 6 out of 178 are misclassified, i.e the k means algorithm is successful in classifying the type of the wine by 96%

```r
library(rpart)
df = data.frame(k=fit.km$cluster, df)
print(str(df))
```

```
## 'data.frame':    178 obs. of  14 variables:
##  $ k                   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Alcohol             : num  1.514 0.246 0.196 1.687 0.295 ...
##  $ Malic.acid          : num  -0.5607 -0.498 0.0212 -0.3458 0.2271 ...
##  $ Ash                 : num  0.231 -0.826 1.106 0.487 1.835 ...
##  $ Acl                 : num  -1.166 -2.484 -0.268 -0.807 0.451 ...
##  $ Mg                  : num  1.9085 0.0181 0.0881 0.9283 1.2784 ...
##  $ Phenols             : num  0.807 0.567 0.807 2.484 0.807 ...
##  $ Flavanoids          : num  1.032 0.732 1.212 1.462 0.661 ...
##  $ Nonflavanoid.phenols: num  -0.658 -0.818 -0.497 -0.979 0.226 ...
##  $ Proanth             : num  1.221 -0.543 2.13 1.029 0.4 ...
##  $ Color.int           : num  0.251 -0.292 0.268 1.183 -0.318 ...
##  $ Hue                 : num  0.361 0.405 0.317 -0.426 0.361 ...
##  $ OD                  : num  1.843 1.11 0.786 1.181 0.448 ...
```

```
##  $ Proline             : num  1.0102 0.9625 1.3912 2.328 -0.0378 ...
## NULL
```

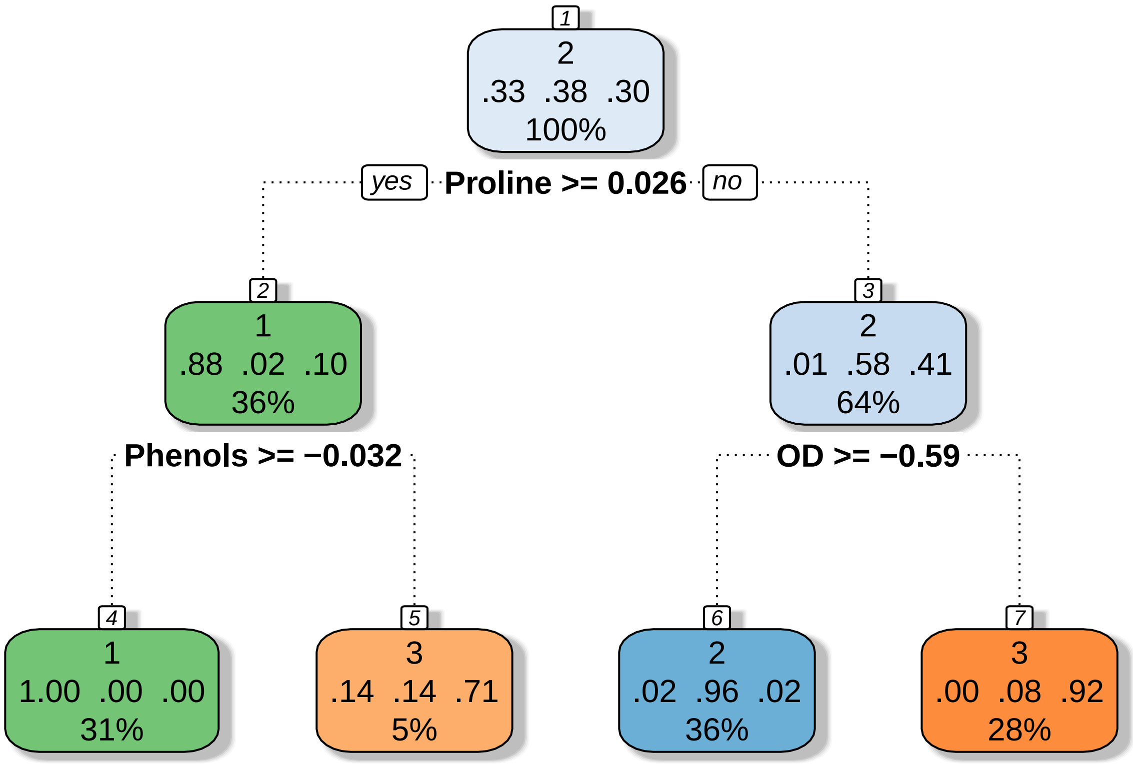
```
#Randomize the dataset
rdf = df[sample(1:nrow(df)), ]
print(head(rdf))
```

```
##     k    Alcohol Malic.acid        Ash        Acl          Mg    Phenols
## 127 2 -0.7028817 -0.7217931 -0.2789084  0.6003946 -0.96212770  0.7108523
## 155 3 -0.5181131 -0.9366262 -0.9714696  0.1512342  0.22814148 -1.3024063
## 72  2  1.0585784 -0.7396958  1.1062139  1.6484357 -0.96212770  1.0463954
## 48  1  1.1078500 -0.5875224 -0.8985684 -1.0465271  0.08810981  1.2860690
## 60  2 -0.7767891 -1.2499245 -3.6688130 -2.6635047 -0.82209603 -0.5034942
## 36  1  0.5904981 -0.4711544  0.1584986  0.3009543  0.01809398  0.6469393
##     Flavanoids Nonflavanoid.phenols    Proanth   Color.int        Hue
## 127  1.1220109            0.2261576  0.3129175 -0.48229163 -1.1700906
## 155 -1.4509256            1.3510772 -0.3335300  1.09646103 -1.6513403
## 72   0.8316795           -1.2201676  0.4876331 -0.72384942  1.7611577
## 48   1.3622851           -1.2201676  0.9593651  0.44943125 -0.2075912
## 60  -1.4609371           -0.6577078 -2.0457425 -1.34068448  0.4049085
## 36   0.9518167           -0.8184106  0.4701615  0.01807806  0.3611585
##             OD    Proline
## 127  0.3215742 -1.2539977
## 155 -1.4953517 -0.3394434
## 72   0.7722845 -1.0698166
## 48   1.0117244  0.7561165
## 60  -1.1150649 -0.7205077
## 36   1.2089101  0.5497067
```

```
train = rdf[1:(as.integer(.8*nrow(rdf))-1), ]
test = rdf[(as.integer(.8*nrow(rdf))):nrow(rdf), ]
#Train the classifier and plot the results
fit = rpart(k ~ ., data=train, method="class")
library(rpart.plot)
library(RColorBrewer)
library(rattle)
```

```
## Loading required package: tibble

## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

##
## Attaching package: 'rattle'

## The following object is masked _by_ '.GlobalEnv':
##
##     wine
```

```r
fancyRpartPlot(fit)
```



Rattle 2020−Nov−29 21:31:26 akhilasaineni

```r
#Now use the predict() function to see how well the model works
pred=predict(fit, test, type="class")
print(table(pred, test$k))
```

```
##
## pred  1  2  3
##    1 14  1  0
##    2  2  9  0
##    3  0  2  9
```

Part 3

Write Up: As mentioned above,we have a 4% misclassification in the K means algorithm. The classification algorithm above clearly predicted the cluster without any misclassifaction (from the truth table). This tells us that the 4 % misclassification will be present in our k means algorithm and this also can be estimated. This misclassifcation will flow through the entire analysis based on the fact that, the classification method used above followed the same clustering format as the k means algorithm.

Q3- Load the dataset of breast cancer. Do the preliminary analysis and implement a KNN (K- nearest neighbors) model for this dataset and don't forget that whenever it is required you should use: set.seed(12345).

```r
bc= read.csv("wisc_bc_data.csv")


nor =function(x) { (x -min(x))/(max(x)-min(x))   }
#bc_norm<-as.data.frame(lapply(bc[,c(-1,-2)], nor))

set.seed(12345)
bc_rand =bc[order(runif(569)), ]


bc_train = bc[1:455, ]
bc_test = bc_rand[456:569, ]

bc_train_norm = as.data.frame(lapply(bc_train[,c(-1,-2)], nor))
bc_test_norm= as.data.frame(lapply(bc_test[,c(-1,-2)], nor))


#install.packages("class")

library(class)

pr=knn(train =bc_train_norm , test =bc_test_norm, cl=bc_train$diagnosis, k=13)



 tab = table(pr,bc_test$diagnosis)

 tab
```

```
##
## pr   B  M
##   B 80  0
##   M  0 34
```

```r
 accuracy = function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
 accuracy(tab)
```

```
## [1] 100
```

Part 4 Q1 Write up: Using the wisc_bc_data.csv dataset, I have implemented a KNN algorithm, where we first split the data into train(80%) & test(20%) with seed set to 12345. Then we have normalized the data, so that all values are between 0 and 1. Various values of K have been tried and the best value of K is 13, Where we achieve a 100% accuracy in the model.

```r
news_p=read.csv("OnlineNewsPopularity_for_R.csv")


head(news_p)
```

```
##                                                              url timedelta
## 1   http://mashable.com/2013/01/07/amazon-instant-video-browser/       731
## 2    http://mashable.com/2013/01/07/ap-samsung-sponsored-tweets/       731
```

```
## 3 http://mashable.com/2013/01/07/apple-40-billion-app-downloads/       731
## 4       http://mashable.com/2013/01/07/astronaut-notre-dame-bcs/       731
## 5               http://mashable.com/2013/01/07/att-u-verse-apps/       731
## 6               http://mashable.com/2013/01/07/beewi-smart-toys/       731
##   n_tokens_title n_tokens_content n_unique_tokens n_non_stop_words
## 1             12              219       0.6635945                1
## 2              9              255       0.6047431                1
## 3              9              211       0.5751295                1
## 4              9              531       0.5037879                1
## 5             13             1072       0.4156456                1
## 6             10              370       0.5598886                1
##   n_non_stop_unique_tokens num_hrefs num_self_hrefs num_imgs num_videos
## 1                0.8153846         4              2        1          0
## 2                0.7919463         3              1        1          0
## 3                0.6638655         3              1        1          0
## 4                0.6656347         9              0        1          0
## 5                0.5408895        19             19       20          0
## 6                0.6981982         2              2        0          0
##   average_token_length num_keywords data_channel_is_lifestyle
## 1             4.680365            5                         0
## 2             4.913725            4                         0
## 3             4.393365            6                         0
## 4             4.404896            7                         0
## 5             4.682836            7                         0
## 6             4.359459            9                         0
##   data_channel_is_entertainment data_channel_is_bus data_channel_is_socmed
## 1                             1                   0                      0
## 2                             0                   1                      0
## 3                             0                   1                      0
## 4                             1                   0                      0
## 5                             0                   0                      0
## 6                             0                   0                      0
##   data_channel_is_tech data_channel_is_world kw_min_min kw_max_min kw_avg_min
## 1                    0                     0          0          0          0
## 2                    0                     0          0          0          0
## 3                    0                     0          0          0          0
## 4                    0                     0          0          0          0
## 5                    1                     0          0          0          0
## 6                    1                     0          0          0          0
##   kw_min_max kw_max_max kw_avg_max kw_min_avg kw_max_avg kw_avg_avg
## 1          0          0          0          0          0          0
## 2          0          0          0          0          0          0
## 3          0          0          0          0          0          0
## 4          0          0          0          0          0          0
## 5          0          0          0          0          0          0
## 6          0          0          0          0          0          0
##   self_reference_min_shares self_reference_max_shares
## 1                       496                       496
## 2                         0                         0
## 3                       918                       918
## 4                         0                         0
## 5                       545                     16000
## 6                      8500                      8500
##   self_reference_avg_sharess weekday_is_monday weekday_is_tuesday
```

```
## 1                 496.000                1                 0
## 2                   0.000                1                 0
## 3                 918.000                1                 0
## 4                   0.000                1                 0
## 5                3151.158                1                 0
## 6                8500.000                1                 0
##   weekday_is_wednesday weekday_is_thursday weekday_is_friday
## 1                    0                   0                 0
## 2                    0                   0                 0
## 3                    0                   0                 0
## 4                    0                   0                 0
## 5                    0                   0                 0
## 6                    0                   0                 0
##   weekday_is_saturday weekday_is_sunday is_weekend     LDA_00     LDA_01
## 1                   0                 0          0 0.50033120 0.37827893
## 2                   0                 0          0 0.79975569 0.05004668
## 3                   0                 0          0 0.21779229 0.03333446
## 4                   0                 0          0 0.02857322 0.41929964
## 5                   0                 0          0 0.02863281 0.02879355
## 6                   0                 0          0 0.02224528 0.30671758
##       LDA_02     LDA_03     LDA_04 global_subjectivity
## 1 0.04000468 0.04126265 0.04012254           0.5216171
## 2 0.05009625 0.05010067 0.05000071           0.3412458
## 3 0.03335142 0.03333354 0.68218829           0.7022222
## 4 0.49465083 0.02890472 0.02857160           0.4298497
## 5 0.02857518 0.02857168 0.88542678           0.5135021
## 6 0.02223128 0.02222429 0.62658158           0.4374086
##   global_sentiment_polarity global_rate_positive_words
## 1                0.09256198                 0.04566210
## 2                0.14894781                 0.04313725
## 3                0.32333333                 0.05687204
## 4                0.10070467                 0.04143126
## 5                0.28100348                 0.07462687
## 6                0.07118419                 0.02972973
##   global_rate_negative_words rate_positive_words rate_negative_words
## 1                0.013698630           0.7692308           0.2307692
## 2                0.015686275           0.7333333           0.2666667
## 3                0.009478673           0.8571429           0.1428571
## 4                0.020715631           0.6666667           0.3333333
## 5                0.012126866           0.8602151           0.1397849
## 6                0.027027027           0.5238095           0.4761905
##   avg_positive_polarity min_positive_polarity max_positive_polarity
## 1             0.3786364            0.10000000                   0.7
## 2             0.2869146            0.03333333                   0.7
## 3             0.4958333            0.10000000                   1.0
## 4             0.3859652            0.13636364                   0.8
## 5             0.4111274            0.03333333                   1.0
## 6             0.3506100            0.13636364                   0.6
##   avg_negative_polarity min_negative_polarity max_negative_polarity
## 1            -0.3500000                -0.600            -0.2000000
## 2            -0.1187500                -0.125            -0.1000000
## 3            -0.4666667                -0.800            -0.1333333
## 4            -0.3696970                -0.600            -0.1666667
## 5            -0.2201923                -0.500            -0.0500000
```

```
## 6           -0.1950000                -0.400           -0.1000000
##   title_subjectivity title_sentiment_polarity abs_title_subjectivity
## 1          0.5000000               -0.1875000             0.00000000
## 2          0.0000000                0.0000000             0.50000000
## 3          0.0000000                0.0000000             0.50000000
## 4          0.0000000                0.0000000             0.50000000
## 5          0.4545455                0.1363636             0.04545455
## 6          0.6428571                0.2142857             0.14285714
##   abs_title_sentiment_polarity shares
## 1                    0.1875000    593
## 2                    0.0000000    711
## 3                    0.0000000   1500
## 4                    0.0000000   1200
## 5                    0.1363636    505
## 6                    0.2142857    855
```

```r
str(news_p)
```

```
## 'data.frame':    39644 obs. of  61 variables:
##  $ url                          : Factor w/ 39644 levels "http://mashable.com/2013/01/07/amazon-insta
##  $ timedelta                    : num  731 731 731 731 731 731 731 731 731 731 ...
##  $ n_tokens_title               : num  12 9 9 9 13 10 8 12 11 10 ...
##  $ n_tokens_content             : num  219 255 211 531 1072 ...
##  $ n_unique_tokens              : num  0.664 0.605 0.575 0.504 0.416 ...
##  $ n_non_stop_words             : num  1 1 1 1 1 ...
##  $ n_non_stop_unique_tokens     : num  0.815 0.792 0.664 0.666 0.541 ...
##  $ num_hrefs                    : num  4 3 3 9 19 2 21 20 2 4 ...
##  $ num_self_hrefs               : num  2 1 1 0 19 2 20 20 0 1 ...
##  $ num_imgs                     : num  1 1 1 1 20 0 20 20 0 1 ...
##  $ num_videos                   : num  0 0 0 0 0 0 0 0 0 1 ...
##  $ average_token_length         : num  4.68 4.91 4.39 4.4 4.68 ...
##  $ num_keywords                 : num  5 4 6 7 7 9 10 9 7 5 ...
##  $ data_channel_is_lifestyle    : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ data_channel_is_entertainment: num  1 0 0 1 0 0 0 0 0 0 ...
##  $ data_channel_is_bus          : num  0 1 1 0 0 0 0 0 0 0 ...
##  $ data_channel_is_socmed       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ data_channel_is_tech         : num  0 0 0 0 1 1 0 1 1 0 ...
##  $ data_channel_is_world        : num  0 0 0 0 0 0 0 0 0 1 ...
##  $ kw_min_min                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ kw_max_min                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ kw_avg_min                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ kw_min_max                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ kw_max_max                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ kw_avg_max                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ kw_min_avg                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ kw_max_avg                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ kw_avg_avg                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ self_reference_min_shares    : num  496 0 918 0 545 8500 545 545 0 0 ...
##  $ self_reference_max_shares    : num  496 0 918 0 16000 8500 16000 16000 0 0 ...
##  $ self_reference_avg_sharess   : num  496 0 918 0 3151 ...
##  $ weekday_is_monday            : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ weekday_is_tuesday           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ weekday_is_wednesday         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ weekday_is_thursday          : num  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ weekday_is_friday            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ weekday_is_saturday          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ weekday_is_sunday            : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ is_weekend                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ LDA_00                       : num  0.5003 0.7998 0.2178 0.0286 0.0286 ...
##  $ LDA_01                       : num  0.3783 0.05 0.0333 0.4193 0.0288 ...
##  $ LDA_02                       : num  0.04 0.0501 0.0334 0.4947 0.0286 ...
##  $ LDA_03                       : num  0.0413 0.0501 0.0333 0.0289 0.0286 ...
##  $ LDA_04                       : num  0.0401 0.05 0.6822 0.0286 0.8854 ...
##  $ global_subjectivity          : num  0.522 0.341 0.702 0.43 0.514 ...
##  $ global_sentiment_polarity    : num  0.0926 0.1489 0.3233 0.1007 0.281 ...
##  $ global_rate_positive_words   : num  0.0457 0.0431 0.0569 0.0414 0.0746 ...
##  $ global_rate_negative_words   : num  0.0137 0.01569 0.00948 0.02072 0.01213 ...
##  $ rate_positive_words          : num  0.769 0.733 0.857 0.667 0.86 ...
##  $ rate_negative_words          : num  0.231 0.267 0.143 0.333 0.14 ...
##  $ avg_positive_polarity        : num  0.379 0.287 0.496 0.386 0.411 ...
##  $ min_positive_polarity        : num  0.1 0.0333 0.1 0.1364 0.0333 ...
##  $ max_positive_polarity        : num  0.7 0.7 1 0.8 1 0.6 1 1 0.8 0.5 ...
##  $ avg_negative_polarity        : num  -0.35 -0.119 -0.467 -0.37 -0.22 ...
##  $ min_negative_polarity        : num  -0.6 -0.125 -0.8 -0.6 -0.5 -0.4 -0.5 -0.5 -0.125 -0.5 ...
##  $ max_negative_polarity        : num  -0.2 -0.1 -0.133 -0.167 -0.05 ...
##  $ title_subjectivity           : num  0.5 0 0 0 0.455 ...
##  $ title_sentiment_polarity     : num  -0.188 0 0 0 0.136 ...
##  $ abs_title_subjectivity       : num  0 0.5 0.5 0.5 0.0455 ...
##  $ abs_title_sentiment_polarity : num  0.188 0 0 0 0.136 ...
##  $ shares                       : int  593 711 1500 1200 505 855 556 891 3600 710 ...
```

```r
colnames(news_p)
```

```
##  [1] "url"                           "timedelta"
##  [3] "n_tokens_title"                "n_tokens_content"
##  [5] "n_unique_tokens"               "n_non_stop_words"
##  [7] "n_non_stop_unique_tokens"      "num_hrefs"
##  [9] "num_self_hrefs"                "num_imgs"
## [11] "num_videos"                    "average_token_length"
## [13] "num_keywords"                  "data_channel_is_lifestyle"
## [15] "data_channel_is_entertainment" "data_channel_is_bus"
## [17] "data_channel_is_socmed"        "data_channel_is_tech"
## [19] "data_channel_is_world"         "kw_min_min"
## [21] "kw_max_min"                    "kw_avg_min"
## [23] "kw_min_max"                    "kw_max_max"
## [25] "kw_avg_max"                    "kw_min_avg"
## [27] "kw_max_avg"                    "kw_avg_avg"
## [29] "self_reference_min_shares"     "self_reference_max_shares"
## [31] "self_reference_avg_sharess"    "weekday_is_monday"
## [33] "weekday_is_tuesday"            "weekday_is_wednesday"
## [35] "weekday_is_thursday"           "weekday_is_friday"
## [37] "weekday_is_saturday"           "weekday_is_sunday"
## [39] "is_weekend"                    "LDA_00"
## [41] "LDA_01"                        "LDA_02"
## [43] "LDA_03"                        "LDA_04"
## [45] "global_subjectivity"           "global_sentiment_polarity"
## [47] "global_rate_positive_words"    "global_rate_negative_words"
## [49] "rate_positive_words"           "rate_negative_words"
```

```
## [51] "avg_positive_polarity"         "min_positive_polarity"
## [53] "max_positive_polarity"         "avg_negative_polarity"
## [55] "min_negative_polarity"         "max_negative_polarity"
## [57] "title_subjectivity"            "title_sentiment_polarity"
## [59] "abs_title_subjectivity"        "abs_title_sentiment_polarity"
## [61] "shares"
```

```
news_p = news_p[,c("n_tokens_title", "n_tokens_content", "n_unique_tokens", "n_non_stop_words", "num_hre
for(i in 1:39644) {


 news_p$fav[i]= if( news_p$shares[i]>=1400) {"YES"} else {"NO"}

}
```

```
set.seed(12345)
news_p_rand = news_p[order(runif(10000)), ]


news_ptrain = news_p_rand[1:9000, ]
news_ptest = news_p_rand[9001:10000, ]

#prop.table(table(news_ptrain$fav))
#prop.table(table(news_ptest$fav))




nor =function(x) { (x -min(x))/(max(x)-min(x))   }


news_ptrain_norm = as.data.frame(lapply(news_ptrain[,c(-18,-19)], nor))
news_ptest_norm= as.data.frame(lapply(news_ptest[,c(-18,-19)], nor))
```

```
pr2=knn(train =news_ptrain_norm , test =news_ptest_norm, cl=news_ptrain$fav, k=499)

 tab2 = table(pr2,news_ptest$fav)

 tab2
```

```
##
## pr2    NO YES
##   NO   24  23
##   YES 390 563
```

```
 accuracy2 = function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
 accuracy2(tab2)
```

```
## [1] 58.7
```

Part 4 Q2 Write up: I have implemented the KNN algorithm on the news popularity dataset with various K values, the best output is when the K value is 499. The accuracy of the model is 59% which is much less than the accuracy achieved with svm polynomial kernel. I have used a trial and error method by trying various K values to see which one yeild higher accuracy.