Clustering

Dr. Marko Mitic

Problem 1: DOCUMENT CLUSTERING WITH DAILY KOS

Document clustering, or text clustering, is a very popular application of clustering algorithms. A web search engine, like Google, often returns thousands of results for a simple query. For example, if you type the search term "jaguar" into Google, around 200 million results are returned. This makes it very difficult to browse or find relevant information, especially if the search term has multiple meanings. If we search for "jaguar", we might be looking for information about the animal, the car, or the Jacksonville Jaguars football team.

Clustering methods can be used to automatically group search results into categories, making it easier to find relavent results. This method is used in the search engines PolyMeta and Helioid, as well as on FirstGov.gov, the official Web portal for the U.S. government. The two most common algorithms used for document clustering are Hierarchical and k-means.

In this problem, we'll be clustering articles published on Daily Kos https://www.dailykos.com/, an American political blog that publishes news and opinion articles written from a progressive point of view. Daily Kos was founded by Markos Moulitsas in 2002, and as of September 2014, the site had an average weekday traffic of hundreds of thousands of visits.

The file dailykos.csv contains data on 3,430 news articles or blogs that have been posted on Daily Kos. These articles were posted in 2004, leading up to the United States Presidential Election. The leading candidates were incumbent President George W. Bush (republican) and John Kerry (democratic). Foreign policy was a dominant topic of the election, specifically, the 2003 invasion of Iraq.

Each of the variables in the dataset is a word that has appeared in at least 50 different articles (1,545 words in total). The set of words has been trimmed according to some of the techniques covered in the previous week on text analytics (punctuation has been removed, and stop words have been removed). For each document, the variable values are the number of times that word appeared in the document.

Let's start by building a hierarchical clustering model.

```r
dailykos = read.csv("dailykos.csv")
str(dailykos)
```

```
## 'data.frame':    3430 obs. of  1545 variables:
##  $ abandon                                : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ abc                                    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ ability                                : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ abortion                               : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ absolute                               : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ abstain                                : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ abu                                    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ abuse                                  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ accept                                 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ access                                 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ accomplish                             : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ account                                : int  0 0 2 0 0 0 0 0 0 0 ...
##  $ accurate                               : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ accusations                            : int  0 0 0 2 0 0 0 0 0 0 ...
##  $ achieve                                : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ acknowledge                            : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ act                                    : int  0 0 0 0 0 0 0 0 0 0 ...
```

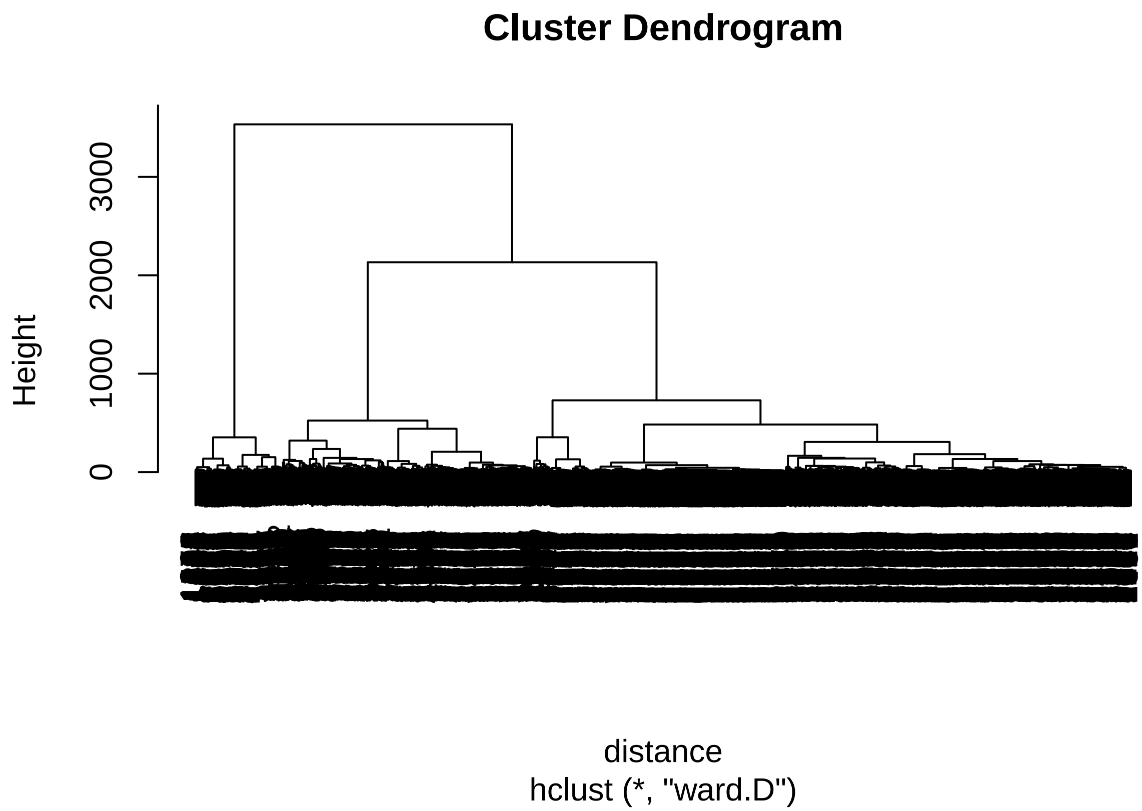
```
##  $ action                                   : int  2 0 0 0 0 0 0 0 0 0 ...
##  $ active                                   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ activist                                 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ actual                                   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ add                                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ added                                    : int  1 0 0 0 1 0 0 0 1 0 ...
##  $ addition                                 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ address                                  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ admin                                    : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ administration                           : int  1 0 0 0 0 0 0 0 0 0 ...
##  $ admit                                    : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ advance                                  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ advantage                                : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ advertise                                : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ advised                                  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ affair                                   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ affect                                   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ affiliate                                : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ afghanistan                              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ afraid                                   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ afternoon                                : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ age                                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ agencies                                 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ agenda                                   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ agree                                    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ ahead                                    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ aid                                      : int  0 0 0 1 1 0 0 0 0 0 ...
##  $ aim                                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ air                                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ alaska                                   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ allegation                               : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ allegory                                 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ allied                                   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ allowed                                  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ alternative                              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ altsite                                  : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ amazing                                  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ amendment                                : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ america                                  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ american                                 : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ amount                                   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ amp                                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ analysis                                 : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ analyst                                  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ anecdotal                                : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ anger                                    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ angry                                    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ announce                                 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ annual                                   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ answer                                   : int  0 0 0 1 0 0 1 0 0 0 ...
##  $ apologies                                : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ apparent                                 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ appeal                                   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ appearance                               : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ applied                                    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ appointed                                  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ approach                                   : int  0 0 0 0 0 0 1 0 0 0 ...
##  $ approval                                   : int  1 0 0 0 1 0 0 0 1 0 ...
##  $ apr                                        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ april                                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ arab                                       : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ area                                       : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ arent                                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ arg                                        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ argue                                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ argument                                   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ arizona                                    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ arm                                        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ armstrong                                  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ army                                       : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ arrest                                     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ arrive                                     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ article                                    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ asap                                       : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ asked                                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ ass                                        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ assess                                     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ assist                                     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ associate                                  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ assume                                     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ atlanta                                    : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ atrios                                     : int  0 0 0 0 0 0 1 0 0 0 ...
##   [list output truncated]
```

```r
distance = dist(dailykos, method = "euclidean")
cluster = hclust(distance, method = "ward.D")
```

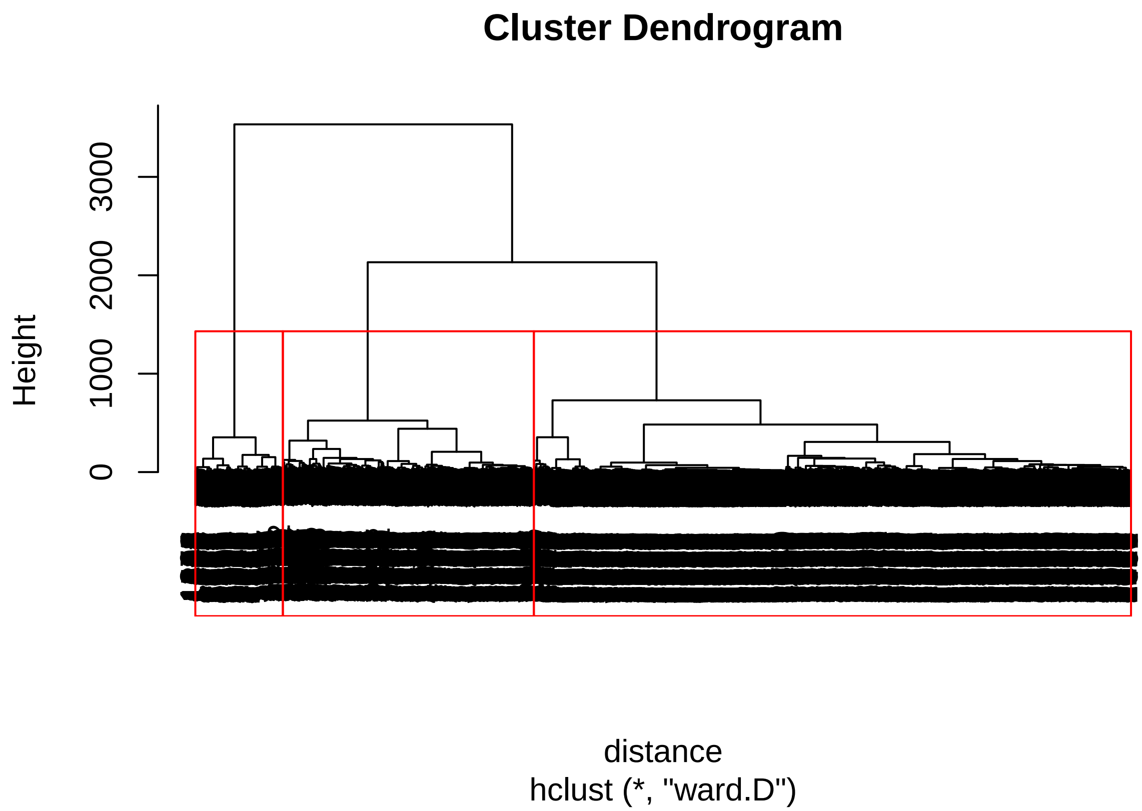
The computation may take some time, since we have lots of observations and variables in the dataset. Let us next plot the dendogram:

```r
plot(cluster)
```



The choices 2 and 3 are good cluster choices according to the dendrogram, because there is a lot of space between the horizontal lines in the dendrogram in those cut off spots (draw a horizontal line across the dendrogram where it crosses 2 or 3 vertical lines). This can be shown by using `rect.hist` function for drawing cluster:

```
plot(cluster)
rect.hclust(cluster, k=3, border="red")
```



However, just thinking about the application, it is probably better to show the reader more categories than 2 or 3. These categories would probably be too broad to be useful. Seven or eight categories seems more reasonable. Let us next subset each of the seven clusters:

```
clusterGroups =cutree(cluster, k = 7)
cluster1 = subset(dailykos, clusterGroups == 1)
cluster2 = subset(dailykos, clusterGroups == 2)
cluster3 = subset(dailykos, clusterGroups == 3)
cluster4 = subset(dailykos, clusterGroups == 4)
cluster5 = subset(dailykos, clusterGroups == 5)
cluster6 = subset(dailykos, clusterGroups == 6)
cluster7 = subset(dailykos, clusterGroups == 7)
```

By using `str` function we observe that cluster1 contains most observations, while cluster4 has lowest number of them. We can also see the frequency for each varaible in each cluster. Combination of `tail`, `sort` and `colMeans` computes the mean frequency values of each of the words in cluster, and then outputs the 6 words that occur the most frequently. The colMeans function computes the column (word) means, the sort function orders the words in increasing order of the mean values, and the tail function outputs the last 6 words listed, which are the ones with the largest column means.

```
tail(sort(colMeans(cluster1)))
```

```
##      state republican       poll   democrat      kerry       bush
##  0.7575039  0.7590837  0.9036335  0.9194313  1.0624013  1.7053712
```

We observe that the word “bush” is most frequent word in this cluster. For cluster 2 this are words “november” and “poll”.

```r
tail(sort(colMeans(cluster2)))
```

```
##      bush  democrat challenge      vote      poll  november
##  2.847352  2.850467  4.096573  4.398754  4.847352 10.339564
```

Next, we can run k-means algorithm, to find new patterns.

```r
k = 7 #seven clusters
set.seed(1000)
KMC = kmeans(dailykos, centers = k)
```

We now subset the `KMC`, as in hierarchical clustering:

```r
dailykosClusters = KMC$cluster
```

The number observations in each cluster can be determined using sum function:

```r
sum(dailykosClusters==1)
```

```
## [1] 146
```

```r
sum(dailykosClusters==2)
```

```
## [1] 144
```

```r
sum(dailykosClusters==3)
```

```
## [1] 277
```

```r
sum(dailykosClusters==4)
```

```
## [1] 2063
```

```r
sum(dailykosClusters==5)
```

```
## [1] 163
```

```r
sum(dailykosClusters==6)
```

```
## [1] 329
```

```r
sum(dailykosClusters==7)
```

```
## [1] 308
```

```r
# or using:
KmeansCluster = split(dailykos, dailykosClusters)
#str(KmeansCluster)
```

It can be observed that cluster 4 and cluster 2 have largest and smallest number of observations. This is, of course, different comparing hierarchical clustering case. Most frequent terms can also be obtained using cobination of `tail`, `sort` and `colMeans`:

```r
KmeansCluster1 = subset(dailykos, KMC$cluster == 1)
KmeansCluster2 = subset(dailykos, KMC$cluster == 2)
KmeansCluster3 = subset(dailykos, KMC$cluster == 3)
KmeansCluster4 = subset(dailykos, KMC$cluster == 4)
KmeansCluster5 = subset(dailykos, KMC$cluster == 5)
KmeansCluster6 = subset(dailykos, KMC$cluster == 6)
KmeansCluster7 = subset(dailykos, KMC$cluster == 7)
tail(sort(colMeans(KmeansCluster1)))
```

```
##          state           iraq          kerry administration       presided
##       1.609589       1.616438       1.636986       2.664384       2.767123
##           bush
##      11.431507
```

Comparing these results with hierarchical clustering, we can determine the similarity of each cluster. For example, using the `table` function, we observe that the hierarchical cluster 7 is most similar to K-means cluster 2:

```r
table(clusterGroups, KMC$cluster)
```

```
##
## clusterGroups    1    2    3    4    5    6    7
##             1    3   11   64 1045   32    0  111
##             2    0    0    0    0    0  320    1
##             3   85   10   42   79  126    8   24
##             4   10    5    0    0    1    0  123
##             5   48    0  171  145    3    1   39
##             6    0    2    0  712    0    0    0
##             7    0  116    0   82    1    0   10
```

Similarly, it is interesting to note that K-means cluster 6 is almost identical to hierarchical cluster 2. We can also conclude that no more than 123 (39.9%) of the observations in K-Means Cluster 7 fall in any hierarchical cluster.

Problem 2: MARKET SEGMENTATION FOR AIRLINES

Market segmentation is a strategy that divides a broad target market of customers into smaller, more similar groups, and then designs a marketing strategy specifically for each group. Clustering is a common technique for market segmentation since it automatically finds similar groups given a data set.

In this problem, we'll see how clustering can be used to find similar groups of customers who belong to an airline's frequent flyer program. The airline is trying to learn more about its customers so that it can target different customer segments with different types of mileage offers.

The file AirlinesCluster.csv contains information on 3,999 members of the frequent flyer program. This data comes from the textbook "Data Mining for Business Intelligence," by Galit Shmueli, Nitin R. Patel, and Peter C. Bruce. For more information, see the website for the book http://www.dataminingbook.com/.

There are seven different variables in the dataset, described below:

- **Balance** = number of miles eligible for award travel
- **QualMiles** = number of miles qualifying for TopFlight status
- **BonusMiles** = number of miles earned from non-flight bonus transactions in the past 12 months
- **BonusTrans** = number of non-flight bonus transactions in the past 12 months
- **FlightMiles** = number of flight miles in the past 12 months
- **FlightTrans** = number of flight transactions in the past 12 months
- **DaysSinceEnroll** = number of days since enrolled in the frequent flyer program

First,let's load the dataset and look at statistical summary:

```r
airlines = read.csv("AirlinesCluster.csv")
str(airlines)
```

```
## 'data.frame':    3999 obs. of  7 variables:
##  $ Balance        : int  28143 19244 41354 14776 97752 16420 84914 20856 443003 104860 ...
##  $ QualMiles      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ BonusMiles     : int  174 215 4123 500 43300 0 27482 5250 1753 28426 ...
##  $ BonusTrans     : int  1 2 4 1 26 0 25 4 43 28 ...
##  $ FlightMiles    : int  0 0 0 0 2077 0 0 250 3850 1150 ...
##  $ FlightTrans    : int  0 0 0 0 4 0 0 1 12 3 ...
##  $ DaysSinceEnroll: int  7000 6968 7034 6952 6935 6942 6994 6938 6948 6931 ...
```

```r
summary(airlines)
```

```
##     Balance          QualMiles         BonusMiles       BonusTrans
##  Min.   :      0   Min.   :    0.0   Min.   :     0   Min.   : 0.0
##  1st Qu.:  18528   1st Qu.:    0.0   1st Qu.:  1250   1st Qu.: 3.0
##  Median :  43097   Median :    0.0   Median :  7171   Median :12.0
##  Mean   :  73601   Mean   :  144.1   Mean   : 17145   Mean   :11.6
##  3rd Qu.:  92404   3rd Qu.:    0.0   3rd Qu.: 23801   3rd Qu.:17.0
##  Max.   :1704838   Max.   :11148.0   Max.   :263685   Max.   :86.0
##   FlightMiles       FlightTrans     DaysSinceEnroll
##  Min.   :    0.0   Min.   : 0.000   Min.   :   2
##  1st Qu.:    0.0   1st Qu.: 0.000   1st Qu.:2330
##  Median :    0.0   Median : 0.000   Median :4096
##  Mean   :  460.1   Mean   : 1.374   Mean   :4119
##  3rd Qu.:  311.0   3rd Qu.: 1.000   3rd Qu.:5790
##  Max.   :30817.0   Max.   :53.000   Max.   :8296
```

It is obious that firstly we need to normalize the data. If we don't normalize the data, the clustering will be dominated by the variables that are on a larger scale.This is done next in our analysis:

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

```
## Warning: package 'caret' was built under R version 3.2.1
```

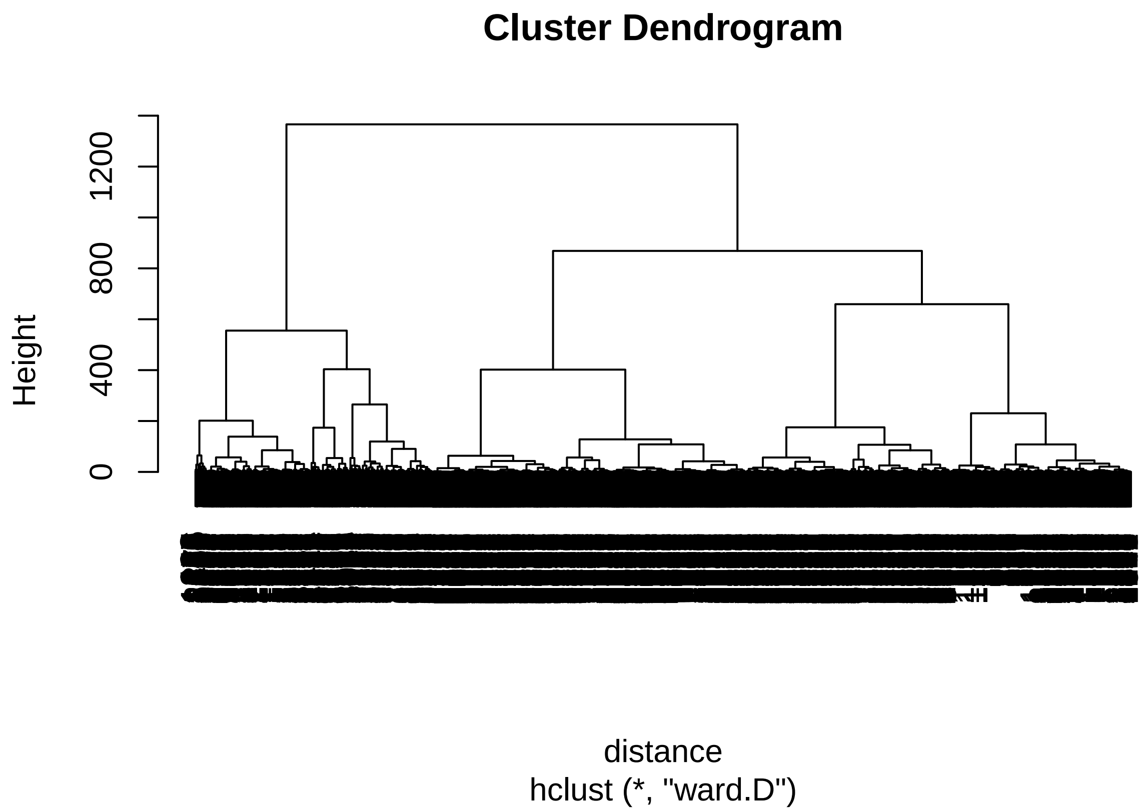
```
## Loading required package: lattice
## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.2.1
```

```
preproc = preProcess(airlines)
airlinesNorm = predict(preproc, airlines)
summary(airlinesNorm)
```

```
##     Balance          QualMiles         BonusMiles        BonusTrans
##  Min.   :-0.7303   Min.   :-0.1863   Min.   :-0.7099   Min.   :-1.20805
##  1st Qu.:-0.5465   1st Qu.:-0.1863   1st Qu.:-0.6581   1st Qu.:-0.89568
##  Median :-0.3027   Median :-0.1863   Median :-0.4130   Median : 0.04145
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 0.1866   3rd Qu.:-0.1863   3rd Qu.: 0.2756   3rd Qu.: 0.56208
##  Max.   :16.1868   Max.   :14.2231   Max.   :10.2083   Max.   : 7.74673
##   FlightMiles       FlightTrans        DaysSinceEnroll
##  Min.   :-0.3286   Min.   :-0.36212   Min.   :-1.99336
##  1st Qu.:-0.3286   1st Qu.:-0.36212   1st Qu.:-0.86607
##  Median :-0.3286   Median :-0.36212   Median :-0.01092
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000
##  3rd Qu.:-0.1065   3rd Qu.:-0.09849   3rd Qu.: 0.80960
##  Max.   :21.6803   Max.   :13.61035   Max.   : 2.02284
```

One can see from the output that FlightMiles now has the largest maximum value, and DaysSinceEnroll now has the smallest minimum value. Note that these were not the variables with the largest and smallest values in the original dataset airlines. Next, we're going to develop hierarchical clustering model:

```
distance = dist(airlinesNorm, method = "euclidean")
HierCluster = hclust(distance, method = "ward.D")
plot(HierCluster)
```



Looking at the denddogram, we can decide that total number of clusters is from 2 to 7. In the next analysis we'll use k=5 clusters. We can subset the data for each of the cluster as follows:

```
clusterGroups =cutree(HierCluster, k = 5)
HierCluster1 = subset(airlinesNorm, clusterGroups == 1)
HierCluster2 = subset(airlinesNorm, clusterGroups == 2)
HierCluster3 = subset(airlinesNorm, clusterGroups == 3)
HierCluster4 = subset(airlinesNorm, clusterGroups == 4)
HierCluster5 = subset(airlinesNorm, clusterGroups == 5)
```

We can use `lapply` to compare the average values in each of the variables for the 5 clusters (the centroids of the clusters):

```
colMeans(subset(airlines, clusterGroups == 1))
```

```
##        Balance       QualMiles      BonusMiles      BonusTrans
##   5.786690e+04    6.443299e-01    1.036012e+04    1.082345e+01
##    FlightMiles     FlightTrans DaysSinceEnroll
##   8.318428e+01    3.028351e-01    6.235365e+03
```

```
colMeans(subset(airlines, clusterGroups == 2))
```

```
##        Balance       QualMiles      BonusMiles      BonusTrans
##   1.106693e+05    1.065983e+03    2.288176e+04    1.822929e+01
##    FlightMiles     FlightTrans DaysSinceEnroll
##   2.613418e+03    7.402697e+00    4.402414e+03
```

```r
colMeans(subset(airlines, clusterGroups == 3))
```

```
##         Balance       QualMiles      BonusMiles      BonusTrans
##    1.981916e+05    3.034615e+01    5.579586e+04    1.966397e+01
##     FlightMiles     FlightTrans DaysSinceEnroll
##    3.276761e+02    1.068826e+00    5.615709e+03
```

```r
colMeans(subset(airlines, clusterGroups == 4))
```

```
##         Balance       QualMiles      BonusMiles      BonusTrans
##    52335.913594        4.847926    20788.766129       17.087558
##     FlightMiles     FlightTrans DaysSinceEnroll
##      111.573733        0.344470     2840.822581
```

```r
colMeans(subset(airlines, clusterGroups == 5))
```

```
##         Balance       QualMiles      BonusMiles      BonusTrans
##    3.625591e+04    2.511177e+00    2.264788e+03    2.973174e+00
##     FlightMiles     FlightTrans DaysSinceEnroll
##    1.193219e+02    4.388972e-01    3.060081e+03
```

```r
lapply(split(airlines, clusterGroups), colMeans)
```

```
## $`1`
##         Balance       QualMiles      BonusMiles      BonusTrans
##    5.786690e+04    6.443299e-01    1.036012e+04    1.082345e+01
##     FlightMiles     FlightTrans DaysSinceEnroll
##    8.318428e+01    3.028351e-01    6.235365e+03
##
## $`2`
##         Balance       QualMiles      BonusMiles      BonusTrans
##    1.106693e+05    1.065983e+03    2.288176e+04    1.822929e+01
##     FlightMiles     FlightTrans DaysSinceEnroll
##    2.613418e+03    7.402697e+00    4.402414e+03
##
## $`3`
##         Balance       QualMiles      BonusMiles      BonusTrans
##    1.981916e+05    3.034615e+01    5.579586e+04    1.966397e+01
##     FlightMiles     FlightTrans DaysSinceEnroll
##    3.276761e+02    1.068826e+00    5.615709e+03
##
## $`4`
##         Balance       QualMiles      BonusMiles      BonusTrans
##    52335.913594        4.847926    20788.766129       17.087558
##     FlightMiles     FlightTrans DaysSinceEnroll
##      111.573733        0.344470     2840.822581
##
## $`5`
##         Balance       QualMiles      BonusMiles      BonusTrans
##    3.625591e+04    2.511177e+00    2.264788e+03    2.973174e+00
##     FlightMiles     FlightTrans DaysSinceEnroll
##    1.193219e+02    4.388972e-01    3.060081e+03
```

We also want to analyze the data using K-means algorithm as follows:

```
k = 5 #five clusters
set.seed(88)
KMC = kmeans(airlinesNorm, centers = k, iter.max = 1000)
```

The number of observations in each cluster is easily determined:

```
sum(KMC$cluster==1)
```

```
## [1] 408
```

```
sum(KMC$cluster==2)
```

```
## [1] 141
```

```
sum(KMC$cluster==3)
```

```
## [1] 993
```

```
sum(KMC$cluster==4)
```

```
## [1] 1182
```

```
sum(KMC$cluster==5)
```

```
## [1] 1275
```

```
# or table(KMC$cluster)
```

We can compare the cluster centroids to each other either by dividing the data points into groups and then using tapply, or by looking at the output of kmeansClust$centers, where "kmeansClust" is the name of the output of the kmeans function.

```
colMeans(subset(airlines, KMC$cluster == 1))
```

```
##          Balance       QualMiles      BonusMiles      BonusTrans
##     2.191614e+05    5.395784e+02    6.247448e+04    2.152451e+01
##      FlightMiles     FlightTrans DaysSinceEnroll
##     6.238725e+02    1.921569e+00    5.605051e+03
```

```
colMeans(subset(airlines, KMC$cluster == 2))
```

```
##          Balance       QualMiles      BonusMiles      BonusTrans
##     174431.51064       673.16312     31985.08511        28.13475
##      FlightMiles     FlightTrans DaysSinceEnroll
##       5859.23404        17.00000      4684.90071
```

```r
colMeans(subset(airlines, KMC$cluster == 3))
```

```
##         Balance       QualMiles      BonusMiles      BonusTrans 
##    6.797744e+04    3.499396e+01    2.449002e+04    1.842900e+01 
##     FlightMiles     FlightTrans DaysSinceEnroll 
##    2.894713e+02    8.851964e-01    3.416783e+03
```

```r
colMeans(subset(airlines, KMC$cluster == 4))
```

```
##         Balance       QualMiles      BonusMiles      BonusTrans 
##    6.016618e+04    5.520812e+01    8.709712e+03    8.362098e+00 
##     FlightMiles     FlightTrans DaysSinceEnroll 
##    2.032589e+02    6.294416e-01    6.109540e+03
```

```r
colMeans(subset(airlines, KMC$cluster == 5))
```

```
##         Balance       QualMiles      BonusMiles      BonusTrans 
##    3.270667e+04    1.264667e+02    3.097478e+03    4.284706e+00 
##     FlightMiles     FlightTrans DaysSinceEnroll 
##    1.814698e+02    5.403922e-01    2.281055e+03
```

```r
lapply(split(airlines, KMC$cluster), colMeans)
```

```
## $`1`
##         Balance       QualMiles      BonusMiles      BonusTrans 
##    2.191614e+05    5.395784e+02    6.247448e+04    2.152451e+01 
##     FlightMiles     FlightTrans DaysSinceEnroll 
##    6.238725e+02    1.921569e+00    5.605051e+03 
## 
## $`2`
##         Balance       QualMiles      BonusMiles      BonusTrans 
##    174431.51064       673.16312     31985.08511        28.13475 
##     FlightMiles     FlightTrans DaysSinceEnroll 
##      5859.23404        17.00000      4684.90071 
## 
## $`3`
##         Balance       QualMiles      BonusMiles      BonusTrans 
##    6.797744e+04    3.499396e+01    2.449002e+04    1.842900e+01 
##     FlightMiles     FlightTrans DaysSinceEnroll 
##    2.894713e+02    8.851964e-01    3.416783e+03 
## 
## $`4`
##         Balance       QualMiles      BonusMiles      BonusTrans 
##    6.016618e+04    5.520812e+01    8.709712e+03    8.362098e+00 
##     FlightMiles     FlightTrans DaysSinceEnroll 
##    2.032589e+02    6.294416e-01    6.109540e+03 
## 
## $`5`
##         Balance       QualMiles      BonusMiles      BonusTrans 
##    3.270667e+04    1.264667e+02    3.097478e+03    4.284706e+00 
##     FlightMiles     FlightTrans DaysSinceEnroll 
##    1.814698e+02    5.403922e-01    2.281055e+03
```

The clusters are not displayed in a meaningful order, so while there may be a cluster produced by the k-means algorithm that is similar to Cluster 1 produced by the Hierarchical method, it will not necessarily be shown first.

Problem 3: PREDICTING STOCK RETURNS WITH CLUSTER-THEN-PREDICT

We'll use cluster-then-predict to predict future stock prices using historical stock data.

When selecting which stocks to invest in, investors seek to obtain good future returns. In this problem, we will first use clustering to identify clusters of stocks that have similar returns over time. Then, we'll use logistic regression to predict whether or not the stocks will have positive future returns.

For this problem, we'll use StocksCluster.csv, which contains monthly stock returns from the NASDAQ stock exchange. The NASDAQ is the second-largest stock exchange in the world, and it lists many technology companies. The stock price data used in this problem was obtained from infochimps http://www.infochimps.com/, a website providing access to many datasets.

Each observation in the dataset is the monthly returns of a particular company in a particular year. The years included are 2000-2009. The companies are limited to tickers that were listed on the exchange for the entire period 2000-2009, and whose stock price never fell below $1. So, for example, one observation is for Yahoo in 2000, and another observation is for Yahoo in 2001. Our goal will be to predict whether or not the stock return in December will be positive, using the stock returns for the first 11 months of the year.

This dataset contains the following variables:

- **ReturnJan** = the return for the company's stock during January (in the year of the observation).
- **ReturnFeb** = the return for the company's stock during February (in the year of the observation).
- **ReturnMar** = the return for the company's stock during March (in the year of the observation).
- **ReturnApr** = the return for the company's stock during April (in the year of the observation).
- **ReturnMay** = the return for the company's stock during May (in the year of the observation).
- **ReturnJune** = the return for the company's stock during June (in the year of the observation).
- **ReturnJuly** = the return for the company's stock during July (in the year of the observation).
- **ReturnAug** = the return for the company's stock during August (in the year of the observation).
- **ReturnSep** = the return for the company's stock during September (in the year of the observation).
- **ReturnOct** = the return for the company's stock during October (in the year of the observation).
- **ReturnNov** = the return for the company's stock during November (in the year of the observation).
- **PositiveDec** = whether or not the company's stock had a positive return in December (in the year of the observation). This variable takes value 1 if the return was positive, and value 0 if the return was not positive.

For the first 11 variables, the value stored is a proportional change in stock value during that month. For instance, a value of 0.05 means the stock increased in value 5% during the month, while a value of -0.02 means the stock decreased in value 2% during the month.

Let us first obtain the structure f the datase:

```
stocks = read.csv("StocksCluster.csv")
str(stocks)
```

```
## 'data.frame':    11580 obs. of  12 variables:
##  $ ReturnJan  : num  0.0807 -0.0107 0.0477 -0.074 -0.031 ...
##  $ ReturnFeb  : num  0.0663 0.1021 0.036 -0.0482 -0.2127 ...
##  $ ReturnMar  : num  0.0329 0.1455 0.0397 0.0182 0.0915 ...
```

```
##  $ ReturnApr  : num  0.1831 -0.0844 -0.1624 -0.0247 0.1893 ...
##  $ ReturnMay  : num  0.13033 -0.3273 -0.14743 -0.00604 -0.15385 ...
##  $ ReturnJune : num  -0.0176 -0.3593 0.0486 -0.0253 -0.1061 ...
##  $ ReturnJuly : num  -0.0205 -0.0253 -0.1354 -0.094 0.3553 ...
##  $ ReturnAug  : num  0.0247 0.2113 0.0334 0.0953 0.0568 ...
##  $ ReturnSep  : num  -0.0204 -0.58 0 0.0567 0.0336 ...
##  $ ReturnOct  : num  -0.1733 -0.2671 0.0917 -0.0963 0.0363 ...
##  $ ReturnNov  : num  -0.0254 -0.1512 -0.0596 -0.0405 -0.0853 ...
##  $ PositiveDec: int  0 0 0 1 1 1 1 0 0 0 ...
```

Proportion of observation which is positive can be determined using `table`:

```r
table(stocks$PositiveDec)
```

```
##
##    0    1
## 5256 6324
```

```r
6324/(6324+5256)
```

```
## [1] 0.546114
```

The correlation between each component in dataset is:

```r
cor(stocks)
```

```
##                ReturnJan   ReturnFeb    ReturnMar    ReturnApr
## ReturnJan    1.000000000  0.06677458 -0.090496798 -0.037678006
## ReturnFeb    0.066774583  1.00000000 -0.155983263 -0.191351924
## ReturnMar   -0.090496798 -0.15598326  1.000000000  0.009726288
## ReturnApr   -0.037678006 -0.19135192  0.009726288  1.000000000
## ReturnMay   -0.044411417 -0.09552092 -0.003892789  0.063822504
## ReturnJune   0.092238307  0.16999448 -0.085905486 -0.011027752
## ReturnJuly  -0.081429765 -0.06177851  0.003374160  0.080631932
## ReturnAug   -0.022792019  0.13155979 -0.022005400 -0.051756051
## ReturnSep   -0.026437153  0.04350177  0.076518327 -0.028920972
## ReturnOct    0.142977229 -0.08732427 -0.011923758  0.048540025
## ReturnNov    0.067632333 -0.15465828  0.037323535  0.031761837
## PositiveDec  0.004728518 -0.03817318  0.022408661  0.094353528
##                ReturnMay  ReturnJune    ReturnJuly     ReturnAug
## ReturnJan   -0.044411417  0.09223831 -0.0814297650 -0.0227920187
## ReturnFeb   -0.095520920  0.16999448 -0.0617785094  0.1315597863
## ReturnMar   -0.003892789 -0.08590549  0.0033741597 -0.0220053995
## ReturnApr    0.063822504 -0.01102775  0.0806319317 -0.0517560510
## ReturnMay    1.000000000 -0.02107454  0.0908502642 -0.0331256580
## ReturnJune  -0.021074539  1.00000000 -0.0291525996  0.0107105260
## ReturnJuly   0.090850264 -0.02915260  1.0000000000  0.0007137558
## ReturnAug   -0.033125658  0.01071053  0.0007137558  1.0000000000
## ReturnSep    0.021962862  0.04474727  0.0689478037  0.0007407139
## ReturnOct    0.017166728 -0.02263599 -0.0547089088 -0.0755945614
## ReturnNov    0.048046590 -0.06527054 -0.0483738369 -0.1164890345
## PositiveDec  0.058201934  0.02340975  0.0743642097  0.0041669657
```

```
##                ReturnSep   ReturnOct   ReturnNov  PositiveDec
## ReturnJan   -0.0264371526  0.14297723  0.06763233  0.004728518
## ReturnFeb    0.0435017706 -0.08732427 -0.15465828 -0.038173184
## ReturnMar    0.0765183267 -0.01192376  0.03732353  0.022408661
## ReturnApr   -0.0289209718  0.04854003  0.03176184  0.094353528
## ReturnMay    0.0219628623  0.01716673  0.04804659  0.058201934
## ReturnJune   0.0447472692 -0.02263599 -0.06527054  0.023409745
## ReturnJuly   0.0689478037 -0.05470891 -0.04837384  0.074364210
## ReturnAug    0.0007407139 -0.07559456 -0.11648903  0.004166966
## ReturnSep    1.0000000000 -0.05807924 -0.01971980  0.041630286
## ReturnOct   -0.0580792362  1.00000000  0.19167279 -0.052574956
## ReturnNov   -0.0197197998  0.19167279  1.00000000 -0.062346556
## PositiveDec  0.0416302863 -0.05257496 -0.06234656  1.000000000
```

The maximum correlatin is detected between `ReturnOct` and `ReturnNov`, and is equal to ~ 0.192. Maximum and minumum of returns per month is calculated next:

```r
summary(stocks)
```

```
##    ReturnJan            ReturnFeb           ReturnMar
##  Min.   :-0.7616205   Min.   :-0.690000   Min.   :-0.712994
##  1st Qu.:-0.0691663   1st Qu.:-0.077748   1st Qu.:-0.046389
##  Median : 0.0009965   Median :-0.010626   Median : 0.009878
##  Mean   : 0.0126316   Mean   :-0.007605   Mean   : 0.019402
##  3rd Qu.: 0.0732606   3rd Qu.: 0.043600   3rd Qu.: 0.077066
##  Max.   : 3.0683060   Max.   : 6.943694   Max.   : 4.008621
##    ReturnApr           ReturnMay          ReturnJune
##  Min.   :-0.826503   Min.   :-0.92207   Min.   :-0.717920
##  1st Qu.:-0.054468   1st Qu.:-0.04640   1st Qu.:-0.063966
##  Median : 0.009059   Median : 0.01293   Median :-0.000880
##  Mean   : 0.026308   Mean   : 0.02474   Mean   : 0.005938
##  3rd Qu.: 0.085338   3rd Qu.: 0.08396   3rd Qu.: 0.061566
##  Max.   : 2.528827   Max.   : 6.93013   Max.   : 4.339713
##    ReturnJuly           ReturnAug           ReturnSep
##  Min.   :-0.7613096   Min.   :-0.726800   Min.   :-0.839730
##  1st Qu.:-0.0731917   1st Qu.:-0.046272   1st Qu.:-0.074648
##  Median :-0.0008047   Median : 0.007205   Median :-0.007616
##  Mean   : 0.0030509   Mean   : 0.016198   Mean   :-0.014721
##  3rd Qu.: 0.0718205   3rd Qu.: 0.070783   3rd Qu.: 0.049476
##  Max.   : 2.5500000   Max.   : 3.626609   Max.   : 5.863980
##    ReturnOct           ReturnNov           PositiveDec
##  Min.   :-0.685504   Min.   :-0.747171   Min.   :0.0000
##  1st Qu.:-0.070915   1st Qu.:-0.054890   1st Qu.:0.0000
##  Median : 0.002115   Median : 0.008522   Median :1.0000
##  Mean   : 0.005651   Mean   : 0.011387   Mean   :0.5461
##  3rd Qu.: 0.074542   3rd Qu.: 0.076576   3rd Qu.:1.0000
##  Max.   : 5.665138   Max.   : 3.271676   Max.   :1.0000
```

After short intial exploration of the dataset, we need to devide it into raining and testing set:

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

```
## Warning: package 'caTools' was built under R version 3.2.1
```

```
set.seed(144)
spl = sample.split(stocks$PositiveDec, SplitRatio = 0.7)
stocksTrain = subset(stocks, spl == TRUE)
stocksTest = subset(stocks, spl == FALSE)
```

Next, let us train simple logistic regression model for predictive positive returns at the end of the year:

```
StocksModel = glm(PositiveDec~., family="binomial", data = stocksTrain)
```

The accuracy of our initial model can be obtained from confusion matrix:

```
pred = predict(StocksModel, type="response")
table(stocksTrain$PositiveDec, pred>0.5)
```

```
##
##     FALSE TRUE
##   0   990 2689
##   1   787 3640
```

```
ACC = (990+3640)/nrow(stocksTrain)
ACC
```

```
## [1] 0.5711818
```

The result is not very gooy - accuracy of 0.57 won't cut it. The test predictions of the same model should be even lower:

```
pred = predict(StocksModel, type="response", newdata = stocksTest)
table(stocksTest$PositiveDec, pred>0.5)
```

```
##
##     FALSE TRUE
##   0   417 1160
##   1   344 1553
```

```
ACC = (417+1553)/nrow(stocksTest)
ACC
```

```
## [1] 0.5670697
```

Comparing to baseline model (model that always predicts the most common outcome), this accuracy is really not much of an improvement:

```
table(stocksTest$PositiveDec)
```

```
##
##    0    1
## 1577 1897
```

```r
ACC = 1897/(1577+1897)
ACC
```

```
## [1] 0.5460564
```

In order to improve our model, we'll cluster the stocks. The first step in this process is to remove the dependent variable using the following commands:

```r
limitedTrain = stocksTrain
limitedTrain$PositiveDec = NULL
limitedTest = stocksTest
limitedTest$PositiveDec = NULL
```

In cases where we have a training and testing set, we'll want to normalize by the mean and standard deviation of the variables in the training set. We can do this by passing just the training set to the preProcess function:

```r
#intall.packages("caret")
library(caret)

preproc = preProcess(limitedTrain)
normTrain = predict(preproc, limitedTrain)
normTest = predict(preproc, limitedTest)
```

Now we can run K-Means algorithm:

```r
k = 3
set.seed(144)
km = kmeans(normTrain, centers = k)
table(km$cluster)
```

```
##
##    1    2    3
## 3157 4696  253
```

We can now use the `flexclust` package to obtain training set and testing set cluster assignments for our observations

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

```
## Warning: package 'flexclust' was built under R version 3.2.1

## Loading required package: grid
## Loading required package: modeltools

## Warning: package 'modeltools' was built under R version 3.2.1

## Loading required package: stats4
```

```r
km.kcca = as.kcca(km, normTrain)
clusterTrain = predict(km.kcca)
clusterTest = predict(km.kcca, newdata=normTest)
```

Similarly to previous studies, we'll devide the dataset according to obtained clusters:

```r
stockTrain1 = subset(stocksTrain, clusterTrain == 1)
stockTrain2 = subset(stocksTrain, clusterTrain == 2)
stockTrain3 = subset(stocksTrain, clusterTrain == 3)

stocksTest1 = subset(stocksTest, clusterTest == 1)
stocksTest2 = subset(stocksTest, clusterTest == 2)
stocksTest3 = subset(stocksTest, clusterTest == 3)
```

From `mean(stocksTrain1$PositiveDec)`, `mean(stocksTrain2$PositiveDec)`, and `mean(stocksTrain3$PositiveDec)`, we see that stocksTrain1 has the observations with the highest average value of the dependent variable. Let us build logistic models for which predict PositiveDec using all the other variables as independent variables that use developed subset of the training set.

```r
StocksModel1 = glm(PositiveDec~., family="binomial", data = stockTrain1)
StocksModel2 = glm(PositiveDec~., family="binomial", data = stockTrain2)
StocksModel3 = glm(PositiveDec~., family="binomial", data = stockTrain3)

summary(StocksModel1)
```

```
##
## Call:
## glm(formula = PositiveDec ~ ., family = "binomial", data = stockTrain1)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7307  -1.2910   0.8878   1.0280   1.5023
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.17224    0.06302   2.733  0.00628 **
## ReturnJan    0.02498    0.29306   0.085  0.93206
## ReturnFeb   -0.37207    0.29123  -1.278  0.20139
## ReturnMar    0.59555    0.23325   2.553  0.01067 *
## ReturnApr    1.19048    0.22439   5.305 1.12e-07 ***
## ReturnMay    0.30421    0.22845   1.332  0.18298
## ReturnJune  -0.01165    0.29993  -0.039  0.96901
## ReturnJuly   0.19769    0.27790   0.711  0.47685
## ReturnAug    0.51273    0.30858   1.662  0.09660 .
## ReturnSep    0.58833    0.28133   2.091  0.03651 *
## ReturnOct   -1.02254    0.26007  -3.932 8.43e-05 ***
## ReturnNov   -0.74847    0.28280  -2.647  0.00813 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 4243.0  on 3156  degrees of freedom
## Residual deviance: 4172.9  on 3145  degrees of freedom
## AIC: 4196.9
##
## Number of Fisher Scoring iterations: 4
```

```r
summary(StocksModel2)
```

```
##
## Call:
## glm(formula = PositiveDec ~ ., family = "binomial", data = stockTrain2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2012  -1.1941   0.8583   1.1334   1.9424
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.10293    0.03785   2.719 0.006540 **
## ReturnJan    0.88451    0.20276   4.362 1.29e-05 ***
## ReturnFeb    0.31762    0.26624   1.193 0.232878
## ReturnMar   -0.37978    0.24045  -1.579 0.114231
## ReturnApr    0.49291    0.22460   2.195 0.028189 *
## ReturnMay    0.89655    0.25492   3.517 0.000436 ***
## ReturnJune   1.50088    0.26014   5.770 7.95e-09 ***
## ReturnJuly   0.78315    0.26864   2.915 0.003554 **
## ReturnAug   -0.24486    0.27080  -0.904 0.365876
## ReturnSep    0.73685    0.24820   2.969 0.002989 **
## ReturnOct   -0.27756    0.18400  -1.509 0.131419
## ReturnNov   -0.78747    0.22458  -3.506 0.000454 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6506.3  on 4695  degrees of freedom
## Residual deviance: 6362.2  on 4684  degrees of freedom
## AIC: 6386.2
##
## Number of Fisher Scoring iterations: 4
```

```r
summary(StocksModel3)
```

```
##
## Call:
## glm(formula = PositiveDec ~ ., family = "binomial", data = stockTrain3)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9146  -1.0393  -0.7689   1.1921   1.6939
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept) -0.181896   0.325182  -0.559   0.5759
## ReturnJan   -0.009789   0.448943  -0.022   0.9826
## ReturnFeb   -0.046883   0.213432  -0.220   0.8261
## ReturnMar    0.674179   0.564790   1.194   0.2326
## ReturnApr    1.281466   0.602672   2.126   0.0335 *
## ReturnMay    0.762512   0.647783   1.177   0.2392
## ReturnJune   0.329434   0.408038   0.807   0.4195
## ReturnJuly   0.774164   0.729360   1.061   0.2885
## ReturnAug    0.982605   0.533158   1.843   0.0653 .
## ReturnSep    0.363807   0.627774   0.580   0.5622
## ReturnOct    0.782242   0.733123   1.067   0.2860
## ReturnNov   -0.873752   0.738480  -1.183   0.2367
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 346.92  on 252  degrees of freedom
## Residual deviance: 328.29  on 241  degrees of freedom
## AIC: 352.29
##
## Number of Fisher Scoring iterations: 4
```

We can now build test prediction on these models:

```
pred1 = predict(StocksModel1, type="response", newdata = stocksTest1)
table(stocksTest1$PositiveDec, pred1>0.5)
```

```
##
##     FALSE TRUE
##   0    30  471
##   1    23  774
```

```
ACC1 = (30+774)/nrow(stocksTest1)
ACC1
```

```
## [1] 0.6194145
```

```
#############
pred2 = predict(StocksModel2, type="response", newdata = stocksTest2)
table(stocksTest2$PositiveDec, pred2>0.5)
```

```
##
##     FALSE TRUE
##   0   388  626
##   1   309  757
```

```
ACC2 = (388+757)/nrow(stocksTest2)
ACC2
```

```
## [1] 0.5504808
```

```
#############
pred3 = predict(StocksModel3, type="response", newdata = stocksTest3)
table(stocksTest3$PositiveDec, pred3>0.5)
```

```
##
##     FALSE TRUE
##   0    49   13
##   1    21   13
```

```
ACC3 = (49+13)/nrow(stocksTest3)
ACC3
```

```
## [1] 0.6458333
```

Finally, to compute the overall test-set accuracy of the cluster-then-predict approach, we can combine all the test-set predictions into a single vector and all the true outcomes into a single vector:

```
AllPredictions = c(pred1, pred2, pred3)
AllOutcomes = c(stocksTest1$PositiveDec, stocksTest2$PositiveDec, stocksTest3$PositiveDec)
table(AllOutcomes, AllPredictions>0.5)
```

```
##
## AllOutcomes FALSE TRUE
##           0   467 1110
##           1   353 1544
```

```
ACC_Total = (467+1544)/length(AllOutcomes)
ACC_Total
```

```
## [1] 0.5788716
```

We see a modest improvement over the original logistic regression model. Since predicting stock returns is a notoriously hard problem, this is a good increase in accuracy. By investing in stocks for which we are more confident that they will have positive returns (by selecting the ones with higher predicted probabilities), this cluster-then-predict model can give us an edge over the original logistic regression model.