Data Mining Project 2

Zheqi Wang - 47711564

Ziling Feng - 47781816

1. Data Processing

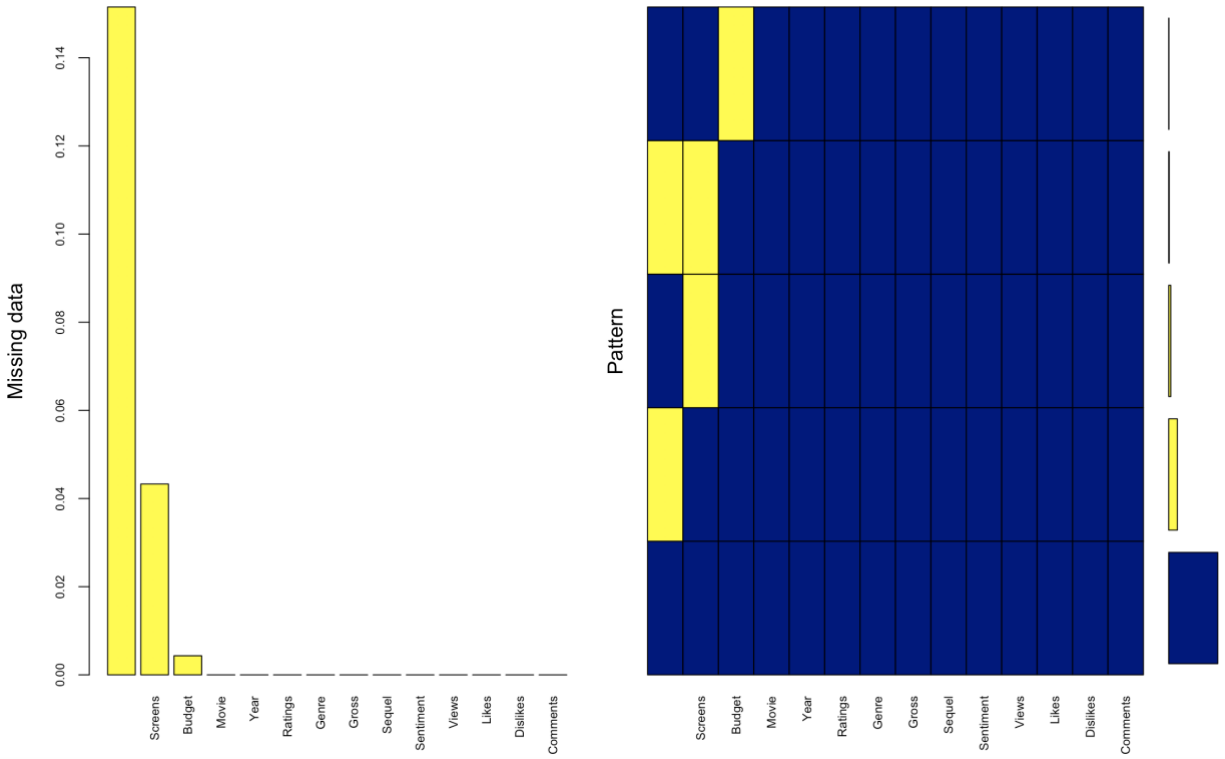
We describe the data in the dataset and summarized the various variables and types. As shown below.

FEATURE NAME	EXPLAINATION	TYPE
Movie	Name of movie	Factor
Year	Year of the movie. Value range: 2014 - 2015	Integer
Ratings	Rating of movie. Value range: 3.1 - 8.7	Numeric
Genre	Value range: 1 - 15	Integer
Gross	Value range: 2470 - 643000000	Integer
Budget	Value range: 70000 – 250000000. **1 NA**	Numeric
Screens	Value range: 2 – 4324. **10 NA**	Integer
Sequel	Value range: 1 - 7	Integer
Sentiment	Value range: -38 - 29	Integer
Views	Value range: 698 - 32626778	Integer
Likes	Value range: 1 - 370552	Integer
Dislikes	Value range: 0 - 13960	Integer
Comments	Value range: 0 - 38363	Integer
Aggregate.Followers	Value range: 1066 – 31030000. **35 NA**	Integer

1.1 Missing values

We count the missing values of each attributes. We could find that 'Budget' has 1 NA value, 'Screens' has 10 NA values, 'Aggregate Followers' has 35 NA values.

We use package “VIM” to visualize the missing value, and plot the missing values into a figure.



The left figure shows the proportion of missing values in each attribute. And right figure is the distribution of missing value.

We decide to fill the missing value with mean value of each attribute.

```
> summary(df)
                    Movie          Year         Ratings          Genre           Gross
 13 Sins                 :  1   Min.   :2014   Min.   :3.100   Min.   : 1.000   Min.   :     2470
 22 Jump Street          :  1   1st Qu.:2014   1st Qu.:5.800   1st Qu.: 1.000   1st Qu.: 10300000
 3 Days to Kill          :  1   Median :2014   Median :6.500   Median : 3.000   Median : 37400000
 300: Rise of an Empire  :  1   Mean   :2014   Mean   :6.442   Mean   : 5.359   Mean   : 68066033
 A Haunted House 2       :  1   3rd Qu.:2015   3rd Qu.:7.100   3rd Qu.: 8.000   3rd Qu.: 89350000
 A Long Way Off          :  1   Max.   :2015   Max.   :8.700   Max.   :15.000   Max.   :643000000
 (Other)                 :225
     Budget             Screens          Sequel        Sentiment          Views              Likes
 Min.   :    70000   Min.   :   2.0   Min.   :1.000   Min.   :-38.00   Min.   :     698   Min.   :     1
 1st Qu.:  9000000   1st Qu.: 563.5   1st Qu.:1.000   1st Qu.:  0.00   1st Qu.:  623302   1st Qu.:  1776
 Median : 28000000   Median :2757.0   Median :1.000   Median :  0.00   Median : 2409338   Median :  6096
 Mean   : 47921730   Mean   :2209.2   Mean   :1.359   Mean   :  2.81   Mean   : 3712851   Mean   : 12732
 3rd Qu.: 65000000   3rd Qu.:3334.0   3rd Qu.:1.000   3rd Qu.:  5.50   3rd Qu.: 5217380   3rd Qu.: 15248
 Max.   :250000000   Max.   :4324.0   Max.   :7.000   Max.   : 29.00   Max.   :32626778   Max.   :370552

    Dislikes          Comments       Aggregate.Followers
 Min.   :    0.0   Min.   :    0.0   Min.   :    1066
 1st Qu.:  105.5   1st Qu.:  248.5   1st Qu.:  250000
 Median :  341.0   Median :  837.0   Median : 1800000
 Mean   :  679.1   Mean   : 1825.7   Mean   : 3038193
 3rd Qu.:  697.5   3rd Qu.: 2137.0   3rd Qu.: 3038193
 Max.   :13960.0   Max.   :38363.0   Max.   :31030000
```

1.2 Duplicate value

We check there is no duplicated value.

```
> summary(duplicated(df))
   Mode    FALSE
logical      231
```

1.3 Standardization

According to previous analysis, we could find some attributes have a large value range. So, we decide to standardization the dataset.

Standardization is essentially a linear transformation. Linear transformation has many good properties. It determines that the change of data will not cause failure, but can improve the performance of data. It has two additional benefits: a mean of 0 and a standard deviation of 1. It allows data to be distributed around 0.

```
> df_stand <- scale(df[ ,-1])
> summary(df_stand)
      Year             Ratings             Genre              Gross              Budget        
 Min.   :-0.6445   Min.   :-3.37953   Min.   :-1.0526   Min.   :-0.7656   Min.   :-0.8834  
 1st Qu.:-0.6445   1st Qu.:-0.64885   1st Qu.:-1.0526   1st Qu.:-0.6498   1st Qu.:-0.7185  
 Median :-0.6445   Median : 0.05911   Median :-0.5697   Median :-0.3449   Median :-0.3678  
 Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000  
 3rd Qu.: 1.5449   3rd Qu.: 0.66592   3rd Qu.: 0.6376   3rd Qu.: 0.2394   3rd Qu.: 0.3153  
 Max.   : 1.5449   Max.   : 2.28410   Max.   : 2.3278   Max.   : 6.4670   Max.   : 3.7304  
    Screens            Sequel           Sentiment            Views              Likes         
 Min.   :-1.5418   Min.   :-0.3715   Min.   :-5.8326   Min.   :-0.8229   Min.   :-0.44168  
 1st Qu.:-1.1496   1st Qu.:-0.3715   1st Qu.:-0.4015   1st Qu.:-0.6849   1st Qu.:-0.38008  
 Median : 0.3826   Median :-0.3715   Median :-0.4015   Median :-0.2890   Median :-0.23023  
 Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000  
 3rd Qu.: 0.7857   3rd Qu.:-0.3715   3rd Qu.: 0.3845   3rd Qu.: 0.3335   3rd Qu.: 0.08725  
 Max.   : 1.4772   Max.   : 5.8317   Max.   : 3.7432   Max.   : 6.4095   Max.   :12.41330  
    Dislikes           Comments        Aggregate.Followers
 Min.   :-0.54589   Min.   :-0.51125   Min.   :-0.6750    
 1st Qu.:-0.46108   1st Qu.:-0.44166   1st Qu.:-0.6197    
 Median :-0.27176   Median :-0.27687   Median :-0.2752    
 Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000    
 3rd Qu.: 0.01483   3rd Qu.: 0.08717   3rd Qu.: 0.0000    
 Max.   :10.67661   Max.   :10.23156   Max.   : 6.2216    
```

2. Clustering Method

In this part, we use three methods--K-means, Gaussian Mixture Model, and hierarchical clustering, to do the cluster. In each method, we need to select the appropriate number of clusters and then do clustering.

2.1 K-means

K-means is an exclusive clustering method based on distance. The steps are as follows:

- Step 1. Given the number of cluster k.
- Step 2. Create an initial partition and randomly select k objects from the dataset, each initially representing a cluster centroid. For other objects, calculate their distance to each centroid and arrange them into the nearest cluster.
- Step 3. An iterative relocation technique is used to improve partitioning by updating centroids. The relocation is to recalculate the average value of the cluster when new objects joining or existing objects leave the cluster. This point with average value is regarded as new centroid. And then, reassign the objects.

2.1.1 Choose number of clusters and clustering

We initialize the number of iterations and initialize an array 'dist'. Take a value of 1 for k and do 10 iterations. After the iteration is finished, calculate the distance between each sample and centroid in each cluster, and sum of total distance of all clusters. Then store in the 'dist' array.

Repeat the above operation for k 2 to 20.

- Withinss is total sum of squares in each cluster, which indicates the square of distance between point and centroid.
- Tot.withinss is the total within-cluster sum of squares, which indicates sum of square of distance between point and centroid.

So, we decide to use these two value to represent distance—which actually is square of distance.

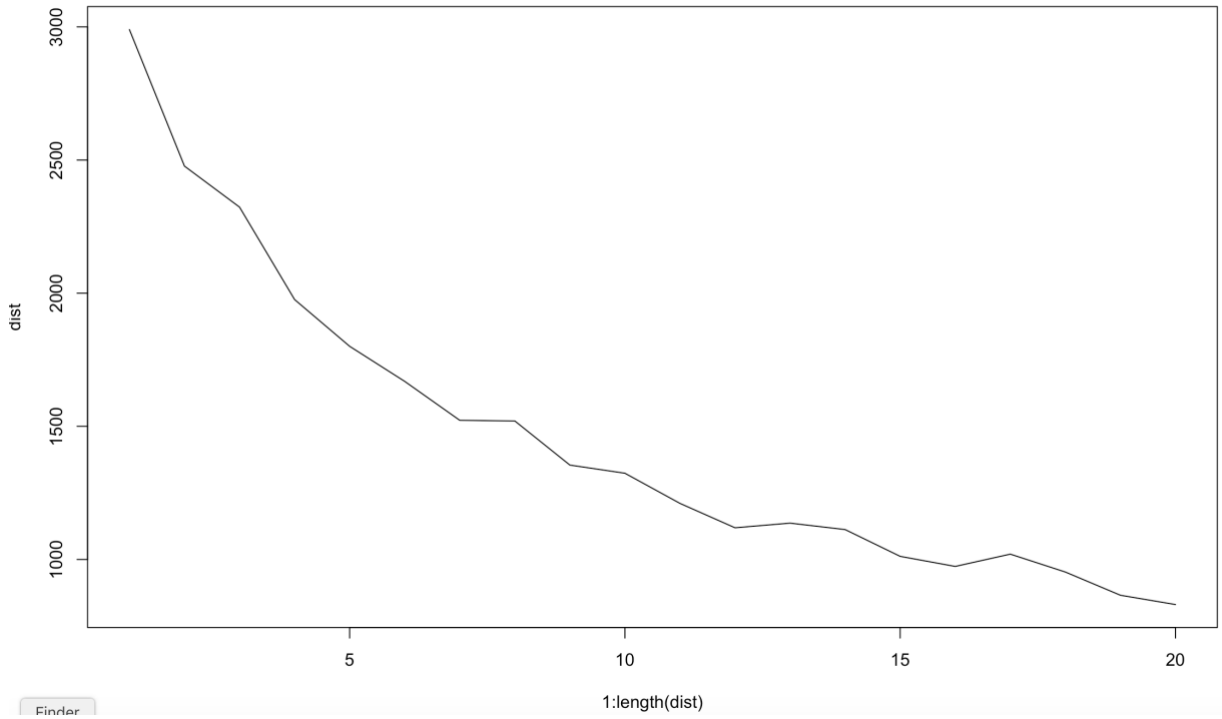
```
## K-means
maxIter = 10

# try k = 1, ..., 20 and check tot.withinss for each model
dist <- NULL
for (k in 1:20){
  m <- kmeans(df_stand, k, maxIter)
  dist[k] <- m$tot.withinss
}
```

We use elbow method to determine k value.

- Elbow Method: This method is suitable for the case where the K value is relatively small. When the selected k value is less than the real value, the cost value will be greatly reduced for each increase of k. When the selected k value is greater than the true K, the change in the cost value will not be so obvious for every increase in k. Thus, the suitable k value will be at this turning point.

Reference: https://www.biaodianfu.com/k-means-choose-k.html

Then, we plot the array ‘dist’ to represent the corresponding change of distance with the change of number of clusters.



From the figure, when x is less than 12, the distance decreases rapidly. When x is greater than 12, the distance also drops, but the rate of decline is not obvious and basically remains stable, so we decided to choose 12.

We do the cluster and show the number of samples in each cluster. As shown below.

```
> m_best <- kmeans(df_stand, 12, maxIter)
> table(m_best$cluster)

 1  2  3  4  5  6  7  8  9 10 11 12
10 28 42 17  1 13 19 22 17  2 30 30
```

2.2 Gaussian Mixture Model

The process of learning is to train several probability distributions. The Gaussian Mixture model refers to the estimation of the probability density distribution of the sample, and the estimated model is the weighted sum of several Gaussian models (specifically, several should be established before the model training). Each Gaussian model represents a class (a Cluster).

Projecting the data in the sample on several Gaussian models separately yields the probability on each class. Then we can select the result of the class with the highest probability.

The function is

$$p(x) = \sum_{k=1}^{K} \pi_k p(x \mid k)$$

K is the number of models, πk is the weight of the kth Gaussian, which is the probability density function of the kth Gaussian, μk is the mean, and σk is the variance.

Estimating the probability density requires variables of πk, μk and σk. When the expression is found, the results of the terms of the summation represent the probability that the sample x belongs to each class.

When making parameter estimates, the maximum likelihood method is often used to maximize the probability value of the sample points on the estimated probability density function. We used the log-likelyhood function. The function shown as follows.

$$\max \sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} \pi_k N(x_i \mid \mu_k, \sigma_k)\right)$$

In order to obtain the extremum of the equation, we could use the EM algorithm. The algorithm could be divided into two steps:

First step is to assume that we know the parameters of each Gaussian model, which can be initialized or based on the results of the previous iteration, to estimate the weight of each Gauss model.

The second step is based on the estimated weight, and then go back to determine the parameters of the Gaussian model. Repeat these two steps until the fluctuations are small and approximate to the extreme value.

Reference: https://blog.csdn.net/jwh_bupt/article/details/7663885
Reference: https://zhuanlan.zhihu.com/p/24546995

2.2.1 Choose number of clusters and clustering

We use package mclust to do clustering. And there is a value called BIC.

The function of BIC as follows:

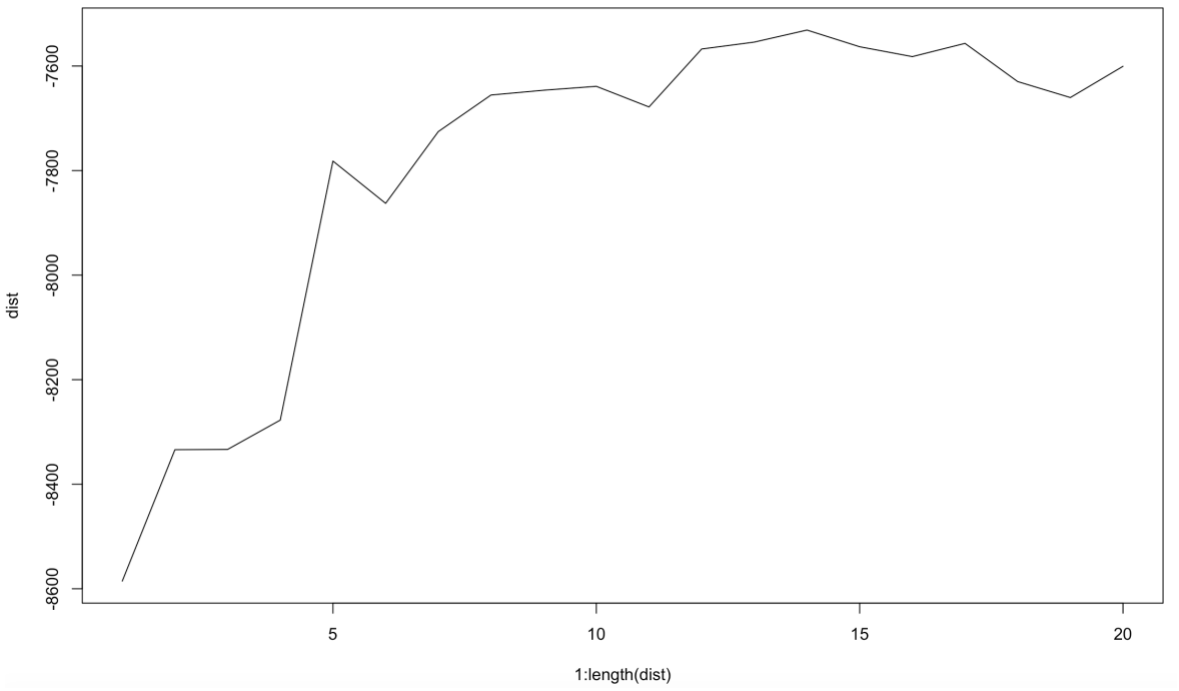
$$2logp(x|M) + constant \approx 2l_M(x, \hat{\theta}) - m_M log(n) \equiv BIC$$

This is the BIC value defined by the author in the Mclust package. It is different from the Bayesian Information Criterion. Here is the BIC defined by the author. It can be seen that the BIC here is directly proportional to the maximum likelihood estimation, so the bigger the BIC value, the better the model.

We initialize the number of iterations and initialize an array 'dist', which is used to store the BIC. Take a value of k for 1 to 20. Do the clustering using package 'Mclust'. Use the 'dist' array to store the BIC values.

```
dist <- NULL
for (k in 1:20) {
  m_gmm <- Mclust(df_stand, k)
  dist[k] <- m_gmm$BIC
}
```

Then, we plot the array 'dist' to represent the corresponding change of BIC value with the change of number of clusters.



We already know the bigger BIC means better model. So, we decide to choose 14 as the number of clusters.

Then, we do the cluster. As shown below.

```
> m_gmm <- Mclust(df_stand, 14)
fitting ...
  |=========================================================================================================| 100%
> summary(m_gmm)
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust EEE (ellipsoidal, equal volume, shape and orientation) model with 14 components:

 log-likelihood   n  df       BIC       ICL
      -2584.32 231 286 -6725.172 -6740.614

Clustering table:
 1  2  3  4  5  6  7  8  9 10 11 12 13 14
34  9 39 16 28 18 10 13  9  8  1 33  1 12
```

2.3 Hierarchical Clustering

The NbClust package provides numerous indices to determine the optimal number of classes in a cluster analysis. There is no guarantee that the results obtained by these indicators are the same, but can be used as a reference for selecting the K value of the cluster number.

The input to the NbClust() function includes a matrix or data frame that needs to be clustered, a distance measure and a clustering method. The number of minimum and maximum clusters would be considered to be the number of clusters. It returns each clustering index and outputs the suggested optimal number of clusters.

Reference: https://www.jianshu.com/p/18dd0ce65bb8

2.3.1 Choose number of clusters and clustering

We install the package and do the NbCluster(). Shown as follows.

```
nc <- NbClust(df_stand, distance = "euclidean",
              min.nc=2, max.nc=15, method="average")
table(nc$Best.n[1,])
```

We use Euclidean distance to represent the distance of each cluster, which is Usual square distance between the two vectors.

The function of Euclidean distance as shown follow:

$$Euclidean\ distance = \sqrt{\sum_i (a_i - b_i)^2}$$

The result shown as follows:

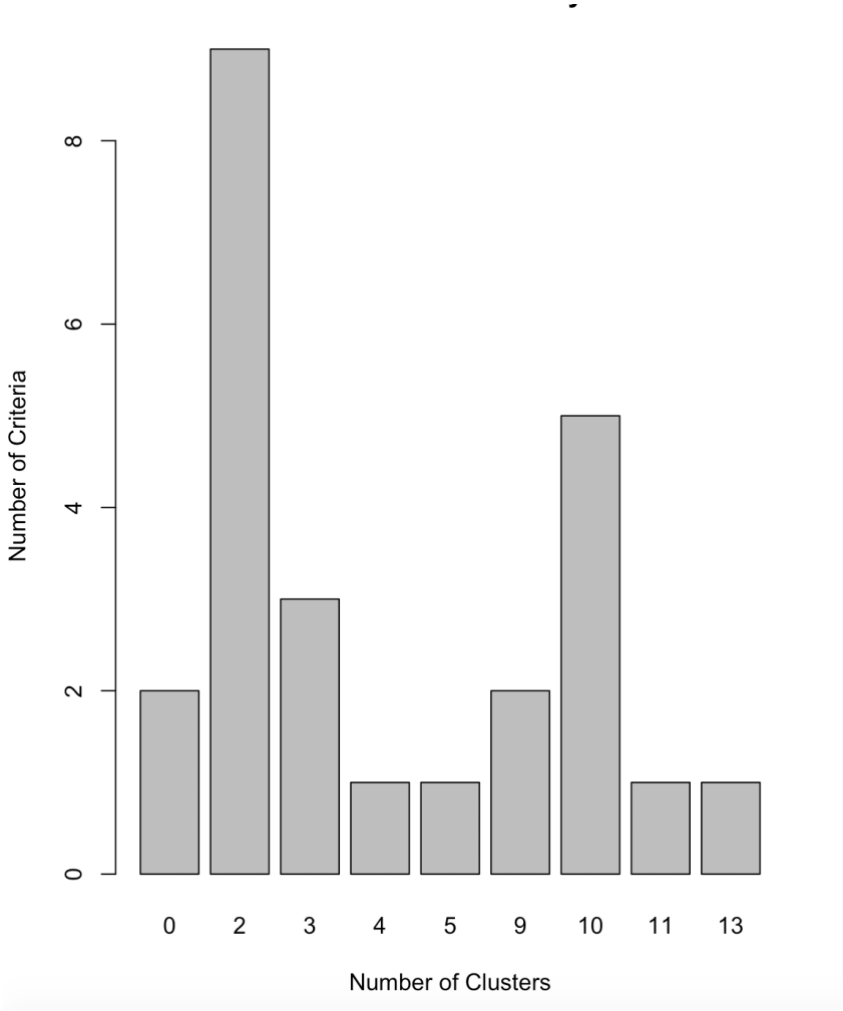
```
*******************************************************************
* Among all indices:
* 9 proposed 2 as the best number of clusters
* 3 proposed 3 as the best number of clusters
* 1 proposed 4 as the best number of clusters
* 1 proposed 5 as the best number of clusters
* 2 proposed 9 as the best number of clusters
* 5 proposed 10 as the best number of clusters
* 1 proposed 11 as the best number of clusters
* 1 proposed 13 as the best number of clusters

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters is  2
```

The return value shows that 2 is the best number of clusters. We use the value 'Best' of the model. And shown in a table. As shown following.

- Best: Best number of clusters proposed by each index and the corresponding index value.

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

 0  2  3  4  5  9 10 11 13
 2  9  3  1  1  2  5  1  1
```

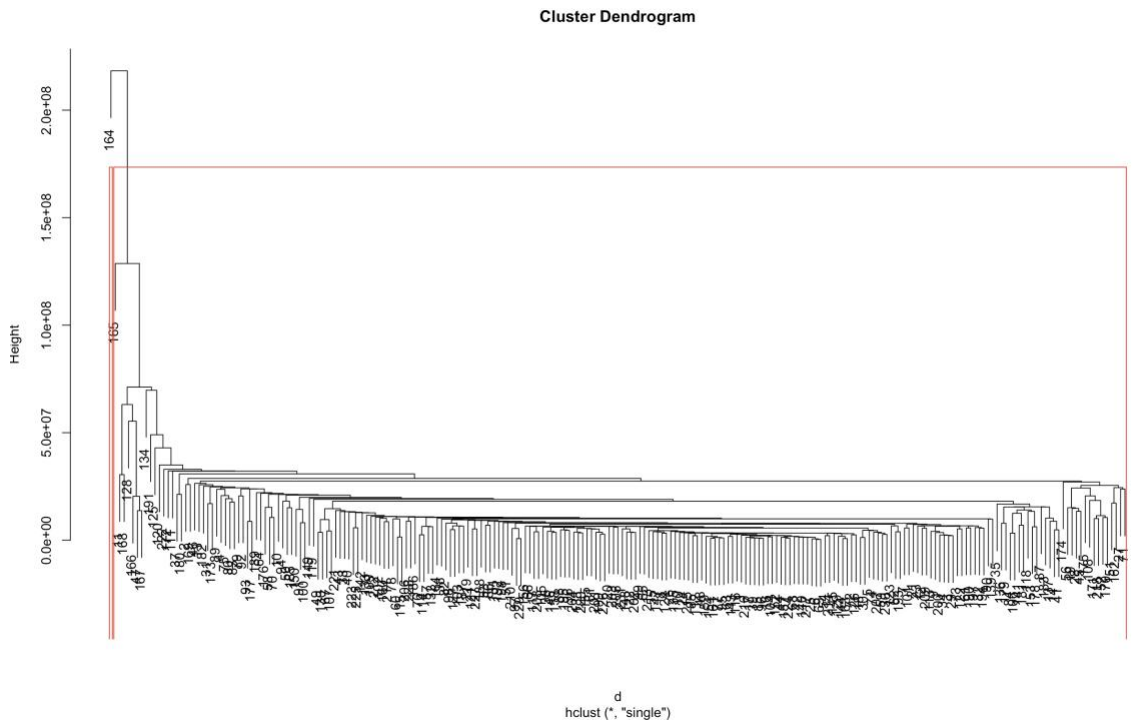
And we also plot the criteria into as a histogram in a picture. As shown below.



So, we choose 2 as the number of clusters. We calculate the distance and do clustering with package 'hclust'.

```
# dissimilarity matrix
d <- dist(df, method = "euclidean")

# clustering using nearest neighbor
hc1 <- hclust(d, method = "single")
plot(hc1)

# draw border of clusters
rect.hclust(hc1, k = 2)
```

Then, we plot the result.



For part 3 of question 2, we already explain the detailed process of these three methods, including mathematical principle, detailed steps and assignment of samples. We also explain the method to choose the number of clusters for each clustering method, including reference measure and mathematical meaning. In addition, we assume the dataset is $X \in R^{N*d}$.

3. K-Means Clustering

3.1 Pseudocode for the K-means Clustering Algorithm

The following is the pseudocode of our own developed k-means function km(). K is the number of clusters. DF is the input data set. There is an initial centroid set called ICS, which default value can be empty. But in our own developed algorithm, we use 2 predefined initial centroids. Iter is the total iteration time which in this case equals 3.

For the algorithm, firstly, we extract centroid points from CIS and store them in c. c[i] means coordinate of each centroid. Then for each iteration, we initialize a column(vector) iter[m] to store each cluster result. M is the number of iterations. After that, for each sample, calculating L2-norm distance of each centroid, choose the closest centroid and assign this sample to this cluster.

After all clustering process, for each cluster, we update our centroids using the following formula:

$$\mu^k = \frac{\sum x^i}{sum(points)}$$

Suppose centroid has coordinate (x, y), x = sum of x in cluster k/ number of points in cluster k; y = sum of y in cluster k/ number of points in cluster k.

PREUDOCODE

```
km(K, df, initial centroid set ICS, iter){
 for(i in K){
  c[i] = ICS(i);
 }
 for( m in iter){
  Add a column iter[m] at the end of df to store the cluster;

  for(each point in the df){
   for(each centroid){
    distance[i] = get distance between point and c[i];
   }
   Get i which distance[i] is minimum;
   //this point belongs to cluster i;
   iter[m] = i;
  }
  for(each centroid c[i]){
   For(each point(x,y) in the cluster i){
    X_sum = X + x;
    Y_sum = Y + y;
   }
   X_centroid= X_sum/number of points;
   Y_centroid= Y_sum/number of points;
   c[i] = (x_centroid, y_centroid);
  }
 }
```

3.2 Implement K-means Clustering Algorithm

3.2.1 Data Processing and Preparation

1) Take the columns ‘Ratings’ and ‘Likes’ from the un-processed dataset, and assign the columns as the input matrix X. Re-scale columns of X such that values in each column range from 0 to 1.

   Firstly, we read original excel data set and take the specific two column into X.

```r
# Take the columns ‘Ratings’ and ‘Likes’ from the un-processed dataset
df_km <- read_excel("movies.xlsx",sheet=1,na="NA")
# Assign the columns as the input matrix X
features <- c("Ratings","Likes")
X <- df_km[features]
```

We find that the range of 'Ratings' and range of 'Likes' are very different. 'Ratings' is 5.6 and 'Likes' is 370551.

```
> summary(X)
    Ratings          Likes
 Min.   :3.100   Min.   :     1
 1st Qu.:5.800   1st Qu.:  1776
 Median :6.500   Median :  6096
 Mean   :6.442   Mean   : 12732
 3rd Qu.:7.100   3rd Qu.: 15248
 Max.   :8.700   Max.   :370552
```

We normalize these two features to have value from 0 to 1. The formula of scale is: Let X = {x1, x2, x3, …, xn}, we can transfer xi(i from 1 to N) to yi(i from 1 to N) by using following formula.

$$y_i = \frac{x_i - \min(X)}{\max(X) - \min(X)}$$

```
# Re-scale columns of X such that values in each column range from 0 to 1.
rescale <- function(x){(x-min(x))/(max(x)-min(x))}
X$Ratings <- rescale(X$Ratings)
X$Likes <- rescale(X$Likes)
```

After scaling, both features value is between 0 and 1.

```
> summary(X)
    Ratings          Likes
 Min.   :0.0000   Min.   :0.000000
 1st Qu.:0.4821   1st Qu.:0.004791
 Median :0.6071   Median :0.016449
 Mean   :0.5967   Mean   :0.034358
 3rd Qu.:0.7143   3rd Qu.:0.041146
 Max.   :1.0000   Max.   :1.000000
```

2) Choose 2 for the number of clusters (K = 2).

   We decide to just give K the value of 2.

3) Implement a function km( ) which takes the input X and K. Internally,

   (a) the algorithm starts with initial centroids, (0, 0) and (1, 1);

   (b) the algorithm terminates after running 3 iterations.

The function will return final cluster-assignment in γ, and final coordinate of centroids in C as described.

3.2.2 Define K-means Algorithm

To developing km() function, let's first look at the procedure of k-means algorithm.

1) Initialize centroids which is (0, 0) and (1, 1) in this case. I make two centroids coordinate into a matrix. Each row represents a centroid ci(x, y), first column means x, second column means y.

```
# initialize centroids, (0, 0) and (1, 1)
c1 <- c(0,0)
c2 <- c(1,1)
cis <- rbind(c1,c2)
```

2) Fix centroids at current location. For each sample, calculate distance of each centroid. Assign the sample to the centroid with minimum distance.
We create a function called clusterDiv() to calculate minimum distance of centroids and samples then assign the points to closest centroid. We use L2-norm distance (Euclidean distance) which equals to:

$$distance = \sqrt{(x_{point} - x_{centroid})^2 + (y_{point} - y_{centroid})^2}$$

After calculating which cluster the samples should belong to, we store it into a vector called vec_cluster and return it.

```
# Function clusterDiv() calculate minimum distance then assige points to closest centroid
# Parameters are: k -> number of clusters, x -> input data, cis -> centroids
clusterDiv <- function(k, x, cis) {
  vec_cluster <- rep(0, nrow(x))
  for(i in 1:nrow(x)) {
    curx <- x[i,1]
    cury <- x[i,2]
    # initialize x and y
    dist <- rep(0, k)
    for (j in 1:nrow(cis)) {
      center_x <- cis[j,1]
      center_y <- cis[j,2]
      # calculate L2 distance
      dist[j] = sqrt((curx - center_x)^2 + (cury - center_y)^2)
    }
    vec_cluster[i] <- which.min(dist)
  }
  return(vec_cluster)
}
```

3) For each cluster, update centroids.
For centroids updating, we write a function called update_centroid(). The input parameter is dataset X and clustering result which stored in a vector. We select the points that are assigned to each cluster, calculate the sum of the abscissa of each point and the ordinate, and divide by the number of each cluster point to update the centroid of the cluster. Then we make two centroids a matrix for next iteration clustering.

```
# Function update_centroid() calculate each cluster's centroid
update_centroid <- function(X, vec) {
  sum_of_1x = 0
  sum_of_1y = 0
  count_1 = 0
  sum_of_2x = 0
  sum_of_2y = 0
  count_2 = 0
  for (i in 1:nrow(X)) {
    if (vec[i] == 1) {
      sum_of_1x = sum_of_1x + X[i,1]
      sum_of_1y = sum_of_1y + X[i,2]
      count_1 = count_1 + 1
    }
    if (vec[i] == 2) {
      sum_of_2x = sum_of_2x + X[i,1]
      sum_of_2y = sum_of_2y + X[i,2]
      count_2 = count_2 + 1
    }
  }
  c1 <- c(sum_of_1x/count_1, sum_of_1y/count_1)
  c2 <- c(sum_of_2x/count_2, sum_of_2y/count_2)
  cis <- rbind(c1, c2)
  return(cis)
}
```

4) After implementing each step of kmeans with a function, we implement a main function called km(). Input parameter is the dataset X, cluster number K, Iteration time IterTime and centroids set cis.
For each iteration, we first cluster according to the centroid, and the result of clustering is represented by plot, and the centroids used for clustering are represented on the graph as well. Then we update of the centroid. The output value is a vector of final clustering result and set of centroids.
The source code of km() function is shown below.

```
km <- function(X, K, IterTime, cis) {
  km_fw <- matrix(rep(0), nrow(X), IterTime)
  for(t in 1:IterTime) {
    km_fw[,t] <- clusterDiv(K, X, cis)
    # cis <- update_centroid(X, km_fw[,t])
    print(count(km_fw[,t]))
    print(cis)
    # draw the plot
    plot(X$Ratings, X$Likes, col = km_fw[,t])
    points(cis[1,1], cis[1,2], col = "green", pch = 15)
    points(cis[2,1], cis[2,2], col = "green", pch = 15)

    cis <- update_centroid(X, km_fw[,t])
  }
  out <- list("cluster membership" = km_fw[,IterTime], "centroids" = cis )
  return(out)
}
```

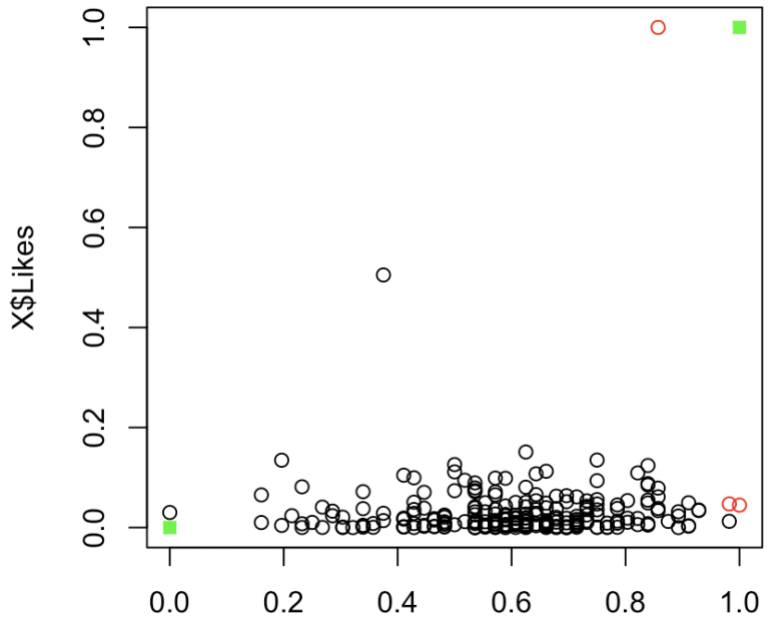
The result is shown as following. The centroids after 3 iterations are (0.552399, 0.02816937) and (0.8625541, 0.07149255).

```
> result
$`cluster membership`
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 2 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 2 2 1 1 1 2 1 1 1 2 1 1 1 1
 [61] 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
[121] 1 1 1 1 2 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 2 1 2 1 1 2 1 1 1 1 2 2 1 1 1 1
[181] 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1

$centroids
   Ratings   Likes
c1 0.552399  0.02816937
c2 0.8625541 0.07149255
```

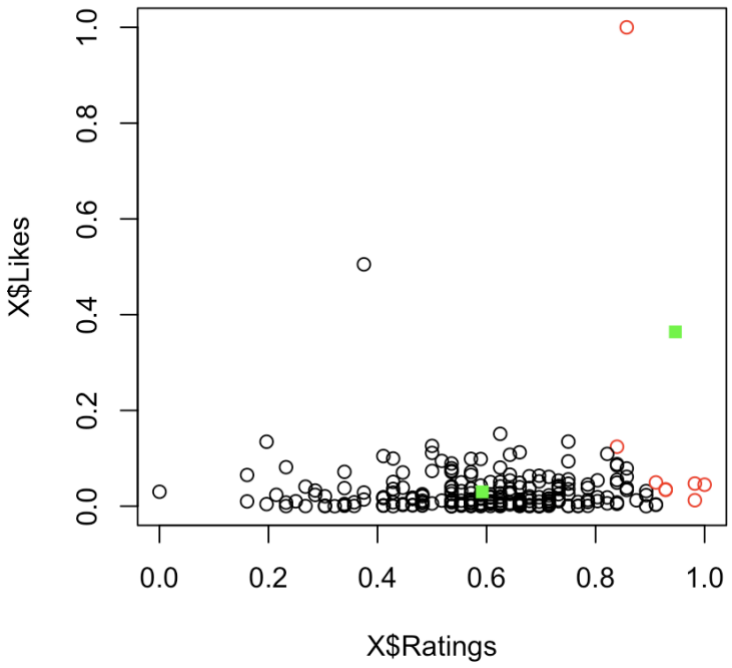
3.2.3 Plot the Cluster

Black points represent one cluster, red points represent another cluster. Green points represent centroid of each cluster.

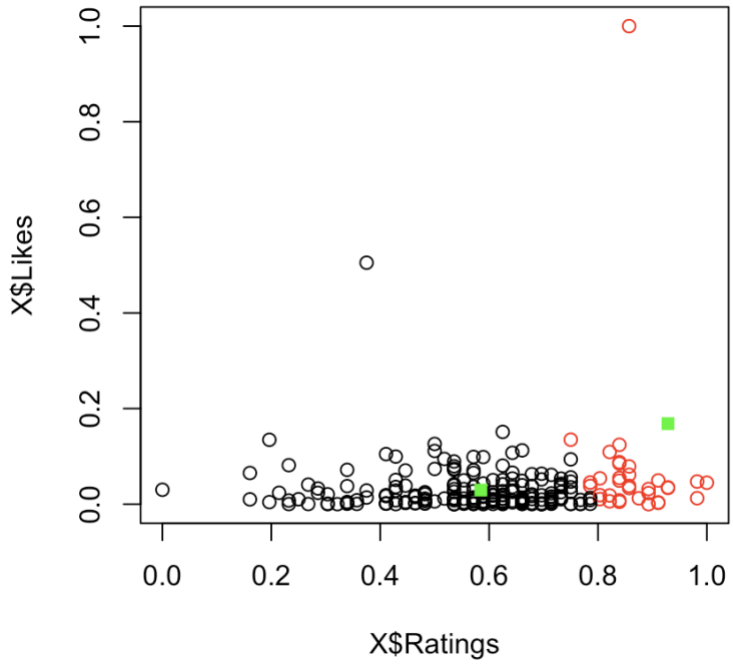
FIRST ITERATION



SECOND ITERATION



THIRD ITERATION



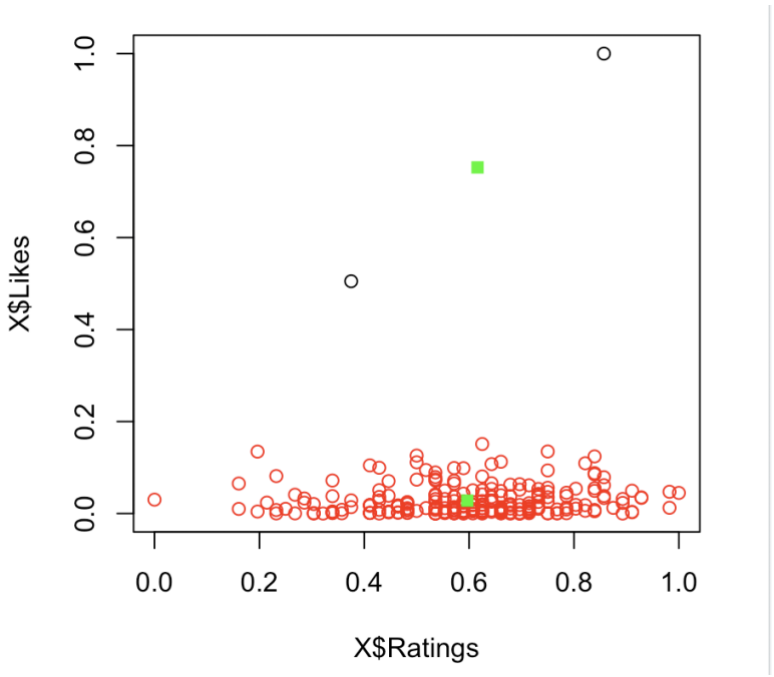
4. Exceptional Work – Improvement of k-means

4.1 Q: What will happened if we use different initialized centroids?

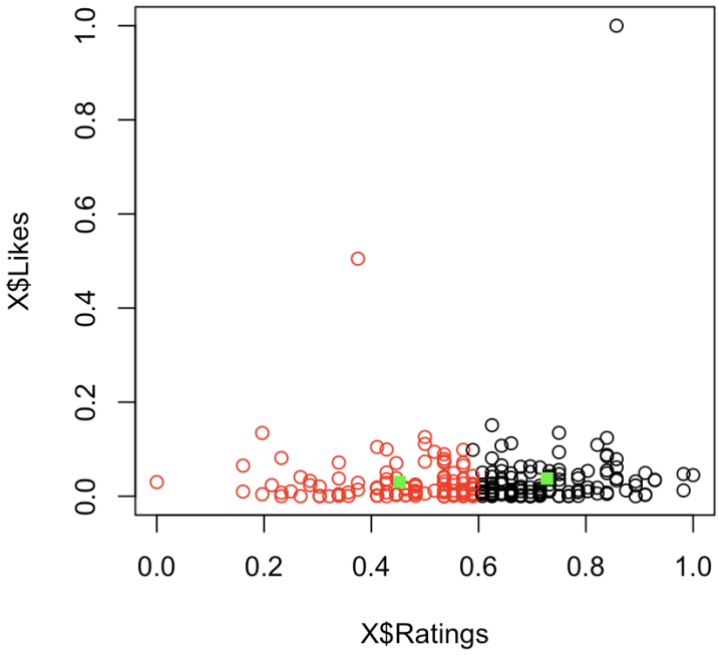
In the example above, we used the centroids (0,0) and (1,1) provided by the course. But if we change our initialized centroids, will the results of k-means change?

We still operate on data ‘Ratings’ and ‘Likes’. We use the initialization centroid (0,0) (1,1) and the initialization centroid (0,1) (1,0) to perform the k-means operation. The two cases are both divided into two clusters, and the number of iterations are both 100.

The result of initial centroid (0, 1) and (1, 0). Comparing with initial centroid (0, 0) and (1, 1).



Initial centroid (0, 1) and (1, 0)



Initial centroid (0, 0) and (1, 1)

According to the comparison of the above two figures, we can see that the initial centroid has a great influence on the clustering results. By doing some research, we found that k-means++ can reduce the impact of initial centroids on clustering results.

K-means++ is a specific way of choosing centers for the k-means algorithm. A better initial centroid could offer a better clustering result.

reference: http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

reference: http://www.cs.columbia.edu/~jebara/6772/proj/oldprojects/bal2123_BATS-Means.pdf

K-mean++ algorithm shows as following.

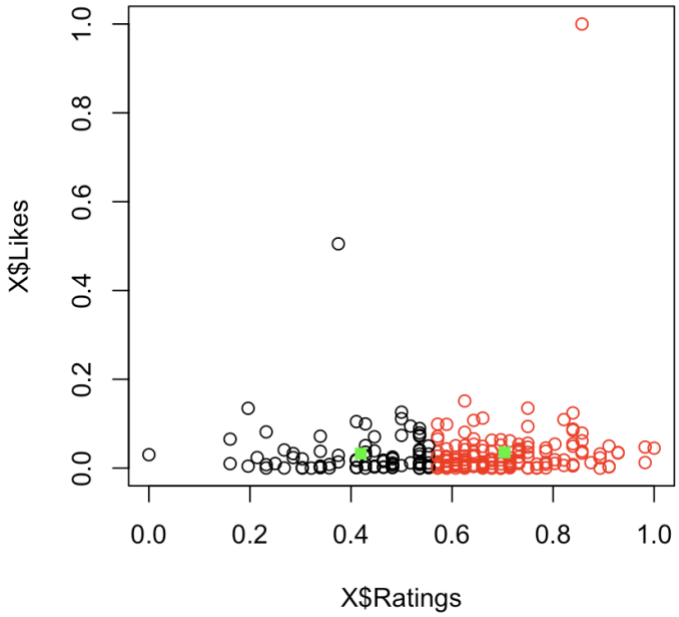
1) Take one center c_1, chosen uniformly at random from X.
2) Take a new center c_i, choosing x ∈ X with probability $\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$
3) Repeat Step 1b. until we have taken k centers altogether.
4) Proceed as with the standard k-means algorithm.

The principle of k-means++ is to make the initial cluster centroids farther apart from each other. This works very well.

We use package LICORS and function kmeanspp() to do k-means++ to our dataset. The input parameters are dataset X, cluster number K and iteration time iter.max.

```
# Solution: using kmean++
# reference: http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf
# reference: http://www.cs.columbia.edu/~jebara/6772/proj/oldprojects/bal2123_BATS-Means.pdf
install.packages("LICORS")
library(LICORS)
k <- kmeanspp(X, K, iter.max = 100)
plot(X$Ratings, X$Likes, col = k$cluster)
points(k$centers, col = "green", pch = 15)
?kmeanspp
# Conclusion: k-means is very sensitive to initialize centroids, so we use k-means++ to improve the cluster method
```

We plot the clustering result, which show as following.



The centroids after 100 iterations shows below. It's (0.419335, 0.03272775) and (0.703869, 0.03534358).

```
> k$centers
   Ratings      Likes
1 0.419335 0.03272775
2 0.703869 0.03534358
```

Conclusion: k-means is very sensitive to initialize centroids, so we use k-means++ to improve the cluster method.

4.2 Q: We find that X has two points far from majority. While calculate centroid, we use mean value, it will influence centroid. 'Outliers' will pull centroid to them.

Because we can't delete the "outliers". By researching, we find that using k_medoids could solve this problem.

reference: https://blog.csdn.net/buracag_mc/article/details/74853787

reference: https://stackoverflow.com/questions/21619794/what-makes-the-distance-measure-in-k-medoid-better-than-k-means

reference: https://blog.csdn.net/u012711335/article/details/78699896/

K-means clustering method is based on the mean, it is sensitive to outliers. A more robust approach is to divide the center point (PAM). Instead of using a centroid (variable mean vector) to represent a class, it is better to use a most representative point (called the center point). K-means clustering generally uses Euclidean distance, while PAM can be calculated using any distance.

The PAM algorithm is as follows:

(1) randomly select K observations (each called the center point);

(2) Calculate the distance/dissimilarity of observations to each center;

(3) Assign each observation to the nearest central point;

(4) Calculate the sum of the distances from each center point to each observation (total cost);

(5) Select a point in the class that is not the center and exchange it with the center point;

(6) Reassign each point to the nearest center point;

(7) Calculate the total cost again;

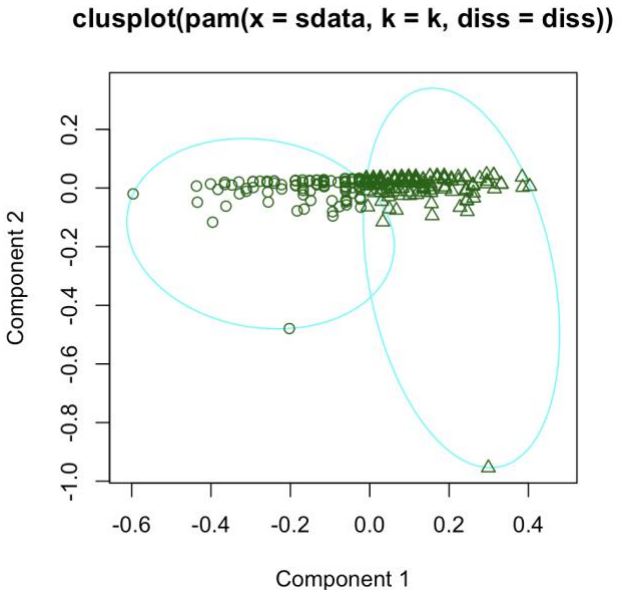
(8) If the total cost is less than the total cost calculated in step (4), the new point is taken as the central point;

(9) Repeat steps (5) to (8) until the center point does not change.

The following is the result of using this package. In this function, 'nc' could tell us the optimal number of clusters which is 2.

```
install.packages("fpc")
library(fpc)
pamk.result <- pamk(X)
pamk.result$nc
# recommand using cluster number K = 2
```

The following is the clustering result graph. This is the package that comes with. We can see the clustering result is very good, the boundary is clear, and it is not affected by the outliers.



Conclusion: k-means is very sensitive to noises and outliers, so we use k-medoids to improve the cluster method.