Institute of Distance and Open Learning

Vidya Nagari, Kalina, Santacruz East – 400098.

A Practical Journal Submitted in fulfillment

of the degree of

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

YEAR 2022-23

Part 2 Semester - 4

Subject Code -

Business Intelligence & Big Data

BY

Ashwin Solaki

Application ID - 117833

(Seat No.-)

INDEX

PRACTICAL NO. : 1

Aim : Pre-process the given data set and hence apply clustering techniques like K-Means , K-Medoids.Interpret the result.

K-Means Clustering Algorithm:

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

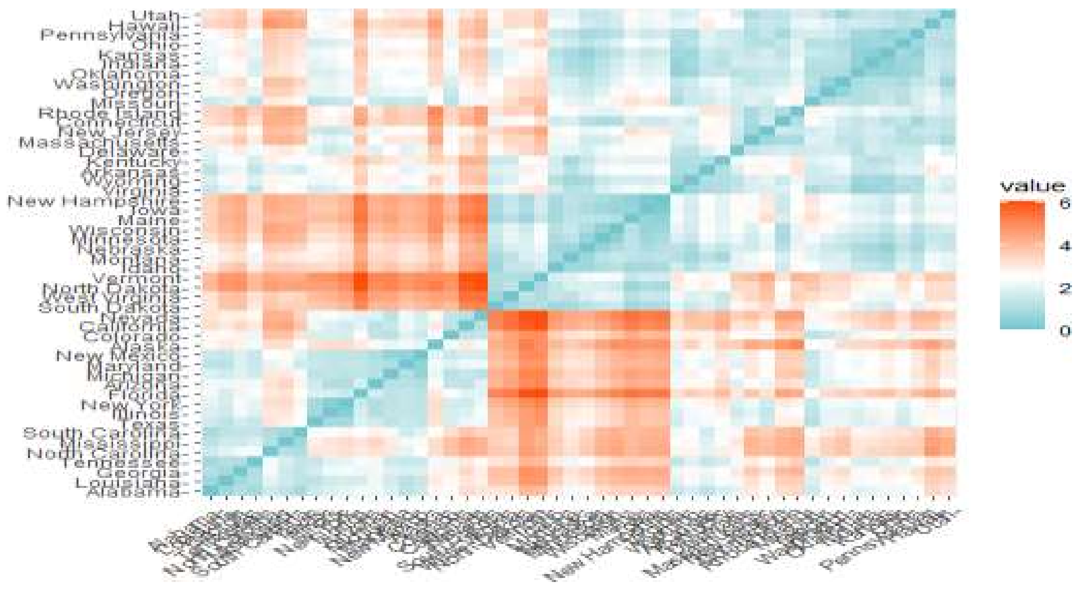
It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

Source code:

```
> install.packages('tidyverse')

> install.packages('cluster')

> install.packages('factoextra')

> library('tidyverse')

> library('cluster')

> library('factoextra')

> df <-USArrests
> df <-na.omit(df)
> df <-scale(df)
> head(df)
```

```
             Murder      Assault     UrbanPop     Rape
Alabama    1.24256408  0.7828393  -0.5209066  -0.003416473
Alaska     0.50786248  1.1068225  -1.2117642   2.484202941
Arizona    0.07163341  1.4788032   0.9989801   1.042878388
Arkansas   0.23234938  0.2308680  -1.0735927  -0.184916602
California 0.27826823  1.2628144   1.7589234   2.067820292
Colorado   0.02571456  0.3988593   0.8608085   1.864967207
```

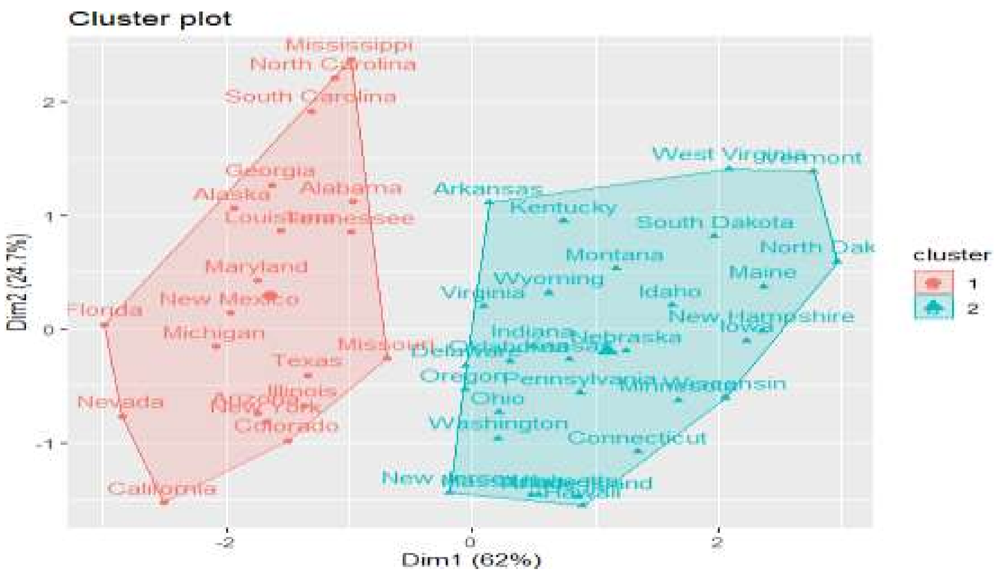
```
> distance <-get_dist(df)
> fviz_dist(distance, gradient = list(low = "#00AFBB", mid = "white", high
  ="#FC4E07"))
```



```
> k2 <- kmeans(df, centers = 2, nstart = 25)
> str(k2)

  List of 9
  $ cluster : Named int [1:50] 1 1 1 2 1 1 2 2 1 1 ...
  ..- attr(*, "names")= chr [1:50] "Alabama" "Alaska" "Arizona" "Arkansas" ...
  $ centers : num [1:2, 1:4] 1.005 -0.67 1.014 -0.676 0.198 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:2] "1" "2"
  .. ..$ : chr [1:4] "Murder" "Assault" "UrbanPop" "Rape"
  $ totss : num 196
  $ withinss : num [1:2] 46.7 56.1
  $ tot.withinss: num 103
  $ betweenss : num 93.1
  $ size : int [1:2] 20 30
  $ iter : int 1
  $ ifault : int 0
  - attr(*, "class")= chr "kmeans"

> k2

  K-means clustering with 2 clusters of sizes 20, 30
  Cluster means:
  Murder Assault UrbanPop Rape
  1 1.004934 1.0138274 0.1975853 0.8469650
```

```
2 -0.669956 -0.6758849 -0.1317235 -0.5646433
Clustering vector:
Alabama Alaska Arizona Arkansas California
1 1 1 2 1
Colorado Connecticut Delaware Florida Georgia
1 2 2 1 1
Hawaii Idaho Illinois Indiana Iowa
2 2 1 2 2
Kansas Kentucky Louisiana Maine Maryland
2 2 1 2 1
Massachusetts Michigan Minnesota Mississippi Missouri
2 1 2 1 1
Montana Nebraska Nevada New Hampshire New Jersey
2 2 1 2 2
New Mexico New York North Carolina North Dakota Ohio
1 1 1 2 2
Oklahoma Oregon Pennsylvania Rhode Island South Carolina
2 2 2 2 1
South Dakota Tennessee Texas Utah Vermont
2 1 1 2 2
Virginia Washington West Virginia Wisconsin Wyoming
2 2 2 2 2
Within cluster sum of squares by cluster:
[1] 46.74796 56.11445/
(between_SS / total_SS = 47.5 %)
Available components:
[1] "cluster" "centers" "totss" "withinss" "tot.withinss"
[6] "betweenss" "size" "iter" "ifault"
```

> fviz_cluster(k2, data = df)



PRACTICAL NO. : 2

Aim : Pre-process the given data set and hence apply partition clustering algorithms .Interpret the result.

CLARANS algorithm:

CLARANS (**C**lustering **L**arge **A**pplications based on **RAN**domized **S**earch) is a Data Mining algorithm designed to cluster spatial data. CLARANS is a partitioning method of clustering particularly useful in spatial data mining. We mean recognizing patterns and relationships existing in spatial data (such as distance-related, direction-relation or topological data, e.g. data plotted on a road map) by spatial data mining.

The CLARA algorithm was introduced as an extension of K-Medoids. It uses only random samples of the input data (instead of the entire dataset) and computes the best medoids in those samples.

CLARANS algorithm takes care of the cons of both K-Medoids and CLARA algorithms besides dealing with difficult-to-handle data mining data, i.e. spatial data. It maintains a balance between the computational cost and the influence of data sampling on clusters' formation.

Source code:

```
> install.packages('CLARA')

> install.packages('cluster')

> library('cluster')

> set.seed(1234)

> df <- rbind(cbind(rnorm(200,0,8), rnorm(200,0,8)),
+ cbind(rnorm(300,50,8), rnorm(300,50,8)))

> colnames(df) <- c("x", "y")

> rownames(df) <- paste0("S", 1:nrow(df))

> head(df, nrow = 6)
```

```
            x          y
 S1  -9.656526   3.881815
 S2  2.219434    5.574150
 S3  8.675529    1.484111
```

```
S4   -18.765582  5.605868
S5   3.432998    2.493448
S6   4.048447    6.083699
```

```
> clara(df, k=2, metric = "euclidean", stand = FALSE,
+ samples = 5, pamLike = FALSE)

 Call: clara(x = df, k = 2, metric = "euclidean", stand = FALSE, samples = 5,
 pamLike = FALSE)
 Medoids:
              x            y
 S1 68 -0.5495492 2.458514
 S3 97 47.5964047 50.892735
 Objective function: 9.9971
 Clustering vector: Named int [1:500] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "names")= chr [1:500] "S1" "S2" "S3" "S4" "S5" "S6" "S7" ...
 Cluster sizes: 200 300
 Best sample:
 [1] S6 S45 S51 S67 S75 S85 S90 S94 S97 S110 S111 S160 S168 S170 S176
 [16] S181 S201 S219 S249 S260 S264 S275 S296 S304 S317 S319 S337 S361
 S362 S369
 [31] S370 S374 S379 S397 S398 S411 S420 S422 S424 S436 S448 S458 S465
 S489
 Available components:
 [1] "sample" "medoids" "i.med" "clustering" "objective"
 [6] "clusinfo" "diss" "call" "silinfo" "data"

> install.packages(c("cluster", "factoextra"))

> library(ggplot2)

> library(fs)

> library(cluster)

> library(factoextra)

> clara.res <- clara(df, 2, samples = 50, pamLike = TRUE)
> print(clara.res)

  Call: clara(x = df, k = 2, samples = 50, pamLike = TRUE)
  Medoids:
               x          y
  S121   -1.531137 1.145057
  S455   48.357304 50.233499
  Objective function: 9.87862
  Clustering vector: Named int [1:500] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
```

```
- attr(*, "names")= chr [1:500] "S1" "S2" "S3" "S4" "S5" "S6" "S7" ...
Cluster sizes: 200 300
Best sample:
[1] S37 S49 S54 S63 S68 S71 S76 S80 S82 S101 S103 S108 S109 S118 S121
[16] S128 S132 S138 S144 S162 S203 S210 S216 S231 S234 S249 S260 S261
S286 S299
[31] S304 S305 S312 S315 S322 S350 S403 S450 S454 S455 S456 S465 S488
S497
Available components:
[1] "sample" "medoids" "i.med" "clustering" "objective"
[6] "clusinfo" "diss" "call" "silinfo" "data"
```

> **dd <- cbind(df, cluster = clara.res$cluster)**

> **head(dd, n = 4)**

```
          x            y        cluster
S1   -9.656526   3.881815     1
S2    2.219434   5.574150     1
S3    8.675529   1.484111     1
S4  -18.765582 5.605868      1
```

> **clara.res$medoids**

```
           x          y
S121 -1.531137 1.145057
S455 48.357304 50.233499
```

> **head(clara.res$clustering, 10)**

```
S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
1 1    1  1 1 1     1 1 1 1
```

> **clarax<-clara(df,2)**

> **clarax**

```
Call: clara(x = df, k = 2)
Medoids:
x y
S168 -0.5495492 2.458514
S397 47.5964047 50.892735
Objective function: 9.9971
Clustering vector: Named int [1:500] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
- attr(*, "names")= chr [1:500] "S1" "S2" "S3" "S4" "S5" "S6" "S7" ...
```

```
 Cluster sizes: 200 300
 Best sample:
 [1] S6 S45 S51 S67 S75 S85 S90 S94 S97 S110 S111 S160 S168 S170 S176
 [16] S181 S201 S219 S249 S260 S264 S275 S296 S304 S317 S319 S337 S361
 S362 S369
 [31] S370 S374 S379 S397 S398 S411 S420 S422 S424 S436 S448 S458 S465
 S489
 Available components:
 [1] "sample" "medoids" "i.med" "clustering" "objective"
 [6] "clusinfo" "diss" "call" "silinfo" "data"
```

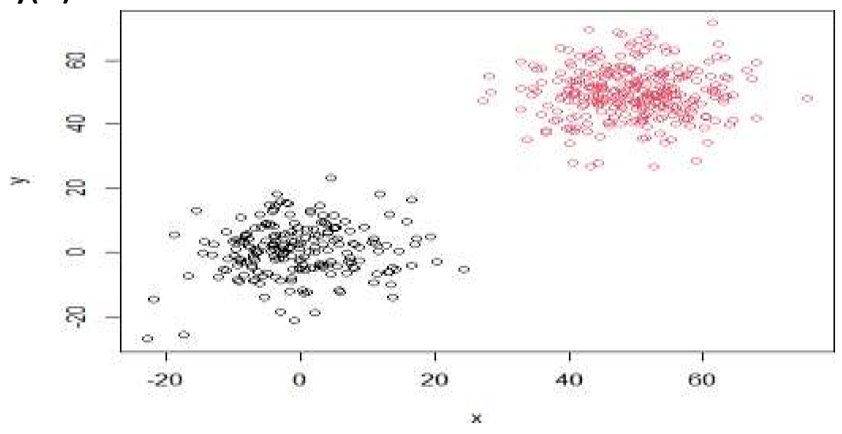
> **plot(df,col=clarax$clustering)**

> **print(clarax)**

```
 Call: clara(x = df, k = 2)
 Medoids:
              x          y
 S168   -0.5495492   2.458514
 S397   47.5964047 50.892735
 Objective function: 9.9971
 Clustering vector: Named int [1:500] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "names")= chr [1:500] "S1" "S2" "S3" "S4" "S5" "S6" "S7" ...
 Cluster sizes: 200 300
 Best sample:
 [1] S6 S45 S51 S67 S75 S85 S90 S94 S97 S110 S111 S160 S168 S170 S176
 [16] S181 S201 S219 S249 S260 S264 S275 S296 S304 S317 S319 S337 S361
 S362 S369
 [31] S370 S374 S379 S397 S398 S411 S420 S422 S424 S436 S448 S458 S465
 S489
 Available components:
 [1] "sample" "medoids" "i.med" "clustering" "objective"
 [6] "clusinfo" "diss" "call" "silinfo" "data"
```

> **library(ggplot2)**

> **library(fs)**



PRACTICAL NO. : 3

Aim : Pre-process the given data set and hence apply hierarchical algorithms and density based clustering techniques. Interpret the result.

Density based clustering algorithm:

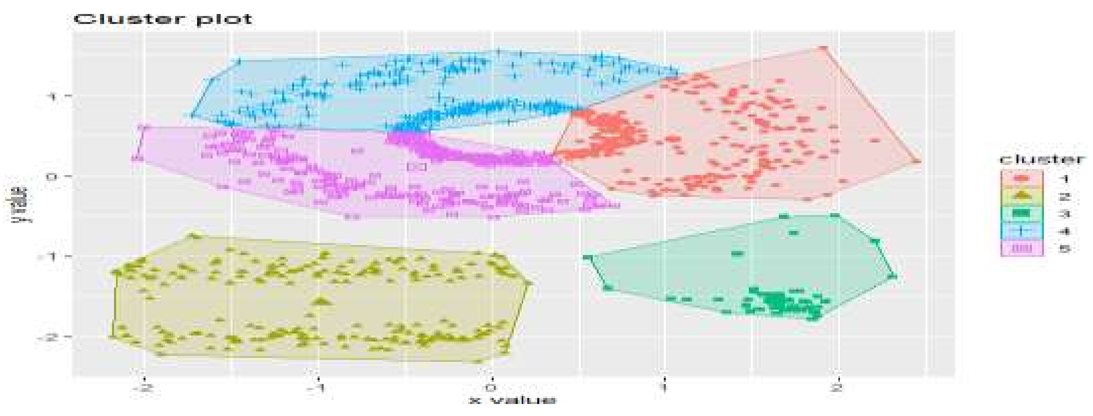
Density based clustering algorithm has played a vital role in finding non linear shapes structure based on the density. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is most widely used density based algorithm. It uses the concept of **density reachability** and **density connectivity**.

Density Reachability - A point "p" is said to be density reachable from a point "q" if point "p" is within ε distance from point "q" and "q" has sufficient number of points in its neighbors which are within distance ε.

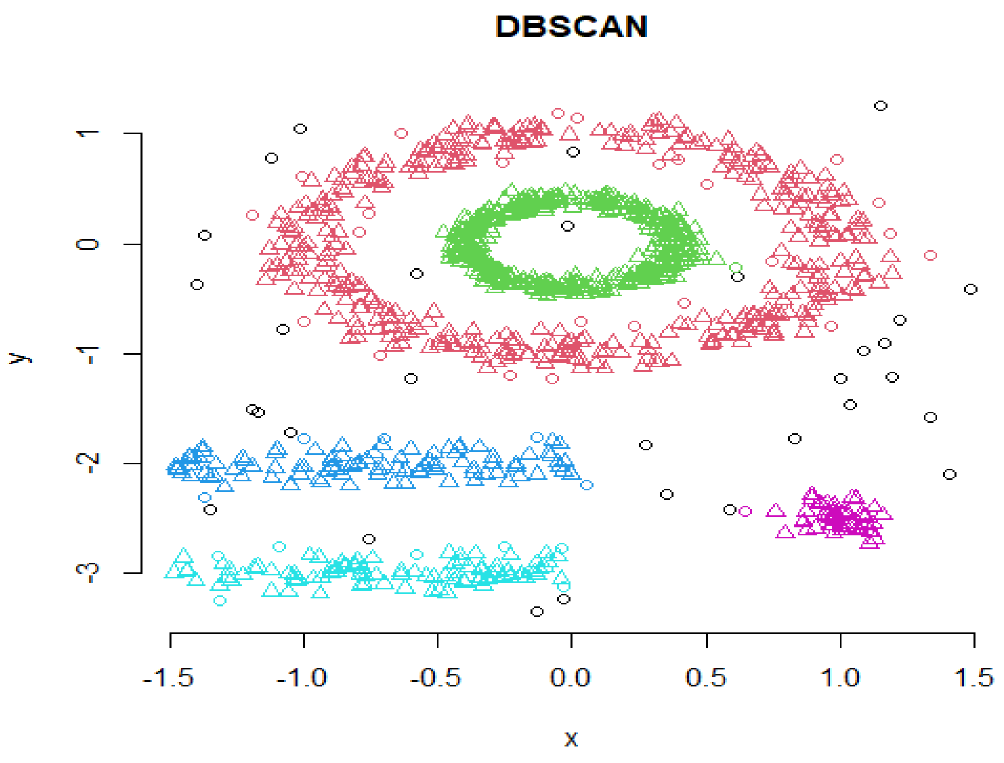
Density Connectivity - A point "p" and "q" are said to be density connected if there exist a point "r" which has sufficient number of points in its neighbors and both the points "p" and "q" are within the ε distance. This is chaining process. So, if "q" is neighbor of "r", "r" is neighbor of "s", "s" is neighbor of "t" which in turn is neighbor of "p" implies that "q" is neighbor of "p".

Source code:

```
> library(factoextra)

> data("multishapes")

> df <- multishapes[, 1:2]

> set.seed(123)

> km.res <-kmeans(df ,5, nstart = 25)

> fviz_cluster(km.res,df, frame= FALSE, geom = "point")
```



```
> install.packages("fpc")

> install.packages("dbscan")

> library(dbscan)

> dbscan(df, eps=5, minPts = 5, weights = NULL)

> data("multishapes", package= "factoextra")

> df<- multishapes[, 1:2]

> library("fpc")

#Compute DBSCAN using fpc package:

> set.seed(123)

> db <- fpc::dbscan(df, eps = 0.15, MinPts = 5)

#Plot DBSCAN results:

> plot(db, df, main= "DBSCAN", frame = FALSE)
```



PRACTICAL NO. : 4

Aim : Pre-process the given data set and hence classify the resultant dataset using tree classification techniques. Interpret the result.

J48 algorithm:

J48 algorithm is one of the best machine learning **algorithms** to examine the data categorically and continuously. When it is used for instance purpose, it occupies more memory space and depletes the performance and accuracy in classifying medical data.

Following are the steps:

1. Open Weka, then Open File i.e “Bank-data.csv” in Weka Explorer.
2. Go to Filter, Click on Choose Filters Unsupervised Attribute Discretrized option.
3. In Discretrized changes the attributeindices and No. of bins Ok Apply.
4. Save the file as “Bank-datafinal.arff” Save.
5. Open File “Bank-datafinal.arff”, then apply Classification algorithm.
6. Click on Classify Choose trees J48 Then Start.
7. Right Click on treesJ48 Visualize tree.

P
RACTICAL NO. : 5

Aim : Pre-process the given data set and hence classify the resultant dataset using statistical based classifiers. Interpret the result.

Naive Bayes Classifiers:

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Following are the steps:

1. Open Weka, then Open file i.e "iris.csv" in Weka Explorer.
2. Click on Choose weka Unsupervised instance RemovePercentage Apply
3. Save file as "iris1.arff" save. Open File "iris1.arff".
4. Click on RemovePercentage Change InvertSelection as TRUE Ok Apply.
5. Save file as "irisfinal.arff" save.
6. Click on Classify Choose Classifier bayes Naivebayes.
7. In Test Options Select Supplied test set set.
8. Open File "irisfinal.arff" , then Start.
9. Right Click bayesNaiveBayes Re-evaluate model on current test set.
10. Click on Start.

PRACTICAL NO. : 6

Aim : Pre-process the given data set and hence classify the resultant dataset using Support Vector Machine. Interpret the result.

Support Vector Machine Algorithm:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

Source code:

```
> x=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)

> y=c(3,4,5,4,8,10,10,11,14,20,23,24,32,34,35,37,42,48,53,60)
```

#Create a data frame of the data:

```
> train=data.frame(x,y)
```

#Plot the dataset:

```
> plot(train,pch=16)
```

#Linear regression:

```
> model <- lm(y ~ x, train)
```

#Plot the model using abline:

```
> abline(model)
```

#SVM:

```
> library(e1071)
```

#Fit a model:

```
> model_svm <- svm(y ~ x , train)
```

#Use the predictions on the data:

```
> pred <- predict(model_svm, train)
```

#Plot the predictions and the plot to see our model fit:

```
> points(train$x, pred, col = "blue", pch=4)
```

#Linear model has a residuals part which we can extract and directly calculate rmse:

```
> error <- model$residuals
> lm_error <- sqrt(mean(error^2))
> print(lm_error)
```

#For svm, we have to manually calculate the difference between actual values (train$y) with our predictions (pred):

```
> error_2 <- train$y - pred
> svm_error <- sqrt(mean(error_2^2))
> print(svm_error)
```

# perform a grid search:

```
> svm_tune <- tune(svm, y ~ x, data = train,
 ranges = list(epsilon = seq(0,1,0.01), cost = 2^(2:9))
)
> print(svm_tune)
```

#The best model:

```
> best_mod <- svm_tune$best.model

> best_mod_pred <- predict(best_mod, train)

> plot(svm_tune)

> plot(train,pch=16)

> points(train$x, best_mod_pred, col = "blue", pch=4)
```

PRACTICAL NO. : 7

Aim : Write a program to explain different functions of Principal Components.

Principal Component Analysis (PCA):-

In order to handle “curse of dimensionality” and avoid issues like over-fitting in high dimensional space, methods like Principal Component analysis is used.

PCA is a method used to reduce number of variables in your data by extracting important one from a large pool. It reduces the dimension of your data with the aim of retaining as much information as possible. In other words, this method combines highly correlated variables together to form a smaller number of an artificial set of variables which is called “principal components” that account for most variance in the data.

Source code:

```
> install.packages('tidyverse')

> install.packages('gridextra')

> library(tidyverse)

> library(gridExtra)

> data("USArrests")

> head(USArrests,10)
```

```
             Murder   Assault  UrbanPop  Rape
Alabama       13.2      236       58      21.2
Alaska        10.0      263       48      44.5
Arizona       8.1       294       80      31.0
Arkansas      8.8       190       50      19.5
California    9.0       276       91      40.6
Colorado      7.9       204       78      38.7
Connecticut 3.3         110       77      11.1
Delaware      5.9       238       72      15.8
Florida 15.4 335 80 31.9
Georgia 17.4 211 60 25.8
```

#Compute variance of each variable:

```
> apply(USArrests,2,var)

Murder     Assault        UrbanPop    Rape
18.97047 6945.16571     209.51878   87.72916
```

Standardizing each variable will fix this issue.
#create new data frame with centered variables:

```
> scaled_df <-apply(USArrests,2,scale)
> head(scaled_df)

      Murder         Assault        UrbanPop      Rape
[1,] 1.24256408   0.7828393    -0.5209066   -0.003416473
[2,] 0.50786248   1.1068225    -1.2117642    2.484202941
[3,] 0.07163341   1.4788032     0.9989801    1.042878388
[4,] 0.23234938   0.2308680    -1.0735927   -0.184916602
[5,] 0.27826823   1.2628144     1.7589234    2.067820292
[6,] 0.02571456   0.3988593     0.8608085    1.864967207
```

#Calculate egienvalues:

```
> arrests.cov <- cov(scaled_df)

> arrests.eigen <- eigen(arrests.cov)

> str(arrests.eigen)

List of 2
$ values : num [1:4] 2.48 0.99 0.357 0.173
$ vectors: num [1:4, 1:4] -0.536 -0.583 -0.278 -0.543 0.418 ...
- attr(*, "class")= chr "eigen"


> (phi <- arrests.eigen$vectors[,1:2])

            [,1]            [,2]
[1,]  -0.5358995    0.4181809
[2,]  -0.5831836    0.1879856
[3,]  -0.2781909   -0.8728062
[4,]  -0.5434321   -0.1673186
```

```
> phi <- -phi

> row.names(phi) <- c("Murder","Assault","UrbanPop","Rape")

> colnames(phi) <- c("PC1","PC2")

> phi

              PC1          PC2
Murder     0.5358995   -0.4181809
Assault    0.5831836   -0.1879856
UrbanPop   0.2781909    0.8728062
Rape       0.5434321    0.1673186


> PC1 <-as.matrix(scaled_df) %*% phi[,1]

> PC2 <-as.matrix(scaled_df) %*% phi[,2]

> pc <- data.frame(State = row.names(USArrests),PC1, PC2)

> head(pc)

       State       PC1          PC2
1    Alabama    0.9756604   -1.1220012
2    Alaska     1.9305379   -1.0624269
3    Arizona    1.7454429    0.7384595
4    Arkansas  -0.1399989   -1.1085423
5    California  2.4986128    1.5274267
6    Colorado    1.4993407    0.9776297


> ggplot(pc, aes(PC1, PC2)) + modelr::geom_ref_line(h = 0) +

+ modelr::geom_ref_line(h = 0) +

+ modelr::geom_ref_line(v = 0) +

+ geom_text(aes(label = State), size =3) +

+ xlab("First Principal Component") +

+ ylab("Second Principal Component") +

+ ggtitle("First Two Principal Components of USArrests Data")
```

PRACTICAL NO. : 8

Aim : Write a program to explain CUR Decomposition technique.

CUR Matrix Decomposition:

CUR matrix decomposition is a low-rank matrix decomposition algorithm that is explicitly expressed in a small number of actual columns and/or actual rows of data matrix.

CUR matrix decomposition was developed as an alternative to Singular Value Decomposition (SVD) and Principal Component Analysis (PCA). CUR matrix decomposition selects columns and rows that exhibit high **statistical leverage** or large **influence** from the data matrix. By implementing the CUR matrix decomposition algorithm, a small number of most important attributes and/or rows can be identified from the original data matrix. Therefore, CUR matrix decomposition is an important tool for exploratory data analysis. CUR matrix decomposition can be applied to a variety of areas and facilitates Regression, Classification, and Clustering.

Source code:

```
> install.packages(“devtools”)

> library(devtools)

> install_github(“cran/rCUR”)

> library(rCUR)

> install.packages(“lattice”)

> library(lattice)

> data(STTm)

> data(STTa)

> n=27

> res=CUR(STTm,31,n,4)

> plotLeverage(res, C=FALSE, top.n=n, xlab=’GeneID’, las=1, top.col=”black”,
top.pch=16, ul.col=”black”, ul.lty=2, col=”grey”)
```

PRACTICAL NO. : 9

Aim : Write a program to explain links to establish higher-order relationships among entities in link analysis.

Page Rank Algorithm:

PageRank (PR) is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages.

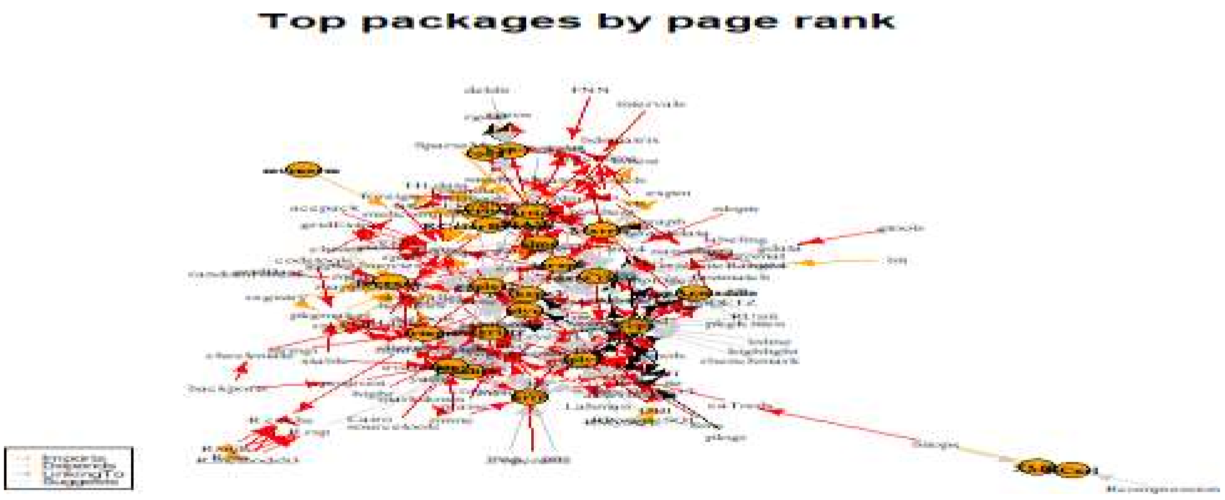
According to Google: PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

It is not the only algorithm used by Google to order search engine results, but it is the first algorithm that was used by the company, and it is the best-known.The above centrality measure is not implemented for multi-graphs.

Source code:

```
> install.packages('miniCRAN')

> install.packages('igraph')

> install.packages('magrittr')

> library(miniCRAN)

> library(igraph)

> library(magrittr)

> MRAN <- http://mran.revolutionanalytics.com/snapshot/2017-02-01/

> pdb <- MRAN %>%

+ contrib.url(type = "source") %>%

+ available.packages(type="source", filters = NULL)
```

```
> g <- pdb[, "Package"] %>%

+ makeDepGraph(availPkgs = pdb, suggests=FALSE, enhances=TRUE,
includeBasePkgs = FALSE)

> pr <- g %>%

+ page.rank(directed = FALSE) %>%

+ use_series("vector") %>%

+ sort(decreasing = TRUE) %>%

+ as.matrix %>%

+set_colnames("page.rank")

> pr2 <- as.data.frame(pr)

> pr2

> set.seed(42)

> pr2 %>%

+ head(25) %>%

+ rownames %>%

+ makeDepGraph(pdb) %>%

+ plot(main="Top packages by page rank", cex=0.5, width=120)
```



P
RACTICAL NO. : 10

Aim : Write a program to implement step-by-step a Collaborative Filtering Recommender System.

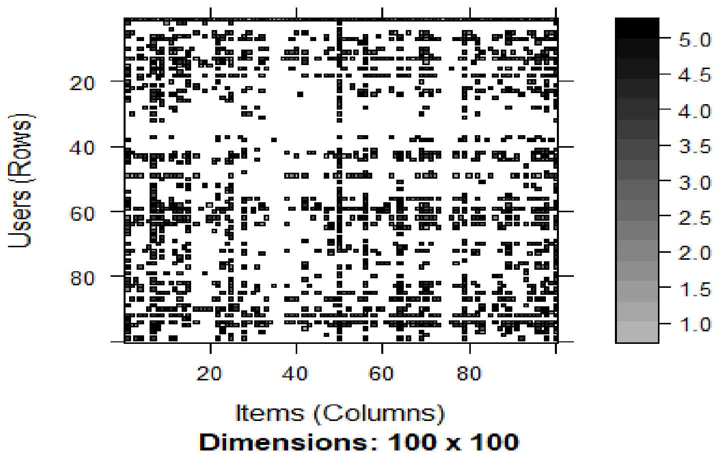
Collaborative Filtering:

To address some of the limitations of content-based filtering, collaborative filtering uses similarities between users and items simultaneously to provide recommendations. This allows for serendipitous recommendations; that is, collaborative filtering models can recommend an item to user A based on the interests of a similar user B. Furthermore, the embeddings can be learned automatically, without relying on hand-engineering of features.

Collaborative Filtering, we tend to find similar users and recommend what similar users like. In this type of recommendation system, we don’t use the features of the item to recommend it, rather we classify the users into the clusters of similar types, and recommend each user according to the preference of its cluster.

Source code:

```
> install.packages(“recommenderlab”)

> install.packages(“stringi”)

> install.packages(“reshape2”)

> library(recommenderlab)

> library(stringi)

> library(reshape2)

> data("MovieLense")

> MovieLense

943 x 1664 rating matrix of class ‘realRatingMatrix’ with 99392 ratings.

> ml10 <- MovieLense[c(1:10),]

> ml10 <- ml10[,c(1:10)]

> as(ml10, "matrix")
> image(MovieLense[1:100,1:100])
```



```
> train <- MovieLense

> our_model <- Recommender(train, method = "UBCF")

> our_model

Recommender of type 'UBCF' for 'realRatingMatrix'
learned using 943 users

> User = 115

> pre <- predict(our_model, MovieLense[User], n = 10)

> pre

Recommendations as 'topNList' with n = 10 for 1 users.

> user_ratings <- train[User]

> as(user_ratings, "list")

> as(pre,"list")

 $`115`
 [1] "Secrets & Lies (1996)"     "Marvin's Room (1996)"
 [3] "Close Shave, A (1995)"     "Mr. Holland's Opus (1995)"
 [5] "Matilda (1996)"            "Fallen (1998)"
 [7] "Mad City (1997)"           "Gattaca (1997)"
 [9] "Tango Lesson, The (1997)" "Welcome To Sarajevo (1997)"
```