library(datasets)

head(iris)

Sepal.Length Sepal.Width Petal.Length Petal.Width Species

1 5.1 3.5 1.4 0.2 setosa

2 4.9 3.0 1.4 0.2 setosa

3 4.7 3.2 1.3 0.2 setosa

4 4.6 3.1 1.5 0.2 setosa

5 5.0 3.6 1.4 0.2 setosa

6 5.4 3.9 1.7 0.4 setosa

After a little bit of exploration, I found that Petal.Length and Petal.Widthwere similar among the same species but varied considerably between different species, as demonstrated below:

library(ggplot2)

ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) + geom\_point()

Here is the plot:

set.seed(20)

irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)

irisCluster

K-means clustering with 3 clusters of sizes 46, 54, 50

Cluster means:

Petal.Length Petal.Width

1 5.626087 2.047826

2 4.292593 1.359259

3 1.462000 0.246000

Clustering vector:

[1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

[35] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

[69] 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1

[103] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1

[137] 1 1 2 1 1 1 1 1 1 1 1 1 1 1

Within cluster sum of squares by cluster:

[1] 15.16348 14.22741 2.02200

(between\_SS / total\_SS = 94.3 %)

Available components:

[1] "cluster" "centers" "totss" "withinss"

[5] "tot.withinss" "betweenss" "size" "iter"

[9] "ifault"

table(irisCluster$cluster, iris$Species)

setosa versicolor virginica

1 0 2 44

2 0 48 6

3 50 0 0

irisCluster$cluster <- as.factor(irisCluster$cluster)

ggplot(iris, aes(Petal.Length, Petal.Width, color = iris$cluster)) + geom\_point()

data <- scale(data)

# Dissimilarity matrix

d <- dist(data, method = "euclidean")

# Hierarchical clustering using Complete Linkage

hc1 <- hclust(d, method = "complete" )

# Plot the obtained dendrogram

plot(hc1, cex = 0.6, hang = -1)

hc2 <- agnes(data, method = "complete")

# Agglomerative coefficient

hc2$ac

## [1] 0.9317012

Let’s compare the methods discussed

# vector of methods to compare

m <- c( "average", "single", "complete", "ward")

names(m) <- c( "average", "single", "complete", "ward")

# function to compute coefficient

ac <- function(x) {

  agnes(data, method = x)$ac

}

map\_dbl(m, ac)

## from ‘purrr’ package

## average single complete ward

## 0.9241325 0.9215283 0.9317012 0.9493598

Ward’s method gets us the highest agglomerative coefficient. Let us look at its dendogram.

hc3 <- agnes(data, method = "ward")

pltree(hc3, cex = 0.6, hang = -1, main = "Dendrogram of agnes")

hc4 <- diana(data)

# Divise coefficient

hc4$dc

## [1] 0.9305939

# plot dendrogram

pltree(hc4, cex = 0.6, hang = -1, main = "Dendrogram of diana")

**#** install.packages("dbscan")

library(dbscan)

iris\_matrix <- as.matrix(iris[, -5])

kNNdistplot(iris\_matrix, k=4)

abline(h=0.4, col="red")

set.seed(1234)

db = dbscan(iris\_matrix, 0.4, 4)

db

hullplot(iris\_matrix, db$cluster)

# Run this program on your local python

# interpreter, provided you have installed

# the required libraries.

# Importing the required packages

import numpy as np

import pandas as pd

from sklearn.metrics import confusion\_matrix

from sklearn.cross\_validation import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

# Function importing Dataset

def importdata():

balance\_data = pd.read\_csv(

'https://archive.ics.uci.edu/ml/machine-learning-databases/blood-transfusion/transfusion.data',

sep= ',', header = 0 )

# Printing the dataswet shape

print ("Dataset Lenght: ", len(balance\_data))

print ("Dataset Shape: ", balance\_data.shape)

# Printing the dataset obseravtions

print ("Dataset: ",balance\_data.head())

return balance\_data

# Function to split the dataset

def splitdataset(balance\_data):

# Seperating the target variable

X = balance\_data.values[:, 0:4]

Y = balance\_data.values[:, 4]

# Spliting the dataset into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size = 0.3, random\_state = 100)

return X, Y, X\_train, X\_test, y\_train, y\_test

# Function to perform training with giniIndex.

def train\_using\_gini(X\_train, X\_test, y\_train):

# Creating the classifier object

clf\_gini = DecisionTreeClassifier(criterion= "gini",

random\_state = 100,max\_depth=3, min\_samples\_leaf=5)

# Performing training

clf\_gini.fit(X\_train, y\_train)

return clf\_gini

# Function to perform training with entropy.

def tarin\_using\_entropy(X\_train, X\_test, y\_train):

# Decision tree with entropy

clf\_entropy = DecisionTreeClassifier(

criterion = "entropy", random\_state = 100,

max\_depth = 3, min\_samples\_leaf = 5)

# Performing training

clf\_entropy.fit(X\_train, y\_train)

return clf\_entropy

# Function to make predictions

def prediction(X\_test, clf\_object):

# Predicton on test with giniIndex

y\_pred = clf\_object.predict(X\_test)

print("Predicted values:")

print(y\_pred)

return y\_pred

# Function to calculate accuracy

def cal\_accuracy(y\_test, y\_pred):

print("Confusion Matrix: ",

confusion\_matrix(y\_test, y\_pred))

print ("Accuracy : ",

accuracy\_score(y\_test,y\_pred)\*100)

print("Report : ",

classification\_report(y\_test, y\_pred))

# Driver code

def main():

# Building Phase

data = importdata()

X, Y, X\_train, X\_test, y\_train, y\_test = splitdataset(data)

clf\_gini = train\_using\_gini(X\_train, X\_test, y\_train)

clf\_entropy = tarin\_using\_entropy(X\_train, X\_test, y\_train)

# Operational Phase

print("Results Using Gini Index:")

# Prediction using gini

y\_pred\_gini = prediction(X\_test, clf\_gini)

cal\_accuracy(y\_test, y\_pred\_gini)

print("Results Using Entropy:")

# Prediction using entropy

y\_pred\_entropy = prediction(X\_test, clf\_entropy)

cal\_accuracy(y\_test, y\_pred\_entropy)

# Calling main function

if \_\_name\_\_=="\_\_main\_\_":

main()

# Run this program on your local python

# interpreter, provided you have installed

# the required libraries.

# Importing the required packages

import numpy as np

import pandas as pd

from sklearn.metrics import confusion\_matrix

from sklearn.cross\_validation import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

# Function importing Dataset

def importdata():

balance\_data = pd.read\_csv('C:/Users/Lenovo/Desktop/tarp.csv',sep= ',', header = 1 )

# Printing the dataswet shape

print ("Dataset Lenght: ", len(balance\_data))

print ("Dataset Shape: ", balance\_data.shape)

# Printing the dataset obseravtions

print ("Dataset: ",balance\_data.head())

return balance\_data

# Function to split the dataset

def splitdataset(balance\_data):

# Seperating the target variable

X = balance\_data.values[:, 1:13]

Y = balance\_data.values[:, 13]

# Spliting the dataset into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size = 0.3, random\_state = 100)

return X, Y, X\_train, X\_test, y\_train, y\_test

# Function to perform training with giniIndex.

def train\_using\_gini(X\_train, X\_test, y\_train):

# Creating the classifier object

clf\_gini = DecisionTreeRegressor(max\_depth=3)

# Performing training

clf\_gini.fit(X\_train, y\_train)

return clf\_gini

# Function to perform training with entropy.

def tarin\_using\_entropy(X\_train, X\_test, y\_train):

# Decision tree with entropy

clf\_entropy = DecisionTreeRegressor(max\_depth = 3)

# Performing training

clf\_entropy.fit(X\_train, y\_train)

return clf\_entropy

# Function to make predictions

def prediction(X\_test, clf\_object):

# Predicton on test with giniIndex

y\_pred = clf\_object.predict(X\_test)

#print("Predicted values:")

#print(y\_pred)

return y\_pred

# Function to calculate accuracy

def cal\_accuracy(y\_test, y\_pred):

print("Confusion Matrix: ",

confusion\_matrix(y\_test, y\_pred))

print ("Accuracy : ",

accuracy\_score(y\_test,y\_pred)\*100)

print("Report : ",

classification\_report(y\_test, y\_pred))

# Driver code

def main():

data = importdata()

X, Y, X\_train, X\_test, y\_train, y\_test = splitdataset(data)

clf\_gini = train\_using\_gini(X\_train, X\_test, y\_train)

clf\_entropy = tarin\_using\_entropy(X\_train, X\_test, y\_train)

print("Results:")

error=0

# Prediction using gini

y\_pred\_gini = prediction(X\_test, clf\_gini)

#cal\_accuracy(y\_test, y\_pred\_gini)

#print(y\_test,y\_pred\_gini)

for i in range(0,len(y\_test)):

error+=math.fabs(y\_pred\_gini[i]-y\_test[i])

error=error/len(y\_test)

print("error=",error)

print(y\_pred\_gini,y\_test)

# Calling main function

if \_\_name\_\_=="\_\_main\_\_":

main()

**KNN:**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **matplotlib.colors** **import** [ListedColormap](http://matplotlib.org/api/_as_gen/matplotlib.colors.ListedColormap.html#matplotlib.colors.ListedColormap)

**from** **sklearn** **import** neighbors, datasets

n\_neighbors = 15

*# import some data to play with*

iris = [datasets.load\_iris](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html#sklearn.datasets.load_iris)()

*# we only take the first two features. We could avoid this ugly*

*# slicing by using a two-dim dataset*

X = iris.data[:, :2]

y = iris.target

h = .02 *# step size in the mesh*

*# Create color maps*

cmap\_light = [ListedColormap](http://matplotlib.org/api/_as_gen/matplotlib.colors.ListedColormap.html#matplotlib.colors.ListedColormap)(['#FFAAAA', '#AAFFAA', '#AAAAFF'])

cmap\_bold = [ListedColormap](http://matplotlib.org/api/_as_gen/matplotlib.colors.ListedColormap.html#matplotlib.colors.ListedColormap)(['#FF0000', '#00FF00', '#0000FF'])

**for** weights **in** ['uniform', 'distance']:

*# we create an instance of Neighbours Classifier and fit the data.*

clf = [neighbors.KNeighborsClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier)(n\_neighbors, weights=weights)

clf.fit(X, y)

**>>> from** **sklearn.neighbors** **import** NearestNeighbors

**>>> import** **numpy** **as** **np**

**>>>** X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])

**>>>** nbrs = NearestNeighbors(n\_neighbors=2, algorithm='ball\_tree').fit(X)

**>>>** distances, indices = nbrs.kneighbors(X)

NAÏVE BAYES:

**>> from** **sklearn** **import** datasets

**>>>** iris = datasets.load\_iris()

**>>> from** **sklearn.naive\_bayes** **import** GaussianNB

**>>>** gnb = GaussianNB()

**>>>** y\_pred = gnb.fit(iris.data, iris.target).predict(iris.data)

**>>>** print("Number of mislabeled points out of a total *%d* points : *%d*"

**...**  % (iris.data.shape[0],(iris.target != y\_pred).sum()))

Number of mislabeled points out of a total 150 points : 6

#Import Library of Gaussian Naive Bayes model

from sklearn.naive\_bayes import GaussianNB

import numpy as np

#assigning predictor and target variables

x= np.array([[-3,7],[1,5], [1,2], [-2,0], [2,3], [-4,0], [-1,1], [1,1], [-2,2], [2,7], [-4,1], [-2,7]])

Y = np.array([3, 3, 3, 3, 4, 3, 3, 4, 3, 4, 4, 4])

#Create a Gaussian Classifier

model = GaussianNB()

# Train the model using the training sets

model.fit(x, y)

#Predict Output

predicted= model.predict([[1,2],[3,4]])

print predicted

**Output: ([3,4])**

R Code:

require(e1071) #Holds the Naive Bayes Classifier

Train <- read.csv(file.choose())

Test <- read.csv(file.choose())

#Make sure the target variable is of a two-class classification problem only

levels(Train$Item\_Fat\_Content)

model <- naiveBayes(Item\_Fat\_Content~., data = Train)

class(model)

pred <- predict(model,Test)

table(pred)

RANDOM FOREST:

**>>> from** **sklearn.ensemble** **import** RandomForestClassifier

**>>> from** **sklearn.datasets** **import** make\_classification

>>>

**>>>** X, y = make\_classification(n\_samples=1000, n\_features=4,

**...**  n\_informative=2, n\_redundant=0,

**...**  random\_state=0, shuffle=**False**)

**>>>** clf = RandomForestClassifier(max\_depth=2, random\_state=0)

**>>>** clf.fit(X, y)

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=2, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=10, n\_jobs=1,

oob\_score=False, random\_state=0, verbose=0, warm\_start=False)

**>>>** print(clf.feature\_importances\_)

[ 0.17287856 0.80608704 0.01884792 0.00218648]

**>>>** print(clf.predict([[0, 0, 0, 0]]))

[1]

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

import pdb

# File Paths

INPUT\_PATH = "../inputs/breast-cancer-wisconsin.data"

OUTPUT\_PATH = "../inputs/breast-cancer-wisconsin.csv"

# Headers

HEADERS = ["CodeNumber", "ClumpThickness", "UniformityCellSize", "UniformityCellShape", "MarginalAdhesion",

           "SingleEpithelialCellSize", "BareNuclei", "BlandChromatin", "NormalNucleoli", "Mitoses", "CancerType"]

def read\_data(path):

    """

    Read the data into pandas dataframe

    :param path:

    :return:

    """

    data = pd.read\_csv(path)

    return data

def get\_headers(dataset):

    """

    dataset headers

    :param dataset:

    :return:

    """

    return dataset.columns.values

def add\_headers(dataset, headers):

    """

    Add the headers to the dataset

    :param dataset:

    :param headers:

    :return:

    """

    dataset.columns = headers

    return dataset

def data\_file\_to\_csv():

    """

    :return:

    """

    # Headers

    headers = ["CodeNumber", "ClumpThickness", "UniformityCellSize", "UniformityCellShape", "MarginalAdhesion",

               "SingleEpithelialCellSize", "BareNuclei", "BlandChromatin", "NormalNucleoli", "Mitoses",

               "CancerType"]

    # Load the dataset into Pandas data frame

    dataset = read\_data(INPUT\_PATH)

    # Add the headers to the loaded dataset

    dataset = add\_headers(dataset, headers)

    # Save the loaded dataset into csv format

    dataset.to\_csv(OUTPUT\_PATH, index=False)

    print "File saved ...!"

def split\_dataset(dataset, train\_percentage, feature\_headers, target\_header):

    """

    Split the dataset with train\_percentage

    :param dataset:

    :param train\_percentage:

    :param feature\_headers:

    :param target\_header:

    :return: train\_x, test\_x, train\_y, test\_y

    """

    # Split dataset into train and test dataset

    train\_x, test\_x, train\_y, test\_y = train\_test\_split(dataset[feature\_headers], dataset[target\_header],

                                                        train\_size=train\_percentage)

    return train\_x, test\_x, train\_y, test\_y

def handel\_missing\_values(dataset, missing\_values\_header, missing\_label):

    """

    Filter missing values from the dataset

    :param dataset:

    :param missing\_values\_header:

    :param missing\_label:

    :return:

    """

    return dataset[dataset[missing\_values\_header] != missing\_label]

def random\_forest\_classifier(features, target):

    """

    To train the random forest classifier with features and target data

    :param features:

    :param target:

    :return: trained random forest classifier

    """

    clf = RandomForestClassifier()

    clf.fit(features, target)

    return clf

def dataset\_statistics(dataset):

    """

    Basic statistics of the dataset

    :param dataset: Pandas dataframe

    :return: None, print the basic statistics of the dataset

    """

    print dataset.describe()

def main():

    """

    Main function

    :return:

    """

    # Load the csv file into pandas dataframe

    dataset = pd.read\_csv(OUTPUT\_PATH)

    # Get basic statistics of the loaded dataset

    dataset\_statistics(dataset)

    # Filter missing values

    dataset = handel\_missing\_values(dataset, HEADERS[6], '?')

    train\_x, test\_x, train\_y, test\_y = split\_dataset(dataset, 0.7, HEADERS[1:-1], HEADERS[-1])

    # Train and Test dataset size details

    print "Train\_x Shape :: ", train\_x.shape

    print "Train\_y Shape :: ", train\_y.shape

    print "Test\_x Shape :: ", test\_x.shape

    print "Test\_y Shape :: ", test\_y.shape

    # Create random forest classifier instance

    trained\_model = random\_forest\_classifier(train\_x, train\_y)

    print "Trained model :: ", trained\_model

    predictions = trained\_model.predict(test\_x)

    for i in xrange(0, 5):

        print "Actual outcome :: {} and Predicted outcome :: {}".format(list(test\_y)[i], predictions[i])

    print "Train Accuracy :: ", accuracy\_score(train\_y, trained\_model.predict(train\_x))

    print "Test Accuracy  :: ", accuracy\_score(test\_y, predictions)

    print " Confusion matrix ", confusion\_matrix(test\_y, predictions)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**EXPERIMENT 1:**

> weather <- read.csv(file="C:/Users/Lenovo/Desktop/weather.csv",head=TRUE,sep=",")

> min=weather$MIN

> rain=weather$RAIN

> mean(max)

[1] 493.1863

> mean(min)

[1] 282.0356

> mean(rain)

[1] 87.59178

> median(max)

[1] 531

> median(min)

[1] 307

> median(rain)

[1] 56

> var(max)

[1] 56612.27

> var(min)

[1] 45797.52

> var(rain)

[1] 6718.11

> sd(max)

[1] 237.9333

> sd(min)

[1] 214.0036

> sd(rain)

[1] 81.96408

> cor(max,min,method='pearson')

[1] 0.9976028

> summary(weather)

STATION STATION\_NAME ELEVATION

GHCND:USC00327027:365 PETERSBURG 2 N ND US:365 Min. :466.3

1st Qu.:466.3

Median :466.3

Mean :466.3

3rd Qu.:466.3

Max. :466.3

LATITUDE LONGITUDE DATE MIN

Min. :48.04 Min. :-98.01 Min. :20100101 Min. :-48

1st Qu.:48.04 1st Qu.:-98.01 1st Qu.:20100402 1st Qu.: 73

Median :48.04 Median :-98.01 Median :20100702 Median :307

Mean :48.04 Mean :-98.01 Mean :20100668 Mean :282

3rd Qu.:48.04 3rd Qu.:-98.01 3rd Qu.:20101001 3rd Qu.:489

Max. :48.04 Max. :-98.01 Max. :20101231 Max. :567

MAX RAIN

Min. :138.0 Min. : 1.00

1st Qu.:247.0 1st Qu.: 27.00

Median :531.0 Median : 56.00

Mean :493.2 Mean : 87.59

3rd Qu.:725.0 3rd Qu.:131.00

Max. :805.0 Max. :376.00

> cor(max,min,method='kendall')

[1] 0.9615269

> cor(max,min,method='spearman')

[1] 0.9959973

> 100\*sd(max)/mean(max)

[1] 48.24411

> 100\*sd(min)/mean(min)

[1] 75.8782

> 100\*sd(rain)/mean(rain)

[1] 93.57508

> IQR(max)

[1] 478

> IQR(min)

[1] 416

> IQR(rain)

[1] 104

> max(max,na.rm=FALSE)

[1] 805

> max(min,na.rm=FALSE)

[1] 567

> max(rain,na.rm=FALSE)

[1] 376

> min(max,na.rm=FALSE)

[1] 138

> min(min,na.rm=FALSE)

[1] -48

> min(rain,na.rm=FALSE)

[1] 1

> range(max)

[1] 138 805

> range(min)

[1] -48 567

> range(rain)

[1] 1 376

> modefunc <- function(v) {

+ uniqv <- unique(v)

+ uniqv[which.max(tabulate(match(v, uniqv)))]

+ }

>

> modefunc(max)

[1] 139

> modefunc(min)

[1] 566

> modefunc(rain)

[1] 12

> sum(max)

[1] 180013

> sum(min)

[1] 102943

> sum(rain)

[1] 31971

> quantile(max)

0% 25% 50% 75% 100%

138 247 531 725 805

> quantile(min)

0% 25% 50% 75% 100%

-48 73 307 489 567

> quantile(rain)

0% 25% 50% 75% 100%

1 27 56 131 376

EXPERIMENT 2

**DATA**

> grade <- read.csv(file="C:/Users/Lenovo/Desktop/class-grades.csv",head=TRUE,sep=",")

**FUNCTIONS**

> length(grade)

[1] 6

PRINT ROWS THAT DON’T HAVE MISSING VALUES:

> grade[!is.na(grade)]

[1] 5.00 8.00 8.00 7.00 8.00 7.00 8.00 7.00 8.00 7.00

[11] 7.00 6.00 8.00 7.00 8.00 7.00 8.00 4.00 8.00 8.00

[21] 8.00 8.00 8.00 7.00 7.00 6.00 8.00 8.00 8.00 7.00

[31] 7.00 7.00 8.00 7.00 8.00 5.00 6.00 7.00 6.00 7.00

**REPLACE MISSING VALUES WITH ZERO:**

> grade[is.na(grade)]=0

> grade

Prefix Assignment Tutorial Midterm TakeHome Final

1 5 57.14 34.09 64.38 51.48 52.50

2 8 95.05 105.49 67.50 99.07 68.33

3 8 83.70 83.17 30.00 63.15 48.89

4 7 81.22 96.06 49.38 105.93 0.00

5 8 91.32 93.64 95.00 107.41 73.89

**NOW CHECK FOR ANY MISSING VALUES BY PRINTINGTOTAL NUMBER OF MISSING VALUES**

> sum(is.na(grade))

[1] 0

RE OPEN FILE TO DO ANOTHER PROCESSING AS NA VALUES ARE REPLACED WITH 0 ALREADY

> grade <- read.csv(file="C:/Users/Lenovo/Desktop/class-grades.csv",head=TRUE,sep=",")

SUM OF MISSING VALUE

> sum(is.na(grade))

[1] 27

**PRINT TRUE FOR MISSING VALUE IN DATA.ELSE FALSE. CHECK WHERE MISSING VALUES ARE IN DATA.**

> is.na(grade)

Prefix Assignment Tutorial Midterm TakeHome Final

[1,] FALSE FALSE FALSE FALSE FALSE FALSE

[2,] FALSE FALSE FALSE FALSE FALSE FALSE

[3,] FALSE FALSE FALSE FALSE FALSE FALSE

[4,] FALSE FALSE FALSE FALSE FALSE TRUE

[5,] FALSE FALSE FALSE FALSE FALSE FALSE

[6,] FALSE FALSE FALSE FALSE FALSE FALSE

**CALCULATE MEAN BY REMOVING AND INCLUDING NA VALUES**

> mean(final,na.rm=TRUE)

[1] 68.28653

> mean(final,na.rm=FALSE)

[1] NA

**PRINT ROW WITH MISSING VALUES**

> grade[!complete.cases(grade),]

Prefix Assignment Tutorial Midterm TakeHome Final

4 7 81.22 96.06 49.38 105.93 NA

8 7 72.85 86.85 60.00 NA 56.11

13 8 NA 103.71 72.50 93.52 63.33

15 8 84.80 89.08 NA 16.91 35.83

18 4 92.01 NA 38.75 86.11 49.17

**PRINT ROW THAT DON’T HAVE ANY MISSING VALUES**

> grade[complete.cases(grade),]

Prefix Assignment Tutorial Midterm TakeHome Final

1 5 57.14 34.09 64.38 51.48 52.50

2 8 95.05 105.49 67.50 99.07 68.33

3 8 83.70 83.17 30.00 63.15 48.89

5 8 91.32 93.64 95.00 107.41 73.89

6 7 95.00 92.58 93.12 97.78 68.06

7 8 95.05 102.99 56.25 99.07 50.00

**REPLACING NA VALUES WITH COLUMN MEANS**

> for(i in 1:ncol(grade)){

+ grade[is.na(grade[,i]), i] <- mean(grade[,i], na.rm = TRUE)

+ }

> grade

**EXPT 3:**

EXPERIMENT 3

> plotdata=c(weather$RAIN)

> barplot(plotdata,main="Rainfall",ylab="Rainfall",names.arg=1:365,legend=rownames(plotdata))

> plotdata=c(weather$RAIN[30:50])

> barplot(plotdata,xlab="days 30 to 50",ylab="Rainfall in cm",col="red",main="Rainfall for days 30 to 50-M.S.Sanjay 15bce0517",names.arg=30:50)

> data=weather$RAIN

> x=c(sum(data[1:31]),sum(data[32:59]),sum(data[60:90]),sum(data[91:120]),sum(data[121:151]),sum(data[152:181]),sum(data[182:212]),sum(data[213:243]),sum(data[244:273]),sum(data[274:304]),sum(data[305:335]),sum(data[336:365]))

> x

[1] 897 588 1114 1296 3657 5636 5508 4431 3323 2969 1321 1231

> pie(x,c('january','february','march','april','may','june','july','august','september','october','november','december'),main="monthly rainfall")

> plot(c(1,2,3,4,5,6,7,8,9,10,11,12),x,main="monthly rainfall -m.s.sanjay 15bce0517",xlab="months",ylab="rainfall in cm")

> boxplot(weather$MAX,weather$MIN,weather$RAIN,xlab="max temp, min temp, rainfall",main="box plot of max temp,min temp and rainfall-m.s.sanjay 15bce0517")

> hist(x,xlab="rainfall classes of range 1000",main="histogram of rainfall-m.s.sanjay 15bce0517")

> plot(x,type="o",col="blue",xlab="months",ylab="rainfall in cm",main="line graph-m.s.sanjay 15bce0517")

**APRIORI**

> library(arules)

> library(datasets)

> data(Groceries)

> rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))

Apriori

Parameter specification:

confidence minval smax arem aval originalSupport maxtime support minlen maxlen target ext

0.8 0.1 1 none FALSE TRUE 5 0.001 1 10 rules FALSE

Algorithmic control:

filter tree heap memopt load sort verbose

0.1 TRUE TRUE FALSE TRUE 2 TRUE

Absolute minimum support count: 9

set item appearances ...[0 item(s)] done [0.00s].

set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].

sorting and recoding items ... [157 item(s)] done [0.00s].

creating transaction tree ... done [0.01s].

checking subsets of size 1 2 3 4 5 6 done [0.02s].

writing ... [410 rule(s)] done [0.00s].

creating S4 object ... done [0.00s].> rules<-sort(rules, by="confidence", decreasing=TRUE)

**PRINT FIRST TEN RULES**

> inspect(rules[1:10])

lhs rhs support confidence lift count

[1] {rice,

sugar} => {whole milk} 0.001220132 1 3.913649 12

[2] {canned fish,

hygiene articles} => {whole milk} 0.001118454 1 3.913649 11

[3] {root vegetables,

butter,

rice} => {whole milk} 0.001016777 1 3.913649 10

[4] {root vegetables,

whipped/sour cream,

flour} => {whole milk} 0.001728521 1 3.913649 17

[5] {butter,

soft cheese,

domestic eggs} => {whole milk} 0.001016777 1 3.913649 10

[6] {citrus fruit,

root vegetables,

soft cheese} => {other vegetables} 0.001016777 1 5.168156 10

[7] {pip fruit,

butter,

hygiene articles} => {whole milk} 0.001016777 1 3.913649 10

[8] {root vegetables,

whipped/sour cream,

hygiene articles} => {whole milk} 0.001016777 1 3.913649 10

[9] {pip fruit,

root vegetables,

hygiene articles} => {whole milk} 0.001016777 1 3.913649 10

[10] {cream cheese ,

domestic eggs,

sugar} => {whole milk} 0.001118454 1 3.913649 11

**TOTAL NUMBER FO RULES**

> length(rules)

[1] 410

15BCE0517

M.S.SANJAY

L3+L4

EXPERIMENT 7

DOCUMENT SIMILARITY TEHNIQUES AND MEASUREMENTS

>library(“stringr”)

>library(“teext2vec”)

> data("movie\_review")

>

> movie\_review = movie\_review[1:500, ]

>

> prep\_fun = function(x) {

+ x %>%

+ # make text lower case

+ str\_to\_lower %>%

+ # remove non-alphanumeric symbols

+ str\_replace\_all("[^[:alnum:]]", " ") %>%

+ # collapse multiple spaces

+ str\_replace\_all("\\s+", " ")

+ }

> movie\_review$review\_clean = prep\_fun(movie\_review$review)

>

> doc\_set\_1 = movie\_review[1:300, ]

> it1 = itoken(doc\_set\_1$review\_clean, progressbar = FALSE)

>

> doc\_set\_2 = movie\_review[301:500, ]

>

> it2 = itoken(doc\_set\_2$review\_clean, progressbar = FALSE)

> it = itoken(movie\_review$review\_clean, progressbar = FALSE)

>

> v = create\_vocabulary(it) %>% prune\_vocabulary(doc\_proportion\_max = 0.1, term\_count\_min = 5)

>

> vectorizer = vocab\_vectorizer(v)

**JACCARD SIMILARITY**

> dtm1 = create\_dtm(it1, vectorizer)

>

> dim(dtm1)

[1] 300 2338

>

>

> dtm2 = create\_dtm(it2, vectorizer)

>

> dim(dtm2)

[1] 200 2338

> dim(dtm2)

[1] 200 2338

> d1\_d2\_jac\_sim = sim2(dtm1, dtm2, method = "jaccard", norm = "none")

> dim(d1\_d2\_jac\_sim)

[1] 300 200

> d1\_d2\_jac\_sim[1:2, 1:5]

2 x 5 sparse Matrix of class "dgCMatrix"

1 2 3 4 5

1 0.02142857 . 0.02362205 0.007575758 0.02597403

2 0.01219512 . 0.02941176 0.013888889 0.02083333

> dtm1\_2 = dtm1[1:200, ]

>

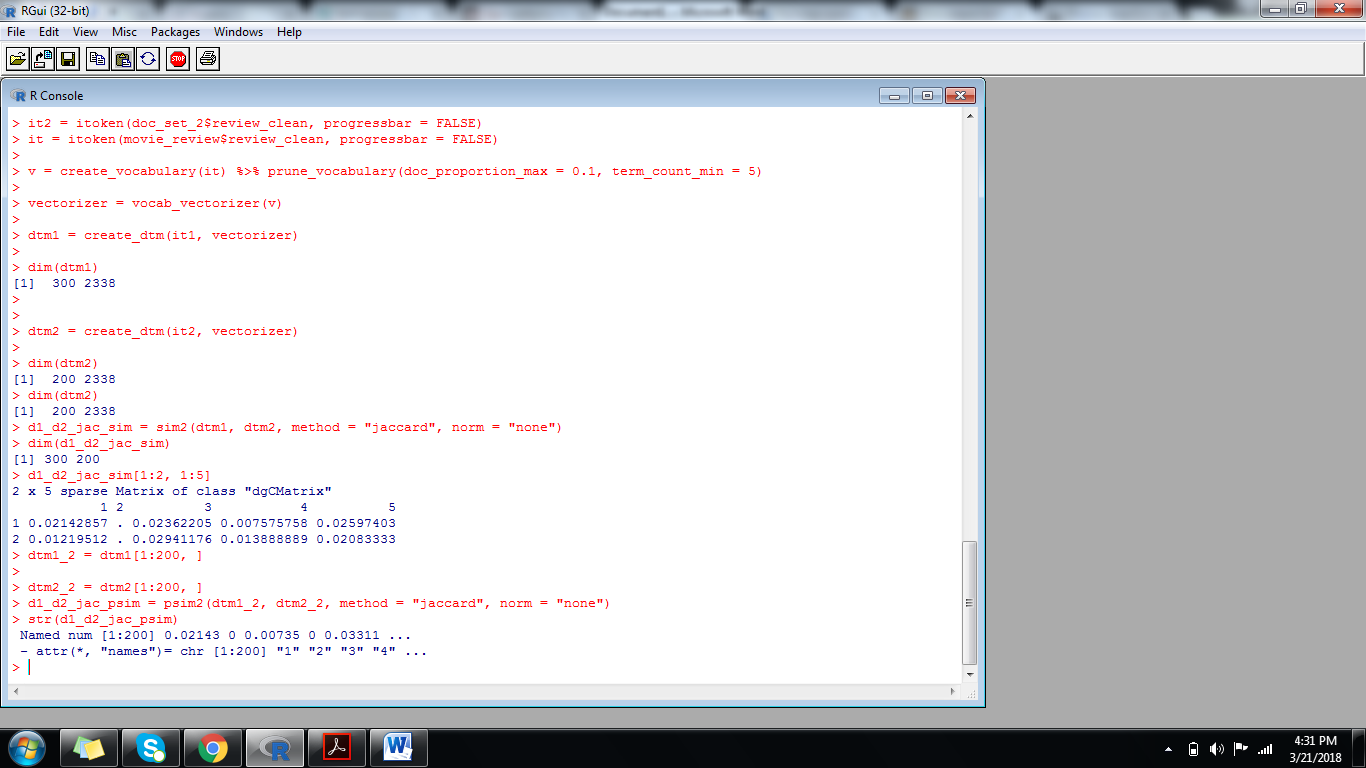
> dtm2\_2 = dtm2[1:200, ]

> d1\_d2\_jac\_psim = psim2(dtm1\_2, dtm2\_2, method = "jaccard", norm = "none")

> str(d1\_d2\_jac\_psim)

Named num [1:200] 0.02143 0 0.00735 0 0.03311 ...

- attr(\*, "names")= chr [1:200] "1" "2" "3" "4" ...



**COSINE SIMILARITY**

> d1\_d2\_cos\_sim = sim2(dtm1, dtm2, method = "cosine", norm = "l2")

> dim(d1\_d2\_cos\_sim)

[1] 300 200

> d1\_d2\_cos\_sim[1:2, 1:5]

2 x 5 sparse Matrix of class "dgCMatrix"

1 2 3 4 5

1 0.02703999 . 0.05063299 0.009500143 0.02753954

2 0.02455143 . 0.06567587 0.034503278 0.04000800

**COSINE SIMILARITY WITHTF-IDF(tern frequency invertd document frequency)**

> dtm = create\_dtm(it, vectorizer)

>

> tfidf = TfIdf$new()

> dtm\_tfidf = fit\_transform(dtm, tfidf)

> d1\_d2\_tfidf\_cos\_sim = sim2(x = dtm\_tfidf, method = "cosine", norm = "l2")

> d1\_d2\_tfidf\_cos\_sim[1:2, 1:5]

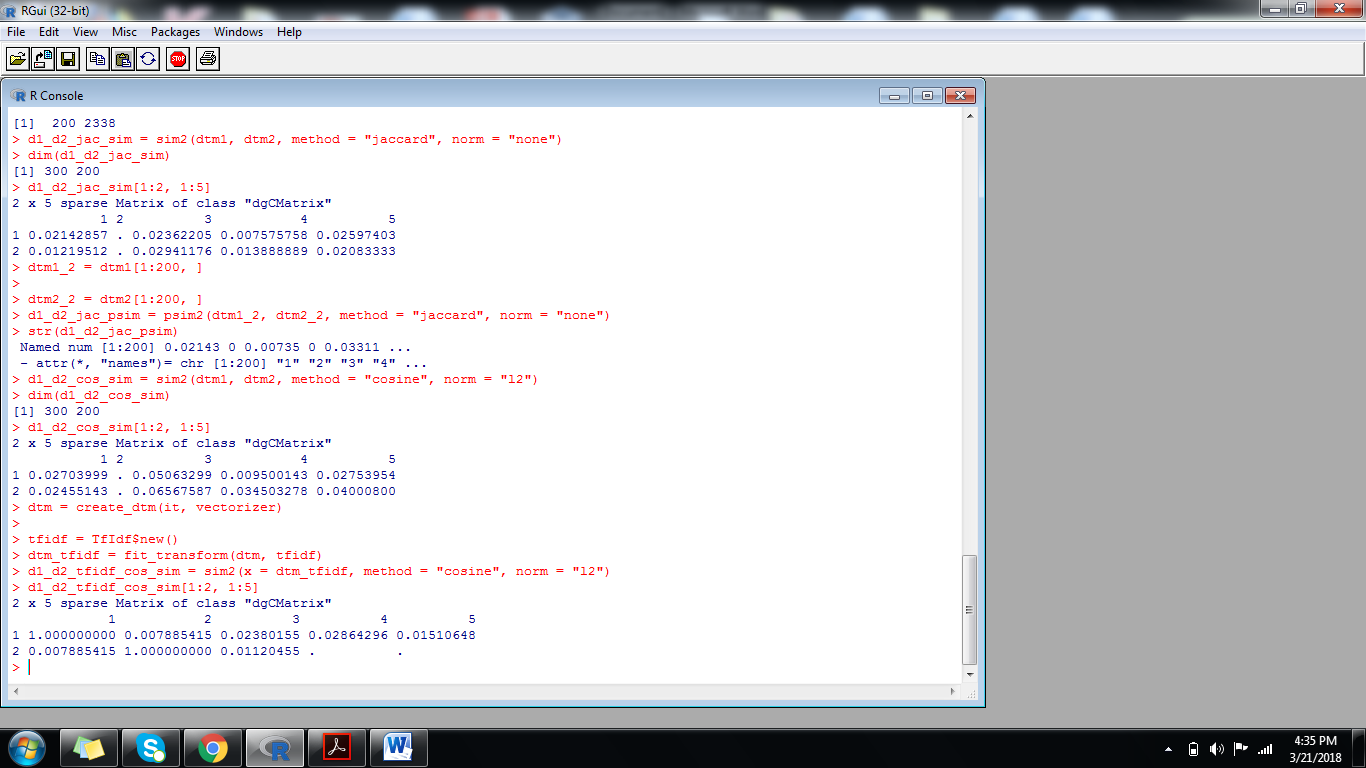
2 x 5 sparse Matrix of class "dgCMatrix"

1 2 3 4 5

1 1.000000000 0.007885415 0.02380155 0.02864296 0.01510648

2 0.007885415 1.000000000 0.01120455 . .

>.



**Cosine similarity with lsa:**

> lsa = LSA$new(n\_topics = 100)

> dtm\_tfidf\_lsa = fit\_transform(dtm\_tfidf, lsa)

> d1\_d2\_tfidf\_cos\_sim = sim2(x = dtm\_tfidf\_lsa, method = "cosine", norm = "l2")

> d1\_d2\_tfidf\_cos\_sim[1:2, 1:5]

1 2 3 4 5

1 1.0000000 0.1131024 0.2441907 0.22305870 0.27371567

2 0.1131024 1.0000000 0.1196754 -0.02698779 -0.04750231

> x = dtm\_tfidf\_lsa[1:250, ]

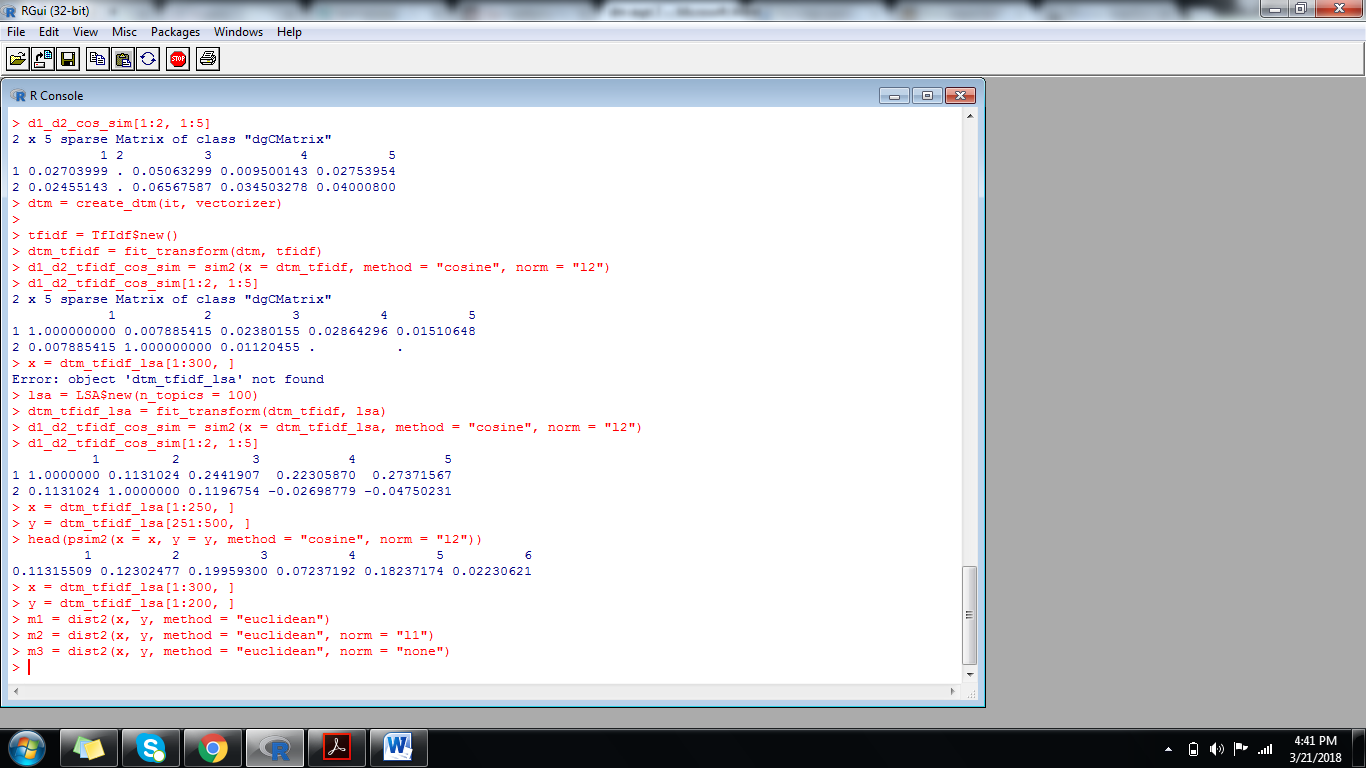
> y = dtm\_tfidf\_lsa[251:500, ]

> head(psim2(x = x, y = y, method = "cosine", norm = "l2"))

1 2 3 4 5 6

0.11315509 0.12302477 0.19959300 0.07237192 0.18237174 0.02230621

>



15BCE0517

M.S.SANJAY

L3+L4

DATA MINING LAB

EXPERIMENT 8

**SIMULATION OF PAGERANK ALGORITHM**

>library(igraph)

> g <- graph(c(

+ 1, 2, 1, 3, 1, 4,

+ 2, 3, 2, 6, 3, 1,

+ 3, 5, 4, 2, 4, 1,

+ 4, 5, 5, 2, 5, 6,

+ 6, 3, 6, 4),

+ directed=TRUE)

> M = get.adjacency(g, sparse = FALSE)

> M = t(M / rowSums(M))

> n = nrow(M)

> U = matrix(data=rep(1/n, n^2), nrow=n, ncol=n)

> beta=0.85

> A = beta\*M+(1-beta)\*U

> e = eigen(A)

> v <- e$vec[,1]

> v <- as.numeric(v) / sum(as.numeric(v))

> v

[1] 0.1547936 0.1739920 0.2128167 0.1388701 0.1547936 0.1647339

> page.rank(g)$vector

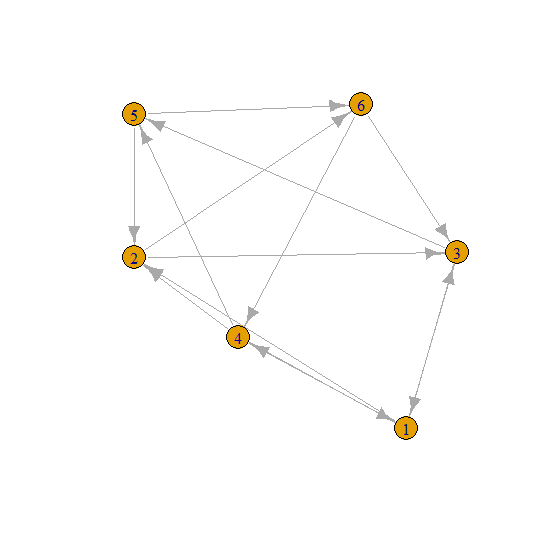
[1] 0.1547936 0.1739920 0.2128167 0.1388701 0.1547936 0.1647339

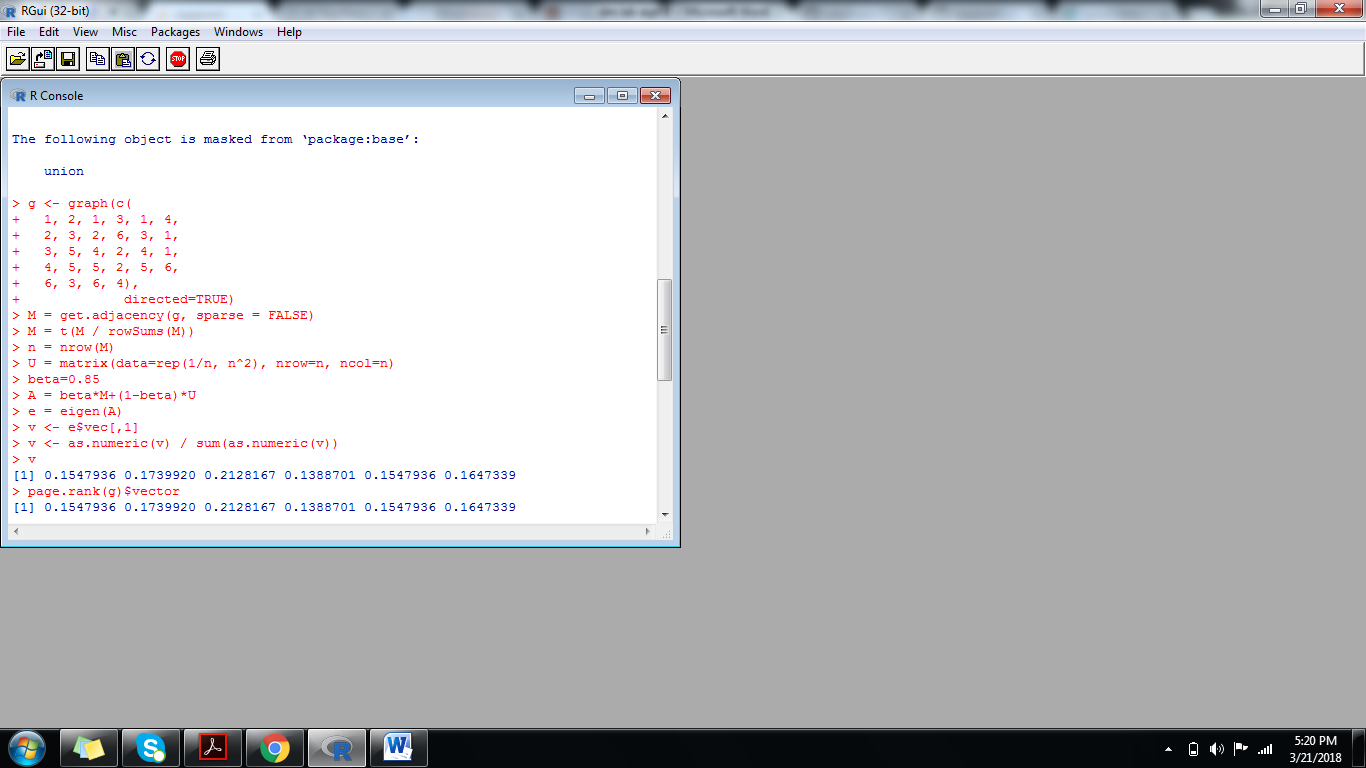
> page.rank(g)$value

[1] 1

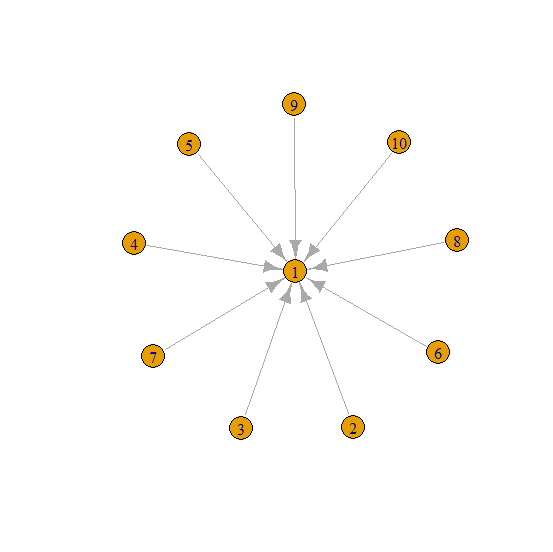
> plot(g)

>





**DEMONSTRATION OF HUBS AND AUTHORITIES:**



> g <- make\_star(10)

> hub\_score(g)$vector

[1] 3.3697e-16 1.0000e+00 1.0000e+00 1.0000e+00 1.0000e+00 1.0000e+00

[7] 1.0000e+00 1.0000e+00 1.0000e+00 1.0000e+00

> authority\_score(g)$vector

[1] 1.000000e+00 2.493665e-18 2.493665e-18 2.493665e-18 2.493665e-18

[6] 2.493665e-18 2.493665e-18 2.493665e-18 2.493665e-18 2.493665e-18

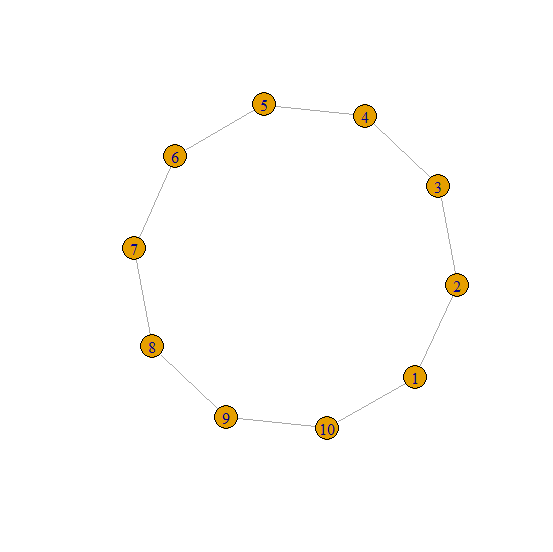
> plot(g)

> authority\_score(g)$value

[1] 9

> hub\_score(g)$value

[1] 9



> g2 <- make\_ring(10)

> hub\_score(g2)$vector

[1] 1 1 1 1 1 1 1 1 1 1

> authority\_score(g2)$vector

[1] 1 1 1 1 1 1 1 1 1 1

> plot(g2)

> authority\_score(g2)$value

[1] 4

> hub\_score(g2)$value

[1] 4

