INTRODUCTION TO R :

R is a programming language and software environment for statistical analysis, graphics representation and reporting. R was created by Ross Ihaka and Robert Gentleman at the university of Auckland, New Zealand, and is currently developed by the R Development core Team. R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.

The various features of R are :

→ R is a well-developed, simple and effective programming language which includes conditionals loops, user defined recursive functions and input and output facilities

→ R has an effective data handling and storage facility

→ R provides a suite of operators for calculations on arrays, lists, vectors and matrices

→ R provides a large, coherent and integrated collection of tools for data analysis

→ R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

R is the best choice of data scientists and

supported by a vibrant community of contributors

R-FUNCTIONS:

A function is a set of statements organised together to perform a specific task. R has a large number of inbuilt functions and the user can create their own functions. In R, a function is an object so the R interpreter is able to pass control to the function, along with arguements that may be necessary for the function to accomplish the actions

An R function is created by using the keyword function. The basic syntax of an R function definition is as follows.

```
function_name ← function (arg1, arg2, ...) {
        function body
}
```

The different parts of a function are:

FUNCTION NAME:

This is the actual name of function

ARGUEMENT:

An arguement is a placeholder and is optional

FUNCTION BODY:

It contains collection of statements that defines

what the function does.

→ RETURN VALUE :

It is the last expression in the function body to be evaluated

R BUILT-IN FUNCTIONS

The functions which are already created or defined in the programming framework are known as built-in functions. These built-in functions are divided into the following based on their functionality.

MATH FUNCTIONS :

R provides various mathematical functions to perform the mathematical calculation.

FUNCTION	DESCRIPTION	EXAMPLE
abs(x)	It returns absolute value of input x	x ← -4 print(abs(x)) output : [1] 4
sqrt(x)	It returns the square root of input x	x ← 4 print (sqrt(x)) output [1] 2
trunc(x)	It returns the truncate value of input x	x ← c(1.2, 2.5, 8.1) print (trunc(x)) output : [1] 128

round (x, digits = n)	It returns round value of input x	x ← -4.8 print (round (x,1)) output : [1] 5
exp(x)	It returns exponent	x ← 4 print (exp (x)) output : [1] 54.598

STRING FUNCTION :

String functions allow us to extract substring from string, search pattern, etc

FUNCTION	DESCRIPTION	EXAMPLE
substr (x, start = n1, stop = n2)	It is used to extract substring in a character vector	a ← "98765432" substr (a, 3, 3) output : [1] "3"
paste (..., sep = " ")	It concatenates string after using sep string to separate them	paste ('one', 2, 'three', 4, 'five') output : [1] one 2 three 4 five
	It splits the elements of character vector x at split point	a ← "split all the character" print (strsplit (a, " ")) output : [[1]] [1] "split" "all" "the" "character"

STATISTICAL PROBABILITY FUNCTIONS:

These are helpful to find normal density, normal quantile, etc.

FUNCTION	DESCRIPTION	EXAMPLE
dnorm (x, m=0, sd=1, log=False)	It is used to find height of probability distribution at each point to a given mean and standard deviation	a ← seq (-7, 7, by=0.1) b ← dnorm (a, mean=2.5 sd=0.5) png (file = "dnorm. png") plot (a, b) dev.off ()
rpois (n, lamba)	It is used to generate random numbers from poisson distribution	rpois (10, 10) [1] 6 10 11 3 10 7 7 8 14 12

TERMWORK - 1

PROBLEM DEFINITION:

Predict the price of a house by applying linear regression to using suitable dataset of real estate business

OBJECTIVES

- To study and analyze BostonHousing Dataset for application and apply modelling technique
- To understand linear regression algorithm.

THEORY :

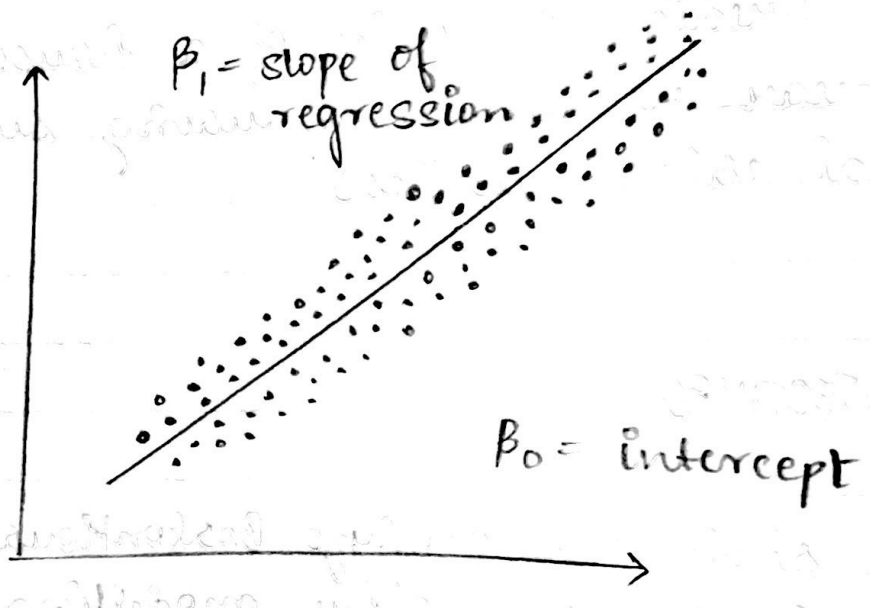
. Naive bayes algorithm is a classification technique based on Bayes Theorem with an assumption of independence among predictors. In simple terms, a naive bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature

For example, a fruit may be considered to be an apple if it is red, round and 3 inches in diameter

Naive Bayes model is easy to build and particularly useful for very large datasets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods

Bayes theorem provides a way of calculation of the posterior probability $P(c/x)$ from $P(x)$, $P(c)$ and $P(x/c)$

$$P(c/x) = \frac{P(x/c)\,P(c)}{P(x)}$$



LINEAR REGRESSION EXPRESSED AS A LINE IN

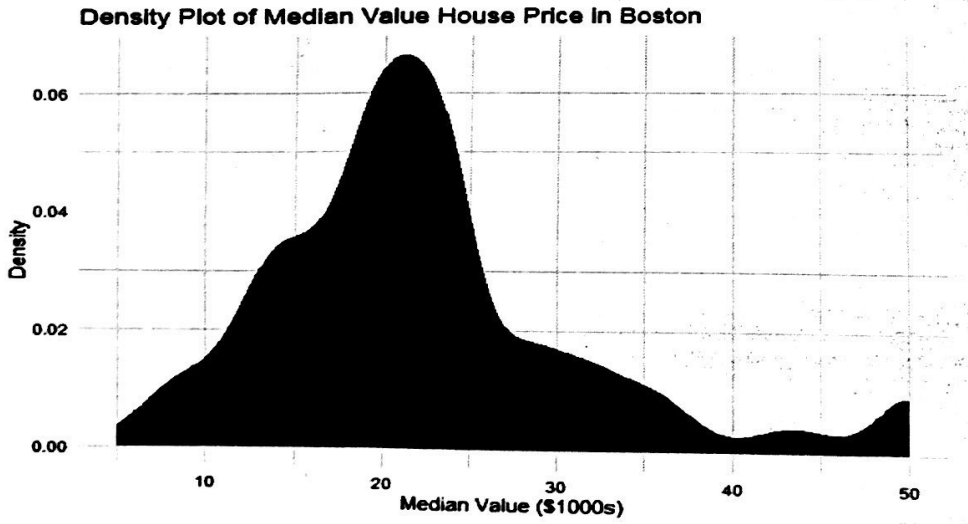
PROGRAM:

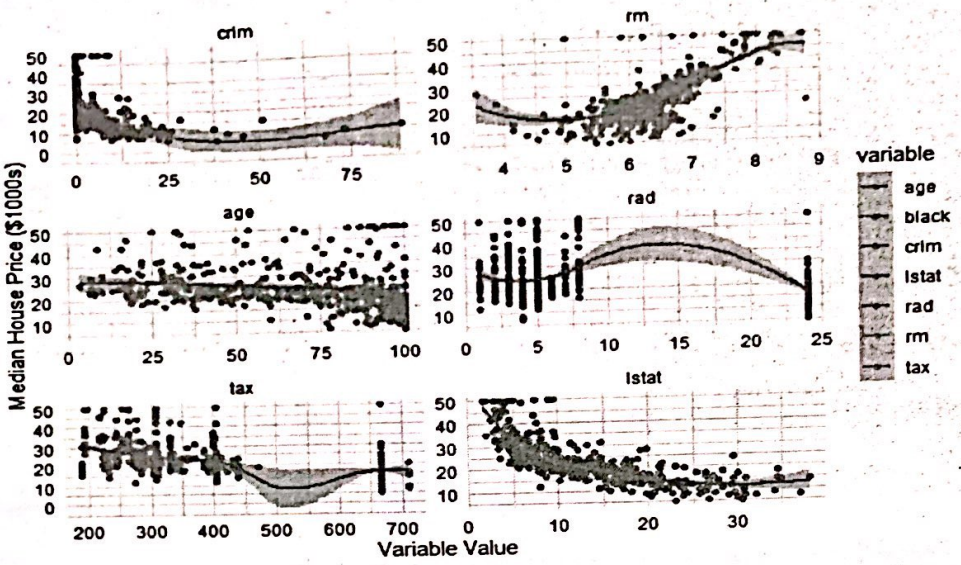
```
#load BostonHousing Data
library(mlbench)
library(dplyr)
library(ggplot2)
library(reshape2)
data("BostonHousing")
housing ← BostonHousing
str(housing)

# ggplot
housing %>%
 ggplot(aes(x = medv)) +
 stat_density() +
 labs(x = "Median value ($1000s)", Y = "Density" title =
  "Density Plot of Median value House Price in Boston")
 + theme_minimal()

# summary
summary(housing $medv)

#predicted v/s original
housing %>%
  select(c(crim, rm, age, rad, tax, lstat, medv)) %>%
  melt(id.vars = "medv") %>%
  ggplot(aes(x= value, y= medv, colour = variable)) +
  geom_point(alpha = 0.7) +
  stat_smooth(aes(colour= "black")) +
```

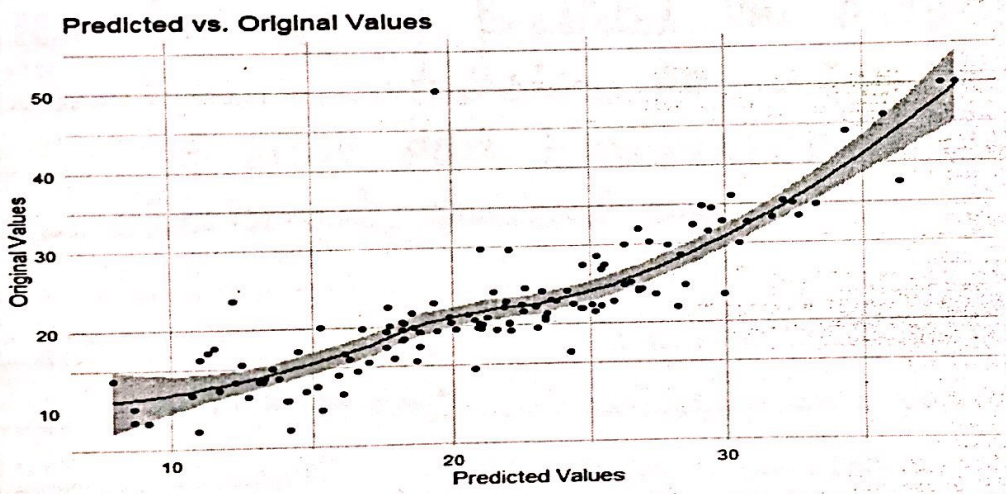
```
facet_wrap(~variable, scales = "free", ncol = 2)+
 labs(x = "Variable Value", y = "Median House Price ($1000s)")
 theme_minimal()

# Set a seed of 123 and split your data into a train
and test set using a 75/25 split. You may find
the caret library helpful here.
 library("caret")
 set.seed(123) # random number generation
 to_train <- createDataPartition(y = housing$medv, p=0.75,
 list = FALSE)
 to_test <- createDataPartition(y= housing$medv, p=0.25,
 list = FALSE)
 train<- housing[to_train,]
 test <- housing[to_test,]

# fit a linear model
 first_lm <- lm(medv ~ crim + rm + tax + lstat, data = train)

 lm1_rsqr <- summary(first_lm1)$r.squared
 print(paste("First linear model has an r-squared
  value of", round(lm1_rsqr, 3), sep = " "))
## [1] "First linear model has an r-squared value
 of 0.672"
 # plot(first_lm)

 second_lm <- lm(log(medv) ~ crim + rm + tax + lstat,
 data = train)
```

```
lm2_rsqu <- summary(second_lm)$r.squared
print(paste("our second linear model has an r-squared value of", round(lm2_rsqu, 3), sep = " "))

abs(mean(second_lm$residuals))

predicted <- predict(second_lm, newdata = test)
results <- data.frame(predicted = mp(predicted), original = test$medv)

results %>%
  ggplot(aes(x = predicted, y = original))+
  geom_point()+
  stat_smooth()+
  labs(x = "Predicted values", y = "original values",
title = "Predicted vs Original values")+
  theme_minimal()
```

```
str(housing)
'data.frame':  506 obs. of  14 variables:
 $ crim    : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
 $ zn      : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
 $ indus   : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
 $ chas    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ nox     : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.5
24 ...
 $ rm      : num  6.58 6.42 7.18 7 7.15 ...
 $ age     : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
 $ dis     : num  4.09 4.97 4.97 6.06 6.06 ...
 $ rad     : num  1 2 2 3 3 3 5 5 5 5 ...
 $ tax     : num  296 242 242 222 222 222 311 311 311 311 ...
 $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
 $ b       : num  397 397 393 395 397 ...
 $ lstat   : num  4.98 9.14 4.03 2.94 5.33 ...
 $ medv    : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
> #ggplot
> housing %>%
+   #The infix operator %>% is not part of base R, but is in fact defined
by the package magrittr (CRAN) and is heavily used by dplyr
+   ggplot(aes(x = medv)) +
+   stat_density() +
+   labs(x = "Median Value ($1000s)", y = "Density", title = "Density Plot
of Median Value House Price in Boston") +
+   theme_minimal()
> #summary
> summary(housing$medv)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   5.00   17.02   21.20   22.53   25.00   50.00
>print(paste("First linear model has an r-squared value of ", round(lm1_rs
qu, 3), sep = ""))
[1] "First linear model has an r-squared value of 0.696"
>print(paste("Our second linear model has an r-squared value of ", round(l
m2_rsqu, 3), sep = ""))
[1] "Our second linear model has an r-squared value of 0.75"
>abs(mean(second_lm$residuals))
[1] 1.588811e-17
```







OUTCOMES :

- Able to study and analyze Boston Housing Dataset for application and apply modelling technique
- Able to understand linear regression algorithm.

CONCLUSION :

Successfully predicted the price of a house by applying linear regression to using Bostonhousing dataset of real state business and analyzed the problem statement successfully

REFERENCES :

Cathy O'Neil, Rachel Schutt "Doing Data Science", O'Reilly Media, Inc.

TERMWORK - 2

PROBLEM STATEMENT:

Apply K-nearest neighbour algorithm to classify and analyze the ionosphere data

OBJECTIVES:

- To study and analyze the ionosphere data
- To understand K-nearest neighbour algorithm.

THEORY :

In pattern recognition, the K-nearest neighbour algorithm is a non-parametric method used for classification and regression. In both cases the input consists of the K- closest training examples in the feature space. The output depends on whether Knn is used for classification or regression

In K-N-N classification, the output is a class membership. An object is classified by a plurality vote of its neighbours, with the object being assigned to the class most common among its K- nearest neighbours

If K=1, then the object is simply assigned to the class of that single nearest neighbours

K-NN is a type of instance-based or lazy learning, where the function is only approximated locally and all computation is deferred until classification

A peculiarity of the K-NN algorithm is that it is sensitive to the local structure of the data

It also produces an arbitrary decision boundary

PROGRAM:

```
install.packages("KernelKnn")
data(ionosphere, package = "KernelKnn")
apply(ionosphere, 2, function(x) length(unique(x)))

ionosphere = ionosphere[-2]
x- scale(ionosphere[,-ncol(ionosphere)])
y = ionosphere[, ncol(ionosphere)]

y = cc1:length(unique(y))[match(ionosphere$class,
sort(unique(ionosphere$class)))]

spl-train = sample(1:length(y), round(length(y)*0.75))

spl-test = setdiff(1:length(y), spl-train)
str(spt-train)
str(spl-test)

acc = function(y-true, preds){
  out = table(y-true, max.col(preds, ties.method='random'))
  acc = sum(diag(out))/sum(out)
  acc
}
library(KernelKnn)
preds-TEST- KernelKnn(x[spl-train,], test_data = x[spl-test,]
y[spl-train], K=5, method='euclidian', weights_function
= null, regression = F, levels = unique(y))
head(preds-TEST)
```

```
preds_TEST_tric = KernelKnn(x[spl_train,], TEST_data=
x[spl_test,], y[spl_train], K=10, method='canberra',
weights_function='tricube', regression=F, Levels=
unique(y))

head(preds_TEST_tric)
norm_Kernel = function(w){
  w = dnorm(w, mean=0, sd=1.0)
  w = w/rowSums(w)
  return(w)
}
preds_TEST_norm = KernelKnn(x[spl_train,], TEST_data=
x[spl_test,], y[spl_train], K=10, method='canberra',
weights_function=norm_Kernel, regression=F,
levels=unique(y))

head(preds_TEST_norm)
fit_CV_pair1 = KernelKnnCV(x, y, K=10, folds=5, methods
='canberra', weights_function='tricube', regression=F,
levels=unique(y), threads=5)

str(fit_CV_pair1)
fit_CV_pair2 = KernelKnnCV(x, y, K=9, folds=5, method=
'canberra', weights_function='epanechnikov', regression
=F levels=unique(y), threads=5)
str(fit_CV_pair2)
acc_pair1 = unlist(lapply(1:length(fit_CV_pair1$preds),
  function(x) acc(y[fit_CV_pair1$folds[[x]]],
    fit_CV_pair1$preds[[x]])))
```

```
acc_pair1
cat('accuracy for params_pair1 is :', mean(acc_pair1) '\n')
acc_pair2 = unlist(lapply(1:length(fit_cv_pair2$preds] function(x) acc(y[fit_cv_pair2$folds[x]]],
        fit_cv_pair2$preds[[x]]]))

acc_pair2
cat('accuracy for params_pair2 is :', mean
        (acc_pair2), '\n')
```

TITLE: K-nearest neighbor algorithm to classify and analyse ionosphere data

```
data(ionosphere, package = 'KernelKnn')
> apply(ionosphere, 2, function(x) length(unique(x)))
  V1   V2   V3   V4   V5   V6   V7   V8   V9  V10  V11
   2    1  219  269  204  259  231  260  244  267  246
 V12  V13  V14  V15  V16  V17  V18  V19  V20  V21  V22
 269  238  266  234  270  254  280  254  266  248  265
 V23  V24  V25  V26  V27  V28  V29  V30  V31  V32  V33
 248  264  256  273  256  281  244  266  243  263  245
 V34 class
 263    2
> ionosphere = ionosphere[, -2]
> X = scale(ionosphere[, -ncol(ionosphere)])
> y = ionosphere[, ncol(ionosphere)]
> y = c(1:length(unique(y)))[ match(ionosphere$class, sort(unique(ionosphere$class))) ]
> spl_train = sample(1:length(y), round(length(y) * 0.75))
> spl_test = setdiff(1:length(y), spl_train)
> str(spl_train)
 int [1:263] 313 106 36 67 90 95 3 316 289 129 ...
> str(spl_test)
 int [1:88] 1 5 9 10 12 23 24 27 39 44 ...
> acc = function (y_true, preds) {
+
+   out = table(y_true, max.col(preds, ties.method = "random"))
+
+   acc = sum(diag(out))/sum(out)
+
+   acc
+ }
> library(KernelKnn)
> preds_TEST = KernelKnn(X[spl_train, ], TEST_data = X[spl_test, ], y[spl_train], k = 5 ,
+                        method = 'euclidean', weights_function = NULL, regression = F,
+                        Levels = unique(y))
> head(preds_TEST)
     class_1 class_2
[1,]     0.2     0.8
[2,]     0.2     0.8
[3,]     0.0     1.0
[4,]     1.0     0.0
[5,]     0.2     0.8
[6,]     0.0     1.0
> preds_TEST_tric = KernelKnn(X[spl_train, ], TEST_data = X[spl_test, ], y[spl_train], k =
10 ,
+                             method = 'canberra', weights_function = 'tricube', regression = F,
```

```
+                     Levels = unique(y))
> head(preds_TEST_tric)
          [,1]     [,2]
[1,] 0.000000e+00 1.0000000
[2,] 0.000000e+00 1.0000000
[3,] 4.892599e-18 1.0000000
[4,] 1.000000e+00 0.0000000
[5,] 4.256383e-01 0.5743617
[6,] 0.000000e+00 1.0000000
> norm_kernel = function(W) {
+
+   W = dnorm(W, mean = 0, sd = 1.0)
+
+   W = W / rowSums(W)
+
+   return(W)
+ }
> preds_TEST_norm = KernelKnn(X[spl_train, ], TEST_data = X[spl_test, ], y[spl_train], k
= 10 ,
+                     method = 'canberra', weights_function = norm_kernel, regression = F,
+                     Levels = unique(y))
> head(preds_TEST_norm)
         [,1]     [,2]
[1,] 0.00000000 1.0000000
[2,] 0.00000000 1.0000000
[3,] 0.07236173 0.9276383
[4,] 1.00000000 0.0000000
[5,] 0.40182415 0.5981758
[6,] 0.00000000 1.0000000
> fit_cv_pair1 = KernelKnnCV(X, y, k = 10 , folds = 5, method = 'canberra',
+
+                     weights_function = 'tricube', regression = F,
+
+                     Levels = unique(y), threads = 5)

cross-validation starts ..
  |                                       |   0%
  |=======================================================|  100%

time to complete : 0.08776522 secs

> str(fit_cv_pair1)
List of 2
 $ preds:List of 5
  ..$ : num [1:71, 1:2] 0 0 0 0 0.283 ...
  ..$ : num [1:70, 1:2] 1.00 9.99e-01 0.00 2.79e-02 1.02e-17 ...
  ..$ : num [1:70, 1:2] 0.198 1 0 0 0 ...
  ..$ : num [1:70, 1:2] 1 0 0 0 0 0 0 0 0 0 ...
  ..$ : num [1:70, 1:2] 1 0 0 1 0 ...
 $ folds:List of 5
```

```
  ..$ fold_1: int [1:71] 228 244 21 250 14 233 15 2 235 43 ...
  ..$ fold_2: int [1:70] 54 58 265 275 65 56 50 61 90 85 ...
  ..$ fold_3: int [1:70] 117 99 284 95 282 299 280 113 129 286 ...
  ..$ fold_4: int [1:70] 177 317 162 142 309 143 304 320 308 168 ...
  ..$ fold_5: int [1:70] 187 348 333 183 343 211 218 346 202 221 ...
> fit_cv_pair2 = KernelKnnCV(X, y, k = 9 , folds = 5,method = 'canberra',
+
+                     weights_function = 'epanechnikov', regression = F,
+
+                     Levels = unique(y), threads = 5)

cross-validation starts ..
  |                                      |  0%
  |======================================================================| 100%

time to complete : 0.07477212 secs

> str(fit_cv_pair2)
List of 2
 $ preds:List of 5
  ..$ : num [1:71, 1:2] 0 0 0 0 0.222 ...
  ..$ : num [1:70, 1:2] 1 0.9981 0 0.0512 0 ...
  ..$ : num [1:70, 1:2] 0.199 1 0 0 0 ...
  ..$ : num [1:70, 1:2] 1 0 0 0 0 0 0 0 0 0 ...
  ..$ : num [1:70, 1:2] 1 0 0 1 0 ...
 $ folds:List of 5
  ..$ fold_1: int [1:71] 228 244 21 250 14 233 15 2 235 43 ...
  ..$ fold_2: int [1:70] 54 58 265 275 65 56 50 61 90 85 ...
  ..$ fold_3: int [1:70] 117 99 284 95 282 299 280 113 129 286 ...
  ..$ fold_4: int [1:70] 177 317 162 142 309 143 304 320 308 168 ...
  ..$ fold_5: int [1:70] 187 348 333 183 343 211 218 346 202 221 ...
> acc_pair1 = unlist(lapply(1:length(fit_cv_pair1$preds),
+
+                     function(x) acc(y[fit_cv_pair1$folds[[x]]],
+
+                                     fit_cv_pair1$preds[[x]])))
> acc_pair1
[1] 0.9154930 0.9142857 0.9142857 0.9285714 0.9571429
> cat('accurcay for params_pair1 is :', mean(acc_pair1), '\n')
accurcay for params_pair1 is : 0.9259557
> acc_pair2 = unlist(lapply(1:length(fit_cv_pair2$preds),
+
+                     function(x) acc(y[fit_cv_pair2$folds[[x]]],
+
+                                     fit_cv_pair2$preds[[x]])))
> acc_pair2
[1] 0.9014085 0.9142857 0.9000000 0.9142857 0.9571429
> cat('accuracy for params_pair2 is :', mean(acc_pair2), '\n')
accuracy for params_pair2 is : 0.9174245
>
```

OUTCOMES:

- Able to study and analyze ionosphere dataset and apply the algorithm successfully
- Able to understand K-nearest neighbours algorithm

CONCLUSION:

Successfully defined in terms of their attributes look at their labels and give unassigned item the majority vote if there is a tie and analyzed the problem statement successfully

REFERENCES:

Cathy O'Neil, Rachel Schutt "Doing Data Science", O'Reilly Media, Inc

TERMWORK - 3

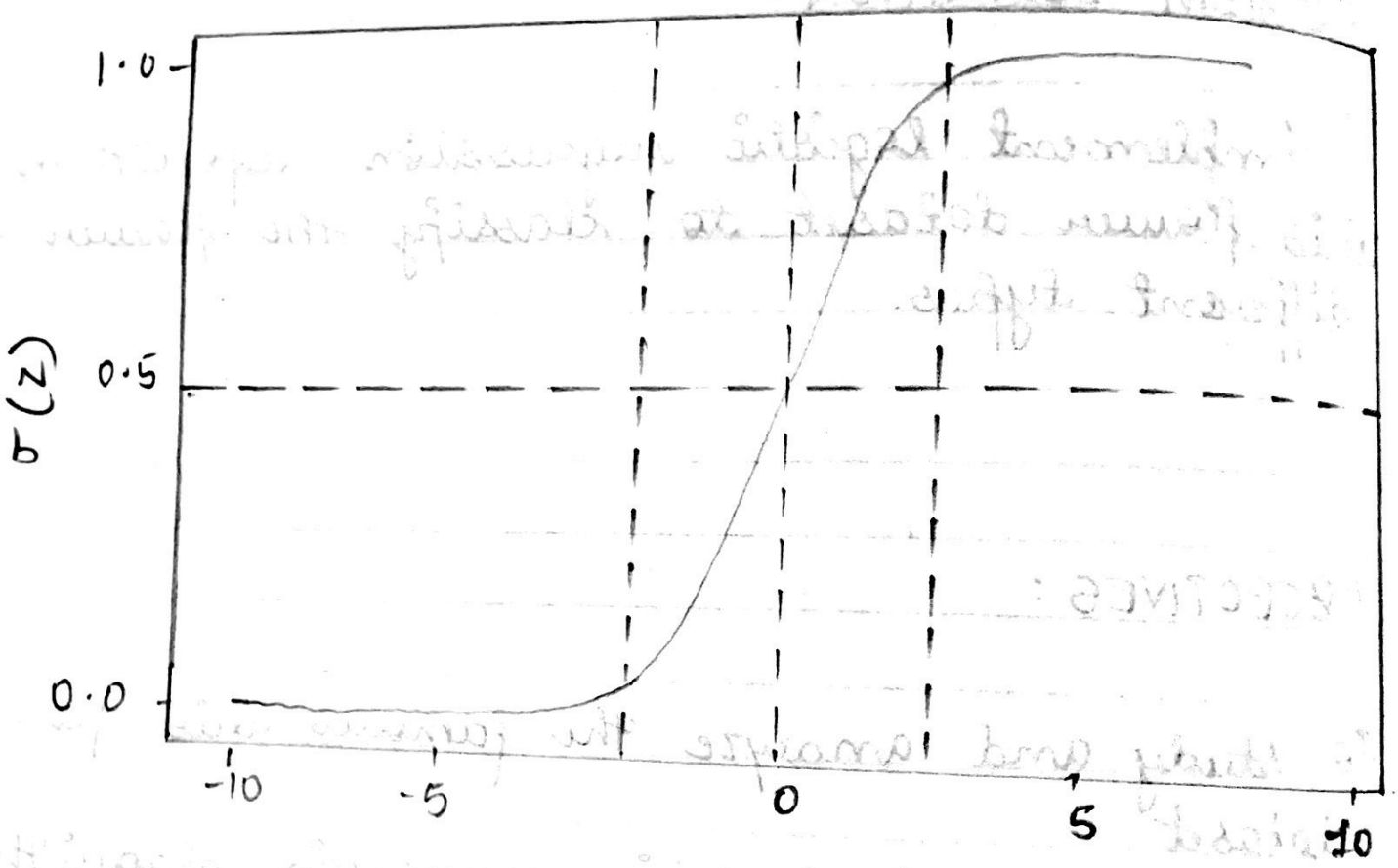
PROBLEM DEFINITION:

Classify the messages as spam and ham using naive bayes algorithm on sms dataset

OBJECTIVES :

- To study and analyze sms dataset
- To understand naive bayes algorithm.

Sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$



$z = \Sigma w_i x_i + \text{bias}$

SIGMOID FUNCTION GRAPH

THEORY:

Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous. Like all regression analysis, the logistic regression is a predictive analysis. It is used to describe data and to explain the relation between one dependent binary variable and one or more nominal, ordinal, interval or ratio level independent variables

Sometimes logistic regression are different to interpret, the Logistic Regression is a classification algorithm used to assign observations to a discrete set of classes unlike linear regression which outputs continuous number value, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to 2 or more discrete classes

In order to map predicted values to probabilities we use the sigmoid function which maps any real value into another value between 0 and 1

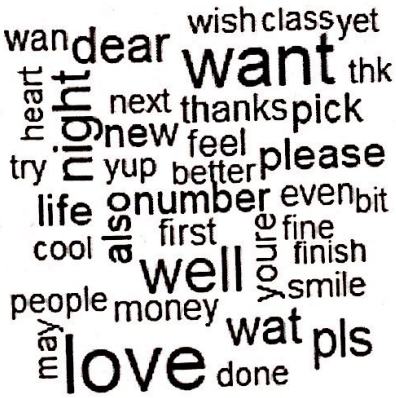
$$S(z) = \frac{1}{1+e^{-z}}$$

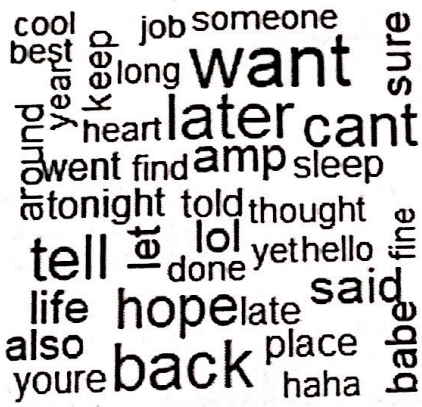
PROGRAM :

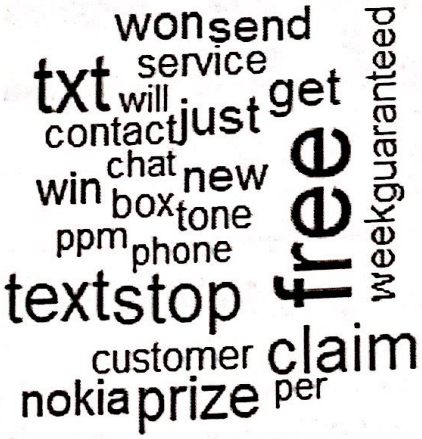
```
install.packages("quanteda")
install.packages("RColorBrewer")
spam = read.csv("spam.csv", header = FALSE, sep=",",
quote = '\n\n', stringsAsFactors = FALSE)
table(spam$V1)
names(spam) <- c("type", "message")
head(spam)
set.seed(2012)
spam <- spam[sample(nrow(spam)),]
library(quanteda)
msg.corpus <- corpus(spam$message)
docvars(msg.corpus) <- spam$type
spam.plot <- corpus_subset(msg.corpus, docvar1 ==
"spam")
spam.plot <- dfm(spam.plot, tolower = TRUE, remove_
punct = TRUE, remove_twitter = TRUE, remove_numbers
= TRUE, remove = stopwords("SMART"))
library(RColorBrewer)
spam.col <- brewer.pal(10, "BrBG")
textplot_wordcloud", col.main = "grey14")
ham.plot <- corpus_subset(msg.corpus, docvar1 == "ham")
ham.plot <- dfm(ham.plot, tolower = TRUE, remove_punct =
TRUE, remove_twitter = TRUE, remove_numbers = TRUE,
remove = c("gt", "lt", stopwords("SMART")))
ham.col = brewer.pal(10, "BrBG")
textplot_wordcloud(ham.plot, min.freq = 50, colors=ham.col
fixed.asp = TRUE)
```

```
title("Ham Wordcloud", col.main= "grey14")

spam.train <- spam[1:4458]
spam.test <- spam[4458 : nrow(spam)]
msg.dfm <- dfm(msg.corpus, tolower = TRUE)
msg.dfm <- dfm_trim(msg.dfm, min.count=5, min_
                                          docfreq

head(msg.dfm)
msg.dfm.train <- msg.df[1:4458]
msg.dfm.test <- msg.dfm[4458 : nrow(spam)]
head(msg.dfm)

nb.classifier <- textmodel_nb(msg.dfm.train, spam.
                                          train[1])

nb.classifier
pred <- predict(nb.classifier, msg.dfm.test)

table(predicted = pred; actual = spam.test[1])
mean(pred == spam.test[,1])*100.
```

```
> library(tm)
> library(wordcloud)
> library(e1071)
> sms_spam_df <- read.csv(file="sms_spam.csv", stringsAsFactors=F)
> sms_corpus <- Corpus(VectorSource(sms_spam_df$text))
> #translate all letters to lower case
> clean_corpus <- tm_map(sms_corpus, content_transformer(tolower))
> # remove numbers
> clean_corpus <- tm_map(clean_corpus, removeNumbers)
> # remove punctuation
> clean_corpus <- tm_map(clean_corpus, removePunctuation)
> stopwords()[1:10]
 [1] "i"          "me"         "my"         "myself"
 [5] "we"         "our"        "ours"       "ourselves"
 [9] "you"        "your"
> clean_corpus <- tm_map(clean_corpus, removeWords, stopwords())
> sms_dtm <- DocumentTermMatrix(clean_corpus)
> spam_indices <- which(sms_spam_df$category == "spam")
> ham_indices <- which(sms_spam_df$category == "ham")
> wordcloud(clean_corpus[ham_indices], min.freq=40)
> wordcloud(clean_corpus[spam_indices], min.freq=40)
> sms_raw_train <- sms_spam_df[1:4169,]
> sms_raw_test <- sms_spam_df[4170:5559,]
> sms_dtm_train <- sms_dtm[1:4169,]
> sms_dtm_test <- sms_dtm[4170:5559,]
> sms_corpus_train <- clean_corpus[1:4169]
> sms_corpus_test <- clean_corpus[4170:5559]
> spam <- subset(sms_raw_train, category == "spam")
> ham <- subset(sms_raw_train, category == "ham")
> five_times_words <- findFreqTerms(sms_dtm_train, 5)
> sms_train <- DocumentTermMatrix(sms_corpus_train, control=list(dictionar
y = five_times_words))
> sms_test <- DocumentTermMatrix(sms_corpus_test, control=list(dictionary
= five_times_words))
> convert_count <- function(x) {
+   y <- ifelse(x > 0, 1,0)
+   y <- factor(y, levels=c(0,1), labels=c("No", "Yes"))
+   y
+ }
> sms_train <- apply(sms_train, 2, convert_count)
> sms_test <- apply(sms_test, 2, convert_count)
> sms_classifier <- naiveBayes(sms_train, factor(sms_raw_train$category))
> sms_test_pred <- predict(sms_classifier, newdata=sms_test)
> k=table(sms_test_pred, sms_raw_test$category)
> k

sms_test_pred  ham spam
         ham  1203   22
         spam    6  159
> accuracy = sum(diag(k))/sum(k)*100
> accuracy
[1] 97.98561
```







TERMWORK - 4

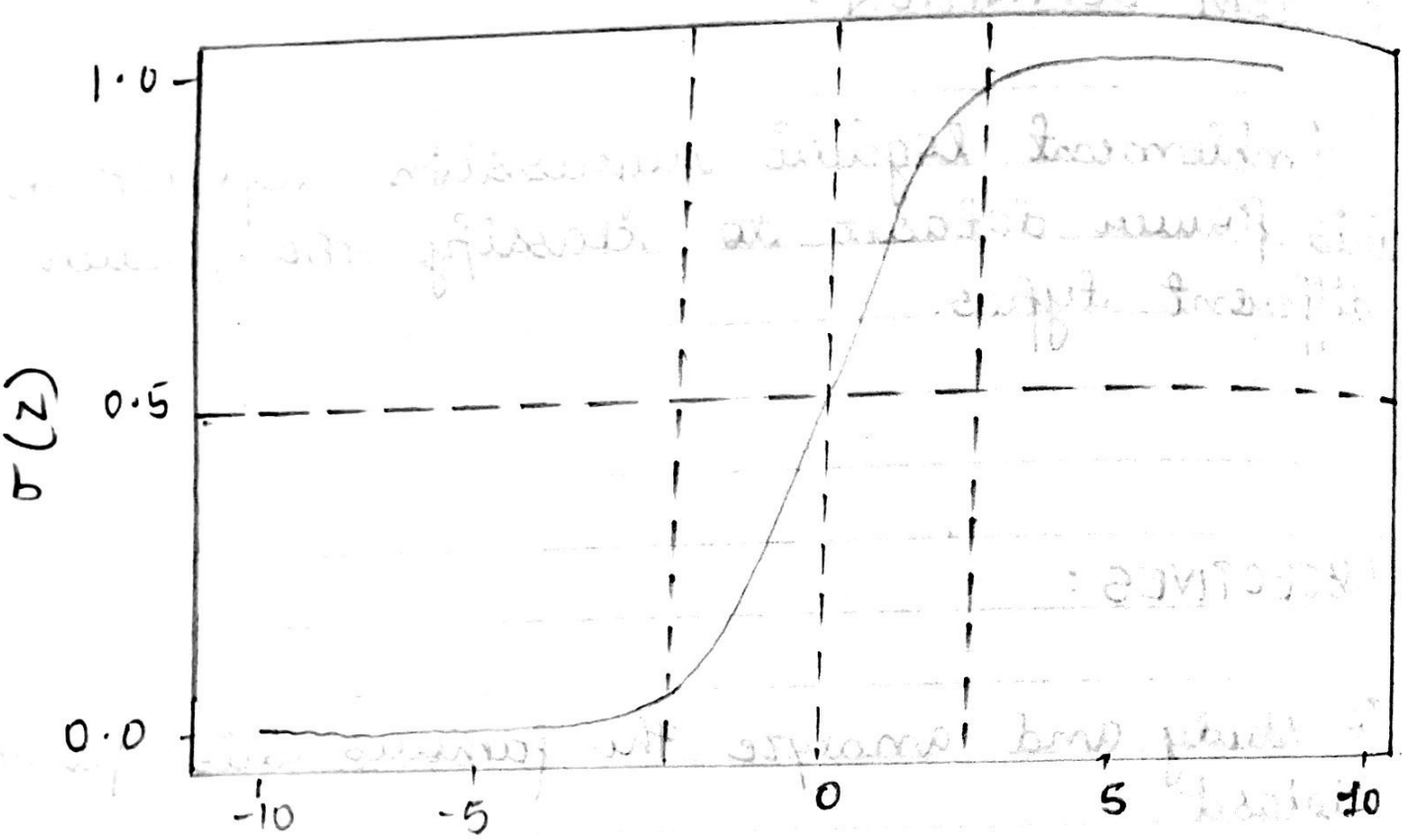
PROBLEM DEFINITION:

Implement logistic regression algorithm on the iris flower dataset to classify the flower into different types.

OBJECTIVES:

- To study and analyze the famous iris flowers dataset
- To understand logistic regression algorithm.

Sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$



$z = \Sigma w_i x_i + \text{bias}$

SIGMOID FUNCTION GRAPH

THEORY:

Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous. Like all regression analysis, the logistic regression is a predictive analysis. It is used to describe data and to explain the relation between one dependent binary variable and one or more nominal, ordinal, interval or ratio level independent variables

Sometimes logistic regression are different to interpret, the Logistic Regression is a classification algorithm used to assign observations to a discrete set of classes unlike linear regression which outputs continuous number value, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to 2 or more discrete classes

In order to map predicted values to probabilities we use the sigmoid function which maps any real value into another value between 0 and 1

$$S(z) = \frac{1}{1+e^{-z}}$$

PROGRAM :

```
#logistic regression for Iris dataset

library (dplyr)
library (ggplot2)
iris.small <- filter (iris, Specis %in% c ("virginica",
  "versicolor"))
glm.out <- glm (species ~ Sepal.width + Sepal.length +
petal.width + petal.length, data = iris.small, family =
binomial)
summary (glm.out)
glm ( formula = species ~ sepal.width + sepal.length +
Petal.length, family - binomial, data - iris.small)
lr_data <- data.frame (predictor = glm.out $ linear.
predictions prob = glm.out $ fitted.values, species =
iris.small $ specis)
ggplot (lr-data, aes (x= predictor, y= prob, color= species))
+ geom-point ()
ggplot ( lr-data, aes (x = predictor, fill = species)) + geom
density ( alpha = 0.5)
plant 1 <- data.frame( sepal.length = 6.4, sepal.width = 2.8,
petal.length = 4.6, Petal.width = 1.8)
plant 2 <- data.frame (sepal.length = 6.8, Sepal.width = 2.5,
Petal.length = 4.1, Petal.width = 1.7)
plant 3 <- data.frame (Sepal.length = 6.7, Sepal.width = 3.3
Petal.length = 5.2, Petal Width = 2.3)
predict (glm.out, plant1, type = "response")
predict (glm.out, plant 2, type = "response")
predict (glm.out, plant 3, type = "response")
```

TERM 4 - To classify the flowers using into one of the categories using Logistic Regression

```
summary(glm.out)

Call:
glm(formula = Species ~ Sepal.Width + Petal.Width + Petal.Length,
    family = binomial, data = iris.small)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.75795  -0.00412   0.00000   0.00290   1.92193

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   -50.527     23.995  -2.106   0.0352 *
Sepal.Width    -8.376      4.761  -1.759   0.0785 .
Petal.Width    21.430     10.707   2.001   0.0453 *
Petal.Length    7.875      3.841   2.050   0.0403 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 138.629  on 99  degrees of freedom
Residual deviance:  13.266  on 96  degrees of freedom
AIC: 21.266

Number of Fisher Scoring iterations: 10

> glm(formula = Species ~ Sepal.Width + Sepal.Length + Petal.Width + Petal
.Length,family =binomial, data=iris.small)

Call:  glm(formula = Species ~ Sepal.Width + Sepal.Length + Petal.Width +
    Petal.Length, family = binomial, data = iris.small)

Coefficients:
 (Intercept)   Sepal.Width  Sepal.Length   Petal.Width  Petal.Length
     -42.638        -6.681        -2.465        18.286         9.429

Degrees of Freedom: 99 Total (i.e. Null);  95 Residual
Null Deviance:      138.6
Residual Deviance: 11.9         AIC: 21.9
> glm.out<- glm(Species ~ Sepal.Width + Petal.Width + Petal.Length,data =i
ris.small,family = binomial)
> summary(glm.out)

Call:
glm(formula = Species ~ Sepal.Width + Petal.Width + Petal.Length,
    family = binomial, data = iris.small)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-1.75795  -0.00412   0.00000   0.00290   1.92193

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   -50.527     23.995  -2.106   0.0352 *
Sepal.Width    -8.376      4.761  -1.759   0.0785 .
Petal.Width    21.430     10.707   2.001   0.0453 *
Petal.Length    7.875      3.841   2.050   0.0403 *
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 138.629  on 99  degrees of freedom
Residual deviance:  13.266  on 96  degrees of freedom
AIC: 21.266

Number of Fisher Scoring iterations: 10

> lr_data<-data.frame(predictor=glm.out$linear.predictors,prob=glm.out$fit
ted.values,Species=iris.small$Species)
> ggplot(lr_data,aes(x=predictor,y=prob,color=Species))+geom_point()
> ggplot(lr_data,aes(x=predictor,fill=Species))+geom_density(alpha=.5)
> plant1<-data.frame(Sepal.Length=6.4,Sepal.Width=2.8,Petal.Length=4.6,Pet
al.Width=1.8)
> plant2<-data.frame(Sepal.Length=6.3,Sepal.Width=2.5,Petal.Length=4.1,Pet
al.Width=1.7)
> plant3<-data.frame(Sepal.Length=6.7,Sepal.Width=3.3,Petal.Length=5.2,Pet
al.Width=2.3)
> predict(glm.out,plant1,type="response")
        1
0.6934611
> predict(glm.out,plant2,type="response")
          1
0.06002675
> predict(glm.out,plant3,type="response")
        1
0.9999943
```