**LAB - 06**

**AIM:**  To develop a program in R language to implement KNN Classifier.

**TOOL USED:**

1. R(Programming Language)
2. RStudio

**THEORY:**

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. KNN algorithms use data and classify new data points based on similarity measures (e.g. distance function). The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data. It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. KNN algorithm at the training phase just stores the dataset and when it gets new data, and then it classifies that data into a category that is much similar to the new data.

## **Advantages of KNN Algorithm:**

## It is very simple algorithm to understand and interpret.

## It is very useful for nonlinear data because there is no assumption about data in this algorithm.

## It is a versatile algorithm as we can use it for classification as well as regression.

## It has relatively high accuracy but there are much better supervised learning models than KNN.

## **Disadvantages of KNN Algorithm:**

* It is computationally a bit expensive algorithm because it stores all the training data.
* High memory storage required as compared to other supervised learning algorithms.
* Prediction is slow in case of big N.
* It is very sensitive to the scale of data as well as irrelevant features.

**DATASET:**

For this Iris dataset is used.

**Table 6.1: Iris Dataset**

**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

…

74 6.1 2.8 4.7 1.2 versicolor

75 6.4 2.9 4.3 1.3 versicolor

76 6.6 3.0 4.4 1.4 versicolor

…

146 6.7 3.0 5.2 2.3 virginica

147 6.3 2.5 5.0 1.9 virginica

148 6.5 3.0 5.2 2.0 virginica

149 6.2 3.4 5.4 2.3 virginica

150 5.9 3.0 5.1 1.8 virginica

**PROGRAM:**

# Installing Packages

install.packages("e1071")

install.packages("caTools")

install.packages("class")

# Loading package

library(e1071)

library(caTools)

library(class)

# Loading data

data(iris)

head(iris)

# Splitting data into train

# and test data

split <- sample.split(iris, SplitRatio = 0.7)

train\_cl <- subset(iris, split == "TRUE")

test\_cl <- subset(iris, split == "FALSE")

# Feature Scaling

train\_scale <- scale(train\_cl[, 1:4])

test\_scale <- scale(test\_cl[, 1:4])

# Fitting KNN Model

# to training dataset

classifier\_knn <- knn(train = train\_scale,

                      test = test\_scale,

                      cl = train\_cl$Species,

                      k = 1)

classifier\_knn

# Confusiin Matrix

cm <- table(test\_cl$Species, classifier\_knn)

cm

# Model Evaluation - Choosing K

# Calculate out of Sample error

misClassError <- mean(classifier\_knn != test\_cl$Species)

print(paste('Accuracy =', 1-misClassError))

# K = 3

classifier\_knn <- knn(train = train\_scale,

                      test = test\_scale,

                      cl = train\_cl$Species,

                      k = 3)

misClassError <- mean(classifier\_knn != test\_cl$Species)

print(paste('Accuracy =', 1-misClassError))

# K = 5

classifier\_knn <- knn(train = train\_scale,

                      test = test\_scale,

                      cl = train\_cl$Species,

                      k = 5)

misClassError <- mean(classifier\_knn != test\_cl$Species)

print(paste('Accuracy =', 1-misClassError))

# K = 7

classifier\_knn <- knn(train = train\_scale,

                      test = test\_scale,

                      cl = train\_cl$Species,

                      k = 7)

misClassError <- mean(classifier\_knn != test\_cl$Species)

print(paste('Accuracy =', 1-misClassError))

# K = 15

classifier\_knn <- knn(train = train\_scale,

                      test = test\_scale,

                      cl = train\_cl$Species,

                      k = 15)

misClassError <- mean(classifier\_knn != test\_cl$Species)

print(paste('Accuracy =', 1-misClassError))

# K = 19

classifier\_knn <- knn(train = train\_scale,

                      test = test\_scale,

                      cl = train\_cl$Species,

                      k = 19)

misClassError <- mean(classifier\_knn != test\_cl$Species)

print(paste('Accuracy =', 1-misClassError))

**OUTPUT:**

classifier\_knn

setosa versicolor virginica

setosa 20 0 0

versicolor 0 19 1

virginica 0 1 19

**Table 6.2: Accuracy of Model for various value of K**

|  |  |
| --- | --- |
| **K** | **Accuracy** |
| 1 | 0.9666 |
| 3 | 0.9333 |
| 5 | 0.9666 |
| 7 | 0.9666 |
| 15 | 0.9333 |
| 19 | 0.9 |

**LAB - 07**

**AIM:**  To develop a program in R language to implement Multiple Linear Regression Model.

**TOOL USED:**

1. R(Programming Language)
2. RStudio

**THEORY:**

It’s a formof linear regression that is used when there are two or more predictors. Multiple linear regression refers to a statistical technique that is used to predict the outcome of a variable based on the value of two or more variables. It is sometimes known simply as multiple regression, and it is an extension of linear regression. The variable that we want to predict is known as the dependent variable, while the variables we use to predict the value of the [dependent variable](https://corporatefinanceinstitute.com/resources/knowledge/terms/dependent-variable/) are known as independent or explanatory variables.

The probabilistic model that includes more than one independent variable is called **multiple regression models**. The general form of this model is:



Where:

yi​ is the dependent or predicted variable

β0 is the y-intercept, i.e., the value of y when both xi and x2 are 0.

β1 and β2 are the regression coefficients representing the change in y relative to a one-unit change in xi1 and xi2, respectively.

βp is the slope coefficient for each independent variable

ϵ is the model’s random error (residual) term.

**PROGRAM:**

Year <- c(2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2017, 2016,2016,2016,2016,2016,2016,2016,2016,2016,2016,2016,2016)

Month <- c(12, 11,10,9,8,7,6,5,4,3,2,1,12,11,10,9,8,7,6,5,4,3,2,1)

Interest\_Rate <- c(2.75,2.5,2.5,2.5,2.5,2.5,2.5,2.25,2.25,2.25,2,2,2,1.75,1.75,1.75, 1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75)

Unemployment\_Rate <- c(5.3,5.3,5.3,5.3,5.4,5.6,5.5,5.5,5.5,5.6,5.7,5.9,6,5.9,5.8,6.1, 6.2,6.1,6.1,6.1,5.9,6.2,6.2,6.1)

Stock\_Index\_Price <- c(1464,1394,1357,1293,1256,1254,1234,1195,1159,1167, 1130,1075,1047,965,943,958,971,949,884,866,876,822,704,719)

plot(x=Interest\_Rate, y=Stock\_Index\_Price)

plot(x=Unemployment\_Rate, y=Stock\_Index\_Price)

 model <- lm(Stock\_Index\_Price ~ Interest\_Rate + Unemployment\_Rate)

 summary(model)

**OUTPUT:**

Call:

lm(formula = Stock\_Index\_Price ~ Interest\_Rate + Unemployment\_Rate)

Residuals:

Min 1Q Median 3Q Max

-158.205 -41.667 -6.248 57.741 118.810

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1798.4 899.2 2.000 0.05861 .

Interest\_Rate 345.5 111.4 3.103 0.00539 \*\*

Unemployment\_Rate -250.1 117.9 -2.121 0.04601 \*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 70.56 on 21 degrees of freedom

Multiple R-squared: 0.8976, Adjusted R-squared: 0.8879

F-statistic: 92.07 on 2 and 21 DF, p-value: 4.043e-11

**LAB - 08**

**AIM:**  To develop a program in R language to implement Logistic Regression Model.

**TOOL USED:**

1. R(Programming Language)
2. RStudio

**THEORY:**

Logistic regression is also known as Binomial logistic regression. It is based on sigmoid function where output is probability and input can be from -infinity to + infinity. Logistic regression is used when the dependent variable is binary(0/1, True/False, Yes/No) in nature. Logistic regression is also known as generalized linear model. As it is used as a classification technique to predict a qualitative response, Value of y ranges from 0 to 1 and can be represented by following equation:

**Odd = P/1-P.**

P is probability of characteristic of interest. The odds ratio is defined as the probability of success in comparison to the probability of failure.

**DATASET:**

**Table 8.1: MTCARS Dataset**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **model** | **mpg** | **cyl** | **disp** | **hp** | **drat** | **wt** | **qsec** | **vs** | **am** | **gear** | **carb** |
| Mazda RX4 | 21 | 6 | 160 | 110 | 3.9 | 2.62 | 16.46 | 0 | 1 | 4 | 4 |
| Mazda RX4 Wag | 21 | 6 | 160 | 110 | 3.9 | 2.875 | 17.02 | 0 | 1 | 4 | 4 |
| Datsun 710 | 22.8 | 4 | 108 | 93 | 3.85 | 2.32 | 18.61 | 1 | 1 | 4 | 1 |
| Hornet 4 Drive | 21.4 | 6 | 258 | 110 | 3.08 | 3.215 | 19.44 | 1 | 0 | 3 | 1 |
| Hornet Sportabout | 18.7 | 8 | 360 | 175 | 3.15 | 3.44 | 17.02 | 0 | 0 | 3 | 2 |

**PROGRAM:**

#Installing the package

install.packages("dplyr")

install.packages("caTools")    # For Logistic regression

install.packages("ROCR")       # For ROC curve to evaluate model

# Loading package

library(dplyr)

library(caTools)

library(ROCR)

  # Summary of dataset in package

summary(mtcars)

  # Splitting dataset

split <- sample.split(mtcars, SplitRatio = 0.8)

split

train\_reg <- subset(mtcars, split == "TRUE")

test\_reg <- subset(mtcars, split == "FALSE")

# Training model

logistic\_model <- glm(vs ~ wt + disp,

                      data = train\_reg,

                      family = "binomial")

logistic\_model

# Summary

summary(logistic\_model)

# Predict test data based on model

predict\_reg <- predict(logistic\_model,

                       test\_reg, type = "response")

predict\_reg

# Changing probabilities

predict\_reg <- ifelse(predict\_reg >0.5, 1, 0)

# Evaluating model accuracy

# using confusion matrix

table(test\_reg$vs, predict\_reg)

   missing\_classerr <- mean(predict\_reg != test\_reg$vs)

print(paste('Accuracy =', 1 - missing\_classerr))

   # ROC-AUC Curve

ROCPred <- prediction(predict\_reg, test\_reg$vs)

ROCPer <- performance(ROCPred, measure = "tpr", x.measure = "fpr")

  auc <- performance(ROCPred, measure = "auc")

auc <- auc@y.values[[1]]

auc

 # Plotting curve

plot(ROCPer)

plot(ROCPer, colorize = TRUE,

     print.cutoffs.at = seq(0.1, by = 0.1),

     main = "ROC CURVE")

abline(a = 0, b = 1)

auc <- round(auc, 4)

legend(.6, .4, auc, title = "AUC", cex = 1)

**OUTPUT:**

**Logistic model**

Call: glm(formula = vs ~ wt + disp, family = "binomial", data = train\_reg)

Coefficients:

(Intercept) wt disp

3.65725 0.50755 -0.02638

Degrees of Freedom: 23 Total (i.e. Null); 21 Residual

Null Deviance: 33.27

Residual Deviance: 17.64 AIC: 23.64

**Summary Logistic model**

Call: glm(formula = vs ~ wt + disp, family = "binomial", data = train\_reg)

Deviance Residuals:

Min 1Q Median 3Q Max

-1.7280 -0.4454 0.1462 0.5425 1.8519

Coefficients:

Estimate Std. Error z value Pr(>|z|)

(Intercept) 3.65725 3.20933 1.140 0.254

wt 0.50755 1.74052 0.292 0.771

disp -0.02638 0.01611 -1.638 0.102

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 33.271 on 23 degrees of freedom

Residual deviance: 17.638 on 21 degrees of freedom

AIC: 23.638

Number of Fisher Scoring iterations: 6

**Predicted Test data on model**

Hornet Sportabout Duster 360 Merc 280C Lincoln Continental

0.01642104 0.01752145 0.72757938 0.00325796

Fiat 128 Dodge Challenger Porsche 914-2 Ford Pantera L

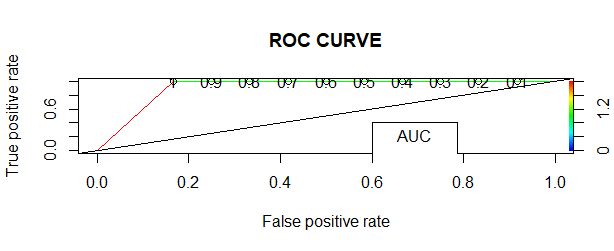
0.93690626 0.05001231 0.82781339 0.01812313

**Confusion matrix** (predict\_reg)

0 1

0 5 1

1 0 2

Accuracy = 0.875

**Fig. 8.1: ROC Curve for TPR and FPR**

**LAB - 09**

**AIM:**  1. Develop a program in PYTHON to read a Dataset in EXCEL / CSV and display its properties.

2. Develop a program in PYTHON to implement Naïve Bayesian (NB) Classifier.

**TOOL USED:**

1. Python (Programming Language)
2. Anaconda

**THEORY:**

Python provides us with three functions to read data from a text file:

1. **read(n)** – This function reads n bytes from the text files or reads the complete information from the file if no number is specified. It is smart enough to handle the delimiters when it encounters one and separates the sentences
2. **readline(n)** – This function allows you to read n bytes from the file but not more than one line of information
3. **readlines()** – This function reads the complete information in the file but unlike **read()**, it doesn’t bother about the delimiting character and prints them as well in a list format

**PROGRAM:**

1. **Using CSV**

import csv

# csv file name

filename = "kddcup\_data\_10\_percent\_corrected.csv"

# initializing the titles and rows list

fields = []

rows = []

# reading csv file

with open(filename, 'r') as csvfile:

# creating a csv reader object

csvreader = csv.reader(csvfile)

# extracting field names through first row

fields = next(csvreader)

# extracting each data row one by one

for row in csvreader:

rows.append(row)

# get total number of rows

print("Total no. of rows: %d"%(csvreader.line\_num))

# printing the field names

print('Field names are:' + ', '.join(field for field in fields))

# printing first 5 rows

print('\nFirst 5 rows are:\n')

for row in rows[:5]:

# parsing each column of a row

for col in row:

print("%10s"%col),

print('\n')

1. **Using Pandas**

import pandas as pd

data = pd.read\_excel("Computer Status.xlsx")

print("##### Printing the Data Set : #####")

print(data)

print("\n \n \n ##### Printing the Top 5 Rows of Data Set : #####")

print(data.head())

print("\n \n \n ##### Columns as List : #####")

print(data.columns.ravel())

spcol = pd.read\_excel("Computer Status.xlsx", usecols=['RAM', 'LAB'])

print("\n \n \n ##### Reading Data from Specific Columns of EXCEL File and Printing it :#####")

print(spcol)

brdataall = (data['Branch'].tolist())

print("\n \n \n ##### Data of Branch Column : #####")

print(brdataall)

**OUTPUT:**

1. **Using Excel**

Total no. of rows: 494022

Field names are:duration, protocol\_type, service, flag, src\_bytes, dst\_bytes, land, wrong\_fragment, urgent, hot, num\_failed\_logins, logged\_in, num\_compromised, root\_shell, su\_attempted, num\_root, num\_file\_creations, num\_shells, num\_access\_files, num\_outbound\_cmds, is\_host\_login, is\_guest\_login, count, srv\_count, serror\_rate, srv\_serror\_rate, rerror\_rate, srv\_rerror\_rate, same\_srv\_rate, diff\_srv\_rate, srv\_diff\_host\_rate, dst\_host\_count, dst\_host\_srv\_count, dst\_host\_same\_srv\_rate, dst\_host\_diff\_srv\_rate, dst\_host\_same\_src\_port\_rate, dst\_host\_srv\_diff\_host\_rate, dst\_host\_serror\_rate, dst\_host\_srv\_serror\_rate, dst\_host\_rerror\_rate, dst\_host\_srv\_rerror\_rate, class.

First 5 rows are:

0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0,0,0,0,1,0,0,9,9,1,0,0.11,0,0,0,0,0,normal.

0,tcp,http,SF,239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0,0,0,0,1,0,0,19,19,1,0,0.05,0,0,0,0,0,normal.

0,tcp,http,SF,235,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0,0,0,0,1,0,0,29,29,1,0,0.03,0,0,0,0,0,normal.

0,tcp,http,SF,219,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0,0,0,0,1,0,0,39,39,1,0,0.03,0,0,0,0,0,normal.

0,tcp,http,SF,217,2032,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0,0,0,0,1,0,0,49,49,1,0,0.02,0,0,0,0,0,normal.

**2. Using Panda**

##### Printing the Data Set : #####

S.\nNo. System No. MotherBoard Company RAM \

0 21 Sys No 21 Intel NaN 4-GB DDR-III

1 22 Sys No 22 Intel NaN 4-GB DDR-III

2 23 Sys No 23 Intel NaN 4-GB DDR-III

3 24 Sys No 24 Intel NaN 4-GB DDR-III

4 25 Sys No 25 Intel NaN 4-GB DDR-III

.. ... ... ... ... ...

530 26 Sys No 26 ASUS Beetel 2-GB DDR-III

531 27 Sys No 27 ASUS Beetel 2-GB DDR-II

532 28 Sys No 28 ASUS Beetel 1-GB DDR-II

533 29 Sys No 29 ASUS Beetel 1-GB DDR-II

534 30 Sys No 30 ASUS Beetel 1-GB DDR-II

Processor HardDisk Monitor Keyboard \

0 Intel®Core I3 2.90GHz 500-GB(sata) 17"" LCD HP HP

1 Intel®Core I3 2.90GHz 500-GB(sata) 17"" LCD HP HP

2 Intel®Core I3 2.90GHz 500-GB(sata) 17"" LCD HP HP

3 Intel®Core I3 2.90GHz 500-GB(sata) 17"" LCD HP HP

4 Intel®Core I3 2.90GHz 500-GB(sata) 17"" LCD HP HP

.. ... ... ... ...

530 Pentium D 2.60 GHz 160-GB(sata) 17"" LCD Acer Logitech K200

531 Pentium D 2.20 GHz 250-GB(sata) 17"" LCD Acer Logitech K200

532 Pentium D 2.60 GHz 160-GB(sata) 17"" LCD Acer HCL

533 Pentium D 2.70 GHz 160-GB(sata) 17"" LCD Acer Logitech K200

534 Pentium D 2.60 GHz 160-GB(sata) 17"" LCD AOC Logitech K201

##### Printing the Top 5 Rows of Data Set : #####

S.\nNo. System No. MotherBoard Company RAM \

0 21 Sys No 21 Intel NaN 4-GB DDR-III

1 22 Sys No 22 Intel NaN 4-GB DDR-III

2 23 Sys No 23 Intel NaN 4-GB DDR-III

3 24 Sys No 24 Intel NaN 4-GB DDR-III

4 25 Sys No 25 Intel NaN 4-GB DDR-III

Processor HardDisk Monitor Keyboard Mouse \

0 Intel®Core I3 2.90GHz 500-GB(sata) 17"" LCD HP HP HP

1 Intel®Core I3 2.90GHz 500-GB(sata) 17"" LCD HP HP Intex

2 Intel®Core I3 2.90GHz 500-GB(sata) 17"" LCD HP HP HP

3 Intel®Core I3 2.90GHz 500-GB(sata) 17"" LCD HP HP HP

4 Intel®Core I3 2.90GHz 500-GB(sata) 17"" LCD HP HP HP

##### Columns as List : #####

['S.\nNo.' 'System No.' 'MotherBoard' 'Company' 'RAM' 'Processor'

'HardDisk' 'Monitor' 'Keyboard' 'Mouse' 'CPU Sr.No.' 'Lab' 'Unnamed: 12'

'Unnamed: 13' 'Unnamed: 14']

##### Reading Data from Specific Columns of EXCEL File and Printing it :#####

RAM Lab

0 4-GB DDR-III 1

1 4-GB DDR-III 1

2 4-GB DDR-III 1

3 4-GB DDR-III 1

4 4-GB DDR-III 1

.. ... ...

530 2-GB DDR-III CE-17

531 2-GB DDR-II CE-17

532 1-GB DDR-II CE-17

533 1-GB DDR-II CE-17

534 1-GB DDR-II CE-17

[535 rows x 2 columns]

##### Data of Mouse Column : #####

['HP', 'Intex', 'HP', 'HP', 'HP', 'HP', 'HP', 'Logitec', 'HP', 'Logitec', 'Logitec', 'Logitec', 'Wipro', 'Wipro', 'Wipro', 'Wipro', nan, 'Wipro', 'Wipro', 'Wipro', nan, 'Wipro', 'Wipro', 'Wipro', 'HCL', 'Wipro', 'HCL', nan, nan, nan, 'HP', 'HP', 'Wipro', 'HP', 'Wipro', 'Wipro', 'Wipro', nan, nan, nan, 'Logitech', nan, 'HCL', 'Logitech', 'Logitech', 'Logitech', 'Intex', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', nan, 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Dell', 'Logitech', nan, 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'HCL', 'Logitech', 'Logitech', 'HP', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'HCL', 'Logitech', 'Logitech', 'Logitech', 'Logitech', nan, 'Logitech', 'Logitech', 'Logitech', 'Logitech', 'HCL', 'HCL', 'HCL', 'HCL', 'HCL', 'HCL', 'Logitec', 'HCL', 'Logitec', 'HCL', 'HCL', 'HCL', 'HCL', 'HCL', 'HCL', 'HCL', 'Logitec', 'Logitec', 'HCL', 'HCL', 'HCL', 'HCL', 'HCL', 'Logitec', 'HCL', 'HCL', 'HCL', 'HCL', 'Logitec', 'HCL', 'HCL', nan, 'HP', 'Logitec', 'Dell', 'HCL', 'HCL', 'Dell', 'Logitec', 'HCL', 'HCL', nan, 'HCL', 'HCL', 'HCL', 'HCL', 'HCL', 'HCL', 'Logitec', 'HCL', 'HCL', 'HCL', 'HCL', 'HCL', 'Logitec', 'HCL', 'HCL', 'Dell', 'HCL', 'Dell', 'HCL', nan, 'HCL', 'HCL', 'Logitec', 'Logitec', 'Dell', 'HCL', 'Logitec', 'HCL', 'HCL', 'HCL', 'Wipro', 'Logitec', 'HCL', nan, 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'Logitec', 'Logitec', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', nan, 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'Logitec', 'HP', 'HP', 'Logitec', 'HP', 'Logitec', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'HP', 'Logitec', 'HP', 'HP', 'Logitec', 'HP', 'Logitec', 'Logitec', 'HP', 'HP', 'HP', 'HCL', 'HCL', 'HP', 'Logitec', 'Logitec', 'HP', 'HP'

**AIM:**  2. Develop a program in PYTHON to implement Naïve Bayesian (NB) Classifier.

**TOOL USED:**

1. Python (Programming Language)
2. Anaconda

**THEORY:**

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.

**BAYE’s Theorem:** It is also known as BAYE‟s Rule or BAYE‟s Law or Bayesian Reasoning, which determines the Probability of an Event with Uncertain Knowledge. In Probability Theory, it relates the Conditional Probability and Marginal Probability of two Random Events. It was named after the British Mathematician „Thomas Bayes‟. The Bayesian Inference is an application of BAYE‟s Theorem, which is fundamental to Bayesian Statistics.

It computes P(B | A), when P(A | B) is already known. Let, A and B be two independent Events, then Conditional Probability of Event A with Event B already known is:

**P (A | B) = P(A ^ B) / P(B)**

It can also be represented, as:

**P(A ^ B) = P(A | B) \* P(B) ….. (1)**

Similarly, Conditional Probability of Event B with Event A already known is:

**P(B | A) = P(A ^ B) / P(A)**

It can also be represented, as:

**P(A ^ B) = P(B | A) \* P(A)….. (2)**

Equating (1) and (2), we get,

**P(A | B) = [P(B | A) \* P(A)] / P(B)**

The above equation is called BAYE‟s Rule or BAYE‟s Theorem. It shows the Relationship between Joint Probability and Conditional Probability.

**DATASET:**

**Table 9.1: Iris Dataset**

**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

…

74 6.1 2.8 4.7 1.2 versicolor

75 6.4 2.9 4.3 1.3 versicolor

76 6.6 3.0 4.4 1.4 versicolor

…

146 6.7 3.0 5.2 2.3 virginica

147 6.3 2.5 5.0 1.9 virginica

148 6.5 3.0 5.2 2.0 virginica

149 6.2 3.4 5.4 2.3 virginica

150 5.9 3.0 5.1 1.8 virginica

**PROGRAM:**

from sklearn.datasets import load\_iris

iris = load\_iris()

# store the feature matrix (X) and response vector (y)

X = iris.data

y = iris.target

# splitting X and y into training and testing sets

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=1)

# training the model on training set

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

# making predictions on the testing set

y\_pred = gnb.predict(X\_test)

# comparing actual response values (y\_test) with predicted response values (y\_pred)

from sklearn import metrics

print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy\_score(y\_test, y\_pred)\*100)

**OUTPUT:**

Gaussian Naive Bayes model accuracy(in %): 95.0

**LAB - 10**

**AIM:**  To develop a program in PYTHON to implement Linear Regression Model.

**TOOL USED:**

1. Python (Programming Language)
2. Anaconda

**THEORY:**

**Regression:** It predicts the Value of a Dependent Variable based on the Known Value of one or more Independent Variables. It is a related technique to assess the Relationship between an Outcome Variable and one or more Risk Factors or Confounding Variables. The Outcome Variable is also called the Response Variable or Dependent Variable (Y) and the Risk Factors or Confounders are called the Predictors or Explanatory or Independent Variables (X). It is a widely used technique for evaluating multiple Independent Variables.

**Simple Linear Regression:** When there is a single Continuous Dependent Variable and a single Independent Variable, the analysis is called Simple Linear Regression Analysis. It assumes that there is a Linear Association between two Variables. The Simple Linear Regression Equation is given as:

**Y = C1 + C2 \* X**

Where, Y is the Predicted or Expected Value of the Outcome, X is the Predictor, C1 and C2 are Constants known as Y Intercept and Estimated Slope respectively. C1 and C2 are estimated from Sample Data in order to minimize the Sum of Squared Differences between the Observed and the Predicted Values of the Outcome.

**Multiple Linear Regression:** It is an extension of Simple Linear Regression Analysis, used to assess the Association between two or more Independent Variables and a single Continuous Dependent Variable. The Multiple Linear Regression Equation is given, as:

**Y = C0 + C1 \* X1 + C2 \* X2 + C3 \* X3 + ----- + CP \* XP**

**Logistic Regression:** It is a popular and widely used analysis, similar to Linear Regression except that the Outcome is Dichotomous (Success / Failure or Yes / No or True / False). Simple Logistic Regression Analysis refers to the Regression application with one Dichotomous Outcome and one Independent Variable. Multiple Logistic Regression Analysis is applicable when there is a single Dichotomous Outcome and more than one Independent Variable.

The Outcome of Logistic Regression Analysis is often coded as 0 or 1, where „1‟ indicates that the Outcome of interest is Present and „0‟ indicates that the Outcome of interest is Absent. IF we define „P‟ as the Probability that the Outcome is „1‟, then the multiple Logistic Regression Model can be given, as:

**Z = EXP(C0 + C1 \* X1 + C2 \* X2 + ----- + CP \* XP) /**

**[1 + EXP(C0 + C1 \* X1 + C2 \* X2 + ----- + CP \* XP)]**

Where, Z is the Expected Probability that the Outcome is Present, X1, X2, ….., XP are Distinct Independent Variables and C0, C1, ….., CP are Regression Coefficients.

**DATASET:**

**Table 10.1: Dataset**

|  |  |
| --- | --- |
| ind\_var | dep\_var |
| 0 | 1 |
| 1 | 3 |
| 2 | 2 |
| 3 | 5 |
| 4 | 7 |
| 5 | 8 |
| 6 | 8 |
| 7 | 9 |
| 8 | 10 |
| 9 | 12 |
| 10 | 12 |
| 11 | 11 |
| 12 | 16 |
| 13 | 15 |
| 14 | 14 |
| 15 | 20 |
| 16 | 17 |
| 17 | 19 |
| 18 | 16 |
| 19 | 25 |
| 20 | 24 |
| 21 | 23 |
| 22 | 18 |
| 23 | 21 |
| 24 | 16 |

**PROGRAM:**

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

from sklearn import metrics

dset = pd.read\_csv("DA LAB 10.csv")

print(dset.head())

xind = pd.DataFrame(dset['ind\_var'])

ydep = pd.DataFrame(dset['dep\_var'])

indtrain, indtest, deptrain, deptest = train\_test\_split(xind, ydep, test\_size = 0.2, random\_state = 1)

print("Training Data for Independent Variable : \n", indtrain)

print("Testing Data for Independent Variable : \n", indtest)

print("Training Data for Dependent Variable : \n", deptrain)

print("Testing Data for Dependent Variable : \n", deptest)

print("Shape of Training Data (Independent) : ", indtrain.shape)

print("Shape of Testing Data (Independent) : ", indtest.shape)

print("Shape of Training Data (Dependent) : ", deptrain.shape)

print("Shape of Testing Data (Dependent) : ", deptest.shape)

regmodel = LinearRegression()

regmodel.fit(indtrain, deptrain)

print("Value of Regression Intercept is : ", regmodel.intercept\_)

print("Value of Regression Coefficient is : ", regmodel.coef\_)

print("Observed Testing Data : \n", deptest)

deppred = regmodel.predict(indtest)

print("Predicted Testing Data : \n", deppred)

print("Mean Absolute Error : ", metrics.mean\_absolute\_error(deptest, deppred))

print("Mean Squared Error : ", metrics.mean\_squared\_error(deptest, deppred))

print("ROOT MEAN SQUARED ERROR : ", np.sqrt(metrics.mean\_squared\_error(deptest, deppred)))

**OUTPUT:**

ind\_var dep\_var

0 0 1

1 1 3

2 2 2

3 3 5

4 4 7

Training Data for Independent Variable :

ind\_var

10 10

18 18

19 19

4 4

2 2

20 20

6 6

7 7

22 22

1 1

16 16

0 0

15 15

24 24

23 23

9 9

8 8

12 12

11 11

5 5

Testing Data for Independent Variable :

ind\_var

14 14

13 13

17 17

3 3

21 21

Training Data for Dependent Variable :

dep\_var

10 12

18 16

19 25

4 7

2 2

20 24

6 8

7 9

22 18

1 3

16 17

0 1

15 20

24 16

23 21

9 12

8 10

12 16

11 11

5 8

Testing Data for Dependent Variable :

dep\_var

14 14

13 15

17 19

3 5

21 23

Shape of Training Data (Independent) : (20, 1)

Shape of Testing Data (Independent) : (5, 1)

Shape of Training Data (Dependent) : (20, 1)

Shape of Testing Data (Dependent) : (5, 1)

Value of Regression Intercept is : [3.18863143]

Value of Regression Coefficient is : [[0.82856626]]

Observed Testing Data :

dep\_var

14 14

13 15

17 19

3 5

21 23

Predicted Testing Data :

[[14.78855902]

[13.95999276]

[17.27425778]

[ 5.6743302 ]

[20.58852281]]

Mean Absolute Error : 1.3280231716147717

Mean Squared Error : 2.1903140060015893

ROOT MEAN SQUARED ERROR : 1.4799709476883622