## AIM: CLASSIFICATION ALGORITHMS - NAIVE BAYES, ID3, C 4.5, K NEAREST NEIGHBOUR

### THEORY:

**Naive Bayesian:**

It 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 about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

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

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

$$\underbrace{P(c\mid x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x\mid c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c\mid X) = P(x_1\mid c)\times P(x_2\mid c)\times\cdots\times P(x_n\mid c)\times P(c)$$

Above,

- P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor. **Pros:**

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

**Cons:**

- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
- On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.
- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

**K nearest Neighbors:**

KNN (K — Nearest Neighbors) is one of many (supervised learning) algorithms used in data mining and machine learning, it's a classifier algorithm where the learning is based "how similar" is a data (a vector) from other .

**The KNN's steps are:**

1. Receive an unclassified data;
2. Measure the distance (Euclidian, Manhattan, Minkowski or Weighted) from the new data to all others data that is already classified;
3. Gets the K(K is a parameter that you difine) smaller distances;
4. Check the list of classes had the shortest distance and count the amount of each class that appears;
5. Takes as correct class the class that appeared the most times; 6. Classifies the new data with the class that you took in step 5;

**Choosing the right value for K:**

To select the K that's right for your data, we run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

Here are some things to keep in mind:

1. As we decrease the value of K to 1, our predictions become less stable. Just think for a minute, imagine K=1 and we have a query point surrounded by several reds and one green (I'm thinking about the top left corner of the colored plot above), but the green is the single nearest neighbor. Reasonably, we would think the query point is most likely red, but because K=1, KNN incorrectly predicts that the query point is green.
2. Inversely, as we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point). Eventually, we begin to witness an increasing number of errors. It is at this point we know we have pushed the value of K too far.
3. In cases where we are taking a majority vote (e.g. picking the mode in a classification problem) among labels, we usually make K an odd number to have a tiebreaker.

**Advantages:**

1. The algorithm is simple and easy to implement.
2. There's no need to build a model, tune several parameters, or make additional assumptions.
3. The algorithm is versatile. It can be used for classification, regression, and search (as we will see in the next section).

**Disadvantages:**

1. The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

**R Packages:**

1) **tidyr:**
The word tidyr comes from the word tidy, which means clear. tidyr package is used to make the data' tidy'

2) **ggplot2:**
R provides the ggplot package for creating graphics declaratively. This package is famous for its elegant and quality graphs which sets it apart from other visualization packages.

3) **dplyr:**
R provides the dplyr library for performing data wrangling and data analysis.This library facilitates several functions for the data frame in R.

4) **caret:**
R allows us to perform classification and regression tasks by providing the caret package. CaretEnsemble is a feature of caret which is used for the combination of different models.

5) **e1071:**
The e1071 library provides useful functionsessential for data analysis like Naive Bayes, Fourier Transforms, SVMs, Clustering, and other miscellaneous functions

## A) IMPLEMENT NAIVE BAYES

### INSTALLING LOADING LIBS:

```
install.packages("e1071")
install.packages("klaR")

# Loading library e1071
library(e1071)

# Loading library k1aR
library(k1aR)

## Loading required package: caret
library(caret)

## Loading required package: lattice
library(lattice)

## Loading required package: ggplot2
library(ggplot2)
```

```
> # Loading library e1071
> library(e1071)
Warning message:
package ‘e1071’ was built under R version 4.0.5
>
> # Loading library klaR
> library(klaR)
Loading required package: MASS
Warning message:
package ‘klaR’ was built under R version 4.0.5
>
> ## Loading required package: caret
> library(caret)
Loading required package: ggplot2
Loading required package: lattice
Warning messages:
1: package ‘caret’ was built under R version 4.0.5
2: package ‘ggplot2’ was built under R version 4.0.3
>
> ## Loading required package: lattice
> library(lattice)
>
> ## Loading required package: ggplot2
> library(ggplot2)
```

### PRINTING DATASET:

```
# iris dataset
data(iris)
head(iris)
```

```
> # iris dataset
> data(iris)
> 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
```

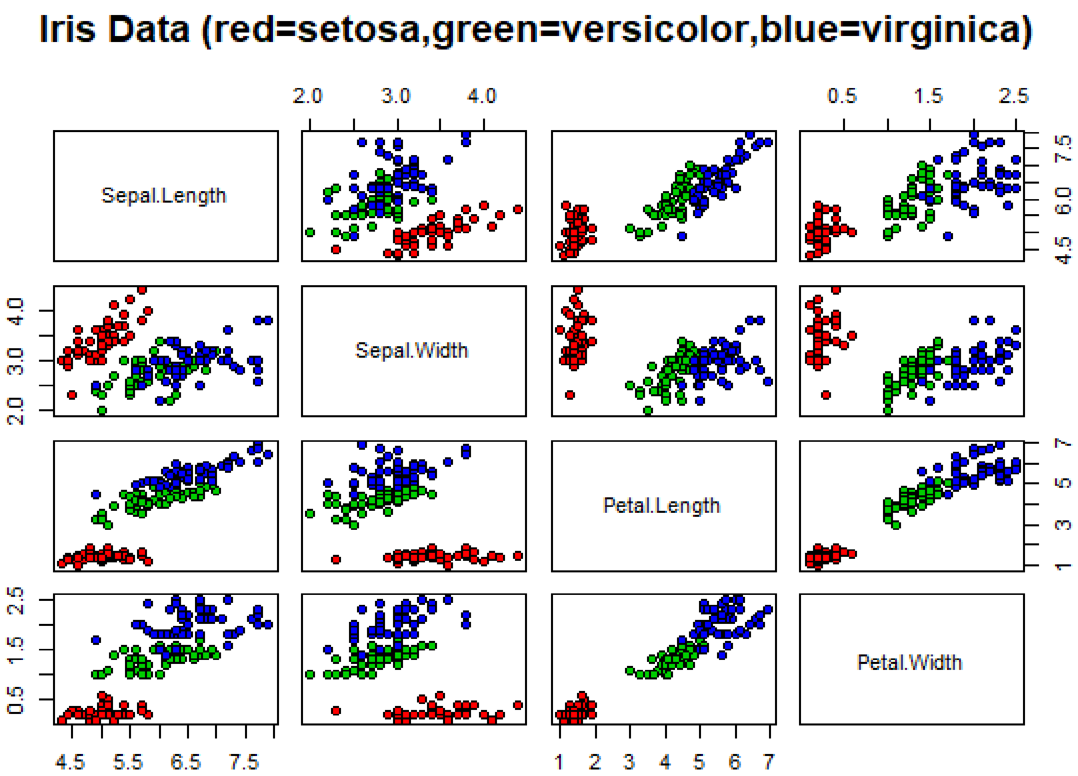
**PRINTING UNIQUE ELEMENTS:**

#finding unique values
unique(iris$Species)

```
> #finding unique values
> unique(iris$Species)
[1] setosa     versicolor virginica
Levels: setosa versicolor virginica
```

**PLOTTING GRAPH:**

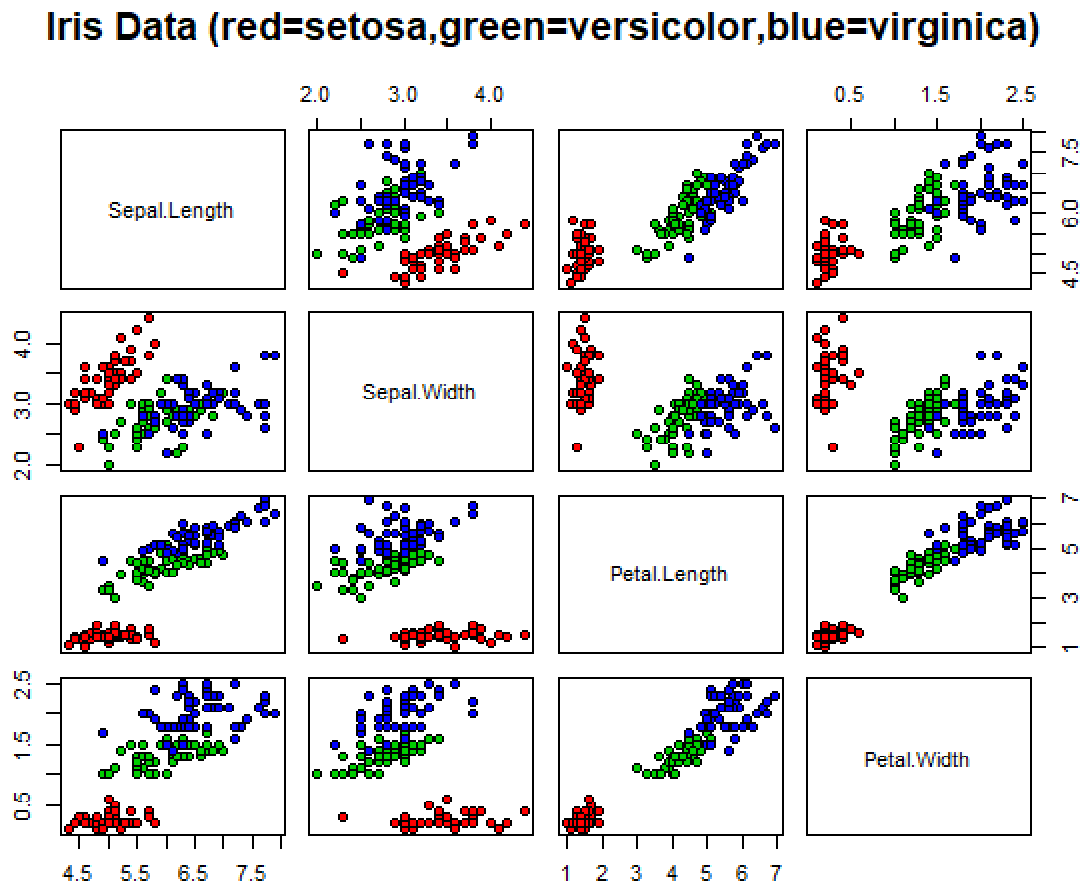
#Plot graph
pairs(iris[1:4], main="Iris Data (red=setosa,green=versicolor,blue=virginica)",
    pch=21, bg=c("red","green3","blue")[unclass(iris$Species)])



## TRAINING NAIVE BAYES:

```
# training a naive Bayes model
index = sample(nrow(iris), floor(nrow(iris) * 0.7)) #70/30 split.
train = iris[index,]
test = iris[-index,]
xTrain = train[,-5] # removing y-outcome variable.
yTrain = train$Species # only y.
xTest = test[,-5]
yTest = test$Species

# nb - tells to use naive bayes
# cv - cross validation
model = train(xTrain,yTrain,'nb',trControl=trainControl(method='cv',number=10))
model

## table() gives frequency table, prop.table() gives freq% table.
prop.table(table(predict(model$finalModel,xTest)$class,yTest))
```

```
> # training a naive Bayes model
> index = sample(nrow(iris), floor(nrow(iris) * 0.7)) #70/30 split.
> train = iris[index,]
> test = iris[-index,]
> xTrain = train[,-5] # removing y-outcome variable.
> yTrain = train$Species # only y.
> xTest = test[,-5]
> yTest = test$Species
>
> # nb - tells to use naive bayes
> # cv - cross validation
> model = train(xTrain,yTrain,'nb',trControl=trainControl(method='cv',number=10))
> model
Naive Bayes

105 samples
  4 predictor
  3 classes: 'setosa', 'versicolor', 'virginica'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 94, 95, 94, 94, 94, 96, ...
Resampling results across tuning parameters:

  usekernel  Accuracy   Kappa
  FALSE      0.9507071  0.9257564
   TRUE      0.9523737  0.9281818

Tuning parameter 'fL' was held constant at a value of 0
Tuning parameter 'adjust' was
 held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were fL = 0, usekernel = TRUE and adjust = 1.
>
> ## table() gives frequency table, prop.table() gives freq% table.
> prop.table(table(predict(model$finalModel,xTest)$class,yTest))
            yTest
                  setosa versicolor  virginica
  setosa     0.33333333 0.00000000 0.00000000
  versicolor 0.00000000 0.31111111 0.04444444
  virginica  0.00000000 0.02222222 0.28888889
```


Iris Data (red=setosa,green=versicolor,blue=virginica)
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
2.0 3.0 4.0
0.5 1.5 2.5
4.5 6.0 7.5
2.0 3.0 4.0
1 3 5 7
0.5 1.5 2.5
4.5 5.5 6.5 7.5
1 2 3 4 5 6 7

## B) IMPLEMENT K NEAREST NEIGHBOUR

### LOADING DATASET:

#K nearest Neighbour
df <- data(iris) ##load data
head(iris) ## see the structure

##Generate a random number that is 90% of the total number of rows in dataset.
ran <- sample(1:nrow(iris), 0.9 * nrow(iris))
ran

```
> #K nearest Neighbour
> df <- data(iris) ##load data
> head(iris) ## see the structure
  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
>
> ##Generate a random number that is 90% of the total number of rows in dataset.
> ran <- sample(1:nrow(iris), 0.9 * nrow(iris))
> ran
  [1]  13 145 149  63  27 119 112  87  94  51 107 114 136  19  65  74  41 100  25   9  99   4
 [23]  64 113  37  15 140  92  50 111  72 134 108  43  49   1  69  54  59 105  36 104 118 121
 [45] 110  39  42 138  40  24 141  30  86 103  21 123  81   6  48  38   2 101  58  61  78 128
 [67]   5  47 130  60  28  84 124  14  75  93  62  70  45  11  53 144  17  56  22  55 146  68
 [89] 150  32  33 127  98   3 148  66 132 117  77  90  85 133  44  95  34  35  83   7  67  12
[111]  71  16 125  79  82  76  57  18  97 135 102  31 120  26  20  88 143 122  29 129  91  52
[133] 139  73 115
```

### NORMILIZATION:

##the normalization function is created
nor <-function(x) { (x -min(x))/(max(x)-min(x)) }
##Run nomalization on first 4 coulumns of dataset because they are the predictors
iris_norm <- as.data.frame(lapply(iris[,c(1,2,3,4)], nor))
iris_norm

```
> ##the normalization function is created
> nor <-function(x) { (x -min(x))/(max(x)-min(x)) }
> ##Run nomalization on first 4 coulumns of dataset because they are the predictors
> iris_norm <- as.data.frame(lapply(iris[,c(1,2,3,4)], nor))
> iris_norm
   Sepal.Length Sepal.Width Petal.Length Petal.Width
1    0.22222222  0.62500000   0.06779661  0.04166667
2    0.16666667  0.41666667   0.06779661  0.04166667
3    0.11111111  0.50000000   0.05084746  0.04166667
4    0.08333333  0.45833333   0.08474576  0.04166667
5    0.19444444  0.66666667   0.06779661  0.04166667
6    0.30555556  0.79166667   0.11864407  0.12500000
7    0.08333333  0.58333333   0.06779661  0.08333333
8    0.19444444  0.58333333   0.08474576  0.04166667
9    0.02777778  0.37500000   0.06779661  0.04166667
10   0.16666667  0.45833333   0.08474576  0.00000000
11   0.30555556  0.70833333   0.08474576  0.04166667
12   0.13888889  0.58333333   0.10169492  0.04166667
13   0.13888889  0.41666667   0.06779661  0.00000000
14   0.00000000  0.41666667   0.01694915  0.00000000
15   0.41666667  0.83333333   0.03389831  0.04166667
16   0.38888889  1.00000000   0.08474576  0.12500000
17   0.30555556  0.79166667   0.05084746  0.12500000
18   0.22222222  0.62500000   0.06779661  0.08333333
19   0.38888889  0.75000000   0.11864407  0.08333333
20   0.22222222  0.75000000   0.08474576  0.08333333
21   0.30555556  0.58333333   0.11864407  0.04166667
22   0.22222222  0.70833333   0.08474576  0.12500000
23   0.08333333  0.66666667   0.00000000  0.04166667
24   0.22222222  0.54166667   0.11864407  0.16666667
25   0.13888889  0.58333333   0.15254237  0.04166667
26   0.19444444  0.41666667   0.10169492  0.04166667
27   0.19444444  0.58333333   0.10169492  0.12500000
28   0.25000000  0.62500000   0.08474576  0.04166667
29   0.25000000  0.58333333   0.06779661  0.04166667
30   0.11111111  0.50000000   0.10169492  0.04166667
31   0.13888889  0.45833333   0.10169492  0.04166667
32   0.30555556  0.58333333   0.08474576  0.12500000
33   0.25000000  0.87500000   0.08474576  0.00000000
34   0.33333333  0.91666667   0.06779661  0.04166667
35   0.16666667  0.45833333   0.08474576  0.04166667
36   0.19444444  0.50000000   0.03389831  0.04166667
37   0.33333333  0.62500000   0.05084746  0.04166667
38   0.16666667  0.66666667   0.06779661  0.00000000
39   0.02777778  0.41666667   0.05084746  0.04166667
40   0.22222222  0.58333333   0.08474576  0.04166667
41   0.19444444  0.62500000   0.05084746  0.08333333
42   0.05555556  0.12500000   0.05084746  0.08333333
43   0.02777778  0.50000000   0.05084746  0.04166667
44   0.19444444  0.62500000   0.10169492  0.20833333
45   0.22222222  0.75000000   0.15254237  0.12500000
46   0.13888889  0.41666667   0.06779661  0.08333333
47   0.22222222  0.75000000   0.10169492  0.04166667
48   0.08333333  0.50000000   0.06779661  0.04166667
49   0.27777778  0.70833333   0.08474576  0.04166667
```

| | | | | |
|---|---|---|---|---|
| 50 | 0.19444444 | 0.54166667 | 0.06779661 | 0.04166667 |
| 51 | 0.75000000 | 0.50000000 | 0.62711864 | 0.54166667 |
| 52 | 0.58333333 | 0.50000000 | 0.59322034 | 0.58333333 |
| 53 | 0.72222222 | 0.45833333 | 0.66101695 | 0.58333333 |
| 54 | 0.33333333 | 0.12500000 | 0.50847458 | 0.50000000 |
| 55 | 0.61111111 | 0.33333333 | 0.61016949 | 0.58333333 |
| 56 | 0.38888889 | 0.33333333 | 0.59322034 | 0.50000000 |
| 57 | 0.55555556 | 0.54166667 | 0.62711864 | 0.62500000 |
| 58 | 0.16666667 | 0.16666667 | 0.38983051 | 0.37500000 |
| 59 | 0.63888889 | 0.37500000 | 0.61016949 | 0.50000000 |
| 60 | 0.25000000 | 0.29166667 | 0.49152542 | 0.54166667 |
| 61 | 0.19444444 | 0.00000000 | 0.42372881 | 0.37500000 |
| 62 | 0.44444444 | 0.41666667 | 0.54237288 | 0.58333333 |
| 63 | 0.47222222 | 0.08333333 | 0.50847458 | 0.37500000 |
| 64 | 0.50000000 | 0.37500000 | 0.62711864 | 0.54166667 |
| 65 | 0.36111111 | 0.37500000 | 0.44067797 | 0.50000000 |
| 66 | 0.66666667 | 0.45833333 | 0.57627119 | 0.54166667 |
| 67 | 0.36111111 | 0.41666667 | 0.59322034 | 0.58333333 |
| 68 | 0.41666667 | 0.29166667 | 0.52542373 | 0.37500000 |
| 69 | 0.52777778 | 0.08333333 | 0.59322034 | 0.58333333 |
| 70 | 0.36111111 | 0.20833333 | 0.49152542 | 0.41666667 |
| 71 | 0.44444444 | 0.50000000 | 0.64406780 | 0.70833333 |
| 72 | 0.50000000 | 0.33333333 | 0.50847458 | 0.50000000 |
| 73 | 0.55555556 | 0.20833333 | 0.66101695 | 0.58333333 |
| 74 | 0.50000000 | 0.33333333 | 0.62711864 | 0.45833333 |
| 75 | 0.58333333 | 0.37500000 | 0.55932203 | 0.50000000 |
| 76 | 0.63888889 | 0.41666667 | 0.57627119 | 0.54166667 |
| 77 | 0.69444444 | 0.33333333 | 0.64406780 | 0.54166667 |
| 78 | 0.66666667 | 0.41666667 | 0.67796610 | 0.66666667 |
| 79 | 0.47222222 | 0.37500000 | 0.59322034 | 0.58333333 |
| 80 | 0.38888889 | 0.25000000 | 0.42372881 | 0.37500000 |
| 81 | 0.33333333 | 0.16666667 | 0.47457627 | 0.41666667 |
| 82 | 0.33333333 | 0.16666667 | 0.45762712 | 0.37500000 |
| 83 | 0.41666667 | 0.29166667 | 0.49152542 | 0.45833333 |
| 84 | 0.47222222 | 0.29166667 | 0.69491525 | 0.62500000 |
| 85 | 0.30555556 | 0.41666667 | 0.59322034 | 0.58333333 |
| 86 | 0.47222222 | 0.58333333 | 0.59322034 | 0.62500000 |
| 87 | 0.66666667 | 0.45833333 | 0.62711864 | 0.58333333 |
| 88 | 0.55555556 | 0.12500000 | 0.57627119 | 0.50000000 |
| 89 | 0.36111111 | 0.41666667 | 0.52542373 | 0.50000000 |
| 90 | 0.33333333 | 0.20833333 | 0.50847458 | 0.50000000 |
| 91 | 0.33333333 | 0.25000000 | 0.57627119 | 0.45833333 |
| 92 | 0.50000000 | 0.41666667 | 0.61016949 | 0.54166667 |
| 93 | 0.41666667 | 0.25000000 | 0.50847458 | 0.45833333 |
| 94 | 0.19444444 | 0.12500000 | 0.38983051 | 0.37500000 |
| 95 | 0.36111111 | 0.29166667 | 0.54237288 | 0.50000000 |
| 96 | 0.38888889 | 0.41666667 | 0.54237288 | 0.45833333 |
| 97 | 0.38888889 | 0.37500000 | 0.54237288 | 0.50000000 |
| 98 | 0.52777778 | 0.37500000 | 0.55932203 | 0.50000000 |
| 99 | 0.22222222 | 0.20833333 | 0.33898305 | 0.41666667 |
| 100 | 0.38888889 | 0.33333333 | 0.52542373 | 0.50000000 |
| 101 | 0.55555556 | 0.54166667 | 0.84745763 | 1.00000000 |
| 102 | 0.41666667 | 0.29166667 | 0.69491525 | 0.75000000 |
| 103 | 0.77777778 | 0.41666667 | 0.83050847 | 0.83333333 |

```
104   0.55555556  0.37500000  0.77966102  0.70833333
105   0.61111111  0.41666667  0.81355932  0.87500000
106   0.91666667  0.41666667  0.94915254  0.83333333
107   0.16666667  0.20833333  0.59322034  0.66666667
108   0.83333333  0.37500000  0.89830508  0.70833333
109   0.66666667  0.20833333  0.81355932  0.70833333
110   0.80555556  0.66666667  0.86440678  1.00000000
111   0.61111111  0.50000000  0.69491525  0.79166667
112   0.58333333  0.29166667  0.72881356  0.75000000
113   0.69444444  0.41666667  0.76271186  0.83333333
114   0.38888889  0.20833333  0.67796610  0.79166667
115   0.41666667  0.33333333  0.69491525  0.95833333
116   0.58333333  0.50000000  0.72881356  0.91666667
117   0.61111111  0.41666667  0.76271186  0.70833333
118   0.94444444  0.75000000  0.96610169  0.87500000
119   0.94444444  0.25000000  1.00000000  0.91666667
120   0.47222222  0.08333333  0.67796610  0.58333333
121   0.72222222  0.50000000  0.79661017  0.91666667
122   0.36111111  0.33333333  0.66101695  0.79166667
123   0.94444444  0.33333333  0.96610169  0.79166667
124   0.55555556  0.29166667  0.66101695  0.70833333
125   0.66666667  0.54166667  0.79661017  0.83333333
126   0.80555556  0.50000000  0.84745763  0.70833333
127   0.52777778  0.33333333  0.64406780  0.70833333
128   0.50000000  0.41666667  0.66101695  0.70833333
129   0.58333333  0.33333333  0.77966102  0.83333333
130   0.80555556  0.41666667  0.81355932  0.62500000
131   0.86111111  0.33333333  0.86440678  0.75000000
132   1.00000000  0.75000000  0.91525424  0.79166667
133   0.58333333  0.33333333  0.77966102  0.87500000
134   0.55555556  0.33333333  0.69491525  0.58333333
135   0.50000000  0.25000000  0.77966102  0.54166667
136   0.94444444  0.41666667  0.86440678  0.91666667
137   0.55555556  0.58333333  0.77966102  0.95833333
138   0.58333333  0.45833333  0.76271186  0.70833333
139   0.47222222  0.41666667  0.64406780  0.70833333
140   0.72222222  0.45833333  0.74576271  0.83333333
141   0.66666667  0.45833333  0.77966102  0.95833333
142   0.72222222  0.45833333  0.69491525  0.91666667
143   0.41666667  0.29166667  0.69491525  0.75000000
144   0.69444444  0.50000000  0.83050847  0.91666667
145   0.66666667  0.54166667  0.79661017  1.00000000
146   0.66666667  0.41666667  0.71186441  0.91666667
147   0.55555556  0.20833333  0.67796610  0.75000000
148   0.61111111  0.41666667  0.71186441  0.79166667
149   0.52777778  0.58333333  0.74576271  0.91666667
150   0.44444444  0.41666667  0.69491525  0.70833333
```

**EXTRACTING TRAINING SET:**

##extract training set
iris_train <- iris_norm[ran,]
iris_train

```
> ##extract training set
> iris_train <- iris_norm[ran,]
> iris_train
    Sepal.Length Sepal.Width Petal.Length Petal.Width
13    0.13888889  0.41666667   0.06779661  0.00000000
145   0.66666667  0.54166667   0.79661017  1.00000000
149   0.52777778  0.58333333   0.74576271  0.91666667
63    0.47222222  0.08333333   0.50847458  0.37500000
27    0.19444444  0.58333333   0.10169492  0.12500000
119   0.94444444  0.25000000   1.00000000  0.91666667
112   0.58333333  0.29166667   0.72881356  0.75000000
87    0.66666667  0.45833333   0.62711864  0.58333333
94    0.19444444  0.12500000   0.38983051  0.37500000
51    0.75000000  0.50000000   0.62711864  0.54166667
107   0.16666667  0.20833333   0.59322034  0.66666667
114   0.38888889  0.20833333   0.67796610  0.79166667
136   0.94444444  0.41666667   0.86440678  0.91666667
19    0.38888889  0.75000000   0.11864407  0.08333333
65    0.36111111  0.37500000   0.44067797  0.50000000
74    0.50000000  0.33333333   0.62711864  0.45833333
41    0.19444444  0.62500000   0.05084746  0.08333333
100   0.38888889  0.33333333   0.52542373  0.50000000
25    0.13888889  0.58333333   0.15254237  0.04166667
9     0.02777778  0.37500000   0.06779661  0.04166667
99    0.22222222  0.20833333   0.33898305  0.41666667
4     0.08333333  0.45833333   0.08474576  0.04166667
64    0.50000000  0.37500000   0.62711864  0.54166667
113   0.69444444  0.41666667   0.76271186  0.83333333
37    0.33333333  0.62500000   0.05084746  0.04166667
15    0.41666667  0.83333333   0.03389831  0.04166667
140   0.72222222  0.45833333   0.74576271  0.83333333
92    0.50000000  0.41666667   0.61016949  0.54166667
50    0.19444444  0.54166667   0.06779661  0.04166667
111   0.61111111  0.50000000   0.69491525  0.79166667
72    0.50000000  0.33333333   0.50847458  0.50000000
134   0.55555556  0.33333333   0.69491525  0.58333333
108   0.83333333  0.37500000   0.89830508  0.70833333
43    0.02777778  0.50000000   0.05084746  0.04166667
49    0.27777778  0.70833333   0.08474576  0.04166667
1     0.22222222  0.62500000   0.06779661  0.04166667
69    0.52777778  0.08333333   0.59322034  0.58333333
54    0.33333333  0.12500000   0.50847458  0.50000000
59    0.63888889  0.37500000   0.61016949  0.50000000
105   0.61111111  0.41666667   0.81355932  0.87500000
36    0.19444444  0.50000000   0.03389831  0.04166667
104   0.55555556  0.37500000   0.77966102  0.70833333
118   0.94444444  0.75000000   0.96610169  0.87500000
121   0.72222222  0.50000000   0.79661017  0.91666667
110   0.80555556  0.66666667   0.86440678  1.00000000
39    0.02777778  0.41666667   0.05084746  0.04166667
42    0.05555556  0.12500000   0.05084746  0.08333333
138   0.58333333  0.45833333   0.76271186  0.70833333
40    0.22222222  0.58333333   0.08474576  0.04166667
24    0.22222222  0.54166667   0.11864407  0.16666667
141   0.66666667  0.45833333   0.77966102  0.95833333
```

**EXTRACT TESTING DATASET:**

```
##extract testing set
iris_test <- iris_norm[-ran,]
iris_test
```

```
> ##extract testing set
> iris_test <- iris_norm[-ran,]
> iris_test
    Sepal.Length Sepal.Width Petal.Length Petal.Width
8     0.19444444   0.5833333   0.08474576  0.04166667
10    0.16666667   0.4583333   0.08474576  0.00000000
23    0.08333333   0.6666667   0.00000000  0.04166667
46    0.13888889   0.4166667   0.06779661  0.08333333
80    0.38888889   0.2500000   0.42372881  0.37500000
89    0.36111111   0.4166667   0.52542373  0.50000000
96    0.38888889   0.4166667   0.54237288  0.45833333
106   0.91666667   0.4166667   0.94915254  0.83333333
109   0.66666667   0.2083333   0.81355932  0.70833333
116   0.58333333   0.5000000   0.72881356  0.91666667
126   0.80555556   0.5000000   0.84745763  0.70833333
131   0.86111111   0.3333333   0.86440678  0.75000000
137   0.55555556   0.5833333   0.77966102  0.95833333
142   0.72222222   0.4583333   0.69491525  0.91666667
147   0.55555556   0.2083333   0.67796610  0.75000000
```

```
##extract 5th column of train dataset because it will be used as 'cl' argument in knn function.
iris_target_category <- iris[ran,5]
iris_target_category
```

```
> ##extract 5th column of train dataset because it will be used as 'cl' argument in knn function.
> iris_target_category <- iris[ran,5]
> iris_target_category
  [1] setosa     virginica  virginica  versicolor setosa     virginica  virginica  versicolor
  [9] versicolor versicolor virginica  virginica  virginica  setosa     versicolor versicolor
 [17] setosa     versicolor setosa     setosa     versicolor setosa     versicolor virginica
 [25] setosa     setosa     virginica  versicolor setosa     virginica  versicolor virginica
 [33] virginica  setosa     setosa     setosa     versicolor versicolor versicolor virginica
 [41] setosa     virginica  virginica  virginica  virginica  setosa     setosa     virginica
 [49] setosa     setosa     virginica  setosa     versicolor virginica  setosa     virginica
 [57] versicolor setosa     setosa     setosa     setosa     virginica  versicolor versicolor
 [65] versicolor virginica  setosa     setosa     virginica  versicolor setosa     versicolor
 [73] virginica  setosa     versicolor versicolor versicolor versicolor setosa     setosa
 [81] versicolor virginica  setosa     versicolor setosa     versicolor virginica  versicolor
 [89] virginica  setosa     setosa     virginica  versicolor setosa     virginica  versicolor
 [97] virginica  virginica  versicolor versicolor versicolor virginica  setosa     versicolor
[105] setosa     setosa     versicolor setosa     versicolor setosa     versicolor setosa
[113] virginica  versicolor versicolor versicolor versicolor setosa     versicolor virginica
[121] virginica  setosa     virginica  setosa     setosa     versicolor virginica  virginica
[129] setosa     virginica  versicolor versicolor virginica  versicolor virginica
Levels: setosa versicolor virginica
```

##extract 5th column if test dataset to measure the accuracy

iris_test_category <- iris[-ran,5]

iris_test_category

```
> ##extract 5th column if test dataset to measure the accuracy
> iris_test_category <- iris[-ran,5]
> iris_test_category
 [1] setosa     setosa     setosa     setosa     versicolor versicolor versicolor virginica
 [9] virginica  virginica  virginica  virginica  virginica  virginica  virginica
Levels: setosa versicolor virginica
```

**EXECUTE KNN FUNCTION:**

##load the package class
library(class)

##run knn function
pr <- knn(iris_train,iris_test,cl=iris_target_category,k=13)
pr

```
> ##load the package class
> library(class)
>
> ##run knn function
> pr <- knn(iris_train,iris_test,cl=iris_target_category,k=13)
> pr
 [1] setosa     setosa     setosa     setosa     versicolor versicolor versicolor virginica
 [9] virginica  virginica  virginica  virginica  virginica  virginica  virginica
Levels: setosa versicolor virginica
```

**CREATE CONFUSION MATRIX:**

##create confusion matrix
tab <- table(pr,iris_test_category)
tab

```
> ##create confusion matrix
> tab <- table(pr,iris_test_category)
> tab
            iris_test_category
pr           setosa versicolor virginica
  setosa          4          0         0
  versicolor      0          3         0
  virginica       0          0         8
```

**CHECKING ACCURACY:**

accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
accuracy(tab)

```
> ##this function divides the correct predictions by total number of predictions that tell us how
 it is accurate
> accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
> accuracy(tab)
[1] 100
```

## C) IMPLEMENT K-MEANS CLUSTERING

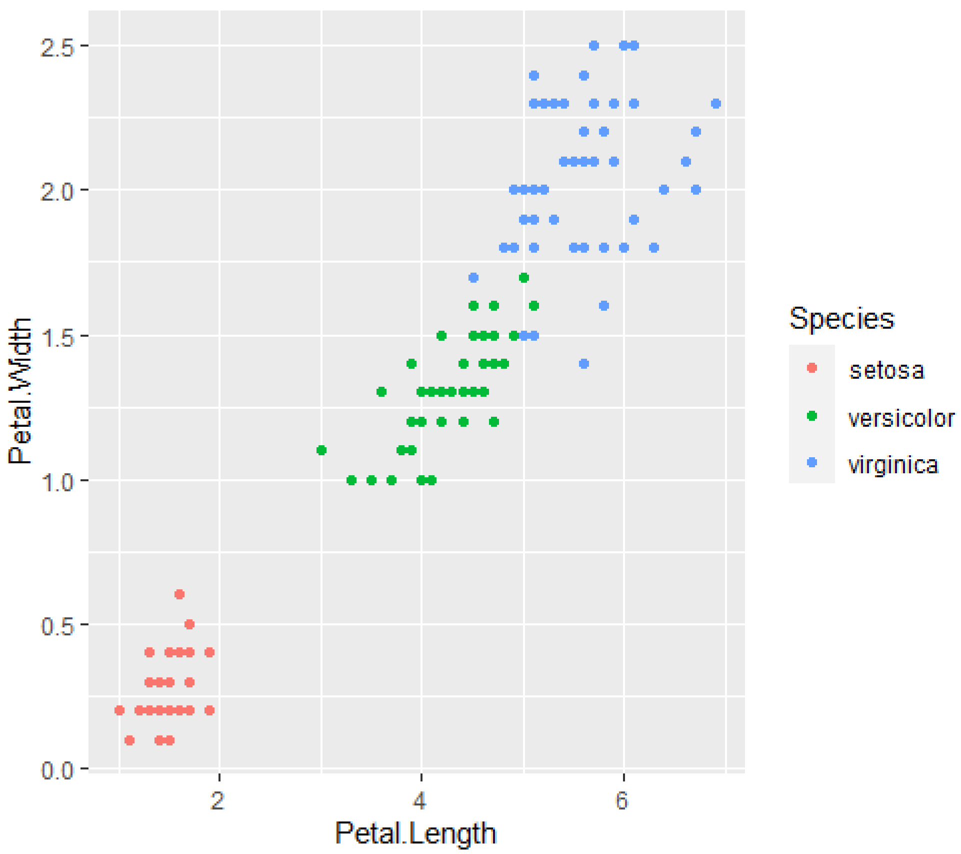
### LOADING DATASET:

```
#K-Means clustering
head(iris)
```

```
> 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
```

### LOADING LIBRARY FOR PLOTING GRAPH:

```
library(ggplot2)
ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) + geom_point()
```



**EXECUTING K-MENAS AND PLOTTING GRAPH:**

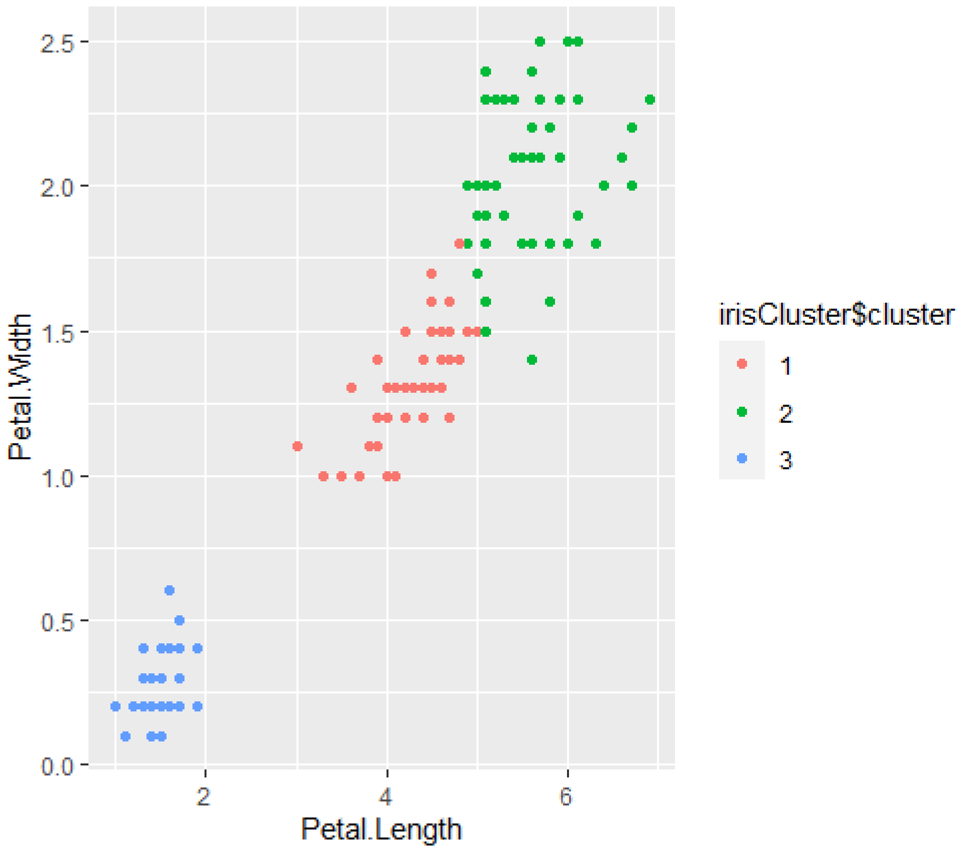
#Setting a seed in R means to initialize a pseudorandom number generator.
set.seed(20)

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

table(irisCluster$cluster, iris$Species)

irisCluster$cluster <- as.factor(irisCluster$cluster)
ggplot(iris, aes(Petal.Length, Petal.Width, color = irisCluster$cluster)) + geom_point()

```
> #Setting a seed in R means to initialize a pseudorandom number generator.
> set.seed(20)
>
> #Executing Kmeans
> irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)
> irisCluster
K-means clustering with 3 clusters of sizes 52, 48, 50

Cluster means:
  Petal.Length Petal.Width
1     4.269231    1.342308
2     5.595833    2.037500
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 3 3 3 3 3 3 3 3 3 3 3
 [46] 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1
 [91] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 2 2 2 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
[136] 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2

Within cluster sum of squares by cluster:
[1] 13.05769 16.29167  2.02200
 (between_SS / total_SS =  94.3 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
>
> table(irisCluster$cluster, iris$Species)

    setosa versicolor virginica
  1      0         48         4
  2      0          2        46
  3     50          0         0
>
> irisCluster$cluster <- as.factor(irisCluster$cluster)
> ggplot(iris, aes(Petal.Length, Petal.Width, color = irisCluster$cluster)) + geom_point()
>
```




**CONCLUSION:**

From this practical, I have learned how to implement naive bayes, k – nn, k – means clustering in R.