PRACTICAL – 8

Aim: Implementation and analysis of Classification algorithms like Naive Bayesian, K-Nearest Neighbor, ID3, C4.5

Theory:

Naïve Bayesian:

Naive Bayes uses the Bayes' Theorem and assumes that all predictors are independent. In other words, this classifier assumes that the presence of one particular feature in a class doesn't affect the presence of another one.

Here's an example: you'd consider fruit to be orange if it is round, orange, and is of around 3.5 inches in diameter. Now, even if these features require each other to exist, they all contribute independently to your assumption that this particular fruit is orange. That's why this algorithm has 'Naive' in its name.

Building the Naive Bayes model is quite simple and helps you in working with vast datasets. Moreover, this equation is popular for beating many advanced classification techniques in terms of performance.

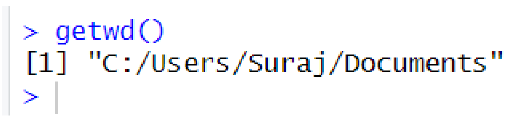
Here's the equation for Naive Bayes:

P (c|x) = P(x|c) P(c) / P(x)

P(c|x) = P(x1 | c) x P(x2 | c) x … P(xn | c) x P(c)

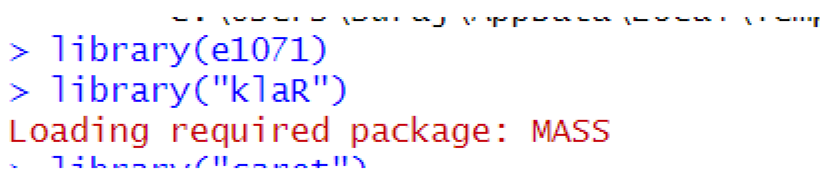
Here, P (c|x) is the posterior probability according to the predictor (x) for the class(c). P(c) is the prior probability of the class, P(x) is the prior probability of the predictor, and P(x|c) is the probability of the predictor for the particular class(c).

Setting working directory

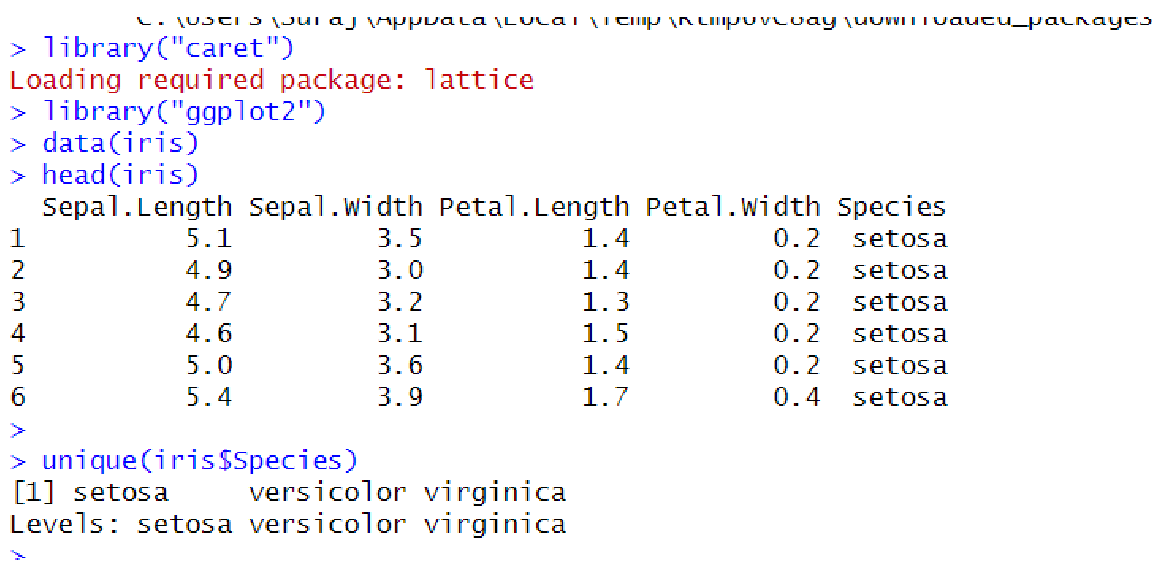


```
> getwd()
[1] "C:/Users/Suraj/Documents"
> 
```

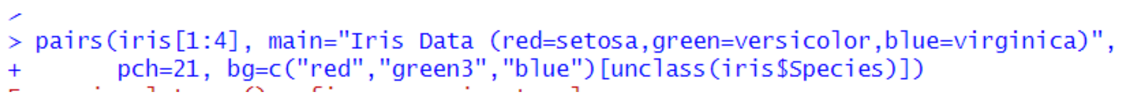
Installing packages: e1071, klaR, caret, lattice, ggplot2



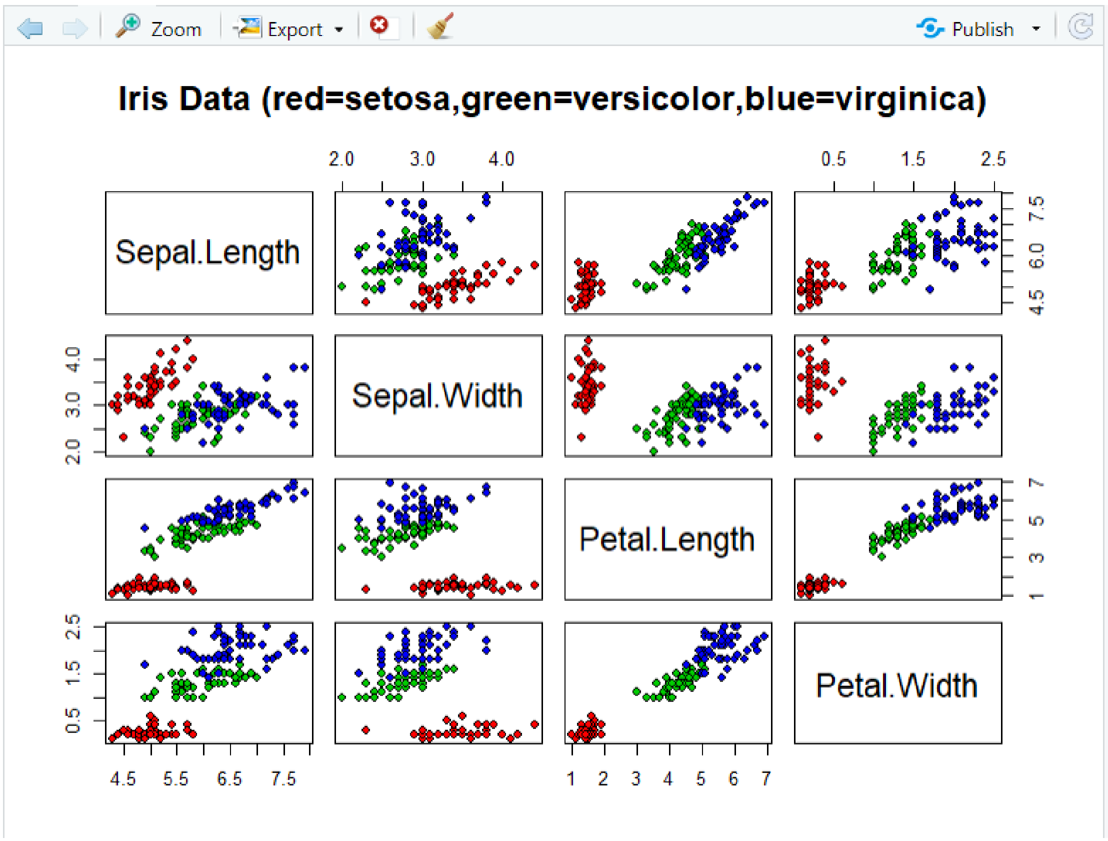
```
> library(e1071)
> library("klaR")
Loading required package: MASS
```



```
> library("caret")
Loading required package: lattice
> library("ggplot2")
> 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
>
> unique(iris$Species)
[1] setosa     versicolor virginica
Levels: setosa versicolor virginica
```



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



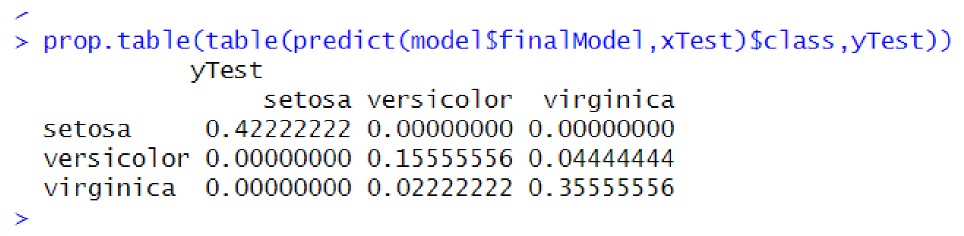
```
>
> index = sample(nrow(iris), floor(nrow(iris) * 0.7)) #70/30 split.
> train = iris[index,]
> test = iris[-index,]
> xTrain = train[,-5] # removing y-outcome variable.
> yTrain = train$Species # only y.
> xTest = test[,-5]
> yTest = test$Species
> model = train(xTrain,yTrain,'nb',trControl=trainControl(method='cv',number=10))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

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

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

  usekernel  Accuracy   Kappa
  FALSE      0.9709091  0.9559400
   TRUE      0.9609091  0.9410216

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 = FALSE and adjust = 1.
>
```



```
> prop.table(table(predict(model$finalModel,xTest)$class,yTest))
            yTest
                 setosa versicolor  virginica
  setosa     0.42222222 0.00000000 0.00000000
  versicolor 0.00000000 0.15555556 0.04444444
  virginica  0.00000000 0.02222222 0.35555556
>
```

K- Nearest neighbour Algorithm:

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. It stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems. It 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, then it classifies that data into a category that is much similar to the new data.

Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So, for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs' images and based on the most similar features it will put it in either cat or dog category.

Working:

The K-NN working can be explained on the basis of the below algorithm:

a) Step-1: Select the number K of the neighbours

b) Step-2: Calculate the Euclidean distance of K number of neighbours

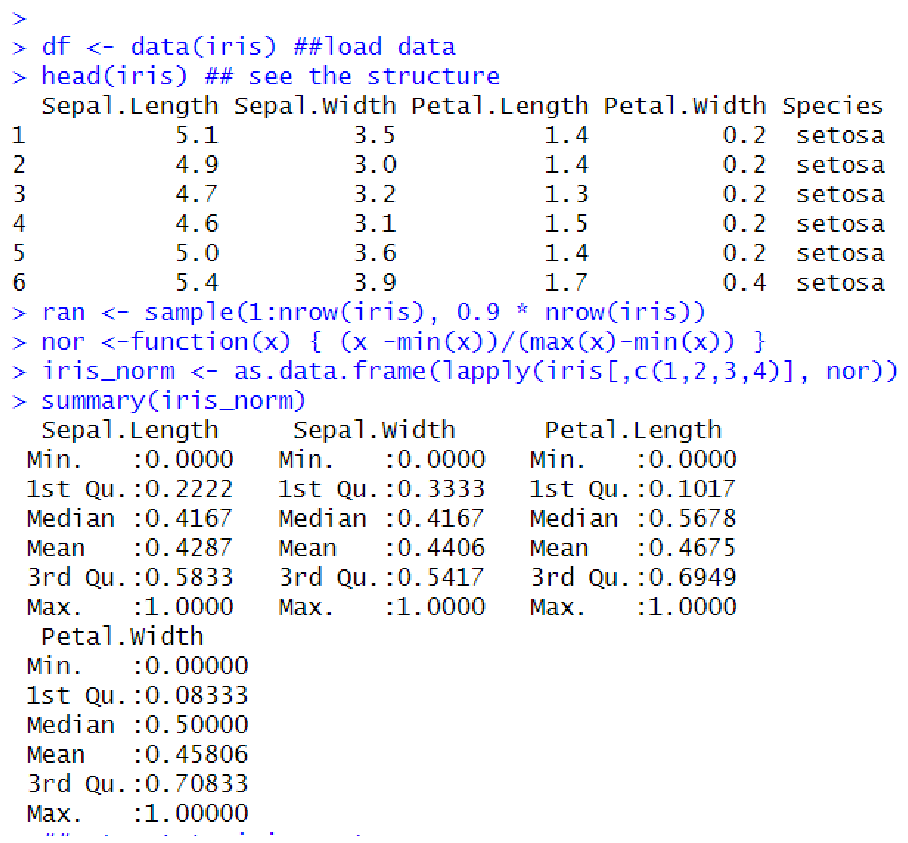
c) Step-3: Take the K nearest neighbours as per the calculated Euclidean distance.

d) Step-4: Among these k neighbours, count the number of the data points in each category.

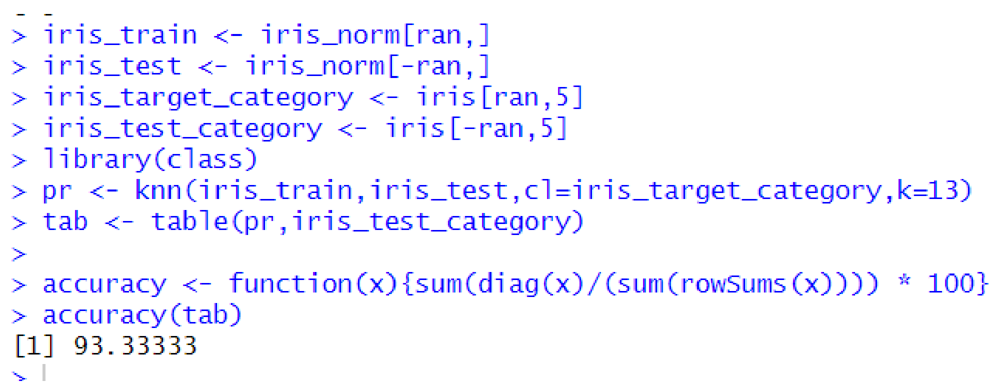
e) Step-5: Assign the new data points to that category for which the number of the neighbour is maximum.

f) Step-6: Our model is ready.

Loading data and printing it's structure.



```
>
> 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
> ran <- sample(1:nrow(iris), 0.9 * nrow(iris))
> nor <-function(x) { (x -min(x))/(max(x)-min(x)) }
> iris_norm <- as.data.frame(lapply(iris[,c(1,2,3,4)], nor))
> summary(iris_norm)
  Sepal.Length      Sepal.Width      Petal.Length
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.2222   1st Qu.:0.3333   1st Qu.:0.1017
 Median :0.4167   Median :0.4167   Median :0.5678
 Mean   :0.4287   Mean   :0.4406   Mean   :0.4675
 3rd Qu.:0.5833   3rd Qu.:0.5417   3rd Qu.:0.6949
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
  Petal.Width
 Min.   :0.00000
 1st Qu.:0.08333
 Median :0.50000
 Mean   :0.45806
 3rd Qu.:0.70833
 Max.   :1.00000
```



```
> iris_train <- iris_norm[ran,]
> iris_test <- iris_norm[-ran,]
> iris_target_category <- iris[ran,5]
> iris_test_category <- iris[-ran,5]
> library(class)
> pr <- knn(iris_train,iris_test,cl=iris_target_category,k=13)
> tab <- table(pr,iris_test_category)
>
> accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
> accuracy(tab)
[1] 93.33333
>
```

Conclusion:

I have successfully implemented and analysed different classification algorithms.