Assignment No: 10

**Title:** Naive Bayes Algorithm for Classification on Pima Indians Diabetes.

**Theory:**

Naive Bayes is a simple, yet effective and commonly-used, machine learning classifier. It is a probabilistic classifier that makes classifications using the Maximum A Posteriori decision rule in a Bayesian setting. It can also be represented using a very simple Bayesian network. Naive Bayes classifiers have been especially popular for text classification, and are a traditional solution for problems such as spam detection.

Windows/Linux Operating Systems, RStudio, jdk.

**Applications**

Real time Prediction:

* **Naive Bayes** is an eager learning classifier and it is sure fast. Thus, it could be used for

making predictions in real time.

* **Multi class Prediction**: This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable. Text classification/ Spam Filtering/ Sentiment Analysis: Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
* **Recommendation System**: Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

**Program:**

**import** csv

**import** pandas **as** pd

**import** numpy **as** no

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

**%matplotlib** inline

names**=**["NumTimesPrg", "PlGlcConc", "BloodP",

"SkinThick", "TwoHourSerIns", "BMI",

"DiPedFunc", "Age", "HasDiabetes"]

dataset**=**pd**.**read\_csv('pima-indians-diabetes.csv',names**=**names);

dataset**.**hist(bins**=**50, figsize**=**(20,15));

plt**.**show();

Load Data:

**def** loadCSV():

lines **=** csv**.**reader(open(r'pima-indians-diabetes.csv'))

data **=** list(lines)

data **=** data[1:]

**for** i **in** range(len(data)):

data[i] **=** [float(x) **for** x **in** data[i]]

**return** data

Split Data:

trainSize **=** int(len(data) **\*** splitRatio)

trainSet **=** []

testSet **=** list(data)

**while** len(trainSet) **<** trainSize:

index **=** random**.**randrange(len(testSet))

trainSet**.**append(testSet**.**pop())

**return** [trainSet, testSet]

**def** seperateByClass(data):

seperated **=** {}

**for** i **in** range(len(data)):

vector **=** data[i]

**if**(vector[**-**1] **not** **in** seperated):

seperated[vector[**-**1]] **=** []

seperated[vector[**-**1]]**.**append(vector)

**return** seperated

## Summarization of data:

**def** summarize(data):

summaries **=** [(mean(attribute), stddev(attribute)) **for** attribute **in** zip(**\***data)]

**del** summaries[**-**1]

**return** summaries

**def** summarizeByClass(data):

seperated **=** seperateByClass(data)

summaries **=** {}

**for** classValue, instances **in** seperated**.**items():

summaries[classValue] **=** summarize(instances)

**return** summaries

## Mathematical functions

**def** mean(data):

**return** sum(data)**/**float(len(data))

**def** stddev(data):

avg **=** mean(data)

var **=** (sum(pow(x**-**avg, 2) **for** x **in** data))**/**float(len(data)**-**1) *# Using Bassel Correction*

**return** math**.**sqrt(var)

**def** calculateProbability(data, mean, stddev):

exp **=** math**.**exp((**-**(math**.**pow(data**-**mean, 2)))**/**float(2**\***math**.**pow(stddev, 2)))

**return** 1**/**(math**.**sqrt(2**\***math**.**pi**\***math**.**pow(stddev, 2))) **\*** exp.

## Predictions and Accuracy

**def** calculateClassProbability(summaries, vector):

probabilities **=** {}

**for** classValue, classSummaries **in** summaries**.**items():

probabilities[classValue] **=** 1

**for** i **in** range(len(classSummaries)):

mean, stddev **=** classSummaries[i]

x **=** vector[i]

probabilities[classValue] **\*=** calculateProbability(x, mean, stddev)

**return** probabilities

**def** predict(summaries, vector):

probabilities **=** calculateClassProbability(summaries, vector)

bestLabel, bestProb **=** **None**, **-**1

**for** classValue, probability **in** probabilities**.**items():

**if** bestLabel **is** **None** **or** probability **>** bestProb:

bestProb **=** probability

bestLabel **=** classValue

**return** bestLabel

**def** getPredictions(summaries, testSet):

predictions **=** []

**for** i **in** range(len(testSet)):

result **=** predict(summaries, testSet[i])

predictions**.**append(result)

**return** predictions

**def** getAccuracy(testSet, predictions):

correct **=** 0

**for** x **in** range(len(testSet)):

**if** testSet[x][**-**1] **==** predictions[x]:

correct **+=** 1

**return** (correct**/**float(len(testSet)))**\***100.0

**def** main(splitRatio, verbose **=** 1):

data **=** loadCSV()

trainingSet, testSet **=** splitData(data, splitRatio)

**if** verbose **is** 1:

print("Split {0} rows into {1} training and {2} testing rows"**.**format(len(data), len(trainingSet), len(testSet)))

summaries **=** summarizeByClass(trainingSet)

predictions **=** getPredictions(summaries, testSet)

accuracy **=** getAccuracy(testSet, predictions)

**if** verbose **is** 1:

print("Accuracy : {0:.3f}%"**.**format(accuracy))

**return** accuracy

main(0.78)

**Output:**

Split 768 rows into 599 training and 169 testing rows

Accuracy : 74.556%

74.55621301775149