# Experiment No-6

AIM:

A program to implement the naive Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Description:

Naive bayes algorithm is a supervised learning algorithm, which is based on bayes theorem and used for solving classification problems.

It is probilistic classifier, which means it predicts on the basis of the probability of an object.

It is not a single algorithm but a family of algorithms where all of them share a common principle.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

P(h|D) is the probability of hypothesis h given data D. This is called the posterior probability

P(D|h) is probability of data d given hypothesis h was true.

P(h) prior Probability of h.

P(D) prior probability of D.

# Source Code

```
import csv
import random
import math
def loadCsv(filename)
    lines = csv.reader(open(filename,"r"))
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset

def splitDataset(dataset, splitRatio):
    trainsize = int(len(dataset) * splitRatio)
    trainset = []
    copy = list(dataset)
    While len(trainset) < trainsize:
        index = random.randrange(len(copy))
        trainset.append(copy.pop(index))
    return [trainset, copy]

def seperateByClass(dataset):
    Seperated = {}
    for i in range(len(dataset)):
        Vector = dataset[i]
        if (vector[-1] not in seperated):
            seperated[vector[-1]] = []
        Seperated[vector[-1]].append(vector)
def mean(numbers):
    return sum(numbers)/float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
```

```
variance = sum([pow(x-avg,2) for x in numbers])/
        float(len(numbers)-1)
    return math.sqrt(variance)

def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute))
for attribute in zip(*dataset)]
    del summaries[-1]
    return summaries
def summarizeByClass(dataset):
    seperated = seperateByClass(dataset)
    summaries = {}
    for classValue, instances in seperated.items():
        summaries[classValue] = summarize(instances)
    return summaries

def calculateClassProbabilities(summaries, inputVector
    probabilities = {}
    for classValue, classSummaries in summaries.
      items():
    probabilities[classValue] = 1
    for i in range (len(classSummaries)):
        mean, stdev = classSummaries[i]
        x = inputVector[i]
        probabilities[classValue] *= calculateProbabili
    (x, mean, stdev) return probabilities

def predict(summaries, inputvector):
    probabilities = calculateClassProbabilities(summa
      inputVector)
    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 i in range(len(testset)):
        if testset[i][-1] = predictions[i]:
            Correct += 1
    return (correct/float(len(testSet)))*100.0

def main():
    filename = 'data.csv'
    splitRatio = 0.67
    dataset = loadcsv(filename)
    trainingSet, testSet = splitDataset(dataset, splitRatio)
    print('Split {0} rows into train={1} and test={2}
rows'.format(len(dataset), len(trainingSet),
len(testset)))
    summaries = summarizeByclass(trainingset)
  predictions = getpredictions(Summaries, testSet)
  accuracy = getAccuracy(testset, predictions)
  print('Accuracy : {0} %'.format(accuracy))
main():
```

# Output

Split 306 rowes into train = 205 and test = 101 rows

Accuracy : 72.27722772277228 %.

# Experiment-7

AIM:

A naive Bayesian classifier model to perform built in java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

Description:

Load the training dataset from the CV file into a list of dictionaries, where each dictionary represents single row and attribute names (columns)

Determine the class variable for each instance in the training data set and add it as a new key value pair

Compute the prior probabilities for each class variable by counting the number of instances in the training dataset that belong to each class variable and dividing by the total number of instances.

For each attribute training data set, compute the conditional probabilities for each attribute value given each class variable by counting the number of instances in the training dataset

Load the test data sets from csv files into lists of dictionaries.

For each instance dataset, posterior probability for each class variable given the attribute values in the instance, using the Naive Bayesian.

Source code:

```
import pandas as pd
msg = pd.read_csv("document.csv", names=
  ["message", "label"])
print("Total Instances of Dataset: ", msg.shape[0])
msg["labelnum"] = msg.label.map({"pos": 1, "neg": 0})

X = msg.message
Y = msg.labelnum
from sklearn.model_selection import train_test_split
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y)
from sklearn.feature_extraction.text import CountVectorizer
count_v = countvectorizer()
Xtrain_dm = count_v.fit_transform(Xtrain)
Xtest_dm = count_v.transform(Xtest)
df = pd.DataFrame(Xtrain_dm.toarray(), columns=
  count_v.get_feature_names_out())
print(df[0:5])

from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(Xtrain_dm, Ytrain)
pred = clf.predict(Xtest_dm)

for doc, p in zip(Xtrain, pred):
    p = 'pos' if p == 1 else 'neg'
    print("%s -> %s" % (doc, p))
from Sklearn.metrics import accuracy_score, confusion_
  matrix, precision_score, recall_score
```

```
print ("Accuracy Metrics : \n")
print ("Accuracy : ", accuracy_score (ytest, pred))
print ("Recall : ", recall_score (ytest, pred))
print ("precision : ", precision_score (ytest, pred))
print ("Confusion Matrix : \n", confusion_matrix (ytest, pred))
```

## Output

Total Instances of Dataset : 18

```
   am  an  and  awesome  bael  best  boss  com'dance  deal  ...
0   0   0    0        0     0     0     0          1     0  ...
1   0   1    0        1     0     0     0          0     0  ...
2   0   0    0        0     0     0     0          0     0  ...
3   1   0    1        0     0     0     0          0     0  ...
4   1   0    0        0     0     0     0          0     0  ...
```

tomorrow view we went what will with work

```
   0   0   0   0   0   0   1   0  ...
0  0   0   0   0   0   0   0   0  ...
1  0   0   0   0   0   0   0   1  ...
2  0   0   0   0   1   0   0   0  ...
3  0   0   0   0   0   0   0   0  ...
4  0   0   0   0   0   0   0   0  ...
```

[5 rows x 44 columns]

I can't deal with this → neg
This is an awesome place → pos
What a great holiday → neg
I am sick and tired of this place → pos
I am tired of this stuff → pos

Accuracy Metrics :

Accuracy : 1.0
Recall : 1.0
precision : 1.0
Confusion Matrix :
[[2 0]
[0 3]]

# Experiment-8

## AIM:

A program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data set.

## Description:

Bayesian network is directed acyclic graph in which each edge corresponds to a cortional dependency, and each node corresponds to a unique random variable.

The directed acyclic graph is a set of random variables represented by nodes.

## Dataset:

The goal is to calculate the posterior conditional probability distribution of each of each of possible unobserved causes given the observed evidence P [Cause | Evidence]

| 1 | Age | Gerda | Family | diet | lifestyle | Cholestrol | heart disease |
|---|---|---|---|---|---|---|---|
| 2 | 0 | 0 | 1 | 1 | 3 | 0 | 1 |
| 3 | 0 | 1 | 1 | 1 | 3 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 2 | 1 | 1 |
| 5 | 4 | 0 | 1 | 1 | 3 | 2 | 0 |
| 6 | 3 | 1 | 1 | 0 | 0 | 2 | 0 |
| 7 | 2 | 0 | 1 | 1 | 1 | 0 | 1 |
| 8 | 4 | 0 | 1 | 0 | 2 | 0 | 1 |
| 9 | 0 | 0 | 1 | 1 | 3 | 0 | 1 |
| 10 | 3 | 1 | 1 | 0 | 0 | 2 | 0 |
| 11 | 1 | 1 | 0 | 0 | 0 | 2 | 1 |
| 12 | 4 | 1 | 0 | 1 | 2 | 0 | 1 |
| 13 | 4 | 0 | 1 | 1 | 3 | 2 | 0 |
| 14 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| 15 | 2 | 0 | 1 | 1 | 1 | 0 | 1 |
| 16 | 3 | 1 | 1 | 0 | 0 | 1 | 0 |
| 17 | 0 | 0 | 1 | 0 | 0 | 2 | 1 |
| 18 | 1 | 1 | 0 | 1 | 2 | 1 | 1 |

Source Code:

```
import pandas as pd
data = pd.read_csv("heartdisease.csv")
heart_disease = pd.DataFrame(data)
print(heart_disease)
from pgmpy.models import BayesianModel
model = BayesianModel([
('age', 'Lifestyle'), ('Gender', 'Lifestyle'), ('family', 'heart
disease'), ('diet', 'cholestrol'), ('Lifestyle', 'diet'),
('cholestrol', 'heartdisease'), ('diet', 'cholestrol')])

from pgmpy.estimators import MaximumLikelihoodEstimator
model.fit(heart_disease, estimator=MaximumLikelihoodEstimato

from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)

print('For age Enter {Supersenior citizen: 0, seniorcitizen: 1,
    Middle Aged: 2, Youth: 3, Teen: 4}')
print('For Gender Enter {Male: 0, Female: 1}')
print('For Family History Enter {yes: 1, NO: 0}')
print('For diet Enter {High: 0, Medium: 1}')
print('For Lifestyle Enter {Athlete: 0, Active: 1, Moderate: 2,
    Sedentary: 3}')

q = HeartDisease_infer.query(variables=['heartdisease'],
  evidence={'age': int(input('Enter age:')),
  'Gender': int(input('Enter Gender:')),
  'Family': int(input('Enter family histor:')),
  'diet': int(input('Enter diet:')),
  'Lifestyle': int(input('Enter Lifestyle:')),
 'Cholestrol': int(input('Enter cholestrd:'))})
```

# Experiment-9

AIM:

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using K-means algorithm. Compare the results of these two algorithms.

Description:

EM algorithm is a iterative approach that cycles between two modes. The first mode attempts to estimate the missing or latent variables called the estimation-step or E-step, the second mode attempts to optimize the parameters of the model to best explain the data, called maximization step.

Initially, a set of initial values are considered. A set of incomplete observed data is given to the system with assumption that observed data.

Use complete data generated in the preceding "Expectation" - step in order to update the values of parameters. It is basically used to update the hypothesis.

Now, in the next step, it is checked whether the values are converging or not, if yes, then stop otherwise repeat step-2 and step-3 until the convergence occurs.

Source Code:

```
from sklearn.cluster import KMeans
from sklearn . import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import Sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = load_iris()
# print(dataset)
X = pd.DataFrame(dataset.data)
X.columns = ['Sepal_length', 'sepal_width', 'petal_length',
  'petal_width']
y = pd.DataFrame(dataset.target)
y.columns = ['Targets']
# print(X)
plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])
# Real plot
plt.subplot(1,3,1)
plt.scatter(X.petal_length, X.petal_width, c=colormap
  [y.Targets], s=40)
plt.title('Real')

# K-plot
plt.subplot(1,3,2)
model = KMeans(n_clusters=3)
model.fit(X)
predY = np.choose(model.labels_, [0,1,2]).astype(np.int64)
plt.scatter(X.petal_length, X.petal_width, c=colormap
  [predY], s=40)
plt.title('KMeans')
```

```
# GM PLOT
scaler = preprocessing, StandardScaler()
scaler.fit(x)
xsa = scaler.transform(x)
xs = pd.DataFrame(xsa, columns = x.columns)
gmm = GaussianMixture(n_components = 3)
gmm.fit(xs)

y_Cluster_gmm = gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(x.petal_Length, x.petal_Width, c= colormap
  [y_cluster_gmm], s=40)
plt.title('GMM classification')
```

## Output

# Experiment - 10

AIM:

A program to implement K-Nearest Neighbour algorithm to the iris data set.

Description:

We are going to write a function, which will find the distance between two given 2-D points in the x-y plane. import numpy, to take help of numpy arrays for storing the coordinates. Finding the distance between two points will help in finding the nearest neighbor of input point, from existing points, classify the point to insert into set of existing points and then insert accordingly.

build a function find_nearest neighbours() to find nearest given point. It will take in point we wish to insert, set of existing points and K-helps with indices as parameters to the function, visualize the situation by plotting the x-y plane filled with points with the help of matplotlib.

Source code:

```
from sklearn.datasets import load_iris
from sklearn.neighbours import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np
```

```
dataset = load_iris
#print(dataset)
X_train, X_test, y_train, y_test = train_test_split(dataset
["data"], dataset["target"], random_state=0)
kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train, y_train)
for i in range(len(X_test)):
    X = X_test[i]
    X_new = np.array([X])
    prediction = kn.predict(X_new)
print("TARGET=", y_test[i], dataset["target_names"]
[y_test[i]], "PREDICTED=", prediction, dataset
["target_names"][prediction])
print(kn.score(X_test, y_test))
```

## Output

```
TARGET = 2  Virginica PREDICTED = [2] ['virginica']
TARGET = 1  Versicolor PREDICTED = [1] ['versicolor']
TARGET = 0  Setosa PREDICTED = [0] ['setosa']
TARGET = 2  Virginica PREDICTED = [2] ['virginica']
TARGET = 0  Setosa PREDICTED = [0] ['setosa']
TARGET = 2  Virginica PREDICTED = [2] ['virginica']
TARGET = 1  Versicolor PREDICTED = [1] ['versicolor']
TARGET = 1  Versicolor PREDICTED = [1] ['versicolor']
TARGET = 2  Virginica PREDICTED = [2] ['virginica']
TARGET = 0  Setosa PREDICTER = [0] ['setosa']
TARGET = 1  Versicolor PREDICTED = [1] ['versicolor']
TARGET = 1  Versicolor PREDICTED = [1] ['versicolor']
TARGET = 0  Setosa PREDICTED = [0] ['setosa']
TARGET = 2  Virginica PREDICTED = [2] ['virginica']
TARGET = 1  Versicolor PREDICTED = [1] ['versicolor']
TARGET = 1  Versicolor PREDICTED = [1] ['versicolor']
0.9736842105263158
```

# Experiment-11

AIM:

Implement the non-parametric Locally Weighted Regression algorithm in order to fit datapoints. Select dataset and draw graph.

Description:

Regression is a technique from statistics that is used to predict value of a desired target quantity when the target quantity is continuous.

X is a independent variable
Y is a dependent variable

Loess/Lowess Regression

Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points of scatter plot.

Lowess Algorithm.

Locally Weighted regression is a very powerful nonparametric model used in statistical learning.

Given dataset X, y, we attempt to find a model parameter $\beta(x)$ minimizer residual sum of weighted squared errors.

value of model term $\beta$ using:

$$\hat{\beta}(x_0) = (X^T W(x) X)^{-1} X^T W y$$

predictions = $x0 * \beta$

Source Code:

```
import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook

def local_regression(x0, X, Y, tau): # add bias term
    x0 = np.r_[1, x0]
    X = np.c_[np.ones(len(X)), X]
    xw = X.T * radial_kernel(x0, X, tau) # XTranspose*W
    beta = np.linalg.pinv(xw@X) @ xw @ Y #@ Matrix
    Multiplication or Dot product
    # predict value
    return x0@beta #@ Matrix multiplication or Dot
    for prediction
def radia_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) /
        (-2 * tau * tau))

# Weight or Radial Kernel Bias Function

n=100
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data set (10 samples) X :\n", X[1:10])

Y = np.log(np.abs(X ** 2 - 1) + 0.5)
print("The Fitting Curve Dataset (10 samples) Y :\n",
    Y[1:10])
#jitter X
X += np.random.normal(scale=1, size=n)
print("Normalised (10 samples) X :\n", X[1:10])
```

```
domain = np.linspace(-3, 3, num=300)
print("x0 Domain space (10 samples):\n", domain[:10])
def plot_lwr(tau):
# prediction through regression
prediction = [local_regression(x0, X, Y, tau) for x0 in
  domain]
plot = figure(plot_width=400, plot_height=400)
plot.title.text = 'tau=%g' % tau
plot.scatter [x, y, alpha=.3]
plot.line(domain, prediction, line_width=2, color='red')
return plot
show (gridplot([[plot_lw(10.0), plot_lwr(10)],
        [plot_lwr(0.1), plot_lwr(0.01)]]))
```

Output: