1) NAIVE BAYES ALGORITHM

In [3]:
```
def groupUnderClass(mydata):
    dict = {}
    for i in range(len(mydata)):
        if (mydata.iloc[i, -1] not in dict):
            dict[mydata.iloc[i, -1]] = []
        dict[mydata.iloc[i, -1]].append(mydata.iloc[i, :])
    return dict

def mean(numbers):
    return sum(numbers) / float(len(numbers))

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

def MeanAndStdDev(mydata):
    info = [(mean(attribute), std_dev(attribute)) for attribute in zip(*mydata)]
    del info[-1]
    return info

def MeanAndStdDevForClass(mydata):
    info = {}
    dict = groupUnderClass(mydata)
    for classValue, instances in dict.items():
        info[classValue] = MeanAndStdDev(instances)
    return info

def calculateGaussianProbability(x, mean, stdev):
    expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * expo
```

In [3]:
```
def groupUnderClass(mydata):
    dict = {}
    for i in range(len(mydata)):
        if (mydata.iloc[i, -1] not in dict):
            dict[mydata.iloc[i, -1]] = []
        dict[mydata.iloc[i, -1]].append(mydata.iloc[i, :])
    return dict

def mean(numbers):
    return sum(numbers) / float(len(numbers))

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

```
def MeanAndStdDev(mydata):
    info = [(mean(attribute), std_dev(attribute)) for attribute in zip(*mydata)]
    del info[-1]
    return info

def MeanAndStdDevForClass(mydata):
    info = {}
    dict = groupUnderClass(mydata)
    for classValue, instances in dict.items():
        info[classValue] = MeanAndStdDev(instances)
    return info

def calculateGaussianProbability(x, mean, stdev):
    expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * expo
```

In [4]:
```
info = MeanAndStdDevForClass(dataset)
predictions = getPredictions(info, X)

accuracy = accuracy_rate(y, predictions)
print("Accuracy of Naive Bayes Model is: ", accuracy)
```

```
Accuracy of your model is:  85.71428571428571
```

In [5]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

y_true = y
y_pred = predictions
print('Confusion Matrix: \n', confusion_matrix(y_true, y_pred))

tp, fn, fp, tn = confusion_matrix(y_true,y_pred,labels=[0,1]).reshape(-1)
print('\nOutcome values : \n', tp, fn, fp, tn)

matrix = classification_report(y_true,y_pred,labels=[0,1])
print('\nClassification report : \n',matrix)
```

```
Confusion Matrix:
 [[4 1]
 [1 8]]

Outcome values :
 4 1 1 8


Classification report :
              precision    recall  f1-score   support

           0       0.80      0.80      0.80         5
           1       0.89      0.89      0.89         9

    accuracy                           0.86        14
   macro avg       0.84      0.84      0.84        14
weighted avg       0.86      0.86      0.86        14
```

In [6]:
```
print('For the Data Instance X = (age <=30,Income = medium,Student = yes,Credit_rating = fair)\n')
X_test = pd.DataFrame([[0, 2, 1, 1]])

predictions = getPredictions(info, X_test)

if predictions[0] == 1:
    print('Prediction is: Yes the student will buy computer')
else:
    print('Prediction is: No the student will not buy computer')
```

```
Classifying Data X = (age <=30,Income = medium,Student = yes,Credit_rating = fair)

Prediction is: Yes
```

2) KNN ALGORITHM

In [11]:
```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, -1].values
```

Out[11]:
```
array([0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0], dtype=int64)
```

In [12]:
```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

In [13]:
```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In [14]:
```
x_train
```

Out[14]:
```
array([[-1.60356745, -1.16599767, -0.81649658,  0.81649658],
       [ 1.06904497,  0.95399809, -0.81649658, -1.22474487],
       [ 1.06904497,  0.95399809,  1.22474487,  0.81649658],
       [-0.26726124, -1.16599767, -0.81649658, -1.22474487],
       [-0.26726124,  0.95399809, -0.81649658,  0.81649658],
       [-0.26726124,  0.95399809,  1.22474487, -1.22474487],
       [ 1.06904497,  0.95399809, -0.81649658,  0.81649658],
       [-0.26726124, -1.16599767, -0.81649658,  0.81649658],
       [ 1.06904497, -0.10599979,  1.22474487, -1.22474487],
       [-1.60356745, -1.16599767,  1.22474487,  0.81649658]])
```

In [15]:
```
x_test
```

Out[15]:
```
array([[-0.26726124, -0.10599979,  1.22474487,  0.81649658],
       [-1.60356745, -0.10599979,  1.22474487, -1.22474487],
       [ 1.06904497, -0.10599979,  1.22474487,  0.81649658],
       [-1.60356745,  0.95399809, -0.81649658, -1.22474487]])
```

In [27]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=4, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
```

Out[27]:
```
KNeighborsClassifier(n_neighbors=4)
```

In [17]:
```
#Predicting the test set result
y_pred= classifier.predict(x_test)
y_pred
```

Out[17]:
```
array([1, 1, 1, 0], dtype=int64)
```

In [28]:
```
    from sklearn.metrics import confusion_matrix
    cm= confusion_matrix(y_test, y_pred)
    print('Confusion Matrix: \n',cm)
```

```
Confusion Matrix:
 [[0 0]
 [1 3]]
```

In [29]:
```
tp, fn, fp, tn = confusion_matrix(y_test,y_pred,labels=[0,1]).reshape(-1)
print('\nOutcome values : \n', tp, fn, fp, tn)
```

```
Outcome values :
 0 0 1 3
```

In [31]:
```
from sklearn.metrics import classification_report
matrix = classification_report(y_test,y_pred,labels=[0,1])
print('\nClassification report : \n',matrix)
```

```
Classification report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       1.00      0.75      0.86         4

    accuracy                           0.75         4
   macro avg       0.50      0.38      0.43         4
weighted avg       1.00      0.75      0.86         4
```

In [34]:
```
print('For the Data Instance X = (age <=30,Income = medium,Student = yes,Credit_rating = fair)\n')
x_test = pd.DataFrame([[0, 2, 1, 1]])

pred = classifier.predict(x_test)
if pred[0] == 1:
    print('Prediction is: Yes the student will buy computer')
else:
    print('Prediction is: No the student will not buy computer')
```

```
For the Data Instance X = (age <=30,Income = medium,Student = yes,Credit_rating = fair)

Prediction is: Yes the student will buy computer
```