##***LAB 10 : Naive Bayes Classifier***##

1. Binary Classification using Naive Bayes Classifier
2. Sentiment Analysis using Naive Bayes

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import joblib
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
%matplotlib inline
```

##Binary Classification using Naive Bayes Classifier##

Useful References :

1. https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/ (https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/)
2. https://www.analyticsvidhya.com/blog/2021/01/a-guide-to-the-naive-bayes-algorithm/ (https://www.analyticsvidhya.com/blog/2021/01/a-guide-to-the-naive-bayes-algorithm/)
3. https://towardsdatascience.com/implementing-naive-bayes-algorithm-from-scratch-python-c6880cfc9c41 (https://towardsdatascience.com/implementing-naive-bayes-algorithm-from-scratch-python-c6880cfc9c41)

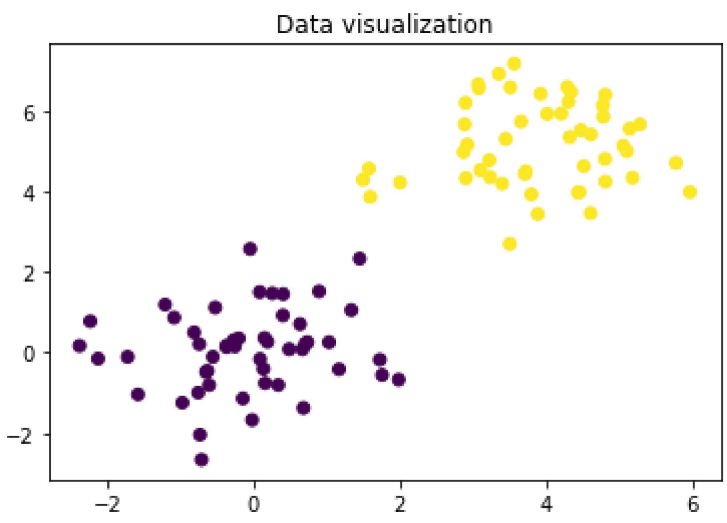
Note : The goal of this experiment is to perform and understand Naive Bayes classification by applying it on the below dataset, you can either fill in the below functions to get the result or you can create a class of your own using the above references to perform classification

1. Generation of 2D training data

In [2]:
```
mean1=np.array([0,0])
mean2=np.array([4,5])
var=np.array([[1,0.1],[0.1,1]])
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,50)
data2=np.random.multivariate_normal(mean2,var,50)
data=np.concatenate((data1,data2))
label=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0])))

plt.figure()
plt.scatter(data[:,0],data[:,1],c=label)
plt.title('Data visualization')
```

Out[2]: Text(0.5, 1.0, 'Data visualization')



2. Split the Dataset by Class Values (Create a Dictionary)

In [3]:
```
def class_dictionary(data, label):
    class_dict = {}

    labels_data = [0, 1]

    for i in labels_data:
        class_dict[i] = []

    for i,d in enumerate(data):
        class_dict[label[i]].append(d)

    return class_dict

class_dict_data = class_dictionary(data, label)
```

3. Calculate Mean, Std deviation and count for each column in a dataset

In [4]:
```
def get_variables(class_dict):
    dataFrame = pd.DataFrame.from_dict(class_dict)

    #vectors for each data vector
    vec11 = []
    vec12 = []

    vec21 = []
    vec22 = []

    for i in range(len(dataFrame)):
        vec11.append(dataFrame[0][i][0])
        vec12.append(dataFrame[0][i][1])

        vec21.append(dataFrame[1][i][0])
        vec22.append(dataFrame[1][i][1])

    out0 = [[np.mean(vec11), np.std(vec11), len(vec11)], [np.mean(vec12), np.std(vec12), len(vec12)]]
    out1 = [[np.mean(vec21), np.std(vec21), len(vec21)], [np.mean(vec22), np.std(vec22), len(vec22)]]

    out_dict = {0: out0, 1: out1}

    return out_dict
```

In [5]:
```
variables_data = get_variables(class_dict_data)
```

3. Calculate Class Probabilities

In [6]:
```
def calculate_probability(x,mean,stdev):
    exponent = np.exp(-((x-mean)**2 / (2 * stdev**2 )))
    return (1 / (np.sqrt(2 * np.pi) * stdev)) * exponent

def calculate_class_probabilities(summaries,row):
    probabilities = dict()

    '''
    You can use the above function (calculate_probability) to calculate
    probability of an individual data point belonging to a particular class
    based on mean and std deviation of that class
    '''

    total = np.sum([summaries[i][0][2] for i in summaries])

    for class_value, class_summaries in summaries.items():
        probabilities[class_value] = (summaries[class_value][0][2])/float(total)

        for i in range(len(class_summaries)):
            mean, stdev, count = class_summaries[i]
            probabilities[class_value] *= calculate_probability(row[i], mean, stdev)

    return probabilities
```

In [ ]:

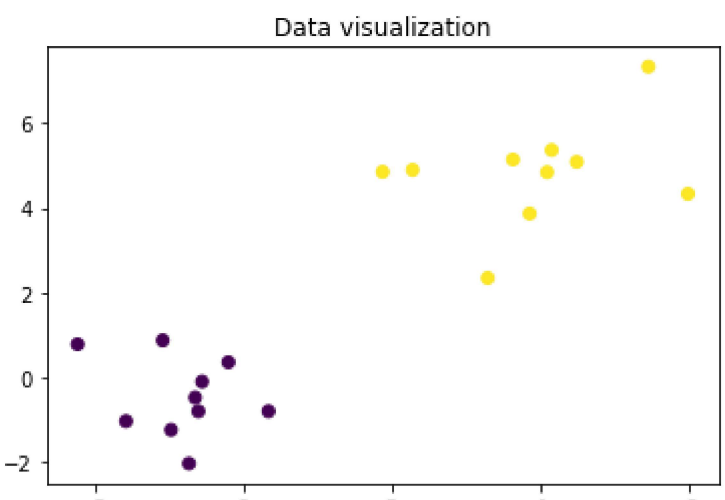
4. Test the model using some samples

In [7]:
```
## Test Data Generation

mean1=np.array([0,0])
mean2=np.array([4,5])
var=np.array([[1,0.1],[0.1,1]])
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,10)
data2=np.random.multivariate_normal(mean2,var,10)
test_data=np.concatenate((data1,data2))
y_test=np.concatenate((np.zeros(data1.shape[0]),np.ones(data2.shape[0])))
print('Test Data Size : ',test_data.shape[0])
plt.figure()
plt.scatter(test_data[:,0],test_data[:,1], c=y_test)
plt.title('Data visualization')
```

```
Test Data Size :  20
```

Out[7]:
```
Text(0.5, 1.0, 'Data visualization')
```



In [8]:
```
test_data
```

Out[8]:
```
array([[-1.57668985, -1.03982262],
       [-2.22908837,  0.77738534],
       [-0.72944014, -2.04059601],
       [-0.60307093, -0.80613793],
       [-0.1988887 ,  0.351987  ],
       [-1.0823819 ,  0.86873075],
       [-0.64602276, -0.4827786 ],
       [-0.55301333, -0.10534224],
       [-0.97041435, -1.24566304],
       [ 0.34076762, -0.80512206],
       [ 5.45488724,  7.3318086 ],
       [ 3.85677639,  3.86105753],
       [ 3.29232307,  2.34108677],
       [ 4.09163132,  4.84049783],
       [ 1.87758468,  4.84893634],
       [ 3.63140878,  5.13876704],
       [ 5.9871582 ,  4.32964087],
       [ 4.15313648,  5.36290063],
       [ 2.28101109,  4.89417293],
       [ 4.49004011,  5.08445841]])
```

Testing for a sample point

In [9]:
```
for i in range(len(test_data)):

    class_dict = class_dictionary(data,label)
    var_dict = get_variables(class_dict)
    out = calculate_class_probabilities(var_dict,test_data[i])
    print('Class Probabilites for the sample ' + str(i+1) + ' of test dataset:')
    print(out)
    print('\n')
```

```
Class Probabilites for the sample 1 of test dataset:
{0: 0.014197204018409897, 1: 8.332975086162198e-16}


Class Probabilites for the sample 2 of test dataset:
{0: 0.005880036890276387, 1: 1.670299581883601e-13}


Class Probabilites for the sample 3 of test dataset:
{0: 0.007806994544567309, 1: 9.895247977223021e-17}


Class Probabilites for the sample 4 of test dataset:
{0: 0.047798541578370046, 1: 3.277207456326854e-13}


Class Probabilites for the sample 5 of test dataset:
{0: 0.07446436058680476, 1: 5.68277354594584e-10}


Class Probabilites for the sample 6 of test dataset:
{0: 0.03435888376009211, 1: 1.0796182888636311e-10}


Class Probabilites for the sample 7 of test dataset:
{0: 0.05790693338553502, 1: 1.5553988414895831e-12}


Class Probabilites for the sample 8 of test dataset:
{0: 0.06896104108236176, 1: 1.5708797068171567e-11}


Class Probabilites for the sample 9 of test dataset:
{0: 0.023376727991490542, 1: 5.037855786228869e-15}


Class Probabilites for the sample 10 of test dataset:
{0: 0.05017154875490167, 1: 1.2593993658550842e-11}


Class Probabilites for the sample 11 of test dataset:
{0: 1.6251054561363263e-19, 1: 0.002461410419103012}


Class Probabilites for the sample 12 of test dataset:
{0: 3.069063988103115e-08, 1: 0.03523745712154807}


Class Probabilites for the sample 13 of test dataset:
{0: 2.0293249571742938e-05, 1: 0.0016571501852476987}


Class Probabilites for the sample 14 of test dataset:
{0: 2.1402220483683901e-10, 1: 0.07089168645216266}


Class Probabilites for the sample 15 of test dataset:
{0: 2.0201055386093997e-07, 1: 0.010351101842332976}


Class Probabilites for the sample 16 of test dataset:
{0: 3.330973809885661e-10, 1: 0.0728828827384983}


Class Probabilites for the sample 17 of test dataset:
{0: 1.0187586090518097e-13, 1: 0.0067536304107634485}
```

```
Class Probabilites for the sample 18 of test dataset:
{0: 1.343314268072207e-11, 1: 0.07112634335881654}


Class Probabilites for the sample 19 of test dataset:
{0: 6.780752872666843e-08, 1: 0.02111004824175014}


Class Probabilites for the sample 20 of test dataset:
{0: 1.1710186512806226e-11, 1: 0.06330676403999416}
```

As seen above the class probability for the 1st sample is given, we can observe that probability is higher for class 0 than 1 and hence imply that this datapoint belongs to class 0

Now Calculate the class probabilities for all the data points in the test dataset and calculate the accuracy by comparing the predicted labels with the true test labels

In [10]:
```
new_labels = list()
acc_metric = 0

for i in range(len(test_data)):
    t = test_data[i]
    out = calculate_class_probabilities(var_dict,t)

    #check if class 0 prob is greater than class 1
    if(out[0] > out[1]):
        #return the class label as 0
        new_labels.append(0)

    #check if class 1 prob is greater than class 0
    else:
        #return the class label as 1
        new_labels.append(1)


new_labels = np.array(new_labels)

for idx,i in enumerate(y_test):
    if(i == new_labels[idx]):
        acc_metric += 1;

#accuracy of the model will be
print("accuracy of the model is:",acc_metric/len(test_data)*100)
```

```
accuracy of the model is: 100.0
```

5. Use the Sci-kit Learn library to perform Gaussian Naive Bayes classifier on the above dataset, also report the accuracy and confusion matrix for the same

In [11]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

nb = GaussianNB();
nb.fit(data,label)

y_predict = nb.predict(test_data)
print(confusion_matrix(y_test,y_predict))

print('The accuracy of this Gausian Naive Bayes classifier is', accuracy_score(y_test,y_predict)*100)
```

```
[[10  0]
 [ 0 10]]
The accuracy of this Gausian Naive Bayes classifier is 100.0
```

##Sentiment Analysis using Naive Bayes Classifier##

Go through the following article (https://www.analyticsvidhya.com/blog/2021/07/performing-sentiment-analysis-with-naive-bayes-classifier/) and implement the same

Keypoints :

1. The link to the dataset is given in the above article, download the same to perform sentiment analysis
2. Understanding how to deal with text data is very important since it requires a lot of preprocessing, you can go through this article (https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/) if you are interested in learning more about it
3. Split the dataset into train-test and train the model
4. Report the accuracy metrics and try some sample prediction outside of those present in the dataset

Note : The goal of this experiment is to explore a practical use case of Naive bayes classifier as well as to understand how to deal with textual data, you can follow any other open source implemetations of sentiment analysis using naive bayes also

Other References :

1. https://towardsdatascience.com/sentiment-analysis-introduction-to-naive-bayes-algorithm-96831d77ac91 (https://towardsdatascience.com/sentiment-analysis-introduction-to-naive-bayes-algorithm-96831d77ac91)
2. https://gist.github.com/CateGitau/6608912ca92733036c090676c61c13cd (https://gist.github.com/CateGitau/6608912ca92733036c090676c61c13cd)

In [12]:
```
!pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\pranay kamal\anaconda3\lib\site-packages (3.6.1)
Requirement already satisfied: regex in c:\users\pranay kamal\anaconda3\lib\site-packages (from nlt
k) (2021.4.4)
Requirement already satisfied: joblib in c:\users\pranay kamal\anaconda3\lib\site-packages (from nlt
k) (1.0.1)
Requirement already satisfied: click in c:\users\pranay kamal\anaconda3\lib\site-packages (from nlt
k) (8.0.3)
Requirement already satisfied: tqdm in c:\users\pranay kamal\anaconda3\lib\site-packages (from nltk)
(4.59.0)
Requirement already satisfied: colorama in c:\users\pranay kamal\anaconda3\lib\site-packages (from c
lick->nltk) (0.4.4)
```

In [13]:
```
data = pd.read_csv('data.csv')
```

In [14]:
```
def preprocess_data(data):
    # Remove package name as it's not relevant
    data = data.drop('package_name', axis=1)

    # Convert text to lowercase
    data['review'] = data['review'].str.strip().str.lower()
    return data

data = preprocess_data(data)
```

In [15]:
```
# Split into training and testing data
x = data['review']
y = data['polarity']

x, x_test, y, y_test = train_test_split(x, y, stratify=y, test_size=0.25, random_state=42)

# Vectorize text reviews to numbers
vec = CountVectorizer(stop_words='english')
x = vec.fit_transform(x).toarray()
x_test = vec.transform(x_test).toarray()


model = MultinomialNB()
model.fit(x, y)

print(model.score(x_test, y_test))

print(model.predict(vec.transform(['Love this app simply awesome!'])))
```

```
0.8565022421524664
[1]
```

In [ ]: