LAB 10 : Naive Bayes Classifier

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

```
import numpy as np
import matplotlib.pyplot as plt
```

Binary Classification using Naive Bayes Classifier

Useful References :

1. https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/
2. 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

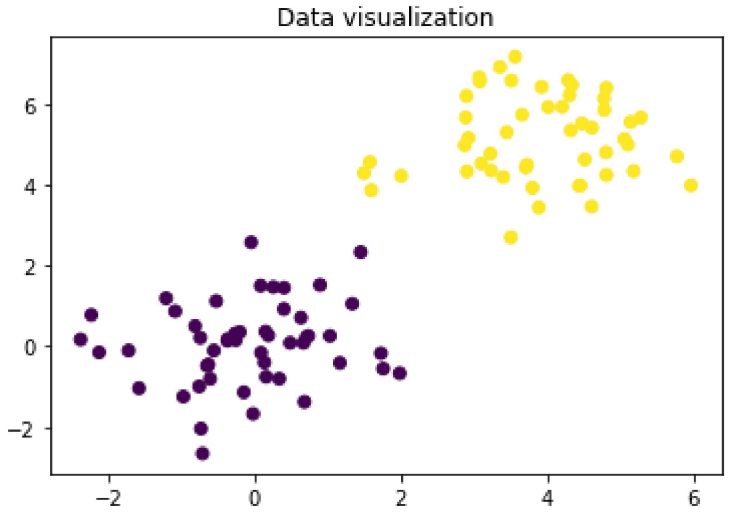
Note : The goal of this experiment is to perform and understand Naive Bayes classification by applying it on the below dataset, you can either fill in the below functions to get the result or you can create a class of your own using the above references to perform classification

1. Generation of 2D training data

```
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')
```

```
Text(0.5, 1.0, 'Data visualization')
```



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

```
def class_dictionary(data,label):
  class_dict = {}
  unique = [0,1]
  for i in unique:
    class_dict[i] = []
  for i,d in enumerate(data):
    class_dict[label[i]].append(d)

  return class_dict
class_dict = class_dictionary(data, label)
class_dict
```

```
 array([-0.75350908, -0.99490358]),
 array([0.08674368, 1.50143912]),
 array([-0.52103706,  1.11904511]),
 array([-0.80961024,  0.50068852]),
 array([-0.73827208,  0.2096953 ]),
 array([-1.20578537,  1.19021132]),
 array([-0.36378319,  0.17554214])],
1: [array([3.50752233, 2.6993138 ]),
 array([4.29192581, 6.59250797]),
 array([3.56619087, 7.17383233]),
 array([4.80815598, 4.80534012]),
 array([1.58074851, 4.56706754]),
 array([2.00718797, 4.22277444]),
 array([3.35739154, 6.92001259]),
 array([3.6604525 , 5.73706073]),
 array([3.40148307, 4.19351521]),
 array([2.92595151, 5.16322159]),
 array([4.45833583, 3.98333488]),
 array([2.88905394, 5.66858736]),
 array([4.61548321, 5.41472741]),
 array([3.08219469, 6.56324233]),
 array([3.22807915, 4.77474657]),
 array([4.20924569, 5.9327244 ]),
 array([4.4787459 , 5.52145107]),
 array([4.01779082, 5.92532132]),
 array([3.71211986, 4.43265775]),
 array([4.43956151, 3.97306561]),
 array([4.81119332, 6.40069914]),
 array([3.45039959, 5.30238367]),
 array([1.59903643, 3.86618863]),
 array([3.92764978, 6.42628439]),
 array([5.28554342, 5.66626269]),
 array([2.90126409, 6.19995457]),
 array([5.10671678, 4.99793313]),
 array([4.51808911, 4.62793958]),
 array([3.88890452, 3.44002436]),
 array([5.14468414, 5.55728691]),
 array([3.0749813, 6.6637202]),
 array([3.23715953, 4.35462184]),
 array([4.3423914 , 6.47522321]),
 array([5.77804351, 4.7055156 ]),

 array([2.90632196, 4.33154509]),
```

```
        array([3.10326381, 4.53088183]),
        array([3.80127866, 3.92783617]),
        array([4.30978752, 6.22424252]),
        array([5.05838861, 5.1332605 ]),
        array([4.32610657, 5.34955732]),
        array([5.18487428, 4.34018354]),
        array([5.96882436, 3.98581601]),
        array([4.61100889, 3.461623  ]),
        array([4.78403921, 5.854026  ]),
        array([3.51338894, 6.58356026]),
        array([4.77966518, 6.13795151]),
        array([4.81271386, 4.24555198]),
        array([3.72700442, 4.49685086]),
        array([2.87520143, 4.98004819]),
        array([1.49913108, 4.29227149])]}
```

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

```
def get_variables(class_dict):
  var_dict = {}
  for i in class_dict:
    l = len(class_dict[i])
    x = np.array(class_dict[i]);
    mu1 = np.mean(x[:,0]);
    mu2 = np.mean(x[:,1]);
    sig1 = sum([(var[0]-mu1)**2 for var in x])/l;
    sig2 = sum([(var[1]-mu2)**2 for var in x])/l;
    var_dict[int(i)]=[(mu1,np.sqrt(sig1),l),(mu2,np.sqrt(sig2),l)];
  return var_dict
var_dict = get_variables(class_dict)
var_dict
```

```
     {0: [(-0.08143074021616499, 0.9927104621843587, 50),
       (0.05650317153356914, 1.0255393627439437, 50)],
      1: [(3.891293527338697, 1.0208299436699904, 50),
       (5.136474984683965, 1.035467336214579, 50)]}
```

3. Calculate Class Probabilities

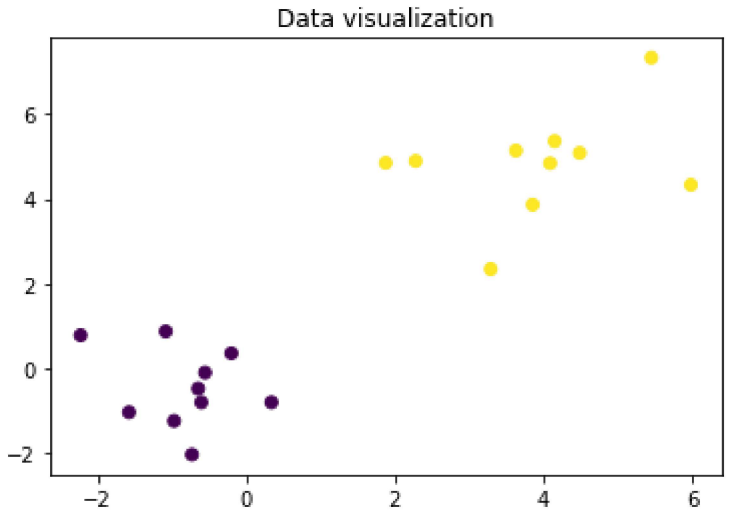
```
def calculate_probability(x,mean,stdev):
  exponent = np.exp(-((x-mean)**2 / (2 * stdev**2 )))
  return (1 / (np.sqrt(2 * np.pi) * stdev)) * exponent

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


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

  '''
  return probabilities
```

4. Test the model using some samples

```
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 =lab )
plt.title('Data visualization')
```

```
plt.title( Data visualization )
```

```
Test Data Size :  20
Text(0.5, 1.0, 'Data visualization')
```



Testing for a sample point

```
class_dict = class_dictionary(data,label)
var_dict = get_variables(class_dict)
out = calculate_class_probabilities(var_dict,test_data[0])
print('Class Probabilites for the first sample of test dataset : ')
print(out)
```

```
Class Probabilites for the first sample of test dataset :
{0: 0.014197204018409897, 1: 8.332975086162255e-16}
```

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

```
lab=[]
scr=0
for i in range(len(test_data)):
  t=test_data[i]
  out = calculate_class_probabilities(var_dict,t)
  if (out[0]>out[1]):
    lab.append(0)
  else:
    lab.append(1)
for idx,i in enumerate(y_test):
  if(i==lab[idx]):
    scr+=1;
print("accuracy of the model is:",scr/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

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix,accuracy_score
nb=GaussianNB();
nb.fit(data,label)
y_predict=nb.predict(test_data)
print(confusion_matrix(y_test,y_predict))
print("accuracy",accuracy_score(y_test,y_predict)*100,"%")
```

```
[[10  0]
 [ 0 10]]
accuracy 100.0 %
```

Sentiment Analysis using Naive Bayes Classifier

Go through the following article 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 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
2. https://gist.github.com/CateGitau/6608912ca92733036c090676c61c13cd

```
!pip install nltk

import pandas as pd
from sklearn.model_selection import train_test_split
import joblib
from sklearn.feature_extraction.text import CountVectorizer

data = pd.read_csv('/content/testing.csv')

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)

# 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()

from sklearn.naive_bayes import MultinomialNB

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

print(model.score(x_test, y_test))

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

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: nltk in /usr/local/lib/python3.7/dist-packages (3.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from nltk) (1.2.0)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from nltk) (7.1.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from nltk) (4.64.1)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.7/dist-packages (from nltk) (2022.6.2)
0.8565022421524664
[1]
```

Colab paid products - Cancel contracts here

✓ 10s completed at 10:20 PM