Naive Bayes Algorithm implementation

By Neeraj Sharma

Date: 10/02/2019

Design:

Description: Implementation of Naïve Bayes Classification algorithm for two data sets has been implemented in three separate programs. API: nbapi.py, Backend: NBPIMADATA.py and NBayesFinal.py

Front End: I have built a restful API POST method to run the program and calculate the accuracy. http://127.0.0.1:5000/calculateAccuracy

Request: json format

```
{
"dataset": "iris"
}
```

We provide the data set name in the request and send the request through postman, SoapUI or curl command. It calls the program and calculates the accuracy and displayed in response payload.

Response: Success 201 Created

```
{
  "dataset": "iris",
  "Accuracy": 78
}
```

Error handling: if the dataset name is not correct then system will throw 400 bad request error with proper error message.

```
{
  "message": "Please enter the correct dataset."
}
```

Dataset:

1. ***Iris dataset description: total number of rows 150***

Attributes:

#1.sepal_length

#2 sepal_width
#3 petal_length
#4 petal_width
#5 species

2. ***Dataset description: Pima-Indians-diabetes.csv***

Attributes:

# 1. Number of times pregnant
# 2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
# 3. Diastolic blood pressure (mm Hg)
# 4. Triceps skin fold thickness (mm)
# 5. 2-Hour serum insulin (mu U/ml)
# 6. Body mass index (weight in kg/(height in m)^2)
# 7. Diabetes pedigree function
# 8. Age (years)
# 9. Class variable (0 or 1) (Diabetic or not)

Number of rows **777**

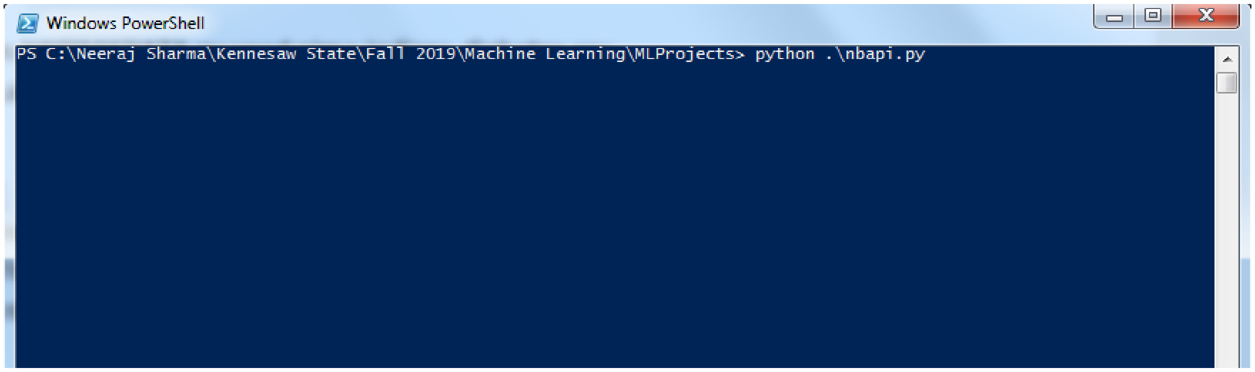
Backend Program:

1. Program NBayesFinal.py – used iris dataset
   - plots the all the data into the scatter graph and it plots the boundary contour and plot the data as per the class.
   - It predicts the class and calculate the accuracy:
     Accuracy is: 78.0%

2. Program NBPIMADATA.py used pima-indians-diebetes.csv
   It also calculates the accuracy of the program for given data set.

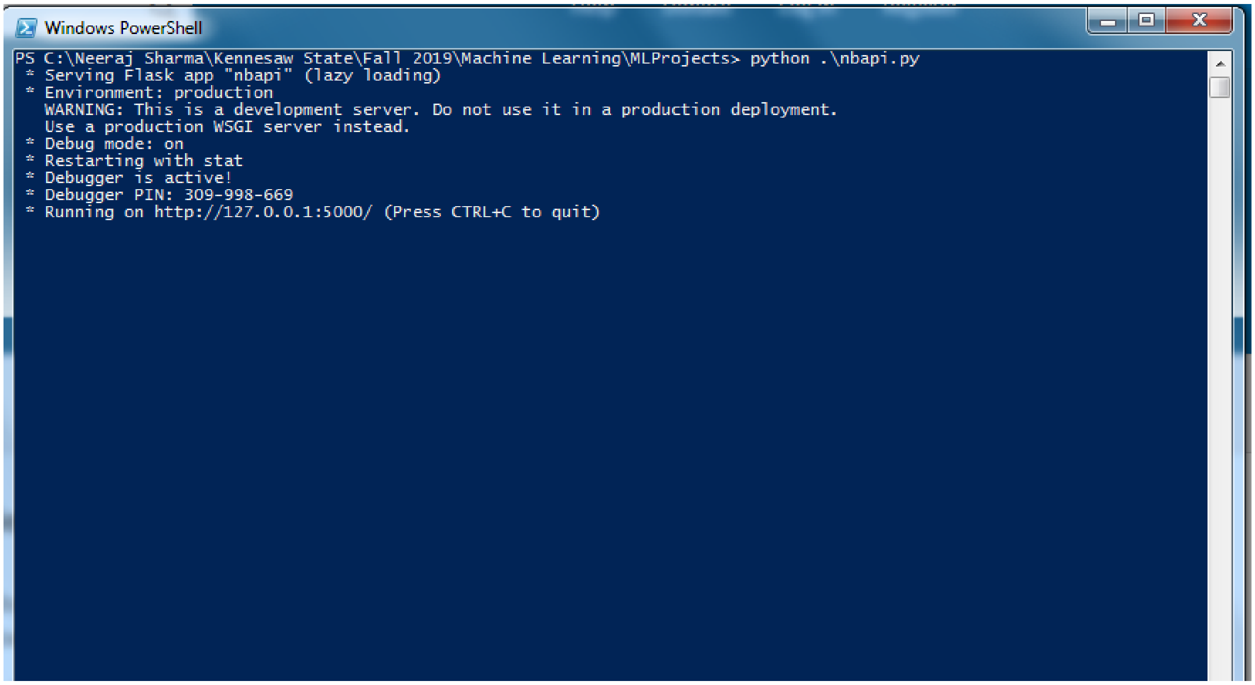
Program Run :

1. Open the windows powerShell or terminal or command prompt and navigate to the project folder. And type the command
   python nbapi.py



```
PS C:\Neeraj Sharma\Kennesaw State\Fall 2019\Machine Learning\MLProjects> python .\nbapi.py
```

2. It will run the nbapi.py program and run the server

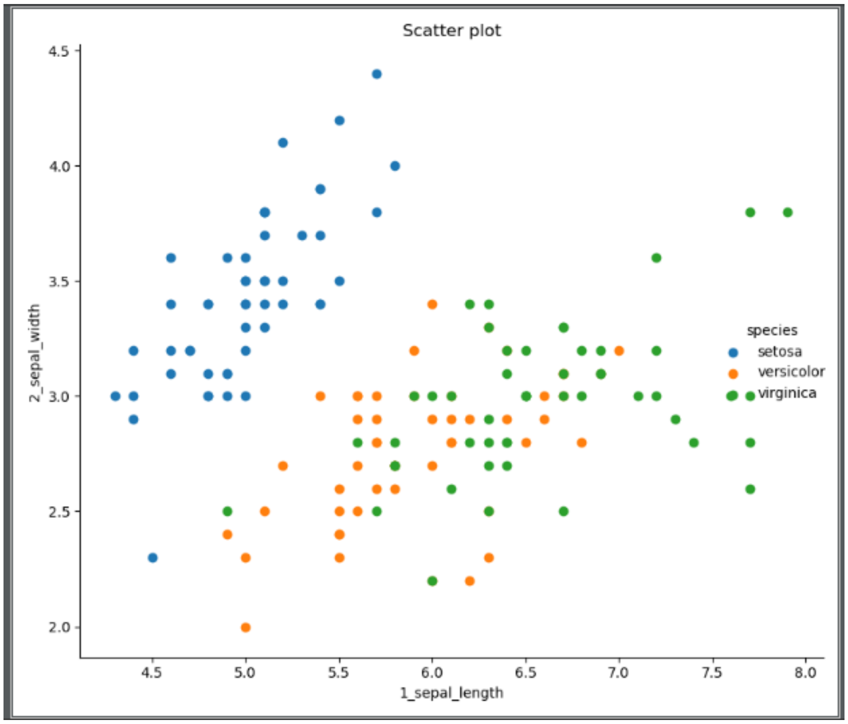


```
PS C:\Neeraj Sharma\Kennesaw State\Fall 2019\Machine Learning\MLProjects> python .\nbapi.py
 * Serving Flask app "nbapi" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 309-998-669
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

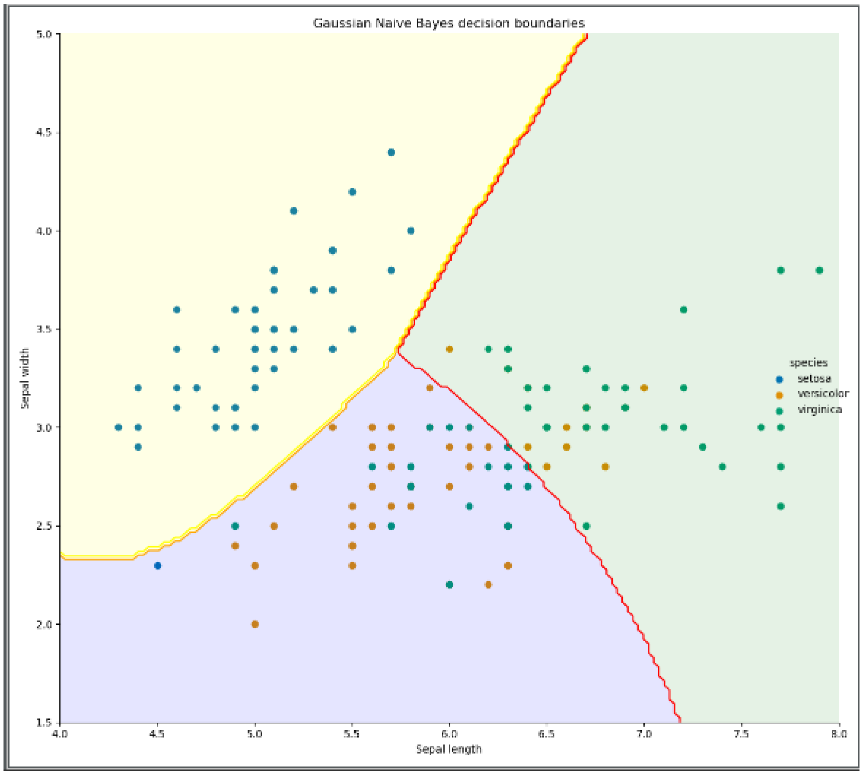
3. Now open postman or soapui and create a post request using below URI and data
   URI: http://127.0.0.1:5000/calculateAccuracy
   Request:

```
{
        "dataset": "iris"
}
```

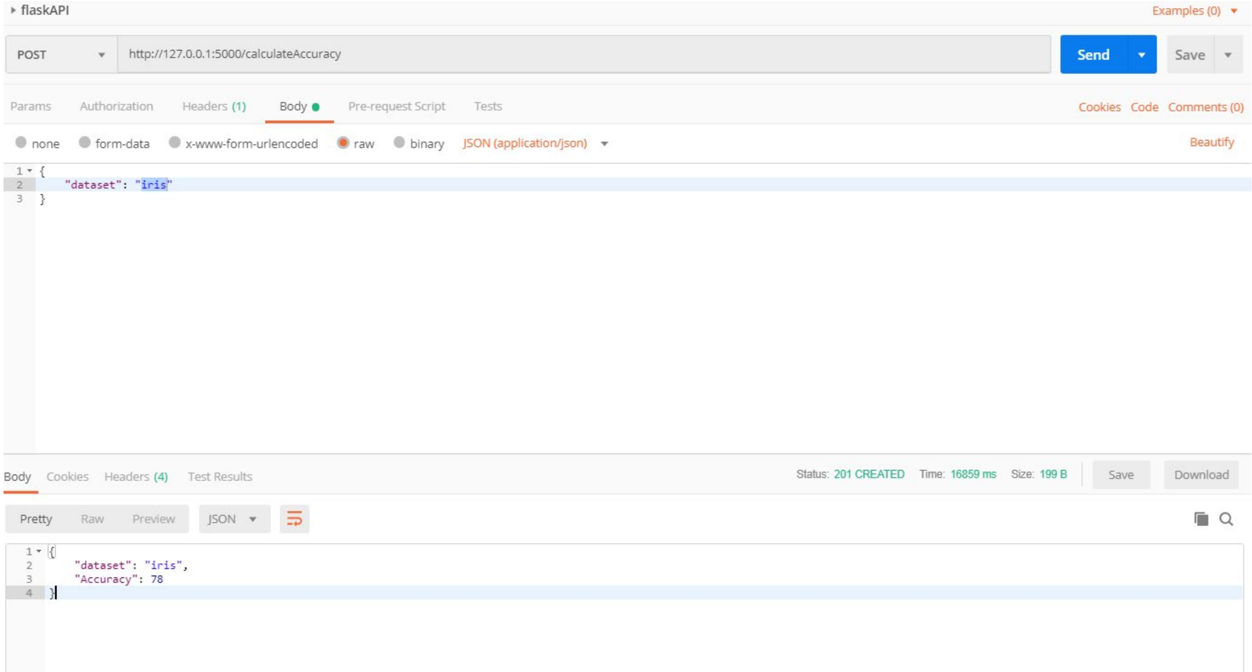
   And click on send button
   It will show the response and provide the accuracy. it will open the scatter plot and boundary contour graph.



Scatter Plot



Decision Boundary graph



POST http://127.0.0.1:5000/calculateAccuracy

JSON (application/json)

```
{
    "dataset": "iris"
}
```

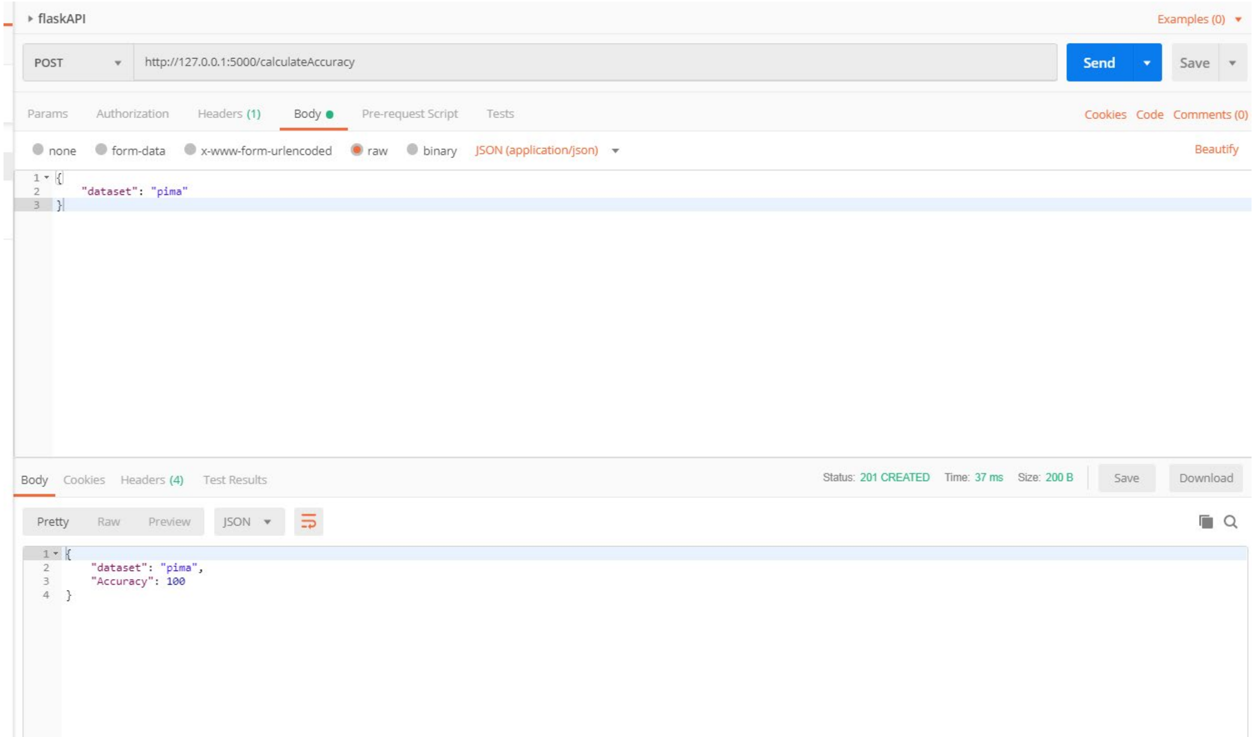
Status: 201 CREATED Time: 16859 ms Size: 199 B

JSON

```
{
    "dataset": "iris",
    "Accuracy": 78
}
```

Accuracy is 78% here.

We change the dataset to pima and send the request it will show the accuracy in the response.



POST http://127.0.0.1:5000/calculateAccuracy

JSON (application/json)

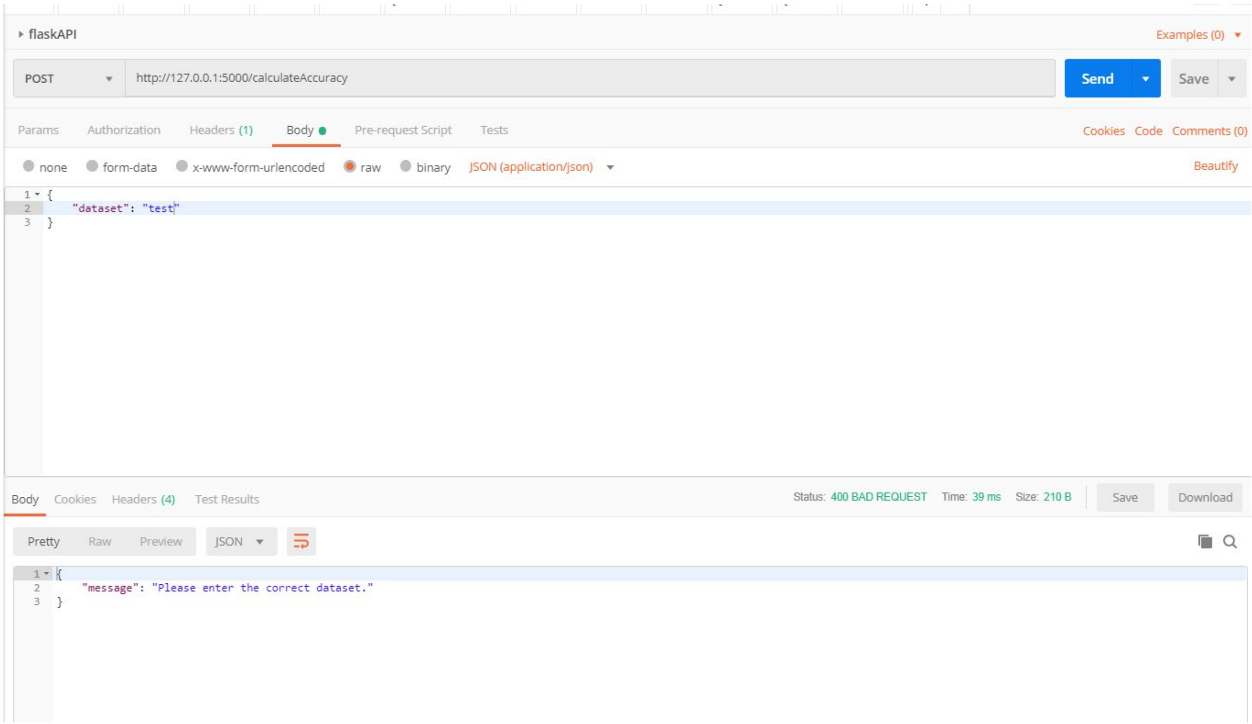
```
{
    "dataset": "pima"
}
```

Status: 201 CREATED Time: 37 ms Size: 200 B

JSON

```
{
    "dataset": "pima",
    "Accuracy": 100
}
```

Also, we can give the wrong dataset and it will throw the error message



POST http://127.0.0.1:5000/calculateAccuracy

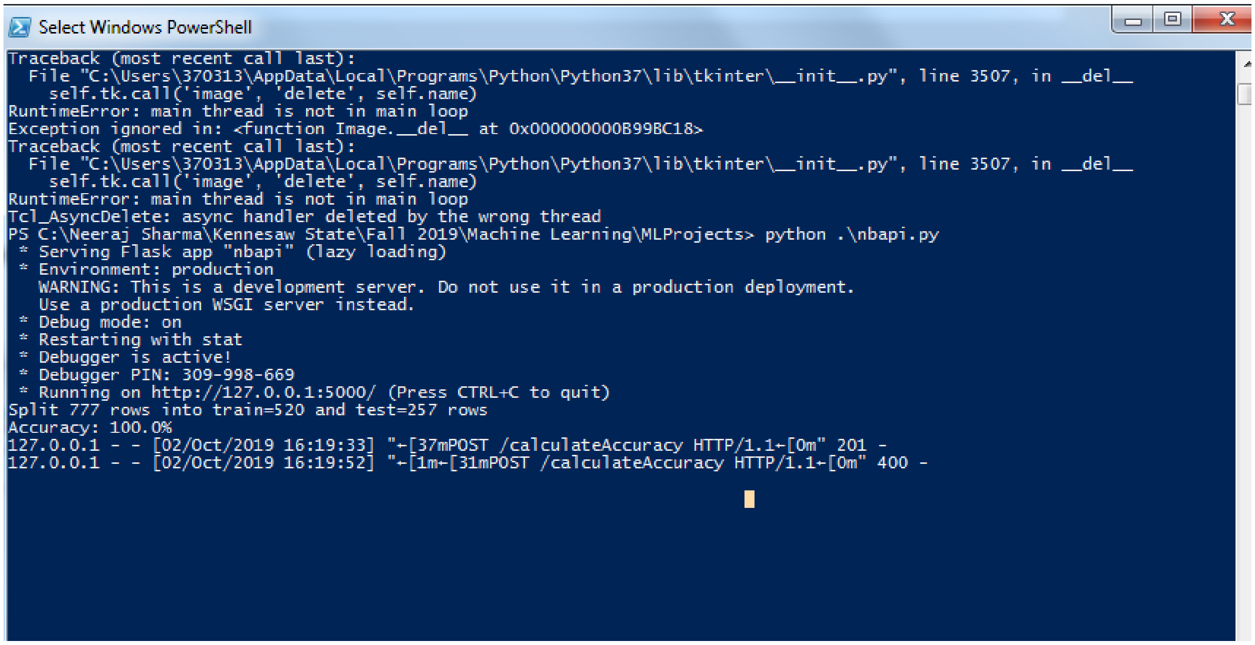
JSON (application/json)

```
{
    "dataset": "test"
}
```

Status: 400 BAD REQUEST Time: 39 ms Size: 210 B

```
{
    "message": "Please enter the correct dataset."
}
```

All the logs will be captured in the PowerShell console



```
Traceback (most recent call last):
  File "C:\Users\370313\AppData\Local\Programs\Python\Python37\lib\tkinter\__init__.py", line 3507, in __del__
    self.tk.call('image', 'delete', self.name)
RuntimeError: main thread is not in main loop
Exception ignored in: <function Image.__del__ at 0x000000000B99BC18>
Traceback (most recent call last):
  File "C:\Users\370313\AppData\Local\Programs\Python\Python37\lib\tkinter\__init__.py", line 3507, in __del__
    self.tk.call('image', 'delete', self.name)
RuntimeError: main thread is not in main loop
Tcl_AsyncDelete: async handler deleted by the wrong thread
PS C:\Neeraj Sharma\Kennesaw State\Fall 2019\Machine Learning\MLProjects> python .\nbapi.py
 * Serving Flask app "nbapi" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 309-998-669
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
Split 777 rows into train=520 and test=257 rows
Accuracy: 100.0%
127.0.0.1 - - [02/Oct/2019 16:19:33] "←[37mPOST /calculateAccuracy HTTP/1.1←[0m" 201 -
127.0.0.1 - - [02/Oct/2019 16:19:52] "←[1m←[31mPOST /calculateAccuracy HTTP/1.1←[0m" 400 -
```

Program Code:

1. ***Front end – API code apiflask.py***
   Front end API calls NBayesFinal.nbmain() and NBPIMADATA.py as per the dataset name in the request provided.

```
from flask import Flask, request, render_template
from flask_restful import Resource, Api
import NBayesFinal
import NBPIMADATA
from flask import abort
#create an object of flask
app = Flask(__name__)
# pass app object to the function  API and create an object of
API function api
api = Api(app)

class nievebays(Resource):
    #defin post method
    def post(self):
        #get the value from the request posted.
        data = request.get_json()
        #get teh value of dataset in the request
        dataset = data['dataset']
        # if statement to call the correct program as per the
dataset
        if dataset == "iris":
            return {'dataset': 'iris',
                    'Accuracy': NBayesFinal.nbmain()}, 201
        elif dataset == "pima":
            return  {'dataset': 'pima',
                    'Accuracy': NBPIMADATA.nbmain()}, 201
        else:
            abort(400, 'Please enter the correct dataset.')
#create the resource path for the api
api.add_resource(nievebays, '/calculateAccuracy')

if __name__ == '__main__':
    app.run(debug=True)
```

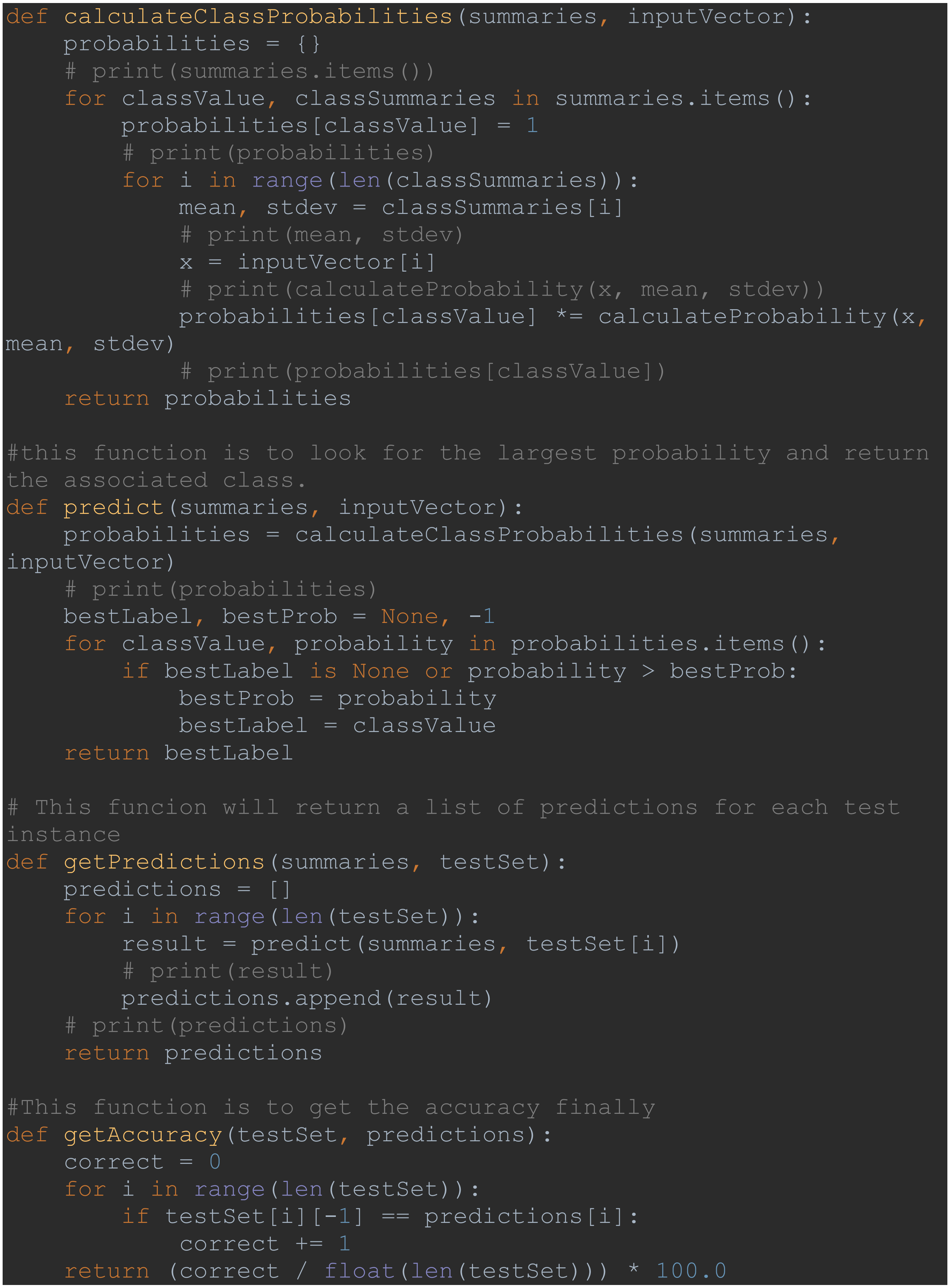
2. ***IRIS data set backend code NBayesFinal.py***

```
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.colors as colors
import seaborn as sns
from scipy.stats import norm
```

```
import scipy.stats


# Load the data set and include the column name and also include
the index for each row
def loaddataset():
    iris = sns.load_dataset("iris")
    iris = iris.rename(index = str, columns =
{'sepal_length':'1_sepal_length','sepal_width':'2_sepal_width',
'petal_length':'3_petal_length', 'petal_width':'4_petal_width'})
    return iris


#Plot the scatter of sepal length vs sepal width
def plotinput(iris):
    sns.FacetGrid(iris, hue="species",
height=7).map(plt.scatter,"1_sepal_length", "2_sepal_width",
).add_legend()
    plt.title('Scatter plot')
    df1 = iris[["1_sepal_length", "2_sepal_width",'species']]
    return df1

# Returns the classes for which the Gaussian Naive Bayes
objective function has greatest value
def predict_NB_gaussian_class(X, mu_list, std_list, pi_list):
    scores_list = []
    classes = len(mu_list)

    for p in range(classes):
        score = (norm.pdf(x=X[0], loc=mu_list[p][0][0],
scale=std_list[p][0][0])
                 * norm.pdf(x=X[1], loc=mu_list[p][0][1],
scale=std_list[p][0][1])
                 * pi_list[p])
        scores_list.append(score)
    return np.argmax(scores_list)


def nbmain():
    iris = loaddataset()
    df1 = plotinput(iris)
    # print(df1)

    #Estimating the parameters
    mu_list =
np.split(df1.groupby('species').mean().values,[1,2])
    std_list =
np.split(df1.groupby('species').std().values,[1,2], axis = 0)
    pi_list = df1.iloc[:,2].value_counts().values / len(df1)

    # Our 2-dimensional distribution will be over variables X 
and Y
    N = 150
    X = np.linspace(4, 8, N)
    Y = np.linspace(1.5, 5, N)
    X, Y = np.meshgrid(X, Y)

    g = sns.FacetGrid(iris, hue="species", height=10, palette = 
'colorblind') .map(plt.scatter, "1_sepal_length", 
"2_sepal_width",)  .add_legend()
    my_ax = g.ax

    #Computing the predicted class function for each value on 
the grid
    zz = np.array(  [predict_NB_gaussian_class( 
np.array([xx,yy]).reshape(-1,1), mu_list, std_list, pi_list) 
                         for xx, yy in zip(np.ravel(X), 
np.ravel(Y)) ] )

    #Reshaping the predicted class into the meshgrid shape
    Z = zz.reshape(X.shape)

    #Plot the filled and boundary contours
    my_ax.contourf( X, Y, Z, 2, alpha = .1, colors = 
('yellow','blue','green'))
    my_ax.contour( X, Y, Z, 2, alpha = 1, colors = 
('yellow','orange','red'))

    # Add axis and title
    my_ax.set_xlabel('Sepal length')
    my_ax.set_ylabel('Sepal width')
    my_ax.set_title('Gaussian Naive Bayes decision boundaries')

    plt.show()

    #Setup X and y data
    X_data = df1.iloc[:,0:2]
    y_labels = 
df1.iloc[:,2].replace({'setosa':0,'versicolor':1,'virginica':2})
.copy()
    # print(X_data)

    #Numpy accuracy
    y_pred = np.array(  [predict_NB_gaussian_class(
np.array([xx,yy]).reshape(-1,1), mu_list, std_list, pi_list)
                         for xx, yy in
zip(np.ravel(X_data.values[:,0]),
np.ravel(X_data.values[:,1]))])
    accuracy = ((np.mean(y_pred == y_labels)))
    # print(f"y_labels: {y_labels}",  f"y_prediction: {y_pred}")
    print(f"Accuracy is: {accuracy*100}%")
    # #Sklearn accuracy
    # display(model_sk.score(X_data,y_labels))
    return accuracy
```

3. ***NBPIMADATA.py – Pima Indians diabetes dataset program***

```
# Example of Naive Bayes implemented from Scratch in Python
import csv
import random
import math


#This function is to load the data from csv filename given
def loadData(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-1]]
    return dataset

#This function is to split the data into training set and test
set
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]

#This function returns a map of class values to lists of data
instances.
def separateByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated

#this function is to calculate the mean
def mean(numbers):
    return sum(numbers) / float(len(numbers))

#this function is to calculate standard deviation
def stdev(numbers):
    avg = mean(numbers)
    # print(avg)
    variance = sum([pow(x - avg, 2) for x in numbers]) /
float(len(numbers) - 1)
    # print(variance)
    # print(math.sqrt(variance))
    return math.sqrt(variance)

#this function is to calculate the mean and the standard
deviation for each attribute.
def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for
attribute in zip(*dataset)]
    del summaries[-1]
    return summaries

#The zip function groups the values for each attribute across
our data instances
def summarizeByClass(dataset):
    separated = separateByClass(dataset)
    summaries = {}
    for classValue, instances in separated.items():
        summaries[classValue] = summarize(instances)
    return summaries

#this function is to calculate probability
def calculateProbability(x, mean, stdev):
    # print(stdev)
    if stdev != 0:
        exponent = math.exp(-(math.pow(x - mean, 2) / (2 *
math.pow(stdev, 2))))
        return (1 / (math.sqrt(2 * math.pi) * stdev)) * exponent
    else:
        return 0.0

# In this function the probability of a given data instance is
calculated by multiplying together the attribute probabilities
for each class
```



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

#this function is to look for the largest probability and return
the associated class.
def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries,
inputVector)
    # print(probabilities)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel

# This funcion will return a list of predictions for each test
instance
def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        # print(result)
        predictions.append(result)
    # print(predictions)
    return predictions

#This function is to get the accuracy finally
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 nbmain():
    filename = r'C:\Neeraj Sharma\Kennesaw State\Fall
2019\Machine Learning\MLProjects\pima-indians-diabetes.csv'
    splitRatio = 0.67
    dataset = loadData(filename)
    trainingSet, testSet = splitDataset(dataset, splitRatio)
    print(f'Split {len(dataset)} rows into
train={len(trainingSet)} and test={len(testSet)} rows')
    # prepare model
    summaries = summarizeByClass(trainingSet)
    # print(summaries)
    # test model
    predictions = getPredictions(summaries, testSet)
    # print(predictions)
    # print(testSet)
    accuracy = getAccuracy(testSet, predictions)
    print(f'Accuracy: {accuracy}%')
    return(accuracy)
```