SRH University Heidelberg

Professional Technical Report

TASK – 1

Classification of handwritten digits based on the Gaussian Naive Bayes

Author:	Abhijith Pavithran
Matriculation Number:	11018118
Email Address:	11018118@stud.hochschule-heidelberg.de
Supervisor:	Prof. Dr. Milan Gnjatovic
Project Submission Date:	09-11-2022

Table of Contents

Table of Figures

ABSTRACT

Nowadays, the scope of machine learning and deep learning studies is increasing day by day. Handwriting recognition is one of the examples in daily life for this field of work. Data storage in digital media is a method that almost everyone is using nowadays. At the same time, it has become a necessity for people to store their notes in digital media and even take notes directly in the digital environment. As a solution to this need, applications have been developed that can recognize numbers, characters, and even text from handwriting using machine learning. I have imported the Gaussian Naïve Bayes Model from the sklearn library for obtaining Hand written digits from dataset.

INTRODUCTION

In this study, tests were performed on the MNIST handwritten digit data set with Gaussian Naïve Bayes model. MNIST (Modified National Institute of Standards and Technology) is a large database of handwritten digits from 0–9. The project uses the modified MNIST dataset with 60000 train data and 10000 test data. It is one of the most commonly used datasets for Machine Learning. Each image in both training and test set is a grey scale image of dimension 28x28 pixels. Naive Bayes is one of the fastest and simple classification algorithms and is usually used for the recognition of handwritten digits.

GAUSSIAN NAÏVE BAYES CLASSIFICATION

The project involves building a Naive Bayes classifier on MNIST dataset.
According to Bayes theorem, let x and y are two events;

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

Conditional probability P(x|y) is referred to the posterior probability.
Conditional probability P(y|x) is referred to the likelihood.
Probability P(x) is referred to the prior probability and P(y) != 0.

Each image in MNIST dataset is a 28x 28 pixels but for our purpose we are converting the images in a single flat array of 784 pixels. Knowing that each pixel from the array can take values between 0–255, it appears like continuous data. To ease the computation, we use Gaussian to get the probabilities of each pixel given a class. Gaussian distribution is :

$$P(x|c) = \frac{1}{\sqrt{(2\pi)^D\ |\Sigma|}} \exp\left(\frac{-1}{2}[(x -| \mu)^T \Sigma^{-1} (x - \mu)]\right)$$

The numbers of dimensions being very large, the probabilities obtained are very small to overcome this we take the log likelihood.

$$\text{Log}\ \ \text{P(x|c)} = -\frac{D}{2}\ln(2\pi) - \frac{1}{2}\ln|\Sigma| - \frac{1}{2}(x - \mu)^T \varepsilon^{-1} (x - \mu)$$

$\mu - Sample\ mean$

$\sum$ - Co-variance

Each pixel in data is assumed to have a Gaussian distribution, the code uses Gaussian Naive Bayes classifier which is imported from sklearn library, each class is assigned with equal probability. Mean and standard deviation is calculated to summarize the distribution of the data, for each class.

Mean(x) = 1/n * sum(x),

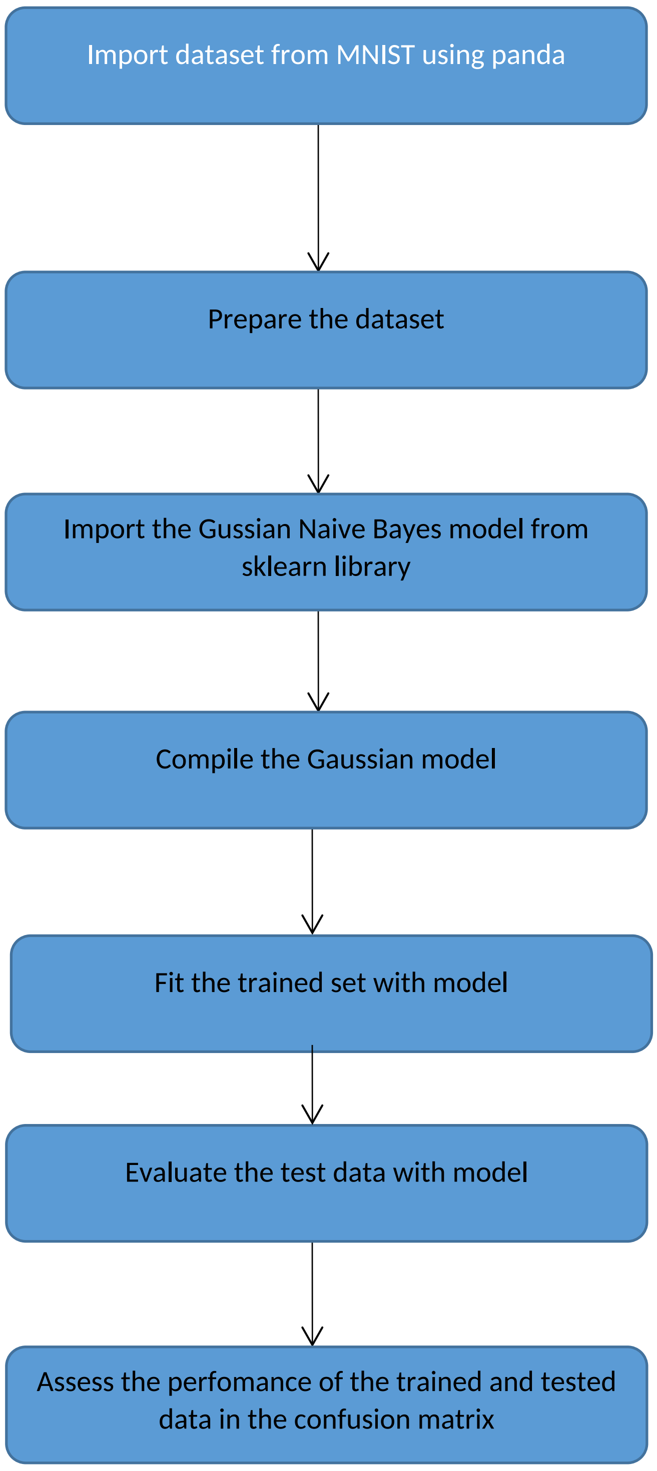
Where, n = number of times the unique value is repeated for an input variable x.

Standard deviation (x) = √ (1/n* Σ(xi-mean(x)^2)),

It is the root of squared distance of input value x from mean value of x where is n is the number of times the unique value is repeated.

Naive Bayes calculations, like calculations for language modeling, are done in log space, to avoid arithmetic underflow and increase speed.

FLOW CHART



Import dataset from MNIST using panda
Prepare the dataset
Import the Gussian Naive Bayes model from sklearn library
Compile the Gaussian model
Fit the trained set with model
Evaluate the test data with model
Assess the perfomance of the trained and tested data in the confusion matrix

Figure 1: Flow chart

IMPLIMENTATION

Importing Libraries

The software libraries used in this project are pandas, numpy, sklearn, seaborn, and matplotlib. The dataset used for training and testing this system is the MNIST are importred through panda library. Numpy library provides the multidimensional arrays. Matplotlibrary is used for creating maps and plots. Sklearn library is used for the importing Gaussian Naïve Bayes algorithm and conclusion matrix.

Removing Constant Features from the Dataset

Those features which contain constant values (i.e. only one value for all the outputs or target values) in the dataset are known as Constant Features. These features don’t provide any information to the target feature. These are redundant data available in the dataset. Presence of this feature has no effect on the target, so it is good to remove these features from the dataset. Drop all such features from the dataset.

Normalization

During the classification phase, the attributes of the data in the picture are compared to the classes in the database to determine which class the picture belongs to. The angle between the horizontal axis in the text correction and the horizontal axis of the written text is known as the slope, and the angle between the vertical axis in the text correction and the vertical axis of the written text is known as the slant. Normalization is another term for handwriting correction using slant and slope.

Reshaping

The MNIST database contains 60,000 training images and 10,000 testing images. Both the data set has 784 number of pixel per image. We need to reshape these parameters into 28x28 pixel format.

Importing Gaussian Naïve Bayes model

Importing Gaussian Naïve Bayes model from the sklearn and then the training data is fit into the Gaussian Naïve Bayes model. The model was evaluated using the "Predict" function on a portion of the dataset that had been put aside for testing.

Prevent Arithmetic Underflow or Overflow

We are multiplying a bunch of numbers between 0 and 1, and so we will get some very very small number (close to zero). When numbers get too large on a computer (exceeding 263 or something), it is called overflow, and results in weird and wrong arithmetic. Our problem is appropriately named underflow, as we can’t handle the precision. To avoid the arithmetic underflow we use log probabilities with likelihood.

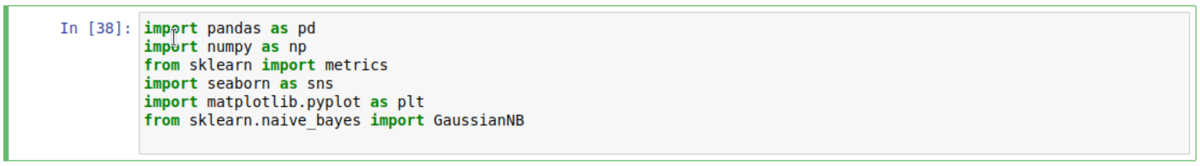
Accuracy

It is a value that indicates the accuracy rate. It is the ratio of predictions classified as correct to all predictions.

Confusion Matrix

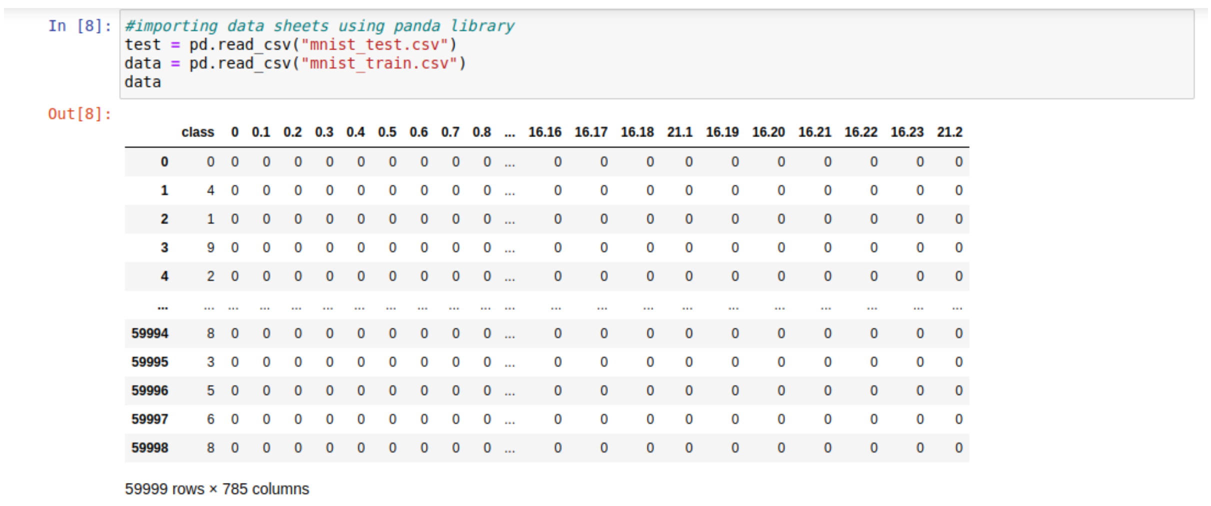
Confusion Matrix or error matrix is a performance measurement method for machine learning classification algorithms. It gives information about the accuracy of the predictions. It is a table that contains combinations of predictions and actual values. Finally, to assess the performance of the trained and tested model the confusion matrix is generated. The confusion matrix function is obtained from the sklearn library's "metrics" module. We have predicted the output by passing y_hat_GB_test and also stored real target in actual y_test.

RESULT



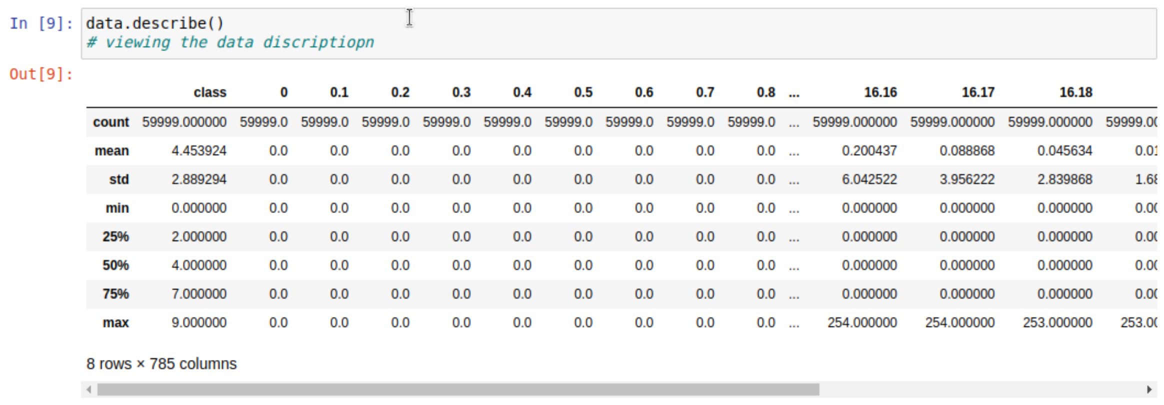
```
In [38]: import pandas as pd
         import numpy as np
         from sklearn import metrics
         import seaborn as sns
         import matplotlib.pyplot as plt
         from sklearn.naive_bayes import GaussianNB
```

Figure 2 : Importing python libraries



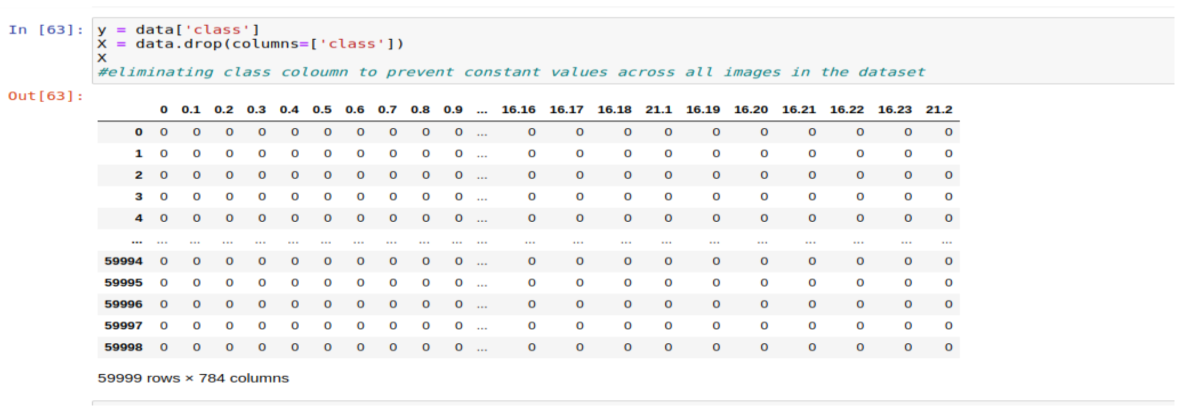
```
In [8]: #importing data sheets using panda library
        test = pd.read_csv("mnist_test.csv")
        data = pd.read_csv("mnist_train.csv")
        data

Out[8]:
       class  0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  ...  16.16  16.17  16.18  21.1  16.19  16.20  16.21  16.22  16.23  21.2
    0      0  0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
    1      4  0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
    2      1  0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
    3      9  0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
    4      2  0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
  ...    ... ...  ...  ...  ...  ...  ...  ...  ...  ...  ...    ...    ...    ...   ...    ...    ...    ...    ...    ...   ...
59994      8  0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
59995      3  0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
59996      5  0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
59997      6  0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
59998      8  0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0

59999 rows × 785 columns
```

Figure 3 : Importing MNIST training datasheet



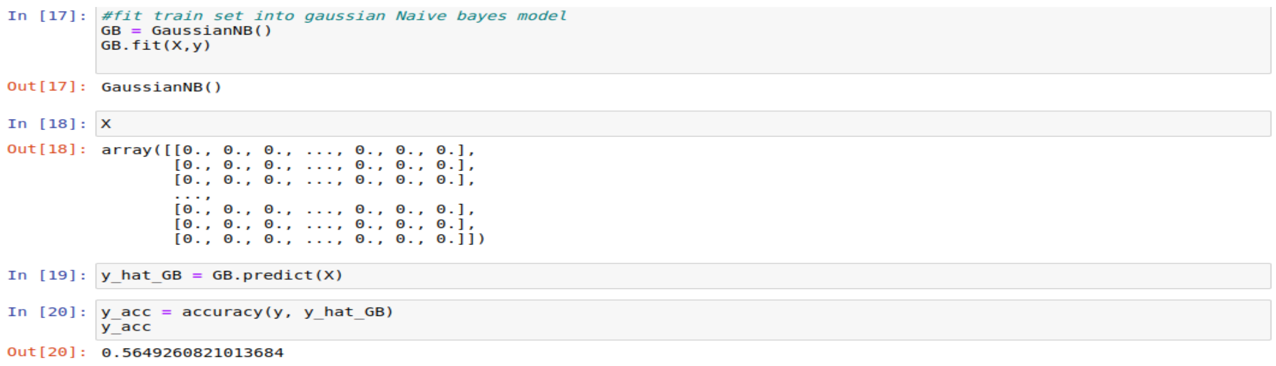
```
In [9]: data.describe()
        # viewing the data discriptiopn

Out[9]:
             class        0      0.1      0.2      0.3      0.4      0.5      0.6      0.7      0.8  ...         16.16         16.17         16.18
count 59999.000000  59999.0  59999.0  59999.0  59999.0  59999.0  59999.0  59999.0  59999.0  59999.0  ...  59999.000000  59999.000000  59999.000000  59999.0
 mean     4.453924      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0  ...      0.200437      0.088868      0.045634  0.0
  std     2.889294      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0  ...      6.042522      3.956222      2.839868  1.6
  min     0.000000      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0  ...      0.000000      0.000000      0.000000  0.0
  25%     2.000000      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0  ...      0.000000      0.000000      0.000000  0.0
  50%     4.000000      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0  ...      0.000000      0.000000      0.000000  0.0
  75%     7.000000      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0  ...      0.000000      0.000000      0.000000  0.0
  max     9.000000      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0  ...    254.000000    254.000000    253.000000  253.0

8 rows × 785 columns
```

Figure 4 : Describing training data



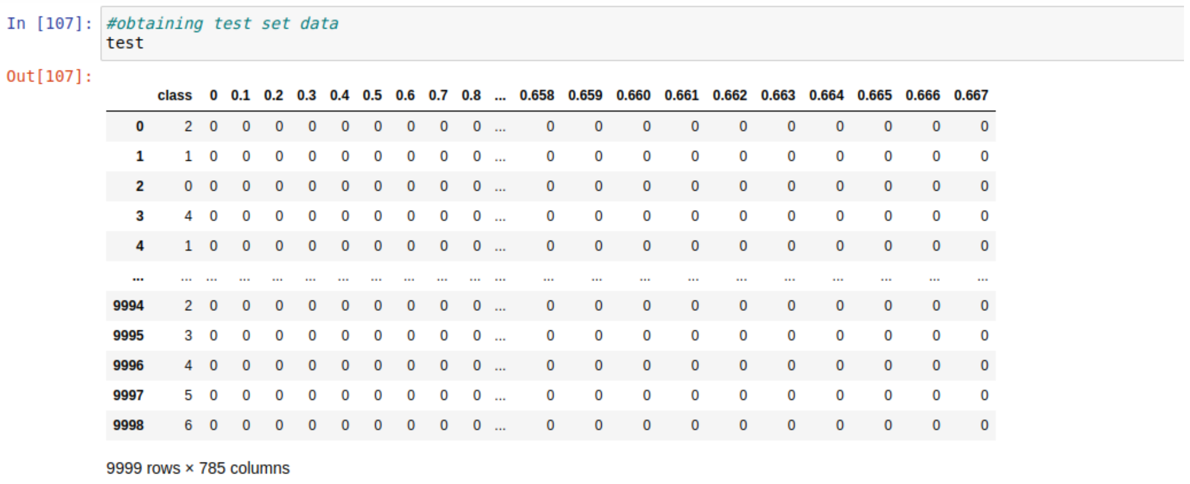
```
In [63]: y = data['class']
         X = data.drop(columns=['class'])
         X
         #eliminating class coloumn to prevent constant values across all images in the dataset

Out[63]:
         0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  ...  16.16  16.17  16.18  21.1  16.19  16.20  16.21  16.22  16.23  21.2
    0    0    0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
    1    0    0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
    2    0    0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
    3    0    0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
    4    0    0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...    ...    ...    ...   ...    ...    ...    ...    ...    ...   ...
59994    0    0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
59995    0    0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
59996    0    0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
59997    0    0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0
59998    0    0    0    0    0    0    0    0    0    0  ...      0      0      0     0      0      0      0      0      0     0

59999 rows × 784 columns
```

Figure 5 : Removing constant features from the dataset by dropping the class column



```
In [17]: #fit train set into gaussian Naive bayes model
         GB = GaussianNB()
         GB.fit(X,y)

Out[17]: GaussianNB()

In [18]: X

Out[18]: array([[0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]])

In [19]: y_hat_GB = GB.predict(X)

In [20]: y_acc = accuracy(y, y_hat_GB)
         y_acc

Out[20]: 0.5649260821013684
```

Figure 6 : Fitting the Gaussian Naïve Bayes model with training data set

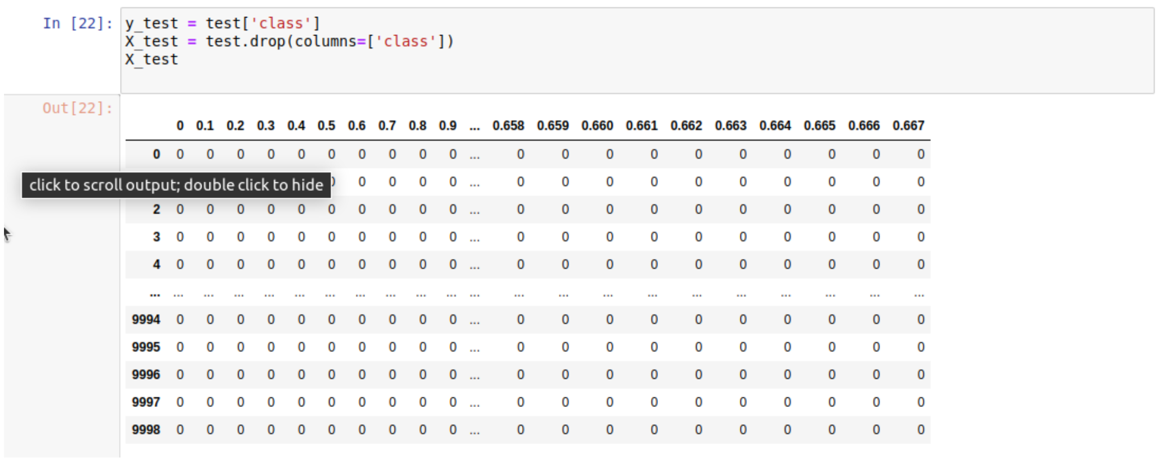


```
In [107]: #obtaining test set data
          test

Out[107]:
```

	class	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	...	0.658	0.659	0.660	0.661	0.662	0.663	0.664	0.665	0.666	0.667
0	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9994	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9995	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9996	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9997	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9998	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

9999 rows × 785 columns

Figure 7 : Importing testing data from MNIST



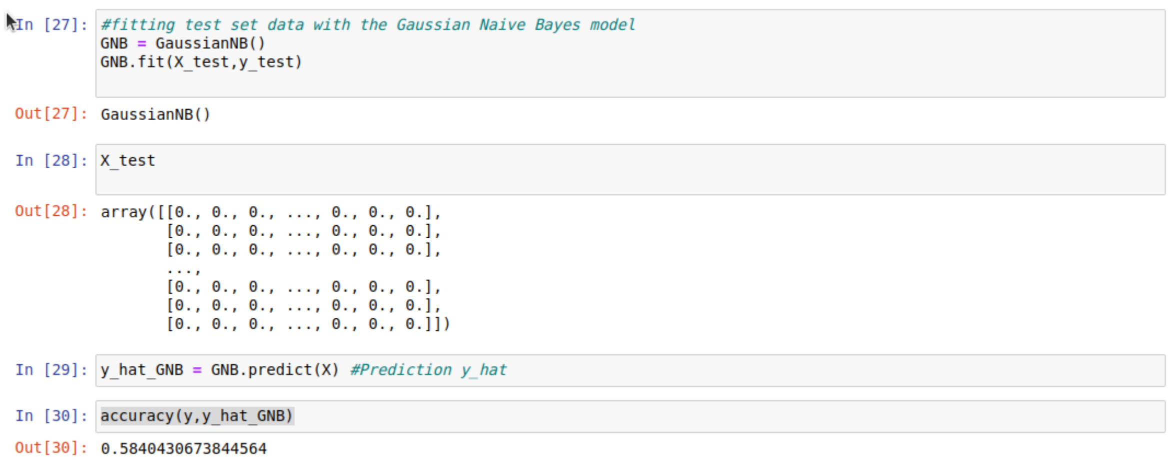
```
In [22]: y_test = test['class']
         X_test = test.drop(columns=['class'])
         X_test
```

Out[22]:

click to scroll output; double click to hide

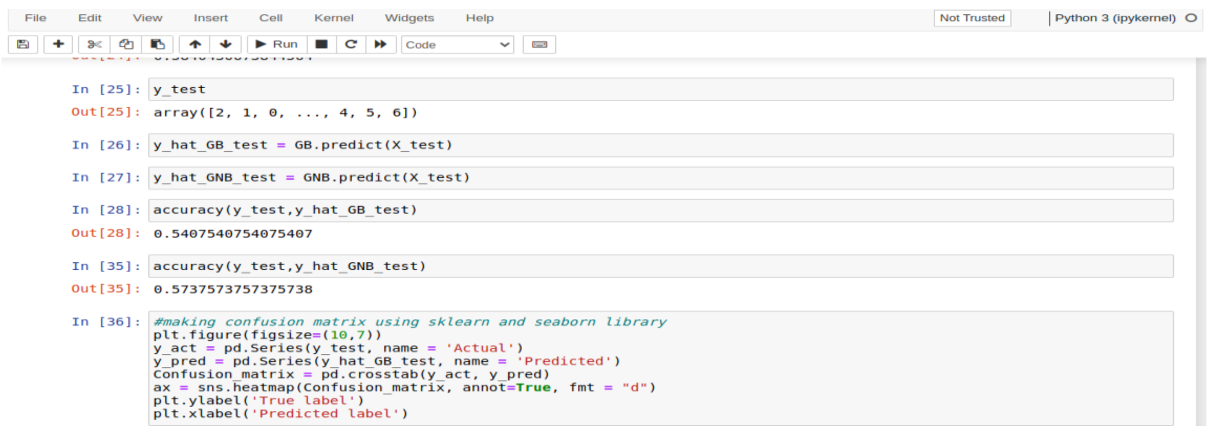
	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	...	0.658	0.659	0.660	0.661	0.662	0.663	0.664	0.665	0.666	0.667
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9994	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Figure 8 : Dropping class attribute from testing data to remove constant features



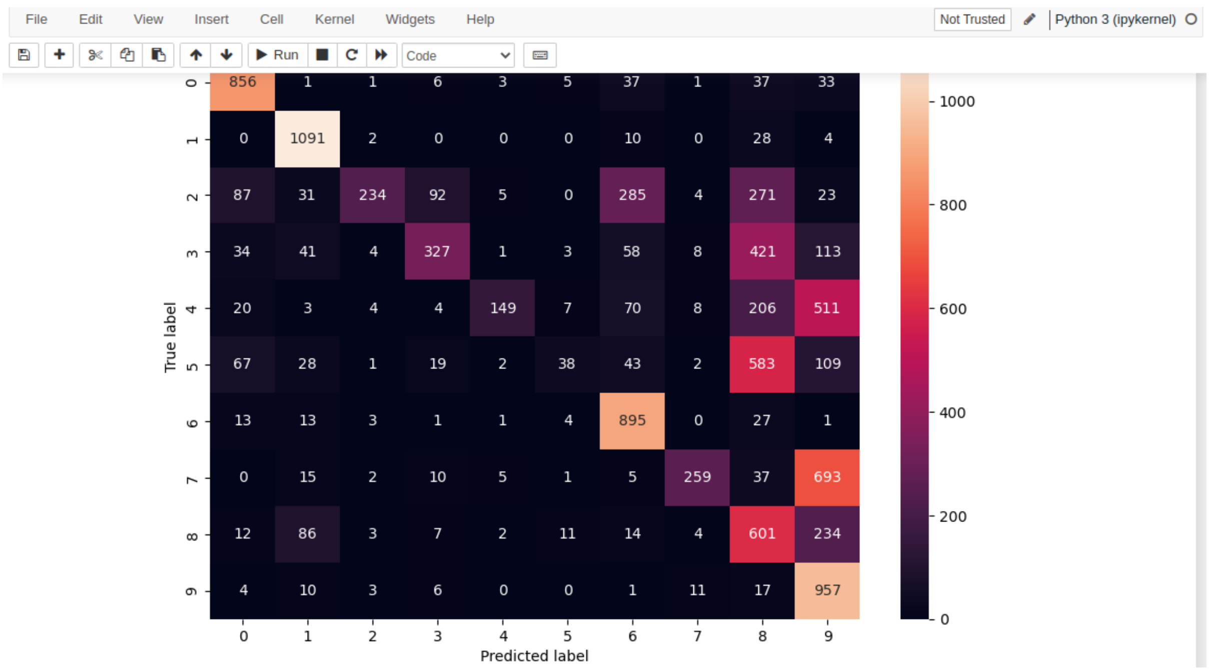
```
In [27]: #fitting test set data with the Gaussian Naive Bayes model
         GNB = GaussianNB()
         GNB.fit(X_test,y_test)

Out[27]: GaussianNB()

In [28]: X_test

Out[28]: array([[0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]])

In [29]: y_hat_GNB = GNB.predict(X) #Prediction y_hat

In [30]: accuracy(y,y_hat_GNB)
Out[30]: 0.5840430673844564
```

Figure 9: Fitting Gaussian model with the testing data



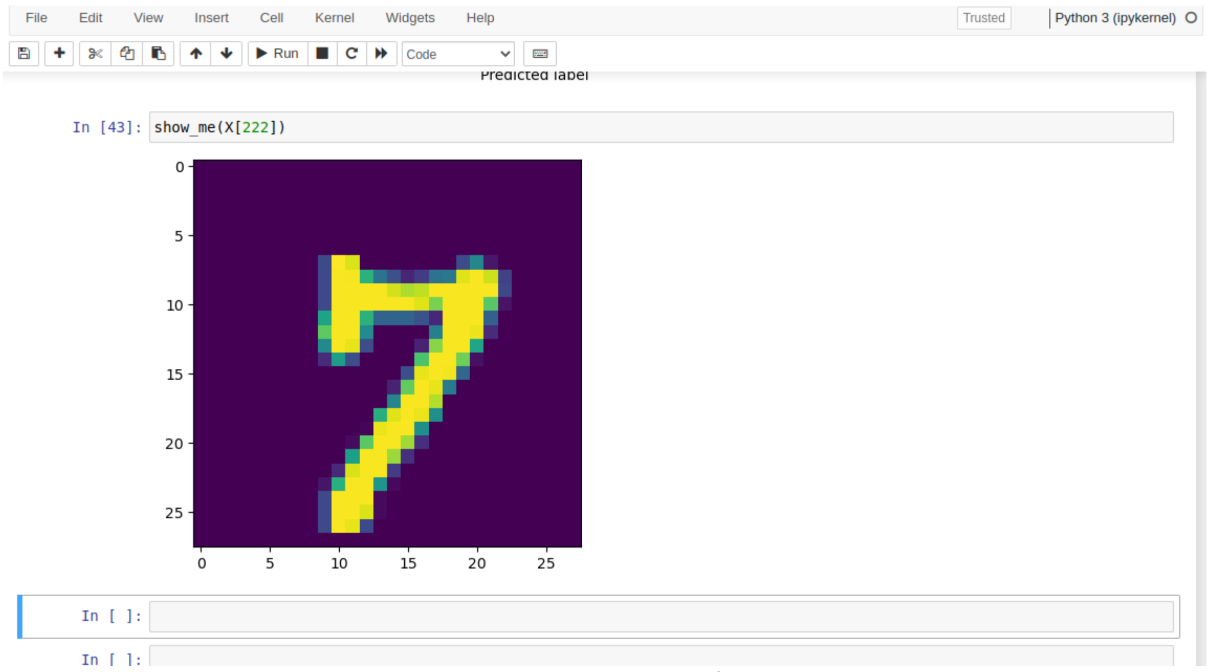
```
In [25]: y_test
Out[25]: array([2, 1, 0, ..., 4, 5, 6])

In [26]: y_hat_GB_test = GB.predict(X_test)

In [27]: y_hat_GNB_test = GNB.predict(X_test)

In [28]: accuracy(y_test,y_hat_GB_test)
Out[28]: 0.5407540754075407

In [35]: accuracy(y_test,y_hat_GNB_test)
Out[35]: 0.5737573757375738

In [36]: #making confusion matrix using sklearn and seaborn library
         plt.figure(figsize=(10,7))
         y_act = pd.Series(y_test, name = 'Actual')
         y_pred = pd.Series(y_hat_GB_test, name = 'Predicted')
         Confusion_matrix = pd.crosstab(y_act, y_pred)
         ax = sns.heatmap(Confusion_matrix, annot=True, fmt = "d")
         plt.ylabel('True label')
         plt.xlabel('Predicted label')
```

Figure 10 : Confusion matrix formation from the trained and tested data.



	0	1	2	3	4	5	6	7	8	9
0	856	1	1	6	3	5	37	1	37	33
1	0	1091	2	0	0	0	10	0	28	4
2	87	31	234	92	5	0	285	4	271	23
3	34	41	4	327	1	3	58	8	421	113
4	20	3	4	4	149	7	70	8	206	511
5	67	28	1	19	2	38	43	2	583	109
6	13	13	3	1	1	4	895	0	27	1
7	0	15	2	10	5	1	5	259	37	693
8	12	86	3	7	2	11	14	4	601	234
9	4	10	3	6	0	0	1	11	17	957

Figure 11 : Confusion matrix

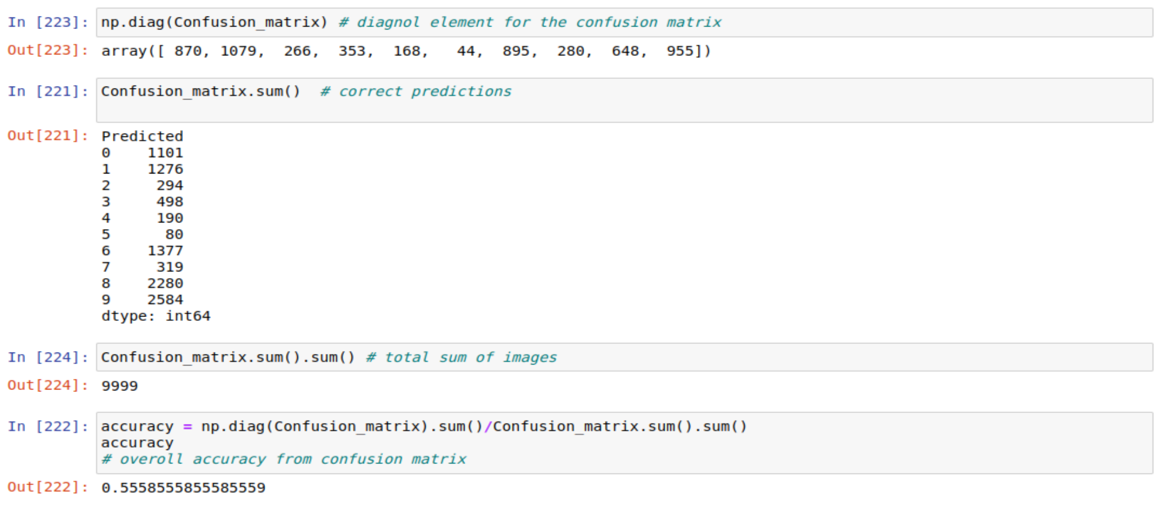


```
In [43]: show_me(X[222])
```

```
In [ ]:
```

```
In [ ]:
```

Figure 12 : Handwritten digit formation



```
In [223]: np.diag(Confusion_matrix) # diagnol element for the confusion matrix
Out[223]: array([ 870, 1079,  266,  353,  168,   44,  895,  280,  648,  955])

In [221]: Confusion_matrix.sum()  # correct predictions

Out[221]: Predicted
          0    1101
          1    1276
          2     294
          3     498
          4     190
          5      80
          6    1377
          7     319
          8    2280
          9    2584
          dtype: int64

In [224]: Confusion_matrix.sum().sum() # total sum of images
Out[224]: 9999

In [222]: accuracy = np.diag(Confusion_matrix).sum()/Confusion_matrix.sum().sum()
          accuracy
          # overoll accuracy from confusion matrix
Out[222]: 0.5558555855585559
```

Figure 13 : Accuracy obtained from confusion matrix

CONCLUSION

In this project, program for the Handwritten Digit Classification based on Gaussian naïve Bayes approach is successfully realized. The program is written using Jupyter Notebook. The classification accuracy obtained by applying Gaussian naïve approach is 55%. Many data does not guarantee a high value of accuracy.

REFERENCE

- https://scikit-learn.org/stable/search.html?q=overloading
- https://towardsdatascience.com/how-to-detect-constant-quasi-constant-features-in-your-dataset-a1ab7aea34b4
- https://courses.cs.washington.edu/courses/cse312/20su/files/student_drive/9.3.pdf
- https://pjreddie.com/projects/mnist-in-csv/
- http://gnjatovic.info/misc/naive.bayesian.classification.pdf
- https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html