Task 1: Handwritten Digit Classification

Name: SUPRAJEET VIJAY PATIL

Matriculation Number: 100004941

Date of Submission: 22.10.2025

Introduction

The **Naive Bayes classifier** is one of the most fundamental probabilistic approaches to supervised learning, derived directly from **Bayes' Theorem**, which provides a framework for updating the probability of a hypothesis as new evidence becomes available.
For a given class $c$and an input instance $x = (x_1, x_2, \dots, x_n)$, Bayes' theorem can be written as:

$$P(c \mid x) = \frac{P(x \mid c)\, P(c)}{P(x)}$$

where:

- $P(c \mid x)$— the **posterior probability** of class $c$given data x;
- $P(x \mid c)$— the **likelihood** of observing $x$if the class is c;
- $P(c)$— the **prior probability** of class c;
- $P(x)$— the **evidence**, acting as a normalization constant.

Since $P(x)$is the same for all classes, the classifier predicts the class with the largest posterior probability:

$$\hat{c} = \arg\max_{c \in C} P(c \mid x) = \arg\max_{c \in C} P(x \mid c) P(c)$$

1.1 The "Naive" Independence Assumption

The distinguishing feature of Naive Bayes is the **conditional independence assumption**:
it assumes that all features $x_i$are **independent of each other given the class**.
Thus, the likelihood term can be factorized as:

$$P(x \mid c) = \prod_{i=1}^{n} P(x_i \mid c)$$

This simplification makes the model extremely efficient, as it avoids the need to estimate a full joint probability distribution over all features.
Although the assumption is rarely true in real-world data (especially for images, where neighboring pixels are correlated), it often yields surprisingly good results and forms the basis for many introductory probabilistic classifiers.

1.2 Gaussian Naive Bayes

In practice, many datasets contain **continuous-valued features** rather than categorical ones.
To model continuous data, the **Gaussian Naive Bayes** variant assumes that the distribution of each feature $x_i$for a given class $c$follows a **normal (Gaussian) distribution**:

$$P(x_i \mid c) = \frac{1}{\sqrt{2\pi\sigma_{ic}^2}} \exp\left(-\frac{(x_i - \mu_{ic})^2}{2\sigma_{ic}^2}\right)$$

Methodology

The goal of this task was to implement and evaluate a **Gaussian Naive Bayes (GNB)** classifier for recognizing handwritten digits using the **MNIST dataset**.
The overall procedure involved four main stages: **data preparation**, **preprocessing**, **model training**, and **evaluation**.

Procedures Carried Out:

1. Data Preparation:
Two datasets were used:

- **mnist_train.csv** — containing 60,000 labeled training samples.
- **mnist_test.csv** — containing 10,000 labeled test samples.

Each record represents one 28×28 pixel grayscale image of a handwritten digit (0–9).
The first column denotes the **class label**, while the remaining 784 columns represent pixel intensity values ranging from 0 to 255.
The data were loaded using the **Pandas** library in Python.
The training set was used to estimate class-conditional statistics, while the test set was reserved exclusively for model evaluation to ensure fair performance measurement.

2. **Preprocessing:**
Because Gaussian Naive Bayes assumes continuous feature values that approximately follow a normal distribution, all pixel intensities were **normalized** by dividing by 255, resulting in a feature range of [0, 1].
This scaling ensures numerical stability and better alignment with the Gaussian assumption.
No additional dimensionality reduction or feature selection was applied, since the intent of this task was to evaluate the algorithm's raw analytical performance under its standard independence assumption.

3. **Model Construction:**
The **Gaussian Naive Bayes** classifier models each feature x_i in every class c as a Gaussian distribution parameterized by its **mean** (μ_{ic}) and **variance** (σ_{ic}^2):

$$P(x_i \mid c) = \frac{1}{\sqrt{2\pi\sigma_{ic}^2}} \exp\left(-\frac{(x_i - \mu_{ic})^2}{2\sigma_{ic}^2}\right)$$

During training, the algorithm estimates μ_{ic} and σ_{ic} from the training samples belonging to each class.
Class priors $P(c)$ are estimated as the relative frequency of each digit in the training data.
During prediction, the posterior probability for each class is computed as:

$$P(c \mid x) \propto P(c) \prod_{i=1}^{784} P(x_i \mid c)$$

and the predicted class is the one that maximizes this posterior probability:

$$\hat{c} = \arg\max_{c \in C} P(c \mid x)$$

To avoid numerical underflow, the product of probabilities was replaced with a sum of logarithms.
The classifier was implemented using the **GaussianNB** class from the **scikit-learn** library, which automates parameter estimation and numerical optimization.

4. **Training Procedure:**
The model was trained on the **normalized training set**.
GaussianNB.fit() computed all required Gaussian parameters for each of the ten digit classes (0–9).
The training process is computationally lightweight since it only requires mean and variance calculations rather than iterative optimization.

5. **Testing and Evaluation:**
After training, the model was applied to the test dataset using the predict() method.
The following metrics were used for performance evaluation:

- **Accuracy** — overall proportion of correctly classified samples.
- **Confusion Matrix** — to visualize class-wise prediction errors.
- **Precision, Recall, and F1-score** — to assess detailed per-class performance.

All computations and analyses were performed in a **Jupyter Notebook environment** using Python 3.
The evaluation results are presented in the following section.

PYTHON IMPLEMENTATION

[1]:

```
# Task 1 - Handwritten Digit Classification using Gaussian Naive Bayes
# Using separate MNIST training and testing datasets
# Strictly analytical version (no plots of digits)

import os
import pandas as pd
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

[2]:

```
# Load MNIST training and test datasets with path checking

def load_csv(filename):
    if not os.path.exists(filename):
        print(f"⚠️ File '{filename}' not found in current directory:", os.getcwd())
        filepath = input(f"Enter full path for {filename}: ").strip()
        return pd.read_csv(filepath)
    else:
        return pd.read_csv(filename)

train_data = load_csv('mnist_train.csv')
test_data = load_csv('mnist_test.csv')

print("Training data shape:", train_data.shape)
print("Test data shape:", test_data.shape)

print("\nFirst few rows of training data:")
display(train_data.head())
```

```
Training data shape: (60000, 785)
Test data shape: (10000, 785)

First few rows of training data:
```

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	28x19	28x20	28x21	28x22	28x23	28x24	28x25	28x26	28x27	28x28
0	5	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	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

[3]:
```
# Split features and labels
X_train = train_data.iloc[:, 1:].values
y_train = train_data.iloc[:, 0].values

X_test = test_data.iloc[:, 1:].values
y_test = test_data.iloc[:, 0].values

print("Training set:", X_train.shape, "Testing set:", X_test.shape)
```

```
Training set: (60000, 784) Testing set: (10000, 784)
```

[4]:
```
# Normalize pixel values (0-255 → 0-1)
X_train = X_train / 255.0
X_test = X_test / 255.0
```

[5]:
```
# Initialize and train Gaussian Naive Bayes model
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Predictions
y_pred = gnb.predict(X_test)
```

[6]:
```
# Evaluate the model
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Accuracy:", round(acc, 4))
print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", report)
```

```
Accuracy: 0.5558

Confusion Matrix:
 [[ 870    0    3    5    2    5   31    1   35   28]
 [   0 1079    2    1    0    0   10    0   38    5]
 [  79   25  266   91    5    2  269    4  271   20]
 [  32   39    6  353    2    3   51    8  409  107]
 [  19    2    5    4  168    7   63    7  210  497]
 [  71   25    1   20    3   44   40    2  586  100]
 [  12   12    3    1    1    7  895    0   26    1]
 [   0   15    2   10    5    1    5  280   39  671]
 [  13   72    3    7    3   11   12    4  648  201]
 [   5    7    3    6    1    0    1   13   18  955]]

Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.89      0.84       980
           1       0.85      0.95      0.90      1135
           2       0.90      0.26      0.40      1032
           3       0.71      0.35      0.47      1010
           4       0.88      0.17      0.29       982
           5       0.55      0.05      0.09       892
           6       0.65      0.93      0.77       958
           7       0.88      0.27      0.42      1028
           8       0.28      0.67      0.40       974
           9       0.37      0.95      0.53      1009

    accuracy                           0.56     10000
   macro avg       0.69      0.55      0.51     10000
weighted avg       0.69      0.56      0.52     10000
```

OUTPUT

```
Training data shape: (60000, 785)
Test data shape: (10000, 785)

First few rows of training data:
```

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	...	28x19	28x20	28x21	28x22	28x23	28x24	28x25	28x26	28x27	28x28
0	5	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	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

```
Accuracy: 0.5558

Confusion Matrix:
 [[ 870    0    3    5    2    5   31    1   35   28]
 [   0 1079    2    1    0    0   10    0   38    5]
 [  79   25  266   91    5    2  269    4  271   20]
 [  32   39    6  353    2    3   51    8  409  107]
 [  19    2    5    4  168    7   63    7  210  497]
 [  71   25    1   20    3   44   40    2  586  100]
 [  12   12    3    1    1    7  895    0   26    1]
 [   0   15    2   10    5    1    5  280   39  671]
 [  13   72    3    7    3   11   12    4  648  201]
 [   5    7    3    6    1    0    1   13   18  955]]

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       980
           1       0.85      0.95      0.90      1135
           2       0.90      0.26      0.40      1032
           3       0.71      0.35      0.47      1010
           4       0.88      0.17      0.29       982
           5       0.55      0.05      0.09       892
           6       0.65      0.93      0.77       958
           7       0.88      0.27      0.42      1028
           8       0.28      0.67      0.40       974
           9       0.37      0.95      0.53      1009

    accuracy                           0.56     10000
   macro avg       0.69      0.55      0.51     10000
weighted avg       0.69      0.56      0.52     10000
```

Evaluation & Comparison

The Gaussian Naive Bayes classifier achieved a **moderate accuracy**, typically around **50–60 %** on MNIST (depending on exact normalization and test subset).
While this is significantly lower than modern models such as **Random Forest** or **Neural Networks**, it aligns with theoretical expectations for GNB:

- **Strengths:**
  - Fast training and prediction.
  - Simple, interpretable probabilistic framework.
  - Works well for data that approximately follows a normal distribution.
- **Limitations:**
  - **Feature independence assumption** rarely holds for image pixels, leading to correlated errors.
  - The Gaussian assumption is only an approximation of the true pixel distribution.

When compared to the **Random Forest** (Task 2) or **Convolutional Neural Network** methods, GNB serves as a computationally efficient baseline but lacks the ability to model complex dependencies between pixels.

Conclusion

The project successfully implemented a **Gaussian Naive Bayes classifier** for handwritten digit recognition following the procedure outlined in Chapter 1 of *Milan Gnjatović's Machine Learning* lecture notes.
Using the MNIST dataset, the model learned class-conditional Gaussian parameters and classified unseen test samples analytically.
The results demonstrate that while **GNB provides a simple and fast baseline**, it is limited by its independence assumption and Gaussian modeling of features.
Nevertheless, it forms a fundamental probabilistic reference point for comparing more advanced algorithms such as Random Forest (Task 2) or Neural Network approaches.

References

- OpenCV Python Documentation: https://docs.opencv.org/
- Took reference from GeeksforGeeks: https://www.geeksforgeeks.org/machine-learning/gaussian-naive-bayes/
- Use of AI Tools ChatGpt, Gemini For Some Reference
- Lecture Notes on Machine Learning by Professor Gnjatovic,