Task 1: Handwritten Digit Classification (Random Forest)

Name: SUPRAJEET VIJAY PATIL

Matriculation Number: 100004941

Date of Submission: 5.11.2025

Introduction

The **Random Forest** algorithm is a robust ensemble-based approach to supervised learning derived from the foundations of **decision tree** classifiers.
A single decision tree recursively partitions the input space using attribute-value tests that maximize class purity—commonly measured by **information gain** or **Gini impurity**. While decision trees are simple and interpretable, they are prone to *overfitting* when trained on noisy or high-dimensional data such as images.
The Random Forest method, introduced by Breiman (2001) and discussed in Chapter 2 of Professor Milan Gnjatović's *Machine Learning* lecture notes, mitigates this limitation through **ensemble learning**.

In a Random Forest, a large number of individual trees are trained on **bootstrap samples** of the dataset—a technique known as *bagging (bootstrap aggregating)*.
During the construction of each tree, only a random subset of features is considered at each split.
This randomization promotes diversity among trees, ensuring that individual models make different errors.
When predicting, each tree casts a "vote" for a class, and the forest outputs the class with the majority of votes.
The overall predictive strength of the forest results from averaging or aggregating across many relatively weak but uncorrelated learners, thereby reducing variance without increasing bias.
Formally, if $h_k(x)$denotes the class predicted by the k^{th}decision tree for an input sample x, the Random Forest prediction is given by the majority-vote rule:

$$\hat{y} = \underset{k \in \{1,\ldots,N\}}{\mathbf{mode}} \, h_k(x)$$

where $N$is the total number of trees in the ensemble.
This task applies the Random Forest classifier to the **MNIST handwritten-digit dataset**, which contains 60 000 training and 10 000 testing examples of 28 × 28 grayscale images of digits 0–9.
Each image is represented as a vector of 784 pixel intensity values (0–255) followed by a label column.
In contrast to the Naïve Bayes approach used in Task 1, no feature normalization or transformation is performed; the algorithm operates directly on the raw pixel values.
The motivation for choosing Random Forest stems from its ability to handle **high-dimensional, nonlinear data** and automatically estimate **feature importance** while maintaining high classification accuracy and generalisation capability.
According to the lecture notes, ensemble methods such as Random Forest combine the low bias of complex models with the low variance of averaged predictions, producing reliable classifiers even when individual trees are unstable.

Methodology

Overview
We applied a Random Forest classifier directly on raw MNIST pixel values (user requirement: *no normalization*). The implementation follows the standard pipeline: load CSVs → split labels/features → train Random Forest → test → evaluate.

Data preparation (steps actually carried out)

1. **Files & format check** — Confirmed the two CSV files mnist_train.csv and mnist_test.csv are present and readable (each row: label, 784-pixel columns). (Task requirement: use MNIST CSVs).
2. **Load CSVs** — Read the files with a CSV reader (pandas read_csv) forcing numeric types to avoid mixed-type issues. No scaling or pixel normalization was applied (pixels kept in 0–255).
3. **Split** — y_train := first column (labels), X_train := remaining 784 columns (features). Same for test set.

Model training

1. **Model choice** — RandomForestClassifier (ensemble of decision trees). This matches the method introduced in the lecture notes: construct many decision trees and aggregate their votes to reduce variance and overfitting. Key concepts used from the notes include tree induction and the idea of ensembles to stabilise predictions.
2. **Hyperparameters (simple, reproducible)** — n_estimators = 100 (default-ish, moderate size), random_state = 42 for reproducibility, n_jobs = -1 to use available CPU cores. No feature selection or dimensionality reduction was performed.
3. **Training** — Fit the Random Forest on X_train and y_train.

Testing / Prediction

1. Use the trained forest to predict labels on X_test.
2. Collect predicted labels y_pred.

PYTHON IMPLEMENTATION

```python
# 📦 Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# 📂 Load MNIST dataset (CSV format)
# Specify header=None as the files do not have headers
train_df = pd.read_csv('/content/sample_data/mnist_train_small.csv', header=None)
test_df = pd.read_csv('/content/sample_data/mnist_test.csv', header=None)

# Rename the first column to 'label' as it contains the labels
train_df.rename(columns={0: 'label'}, inplace=True)
test_df.rename(columns={0: 'label'}, inplace=True)

# Split into features and labels
X_train = train_df.drop('label', axis=1)
y_train = train_df['label']
X_test = test_df.drop('label', axis=1)
y_test = test_df['label']

# 🌲 Build and train the improved Random Forest classifier
rf = RandomForestClassifier(
    n_estimators=400,        # number of trees
    max_depth=30,            # limit tree depth
    min_samples_split=5,     # minimum samples to split a node
    min_samples_leaf=2,      # minimum samples at leaf node
    max_features='sqrt',     # number of features considered per split
    criterion='entropy',     # use information gain
    bootstrap=True,          # enable bootstrap sampling
    class_weight='balanced', # handle class imbalance
    n_jobs=-1,               # use all CPU cores
    random_state=42          # reproducibility
)
rf.fit(X_train, y_train)

# 📊 Evaluate the model
y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:")
print(classification_report(y_test, y_pred))

# 🔍 Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

TEST CASES OUTPUT

Test case 1: -

```
// TEST CASE 1 :-

import pandas as pd  # Data handling
import numpy as np  # Numerical operations
import matplotlib.pyplot as plt  # Plotting library
from sklearn.ensemble import RandomForestClassifier  # Random Forest model
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score  # Evaluation metrics

# Load MNIST data (ensure CSVs are uploaded in Colab)
train = pd.read_csv('/content/sample_data/mnist_train_small.csv', header=None, dtype=int)  # Read training data as integers
test = pd.read_csv('/content/sample_data/mnist_test.csv', header=None, dtype=int)  # Read test data as integers

# Split dataset into features (X) and labels (y)
y_train = train.iloc[:, 0]  # First column = label
X_train = train.iloc[:, 1:]  # Remaining columns = pixel values
y_test = test.iloc[:, 0]  # Labels for test set
X_test = test.iloc[:, 1:]  # Features for test set

# Train Random Forest classifier
rf = RandomForestClassifier(n_estimators=450, random_state=42, n_jobs=-1, min_samples_leaf=1,  min_samples_split=2,  max_depth=25,  max_features='sqrt')
rf.fit(X_train, y_train)  # Train model

# Predict on test data
y_pred = rf.predict(X_test)  # Generate predictions

# Evaluate performance
acc = accuracy_score(y_test, y_pred)  # Calculate accuracy
print(f"Test Accuracy: {acc:.4f}\n")  # Display accuracy

print("Classification Report:")  # Show precision, recall, F1-score
print(classification_report(y_test, y_pred, digits=4))

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)  # Compute confusion matrix
plt.figure(figsize=(8,6))  # Set plot size
plt.imshow(cm, interpolation='nearest', cmap='Blues')  # Visualize matrix
plt.title('Confusion Matrix')  # Add title
plt.xlabel('Predicted Label')  # X-axis label
plt.ylabel('True Label')  # Y-axis label
plt.colorbar()  # Add color scale
plt.show()  # Display plot
```

```
Test Accuracy: 0.9605

Classification Report:
              precision    recall  f1-score   support

           0     0.9613    0.9888    0.9748       980
           1     0.9859    0.9868    0.9863      1135
           2     0.9493    0.9612    0.9552      1032
           3     0.9476    0.9495    0.9486      1010
           4     0.9681    0.9582    0.9632       982
           5     0.9679    0.9451    0.9563       892
           6     0.9648    0.9739    0.9694       958
           7     0.9691    0.9455    0.9572      1028
           8     0.9553    0.9446    0.9499       974
           9     0.9345    0.9475    0.9409      1009

    accuracy                         0.9605     10000
   macro avg     0.9604    0.9601    0.9602     10000
weighted avg     0.9606    0.9605    0.9605     10000
```

Test Case 2 :-

```
# TEST CASE 2 :-

import pandas as pd  # Data handling
import numpy as np  # Numerical operations
import matplotlib.pyplot as plt  # Plotting library
from sklearn.ensemble import RandomForestClassifier  # Random Forest model
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score  # Evaluation metrics

# Load MNIST data (ensure CSVs are uploaded in Colab)
train = pd.read_csv('/content/sample_data/mnist_train_small.csv', header=None, dtype=int)  # Read training data as integers
test = pd.read_csv('/content/sample_data/mnist_test.csv', header=None, dtype=int)  # Read test data as integers

# Split dataset into features (X) and labels (y)
y_train = train.iloc[:, 0]  # First column = label
X_train = train.iloc[:, 1:]  # Remaining columns = pixel values
y_test = test.iloc[:, 0]  # Labels for test set
X_test = test.iloc[:, 1:]  # Features for test set

# Train Random Forest classifier
rf = RandomForestClassifier(n_estimators=300, random_state=42, n_jobs=-1, min_samples_leaf=2,  min_samples_split=5,  max_depth=30,  max_features='sqrt')  # 100 trees, parallel processing
rf.fit(X_train, y_train)  # Train model

# Predict on test data
y_pred = rf.predict(X_test)  # Generate predictions

# Evaluate performance
acc = accuracy_score(y_test, y_pred)  # Calculate accuracy
print(f"Test Accuracy: {acc:.4f}\n")  # Display accuracy

print("Classification Report:")  # Show precision, recall, F1-score
print(classification_report(y_test, y_pred, digits=4))

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)  # Compute confusion matrix
plt.figure(figsize=(8,6))  # Set plot size
plt.imshow(cm, interpolation='nearest', cmap='Blues')  # Visualize matrix
plt.title('Confusion Matrix')  # Add title
plt.xlabel('Predicted Label')  # X-axis label
plt.ylabel('True Label')  # Y-axis label
plt.colorbar()  # Add color scale
plt.show()  # Display plot
```

```
Test Accuracy: 0.9598

Classification Report:
              precision    recall  f1-score   support

           0     0.9651    0.9878    0.9763       980
           1     0.9851    0.9885    0.9868      1135
           2     0.9491    0.9583    0.9537      1032
           3     0.9515    0.9515    0.9515      1010
           4     0.9631    0.9572    0.9602       982
           5     0.9690    0.9473    0.9580       892
           6     0.9630    0.9770    0.9699       958
           7     0.9651    0.9416    0.9532      1028
           8     0.9533    0.9425    0.9479       974
           9     0.9324    0.9425    0.9374      1009

    accuracy                         0.9598     10000
   macro avg     0.9597    0.9594    0.9595     10000
weighted avg     0.9598    0.9598    0.9598     10000
```

Test Case 3:

```python
# TEST CASE 3 :-

import pandas as pd  # Data handling
import numpy as np  # Numerical operations
import matplotlib.pyplot as plt  # Plotting library
from sklearn.ensemble import RandomForestClassifier  # Random Forest model
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score  # Evaluation metrics

# Load MNIST data (ensure CSVs are uploaded in Colab)
train = pd.read_csv('/content/sample_data/mnist_train_small.csv', header=None, dtype=int)  # Read training data as integers
test = pd.read_csv('/content/sample_data/mnist_test.csv', header=None, dtype=int)  # Read test data as integers

# Split dataset into features (X) and labels (y)
y_train = train.iloc[:, 0]  # First column = label
X_train = train.iloc[:, 1:]  # Remaining columns = pixel values
y_test = test.iloc[:, 0]  # Labels for test set
X_test = test.iloc[:, 1:]  # Features for test set

# Train Random Forest classifier
rf = RandomForestClassifier(n_estimators=250, random_state=45, n_jobs=-1, min_samples_leaf=2,  min_samples_split=2,  max_depth=30,  max_features='sqrt')  # 100 trees, parallel processing
rf.fit(X_train, y_train)  # Train model

# Predict on test data
y_pred = rf.predict(X_test)  # Generate predictions

# Evaluate performance
acc = accuracy_score(y_test, y_pred)  # Calculate accuracy
print(f"Test Accuracy: {acc:.4f}\n")  # Display accuracy

print("Classification Report:")  # Show precision, recall, F1-score
print(classification_report(y_test, y_pred, digits=4))

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)  # Compute confusion matrix
plt.figure(figsize=(8,6))  # Set plot size
plt.imshow(cm, interpolation='nearest', cmap='Blues')  # Visualize matrix
plt.title('Confusion Matrix')  # Add title
plt.xlabel('Predicted Label')  # X-axis label
plt.ylabel('True Label')  # Y-axis label
plt.colorbar()  # Add color scale
plt.show()  # Display plot
```

```
Test Accuracy: 0.9587

Classification Report:
              precision    recall  f1-score   support

           0     0.9613    0.9888    0.9748       980
           1     0.9833    0.9885    0.9859      1135
           2     0.9508    0.9554    0.9531      1032
           3     0.9486    0.9505    0.9496      1010
           4     0.9660    0.9552    0.9606       982
           5     0.9611    0.9417    0.9513       892
           6     0.9638    0.9718    0.9678       958
           7     0.9662    0.9455    0.9558      1028
           8     0.9522    0.9415    0.9468       974
           9     0.9315    0.9435    0.9375      1009

    accuracy                         0.9587     10000
   macro avg     0.9585    0.9582    0.9583     10000
weighted avg     0.9587    0.9587    0.9587     10000
```

Evaluation & Comparison

Evaluation follows the metrics and recommendations in the lecture notes (confusion matrix, accuracy, precision, recall) since accuracy alone can be misleading for imbalanced classes; confusion matrix and class-wise precision/recall give more insight

Metrics computed

- **Accuracy** — fraction of test instances correctly classified (diagonal of confusion matrix divided by total).
- **Confusion matrix** — a K×K contingency table showing true vs predicted classes; used to observe per-digit confusions
- **Precision** (per class) — fraction of examples predicted as class *c* that are actually *c*.
- **Recall (Sensitivity)** (per class) — fraction of true examples of class *c* that were correctly predicted.
- **Classification report** — presents precision, recall, F1-score per class and macro/weighted averages.

Practical comparison (Random Forest vs. other methods)

- The lecture notes present Gaussian Naive Bayes as an alternative for numerical features (Task 1). Gaussian NB is computationally cheaper but typically less accurate on high-dimensional image data where pixel dependencies matter. Random Forests, by contrast, learn non-linear decision boundaries and can model feature interactions via tree splits; we therefore expect higher accuracy at the cost of higher training time and memory. This trade-off is consistent with Chapter 1 (Gaussian NB) and Chapter 2 (Random Forests) discussions in the lecture notes.

How evaluation was performed

1. Compute predictions y_pred on the test set.
2. Compute overall accuracy and display the confusion matrix.
3. Produce the classification report (precision/recall/F1) per digit to identify which digits are most often confused. This follows the recommended evaluation procedure outlined in the lecture notes.

Conclusion

The Random Forest classifier implemented for Task 2 provides a straightforward, robust method for MNIST digit classification without feature preprocessing. It leverages ensemble averaging of decision trees to reduce variance and typically achieve good accuracy on the dataset (as suggested by the theory in the lecture notes).

Compared to Gaussian Naive Bayes (Task 1 approach), Random Forests generally yield superior accuracy on raw pixel data because trees can capture interactions between pixels; however, they require more computational resources.

For completeness, evaluation must show the confusion matrix and class-wise scores: these help identify systematic errors (e.g., commonly confused digits) and guide future improvements such as feature engineering, pixel normalization, or switching to more powerful models (CNNs) if required.

References

- OpenCV Python Documentation: https://docs.opencv.org/
- Took reference from GeeksforGeeks: https://www.geeksforgeeks.org/machine-learning/RandomForestClassifier /
- Use of AI Tools ChatGPT, Gemini for Some Reference
- M. Gnjatović, Machine Learning — Lecture Notes, Chapter 1 (evaluation metrics) and Chapter 2 (Decision Trees and Random Forests). (Provided lecture notes).