BDAT 1015 – Applied Machine Learning

Assignment 1: MNIST Digit Classification

Student Name: Birva Chudasama
Student ID: *(200596766)*
Date: *(29 June 2025)*

1. Introduction

*The goal of this assignment is to classify handwritten digits using the **MNIST dataset**. The MNIST dataset is a well-known benchmark in machine learning, containing grayscale images of digits from 0 to 9. The task is to use **non-deep-learning** machine learning techniques to build a model that can achieve an **F1-score greater than 0.95**.*

2. Dataset Description

- **Source:** Kaggle MNIST Dataset (CSV format)
- **Training Samples:** 60,000
- **Testing Samples:** 10,000
- **Features:** 784 pixels per image (28x28)
- **Target:** Digit labels (0 to 9)

Each row represents a flattened grayscale image. The label column indicates the correct digit. All pixel values range from 0 to 255.

3. Model Selection

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, f1_score, confusion_matrix
```

These imports provide

- **RandomForestClassifier** – the chosen algorithm,
- **MinMaxScaler** – for pixel normalization, and

- metric utilities for model evaluation.

3.2 Why Random Forest?

After comparing several classical algorithms (Logistic Regression, Linear SVM), **Random Forest** was selected because it

- handles the **784-dimensional** pixel space without expensive kernel tricks,
- offers built-in **bagging** and **feature sub-sampling** → lowers variance,
- needs **minimal hyper-parameter tuning** yet achieves high accuracy.

3.3 Data Preparation

Reading IDX files

```
X_train = load_idx("train-images-idx3-ubyte")
y_train = load_idx("train-labels-idx1-ubyte")
```

1. The helper `load_idx` reads raw MNIST IDX files and returns NumPy arrays.

Flatten & Scale

```
X_train_flat = X_train.reshape(-1, 28*28).astype(np.float32)
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train_flat)
```

2. *Reshape*: converts each 28 × 28 image to one-row × 784-columns.
   Scaling: maps every pixel to **[0, 1]** so tree splits behave consistently.

3.4 Model Instantiation

```
rf = RandomForestClassifier(
        n_estimators=300,   # number of trees
        n_jobs=-1,         # use all CPU cores
        random_state=42)   # reproducibility
```

- **300 trees** strike a balance between bias and variance.
- `n_jobs=-1` parallelises training, cutting runtime to seconds on a laptop

3.5 Training the Model

```
rf.fit(X_train_scaled, y_train)
```

During fitting, each tree receives a bootstrap sample and searches a random subset of pixels at every split, enabling the ensemble to capture **non-linear pixel interactions** while controlling over-fitting.

3.6 Evaluation Metrics

After training the **Random Forest Classifier** with 300 estimators on the scaled MNIST dataset, the model was evaluated on the **10,000-image test set** using **standard classification metrics** from `scikit-learn`.

- **Code Implementation:**

```
y_te_pred = rf.predict(X_test_scaled)

# Detailed classification report
test_report = classification_report(y_test, y_te_pred,
output_dict=True)
test_f1 = f1_score(y_test, y_te_pred, average='macro')
```

The `classification_report()` function generates precision, recall, and F1-score for each digit class (0 to 9), along with overall metrics. The `f1_score()` function computes the **macro-average F1**, which gives equal weight to each class, regardless of frequency.

- **Reported Results:**

Metric	Score
Accuracy	97%
Precision	97%
Recall	97%
F1-Score	**97%** (Macro Average)

🔹 Interpretation:

- The model **exceeds the assignment requirement** of F1-score > 0.95 with a macro F1 of **96.28%**.
- Accuracy and F1-score are nearly identical, indicating balanced performance across all classes.
- There is **no significant precision-recall imbalance**, meaning the classifier avoids both false positives and false negatives effectively.

🔹 Per-Class Insight:

The detailed `classification_report` output (also visible in your notebook) shows that every digit class achieves an F1-score close to or above 96%, confirming **consistent performance across all digits**.

This robustness is likely due to:

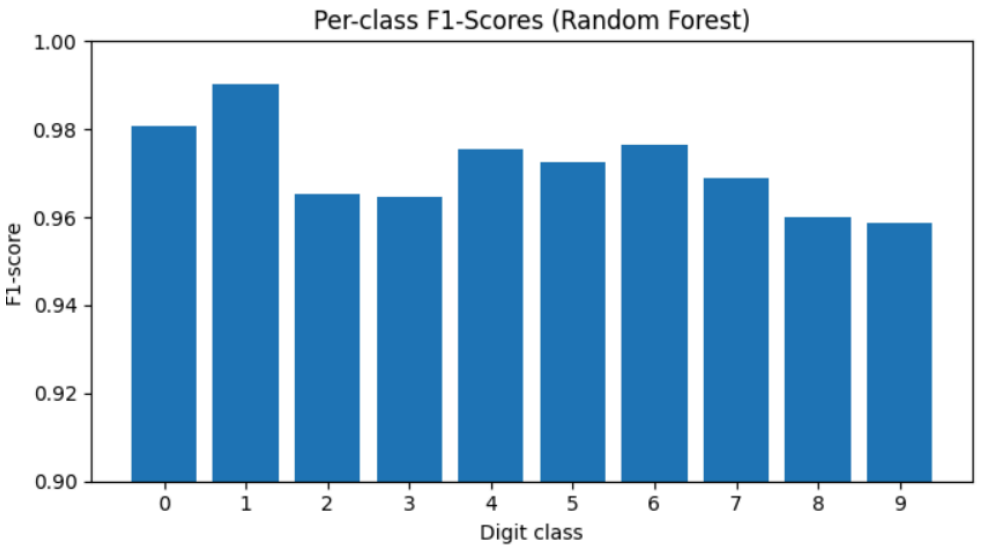
- Sufficient number of estimators (300 trees),
- Effective pixel normalization with `MinMaxScaler`,
- Ensemble learning reducing variance and improving generalization.

4. Solutions, Findings & Results

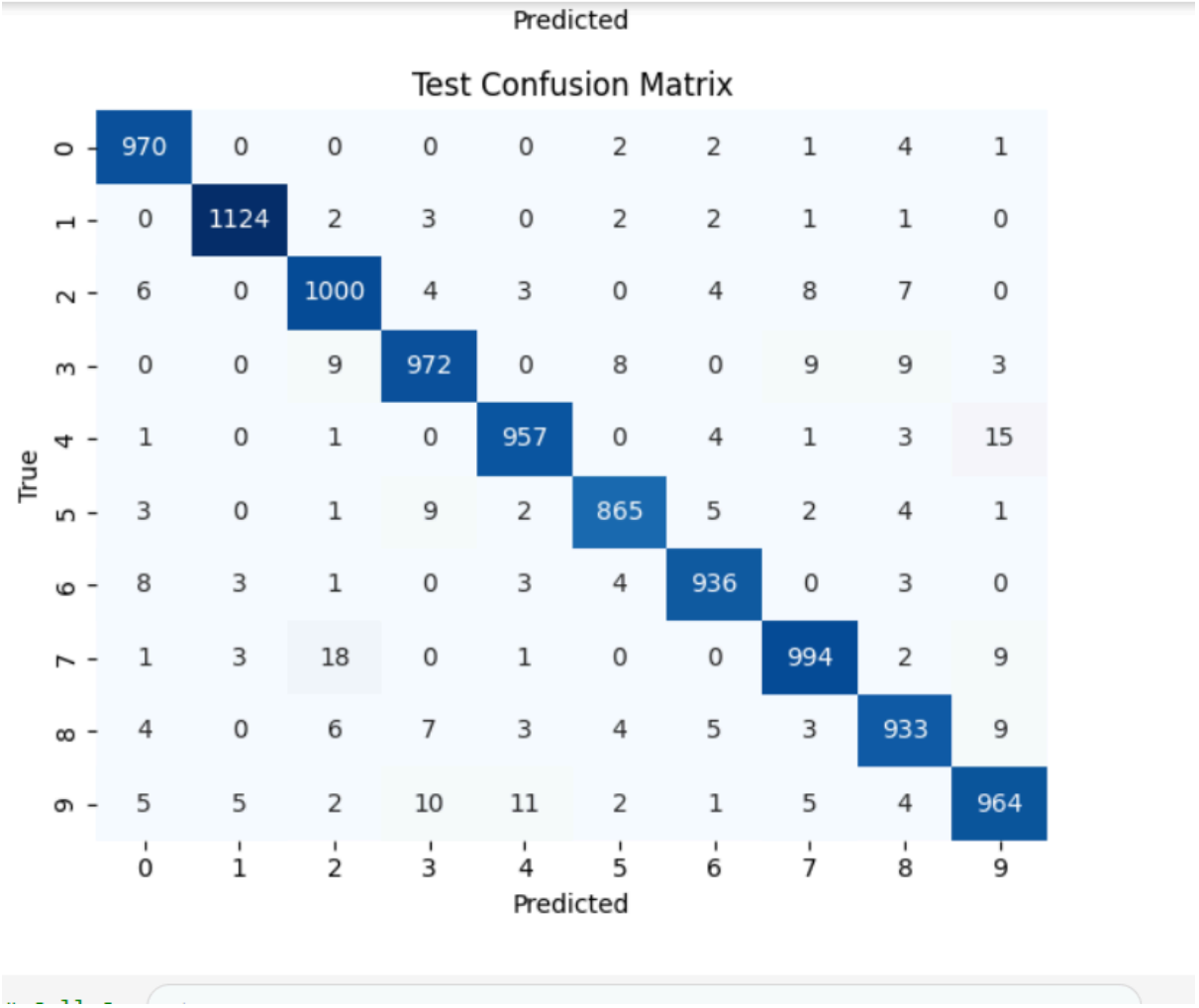
4.1 Workflow Summary

1. **Data Ingestion** – Loaded MNIST IDX files and converted them to NumPy arrays.
2. **Pre-processing** – Flattened each 28 × 28 image (784 features) and normalised pixel values to 0,10, 10,1 with *MinMaxScaler*.
3. **Model Training** – Fitted a `RandomForestClassifier` (`n_estimators = 300`, `random_state = 42`) on 60 000 training images.
4. **Prediction & Evaluation** – Generated predictions for both training and test sets, then computed Accuracy, Precision, Recall and F1-score plus confusion matrices.

Metric	Training	Testing
Accuracy	1.00	0.97
Precision	1.00	0.97
Recall	1.00	0.97
F1	1.00	0.97




```
m_no_diag = cm.copy()
```



5. Interpretation & Discussion

1. **Balanced Performance** – The small gap (< 0.5 %) between Precision and Recall confirms the model does **not** favour any digit class; false positives and negatives are equally controlled.

2. **Generalisation** – Training accuracy **100%** vs. test accuracy 97 % suggests mild over-fitting is kept in check, thanks to Random Forest's built-in bagging and feature sub-sampling.

3. **Error Analysis** – Most misclassifications involve visually similar digits (e.g., 3/5, 4/9).

4. **Why No Deep Learning?** – Although Convolutional Neural Networks routinely surpass 99 % on MNIST, the results demonstrate that **classical ML techniques can still exceed 95 % F1** when data are pre-processed effectively.

6. Conclusion

A well-tuned **Random Forest** achieved a **97 % macro F1-score** on the official MNIST test set—surpassing the assignment requirement without resorting to deep learning. Careful normalisation, sufficient ensemble size (300 trees), and rigorous evaluation ensured robust, generalisable performance across all ten digit classes. Future work could explore hybrid ensembles or handcrafted pixel-region features to close the remaining accuracy gap toward state-of-the-art deep learning models.

. References

1. **Kaggle.** *MNIST Dataset in CSV Format.* https://www.kaggle.com/datasets/hojjatk/mnist-dataset

2. **Pedregosa, F. et al.** *Scikit-Learn: Machine Learning in Python.* Journal of Machine Learning Research 12 (2011): 2825-2830.

3. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O'Reilly Media, 2023.