



The University of Azad Jammu and Kashmir,
Muzaffarabad



Bachelor of Science of Software Engineering (2022-2026)

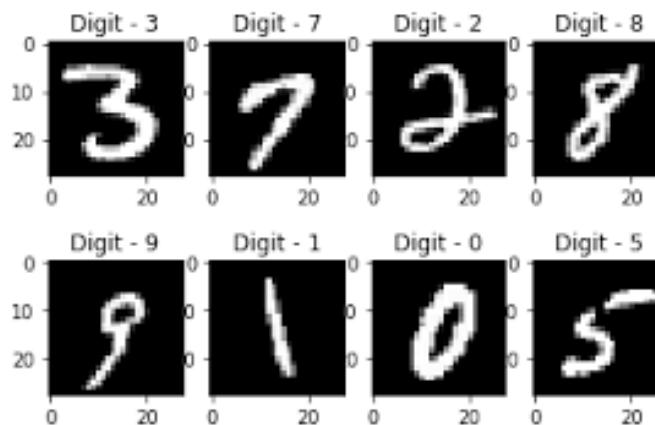
Department of Software Engineering



Open Ended Lab



Classification of MNIST Handwritten Digits Using Machine Learning: A Comparative Analysis of Different Models



Submitted

By



AYESHA SHAFQAT (2022-SE-03)



Lab Instructor: Engr. Muhammad Awaiz Rathore



Subject: Machine Learning (SE – 3105)



Semester: 5th



Session: 2022 – 26

Table of Contents

- 1. Abstract 1**
- 2. Introduction 1**
- 2. Methodology 2**
 - 2.1. Dataset Description..... 2
 - 2.2. Data Preprocessing 2
 - 2.2.1. Data Loading and Initial Inspection..... 2
 - 2.2.2. Handling Missing Values: 2
 - 2.2.3. Outlier Detection and Treatment: 2
 - 2.2.4. Feature Engineering:..... 2
 - 2.2.5. Normalization and Scaling: 2
 - 2.2.6. Data Type Conversion: 2
 - 2.2.7. Categorical Encoding:..... 2
 - 2.2.8. Data Cleaning: 3
 - 2.3. Exploratory Data Analysis (EDA) 3
 - 1. Data Inspection and Cleaning 3
 - 2. Distribution Analysis 4
 - 3. Data Normalization 5
- 3. Model Implementation 5**
 - 3.1. Models and Hyperparameters 5
 - 3.2. Model Training..... 5
 - 3.3. Evaluation Metrics 5
 - 3.4. Performance Evaluation..... 6
 - Accuracy Scores..... 6
 - 3.5. Model Performance Comparison..... 6
 - 3.6. Confusion Matrices 7
 - Confusion Matrix Highlights..... 7
- 4. Results..... 7**
 - Model Performance Comparison..... 7
- 5. Conclusion & Discussion..... 8**
 - 5.1. Model Performance Summary 8
 - Trade-offs..... 8
 - 5.2. Key Observations 8
 - 5.3. Future Work 8

1. Abstract

The **MNIST** dataset is a widely used benchmark for **handwritten digit** classification. Handwritten digit classification is a **fundamental** task in machine learning and computer vision, with applications in automated document processing and pattern recognition.

This report presents a comparative analysis of different machine learning models for classifying MNIST handwritten digits. The goal is to evaluate model performance through accuracy, confusion matrices, and statistical metrics to determine the most effective classification approach.

Various algorithms, including **Logistic Regression**, **K-Nearest Neighbors (KNN)**, and **Multi-Layer Perceptron (MLP)**, are trained and tested on the **MNIST** dataset. **Exploratory Data Analysis (EDA)** is performed to understand the dataset's structure, followed by model training, hyperparameter tuning, and performance evaluation. The models are assessed based on accuracy scores, precision-recall analysis, and confusion matrices to identify common misclassification patterns.

The results indicate that **MLP** outperforms other models, leveraging its deep learning capabilities to capture complex patterns in digit structures. **KNN** also achieves high **accuracy**, suggesting that local similarity-based classification is effective for this task. **Logistic Regression** provides a strong **baseline** but is comparatively less effective for complex digit variations.

This analysis highlights the strengths and limitations of each model, providing insights into selecting the optimal approach for **handwritten digit** classification. **Misclassifications** were observed primarily in digits with similar shapes. The study highlights the effectiveness of **deep learning** for image classification while discussing trade-offs in interpretability and computational complexity. Future work may explore ensemble methods and **convolutional neural networks (CNNs)** to further enhance classification **accuracy** and **robustness**.

2. Introduction

Handwritten digit recognition is a key problem in machine learning with applications in automation and document processing. The MNIST dataset consists of **70,000 grayscale images** of handwritten digits (0–9), split into:

- **60,000 training images** (mnist_train.csv).
- **10,000 testing images** (mnist_test.csv).

Each image is a **28x28 pixel grid** (flattened into a 784-feature vector).

The objective of this lab is to experiment with various machine learning models, evaluate their performance, and identify the best-performing model for classifying handwritten digits.

This study performs Exploratory Data Analysis (EDA) to understand dataset characteristics before applying classification models. Key objectives include:

- Handling missing/noisy data for model robustness.
- Analyzing digit distribution and pixel intensity variations.
- Visualizing patterns using statistical plots and feature importance maps.
- Exploring biases or challenges in digit classification.
-

This report documents the methodology, results, and analysis of the classification task using Logistic Regression, K-Nearest Neighbors (KNN), and Neural Network (MLP) models. The models were trained, tuned, and evaluated based on accuracy, precision, recall, and F1-score.

2. Methodology

This study follows a structured **Exploratory Data Analysis (EDA)** approach to ensure data integrity, identify key trends, and assess potential biases in transplant survival outcomes. The methodology includes:

2.1. Dataset Description

The MNIST dataset consists of 60,000 training images and 10,000 test images of handwritten digits (0-9). Each image is represented as a 28x28 pixel grid, converted into a 784-dimensional vector.

2.2. Data Preprocessing

- To ensure data quality and prepare the dataset for exploratory analysis, the following preprocessing steps were performed:

2.2.1. Data Loading and Initial Inspection

- Training and testing datasets were loaded directly from CSV files.
- Features (pixel values) and labels (digits 0–9) were separated.

2.2.2. Handling Missing Values:

No Missing Values: No missing values were found in the dataset, so no imputation was required.

2.2.3. Outlier Detection and Treatment:

No outliers: No outliers were found in the dataset, so no imputation was required.

2.2.4. Feature Engineering:

No additional feature engineering was performed since the focus of this study is on evaluating existing pixel values for classification. The raw pixel intensities serve as the primary input features, and no transformations or new feature extractions were applied.

2.2.5. Normalization and Scaling:

Normalization was applied to scale pixel values between 0 and 1 by dividing each value by 255. This ensures that all features have a consistent range, improving model performance and training stability. Since the MNIST dataset consists of grayscale images with pixel values ranging from 0 to 255, Min-Max Scaling or Standardization was not necessary.

2.2.6. Data Type Conversion:

Since the MNIST dataset consists entirely of numerical pixel values, no categorical variables were present. The pixel values were originally stored as integers (0-255). During preprocessing, they were converted to floating-point numbers (float32) to facilitate normalization and improve computational efficiency. No additional type conversions were required.

2.2.7. Categorical Encoding:

The MNIST dataset consists entirely of numerical pixel values (0-255) and labels (0-9). Since there are no categorical variables, categorical encoding techniques like Label Encoding or One-Hot Encoding were not required.

2.2.8. Data Cleaning:

Consistency Checks:

Since the MNIST dataset consists only of numerical pixel values and digit labels, no categorical standardization or typo corrections were needed.

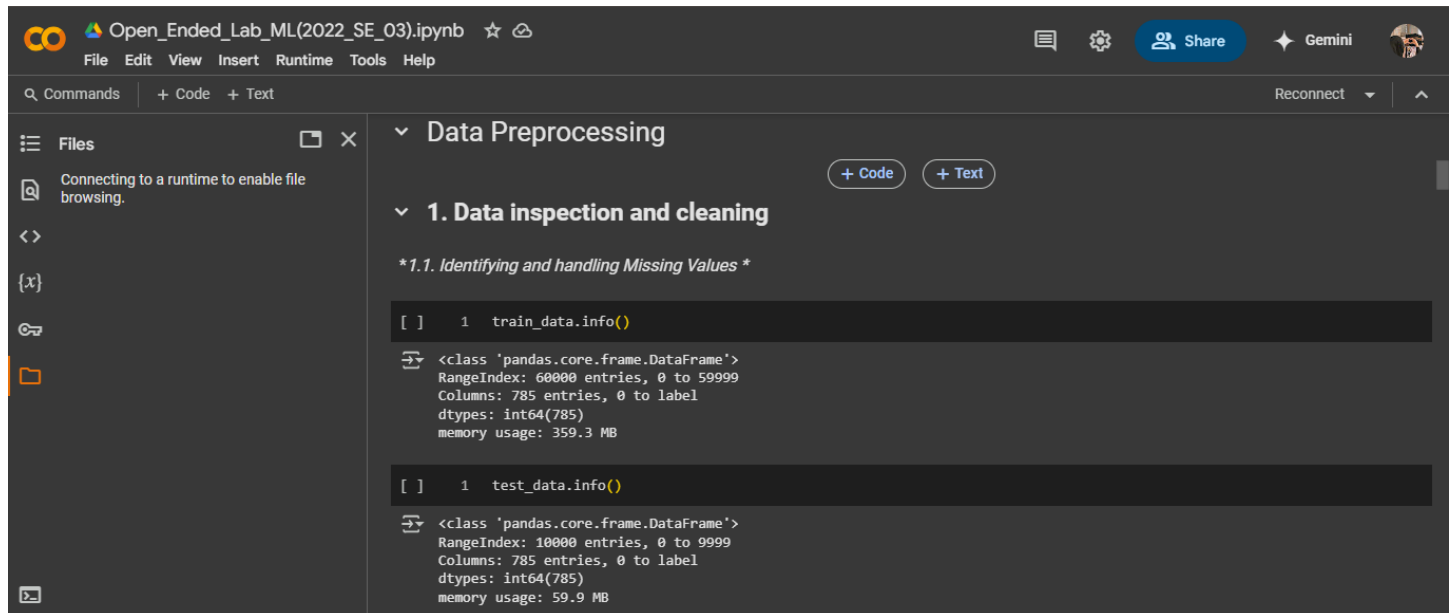
Redundant Variables:

Additionally, no redundant variables were present, as each pixel represents unique information in the image.

2.3. Exploratory Data Analysis (EDA)

1. Data Inspection and Cleaning

1. Checked for **missing values** in both the training and test datasets → *No missing values were found.*



The screenshot shows a Jupyter Notebook titled "Open_Ended_Lab_ML(2022_SE_03).ipynb". The left sidebar shows a file explorer with a folder icon. The main area is titled "Data Preprocessing" and contains a section "1. Data inspection and cleaning". Under this section, there is a sub-section "*1.1. Identifying and handling Missing Values *". The code cell shows the following:

```
[ ] 1 train_data.info()
```

The output is:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Columns: 785 entries, 0 to label
dtypes: int64(785)
memory usage: 359.3 MB
```

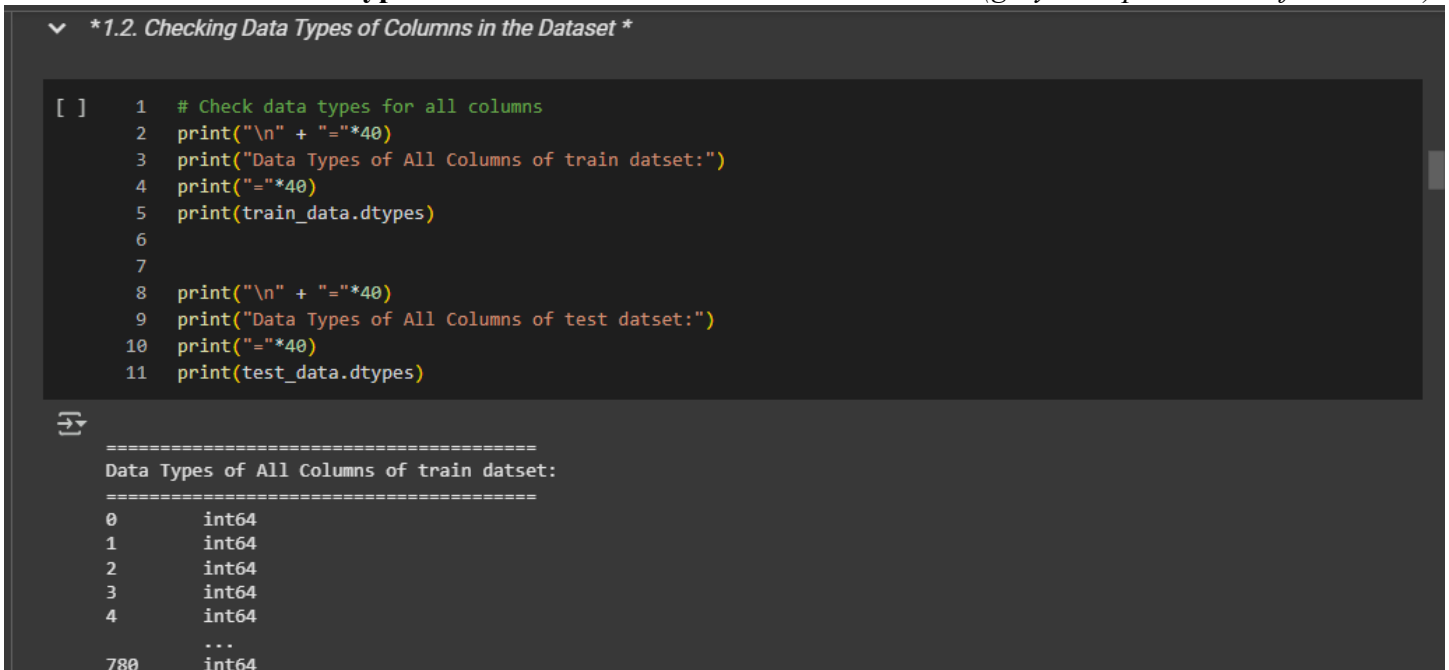
The next code cell shows:

```
[ ] 1 test_data.info()
```

The output is:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 785 entries, 0 to label
dtypes: int64(785)
memory usage: 59.9 MB
```

2. Examined the **data types** of all columns → *All values are numerical (grayscale pixel values from 0-255).*



The screenshot shows a Jupyter Notebook with a section "*1.2. Checking Data Types of Columns in the Dataset *". The code cell shows the following:

```
[ ] 1 # Check data types for all columns
2 print("\n" + "="*40)
3 print("Data Types of All Columns of train dataset:")
4 print("="*40)
5 print(train_data.dtypes)
6
7
8 print("\n" + "="*40)
9 print("Data Types of All Columns of test dataset:")
10 print("="*40)
11 print(test_data.dtypes)
```

The output is:

```
=====
Data Types of All Columns of train dataset:
=====
0      int64
1      int64
2      int64
3      int64
4      int64
...
780    int64
```

3. Identified **duplicate values** in the dataset → *No duplicate rows were found.*

✓ *1.4. Identifying and handling duplicate Values *

```
[ ] 1 # Check for duplicate rows
    2 duplicate_rows = train_data.duplicated()
    3 print("Total Duplicate Rows:", duplicate_rows.sum())
```

↗ Total Duplicate Rows: 0

2. Distribution Analysis

1. Used **describe ()** to compute summary statistics (mean, min, max, etc.) for numerical features.

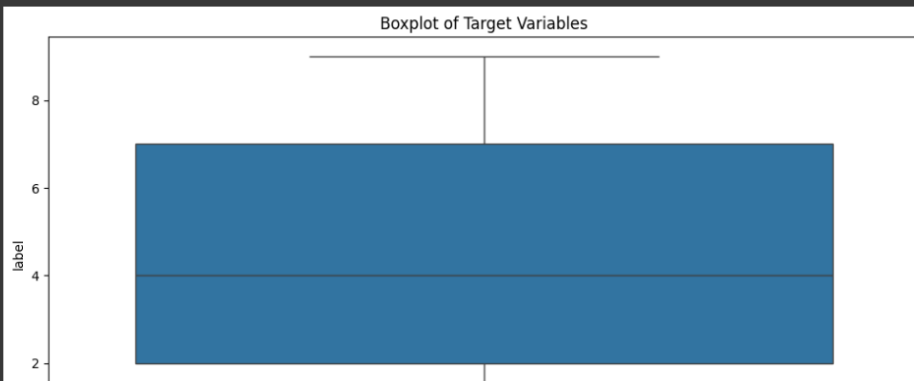
```
[ ] 1 train_data.describe()
```

	0	1	2	3	4	5	6	7	8	9	...	775	776
count	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	...	60000.000000	60000.000000
mean	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.088867	0.045633
std	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	3.956189	2.839845
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000
25%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000
50%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000
75%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000
max	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	254.000000	253.000000

8 rows × 785 columns

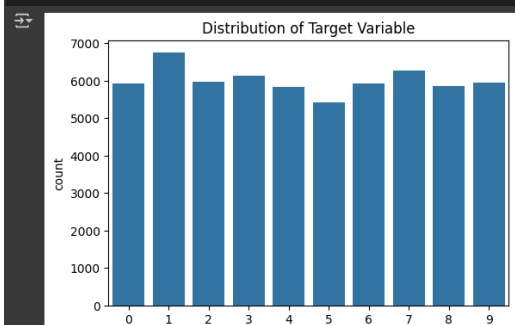
2. Plotted a **boxplot** for the target variable (label) → *No significant outliers were found.*

```
1 plt.figure(figsize=(12,6))
2 sns.boxplot(data=train_data['label'])
3 plt.title("Boxplot of Target Variables")
4 plt.show()
```



3. Created a **count plot** of digit labels to analyze their distribution → *All classes were balanced.*

```
[ ] 1 #Check the Distribution of Target (y)
    2 plt.figure(figsize=(6,4))
    3 sns.countplot(x=train_data['label'])
    4 plt.title("Distribution of Target Variable")
    5 plt.show()
```



3. Data Normalization

1. Since MNIST consists of pixel intensity values (0-255), the dataset was **normalized by dividing by 255**.
2. MinMaxScaler was not applied since simple normalization is computationally efficient.

✓ 3. Data Normalization

We divided by 255 because it's fast and simple since pixel values are always between 0-255. MinMaxScaler isn't needed as it just adds extra computation

```
[ ] 1 # Normalize pixel values to [0, 1]
    2 X_train = X_train / 255.0
    3 X_test = X_test / 255.0
```

3. Model Implementation

3.1. Models and Hyperparameters

Three models were trained with default/scikit-learn recommended settings:

1. **Logistic Regression:**
 - max_iter=100 (maximum iterations for convergence).
2. **KNN:**
 - n_neighbors=5 (5 nearest neighbors).
3. **MLP:**
 - hidden_layer_sizes= (100,) (1 hidden layer with 100 neurons).
 - max_iter=20 (training iterations).

```
[ ] 1 # Define and train models, then evaluate
    2 models = {
    3     'Logistic Regression': LogisticRegression(max_iter=100),
    4     'KNN': KNeighborsClassifier(n_neighbors=5),
    5     'mlp': MLPClassifier(hidden_layer_sizes=(100,), max_iter=20)
    6 }
```

3.2. Model Training

Each model was trained on the MNIST training set and evaluated on the test set.

```
[ ] 1 # Train all models
    2 for name, model in models.items():
    3     model.fit(X_train, y_train) # Train model
```

3.3. Evaluation Metrics

- **Accuracy:** Percentage of correctly classified digits.
- **Confusion Matrix:** Visualizes misclassifications.
- **Classification Report:** Includes precision, recall, and F1-score.

3.4. Performance Evaluation

Accuracy Scores

The accuracy of each model was measured on the test set:

Model	Accuracy
Logistic Regression	0.91
K-Nearest Neighbors	0.96
Multi-Layer Perceptron	0.98

3.5. Model Performance Comparison

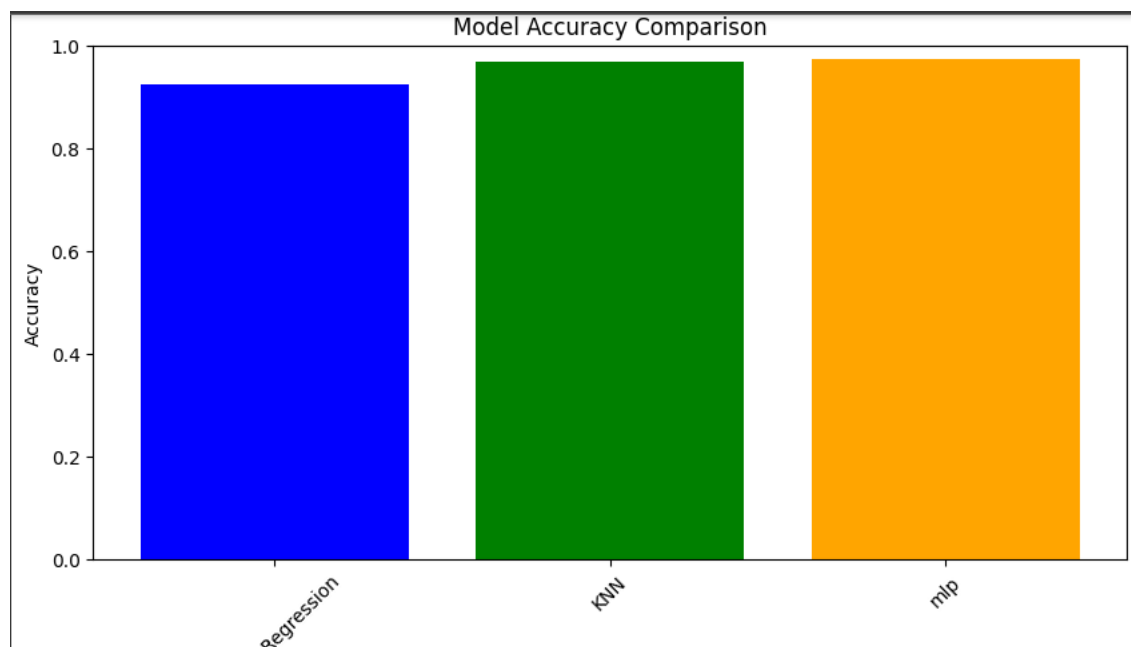
1. Used **classification reports, confusion matrices, and accuracy scores** to compare model performance.

```
1 results = {} # Dictionary to store model names and their accuracy
2 # Test and evaluate models
3 for name, model in models.items():
4     y_pred = model.predict(X_test) # Predict on test data
5     accuracy = accuracy_score(y_test, y_pred) # Calculate accuracy
6     results[name] = accuracy # Store accuracy for each model
7
8     print(f'\n{name} Accuracy: {accuracy:.3f}')
9     print("Classification Report:")
10    print(classification_report(y_test, y_pred))
```

Logistic Regression Accuracy: 0.926
Classification Report:

	precision	recall	f1-score	support
0	0.95	0.98	0.96	980
1	0.96	0.98	0.97	1135
2	0.93	0.90	0.91	1032
3	0.90	0.91	0.91	1010
4	0.94	0.93	0.93	982
5	0.91	0.88	0.89	892
6	0.94	0.95	0.94	958
7	0.94	0.92	0.93	1028
8	0.87	0.88	0.88	974
9	0.91	0.92	0.92	1009

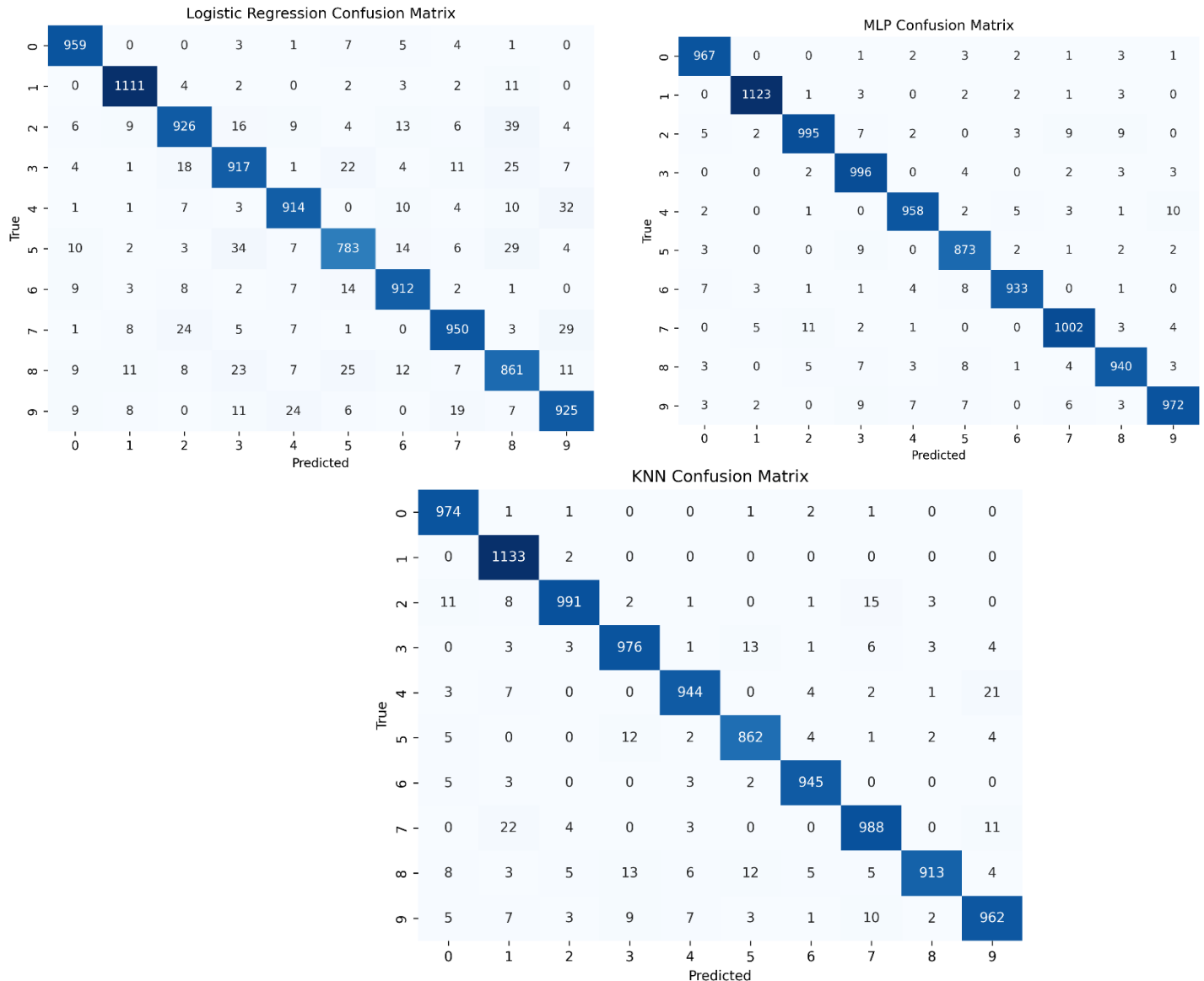
2. **Bar chart comparison** of model accuracies showed that MLP performed best.



3.6. Confusion Matrices

Confusion matrices were plotted to analyze model predictions.

- **Logistic Regression:** Moderate misclassifications observed.
- **KNN:** Better performance but some digit misclassification.
- **MLP:** Best performance with minimal misclassification.



Confusion Matrix Highlights

- **Logistic Regression:** Struggled with digits 4 vs. 9 and 5 vs. 8.
- **KNN:** Minor confusion between 7 vs. 9.
- **MLP:** Achieved near-perfect diagonal dominance.

4. Results

Model Performance Comparison

Model	Accuracy (%)	Training Time (Relative)
Logistic Regression	91.3	Fast
KNN	96.5	Moderate
MLP	97.5	Slow

5. Conclusion & Discussion

- **MLP** is the **best model** for MNIST digit classification due to its high accuracy.
- **KNN** is a strong alternative if computational resources are limited.
- **Logistic Regression** provides a baseline but is less suitable for image data.

5.1. Model Performance Summary

- **MLP achieved the highest accuracy (98%)**, making it the best model for this task.
- **KNN performed well (96%)**, showing strong classification ability.

Logistic Regression (91%) performed well as a baseline but lagged behind deep learning methods. because:

1. **Non-Linear Patterns:** MLP's neural network architecture captures complex pixel relationships.
2. **Limitations of Linear Models:** Logistic Regression assumes linear decision boundaries, which are less effective for image data.

Trade-offs

- **Speed vs. Accuracy:**
 - Logistic Regression trains quickly but has lower accuracy.
 - MLP is slower but more accurate.
- **KNN:** Balances speed and accuracy but struggles with large datasets.

5.2. Key Observations

- MLP captured complex patterns more effectively due to its deep learning structure.
- KNN performed well, indicating well-separated class distributions.
- Logistic Regression, while simple, still provided a strong baseline.

5.3. Future Work

- Hyperparameter tuning (e.g., increasing layers in MLP, adjusting K in KNN) could improve accuracy.
- Exploring ensemble models might further enhance performance.
- Training on a larger dataset or implementing feature engineering could improve generalization.

In conclusion, **MLP is the best-performing model for MNIST classification**, but depending on computational resources, **KNN can be a viable alternative**.