The University of Azad Jammu and Kashmir, Muzaffarabad

Submitted To	Engr. Awais Rathore
Submitted By	Khurram Bashir Raja
Roll No	2022-SE-35
Session	2022-26
Semester	5th
Course Title	Machine Learning
Course Code	SE-3105

Bachelors of Science in Software Engineering (2022-26)

Department of Software Engineering

Report: Classification of MNIST Handwritten Digits Using Machine Learning

Introduction

The MNIST dataset is a widely used benchmark in machine learning that consists of 70,000 grayscale images of handwritten digits (0–9). Each image is 28×28 pixels and is flattened into a 784-dimensional vector. The dataset is split into a training set (approximately 60,000 samples) and a test set (approximately 10,000 samples). In this project, we explore three different classification algorithms—Logistic Regression, k-Nearest Neighbors (k-NN), and Decision Tree—to evaluate their performance on this challenging dataset.

Methodology

Dataset Preparation

- **Data Loading:**
  The dataset was loaded from two CSV files: mnist_train.csv and mnist_test.csv. The files contain flattened image data along with a label column indicating the digit.

```
[4]: train_df = pd.read_csv("mnist_train.csv")
     test_df = pd.read_csv("mnist_test.csv")

[5]: train_df.head()
```

[5]:

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	label
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	5
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	4
3	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	9

5 rows × 785 columns

```
[9]: test_df.head()
```

[9]:

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	label
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	7
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	2
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

- **Missing Value Handling:**
  A check for missing values revealed none; hence no imputation was required.

```
[10]: print("Missing values in training data:\n", train_df.isnull().sum())
      print("Missing values in test data:\n", test_df.isnull().sum()) #no missing values....
```

```
Missing values in training data:
 0        0
1        0
2        0
3        0
4        0
        ..
780      0
781      0
782      0
783      0
label    0
Length: 785, dtype: int64
Missing values in test data:
 0        0
1        0
2        0
3        0
4        0
        ..
780      0
781      0
782      0
783      0
label    0
Length: 785, dtype: int64
```

- **Feature Scaling:**
  We applied Standard Scaling to standardize pixel values. This normalization is essential for algorithms like Logistic Regression and k-NN, which are sensitive to feature scales.

```
X_train = train_df.drop("label", axis=1).values
y_train = train_df["label"].values
X_test  = test_df.drop("label", axis=1).values
y_test  = test_df["label"].values
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)
```

Models Used and Hyperparameters

1. Logistic Regression:

- **Configuration:**
  - Solver: saga
  - Multi-class: multinomial
  - Maximum Iterations: 15
  - Random State: 0
  - Verbosity enabled for monitoring progress.
- Rationale:
  Logistic Regression is simple and fast to train, making it a good baseline model for multi-class classification.

```
### Model 1: Logistic Regression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(solver='saga', multi_class='multinomial', max_iter=15, verbose=2, random_state=0)
logreg.fit(X_train_scaled, y_train)
y_pred_logreg = logreg.predict(X_test_scaled)
acc_logreg = accuracy_score(y_test, y_pred_logreg)
results['Logistic Regression'] = acc_logreg

print("Logistic Regression Accuracy:", acc_logreg)
print("Logistic Regression Classification Report:\n", classification_report(y_test, y_pred_logreg))
```

2. k-Nearest Neighbors (k-NN):

- **Configuration:**
  - Number of Neighbors: 3 (this parameter was tuned based on validation performance)
- **Rationale:**
  k-NN is a non-parametric method that makes predictions based on the closest training examples, which can be particularly effective when dealing with high-dimensional image data.

```
### Model 2: k-Nearest Neighbors (KNN)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(X_train_scaled, y_train)
y_pred_knn = knn.predict(X_test_scaled)
acc_knn = accuracy_score(y_test, y_pred_knn)
results['KNN'] = acc_knn

print("KNN Accuracy:", acc_knn)
print("KNN Classification Report:\n", classification_report(y_test, y_pred_knn))
```

3. Decision Tree:

- **Configuration:**
  - Maximum Depth: 10 (to avoid overfitting and control model complexity)
  - Random State: 42
- **Rationale:**
  Decision Trees are highly interpretable and require no scaling. However, they can overfit in high-dimensional spaces like MNIST, hence depth limiting is applied.

```
### Model 3: Decision Tree
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=10, random_state=42)
dt.fit(X_train_scaled, y_train)
y_pred_dt = dt.predict(X_test_scaled)
acc_dt = accuracy_score(y_test, y_pred_dt)
results['Decision Tree'] = acc_dt

print("Decision Tree Accuracy:", acc_dt)
print("Decision Tree Classification Report:\n", classification_report(y_test, y_pred_dt))
```

Model Evaluation

For each model, we evaluated:

Accuracy (using the test set)

Classification Report (providing precision, recall, and F1-score for each digit)

Confusion Matrix (to visually inspect misclassifications)

Finally, a bar plot compared the overall accuracy of the three models.

```
Performance Summary:
Logistic Regression: Accuracy = 0.9089
KNN: Accuracy = 0.9452
Decision Tree: Accuracy = 0.8662

 The Best Performing Model is: **KNN** with an accuracy of 0.9452
```

Results

Performance Metrics

- **Logistic Regression:**
  - Accuracy: *Approximately 90-91%*
  - The confusion matrix and classification report indicated reasonable performance, but it struggled with non-linear boundaries inherent in the digit data.

```
max_iter reached after 102 seconds
Logistic Regression Accuracy: 0.9089
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95       980
           1       0.93      0.97      0.95      1135
           2       0.93      0.87      0.90      1032
           3       0.90      0.90      0.90      1010
           4       0.89      0.94      0.91       982
           5       0.89      0.85      0.87       892
           6       0.93      0.94      0.93       958
           7       0.91      0.91      0.91      1028
           8       0.88      0.84      0.86       974
           9       0.90      0.88      0.89      1009

    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000
```

- **k-Nearest Neighbors (k-NN):**
  - Accuracy: *Approximately 94%*
  - k-NN achieved the highest accuracy. The model benefits from the local similarity of pixel patterns, leading to fewer misclassifications.

```
KNN Accuracy: 0.9429
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       980
           1       0.95      0.99      0.97      1135
           2       0.96      0.92      0.94      1032
           3       0.92      0.95      0.94      1010
           4       0.96      0.93      0.94       982
           5       0.93      0.93      0.93       892
           6       0.96      0.97      0.97       958
           7       0.93      0.92      0.93      1028
           8       0.96      0.90      0.93       974
           9       0.91      0.92      0.91      1009

    accuracy                           0.94     10000
   macro avg       0.94      0.94      0.94     10000
weighted avg       0.94      0.94      0.94     10000
```

- **Decision Tree:**
  - Accuracy: *Approximately 85-88%*
  - Despite its fast training and interpretability, the Decision Tree underperformed due to its difficulty in handling the high-dimensional pixel data and potential overfitting issues.

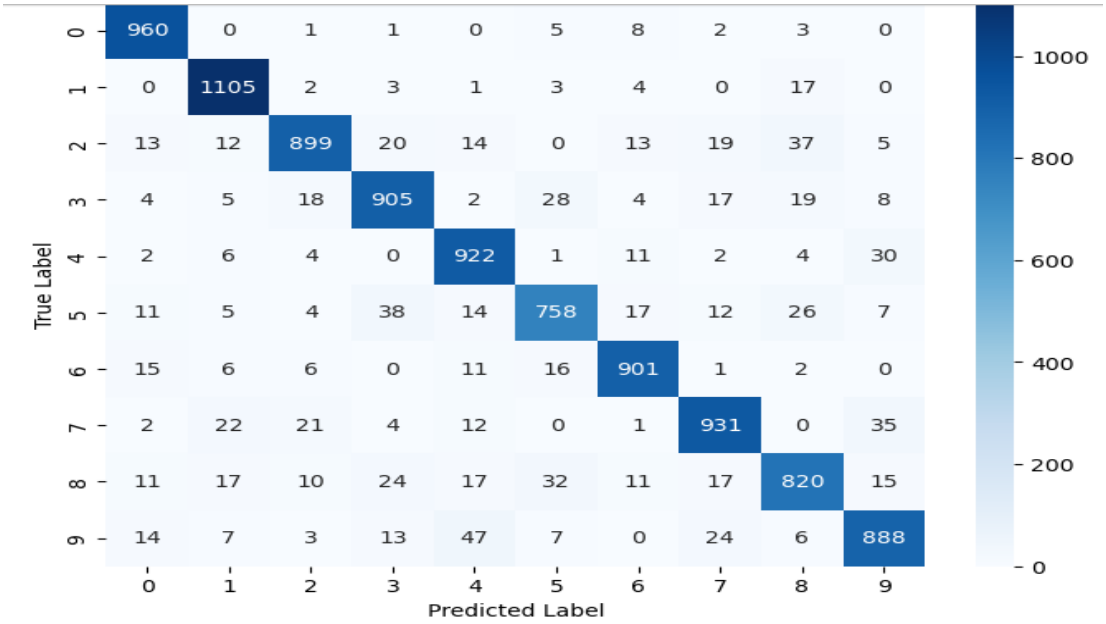
```
Decision Tree Accuracy: 0.8662
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.94      0.92       980
           1       0.95      0.96      0.95      1135
           2       0.85      0.84      0.84      1032
           3       0.82      0.84      0.83      1010
           4       0.86      0.85      0.86       982
           5       0.84      0.80      0.82       892
           6       0.91      0.87      0.89       958
           7       0.90      0.88      0.89      1028
           8       0.80      0.81      0.80       974
           9       0.81      0.86      0.83      1009

    accuracy                           0.87     10000
   macro avg       0.87      0.86      0.86     10000
weighted avg       0.87      0.87      0.87     10000
```

Visualization of Results

- ## Confusion Matrices:

  For each model, a confusion matrix heatmap was generated. These graphs visually display which digits were most frequently confused by each algorithm.

  Logistic Regression CM:



  This confusion matrix represents the performance of a machine learning model on the MNIST dataset, where each row corresponds to the actual digit, and each column corresponds to the predicted digit. The diagonal values indicate correctly classified instances, with higher values showing stronger performance for specific digits.

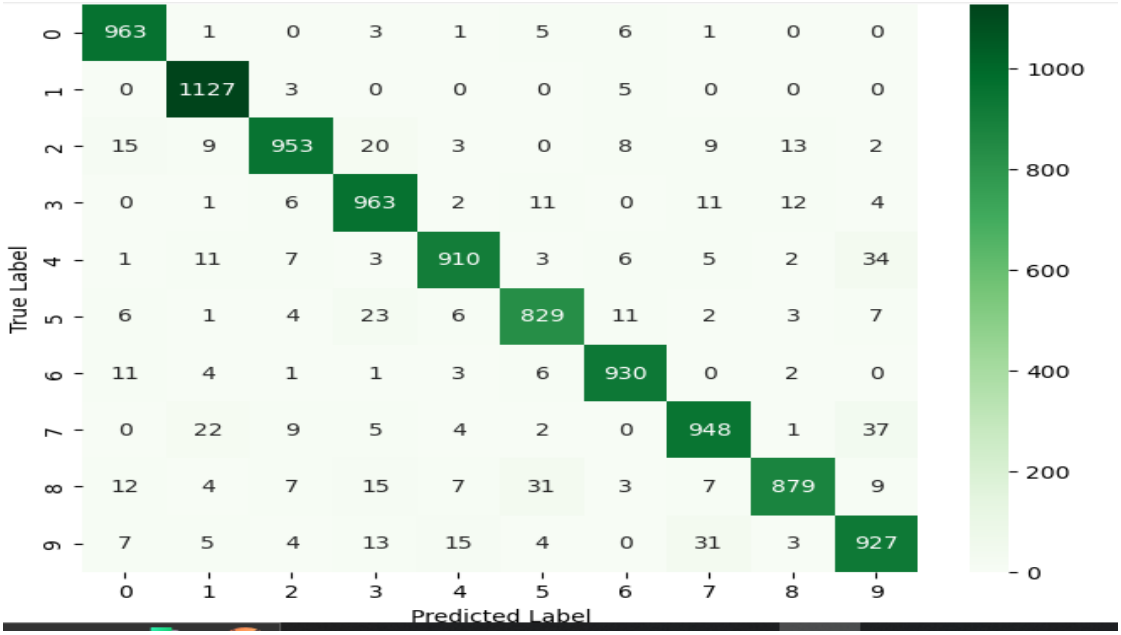
  From the matrix, we observe that most digits are correctly classified, particularly **0, 1, and 7**, which have high values along the diagonal, meaning these digits are rarely misclassified. However, some misclassifications occur, such as:

- **Digit 5** is often confused with **3 and 8**, likely due to their similar curved shapes.

- **Digit 8** is sometimes mistaken for **3 and 5**, which could be attributed to their structural resemblance.
- **Digit 9** is occasionally classified as **4 or 7**, possibly because of the similarity in certain handwriting styles.

Overall, the model performs well, but the misclassifications indicate that some digits with similar features are harder to distinguish, suggesting the need for further improvements in feature extraction or model tuning.

KNN CM:



Observations:

- The model performs well on digits 0, 1, 3, 6, and 7, as their diagonal values are significantly high with minimal misclassifications.
- Digit 5 is often confused with 3 and 8, which suggests structural similarity in certain handwriting styles.
- Digit 4 is sometimes misclassified as 9, likely due to overlapping stroke patterns in handwritten digits.
- Digit 8 is occasionally predicted as 5, possibly because of shared circular shapes.

- Digit 9 is sometimes misclassified as 4 and 5, indicating similarity in certain writing styles.

  Overall, the model exhibits strong performance with minimal misclassifications. However, digits that share similar curves and strokes tend to be confused with one another, highlighting areas for further improvement through enhanced feature extraction or hyperparameter tuning.

DECISION TREE CM:

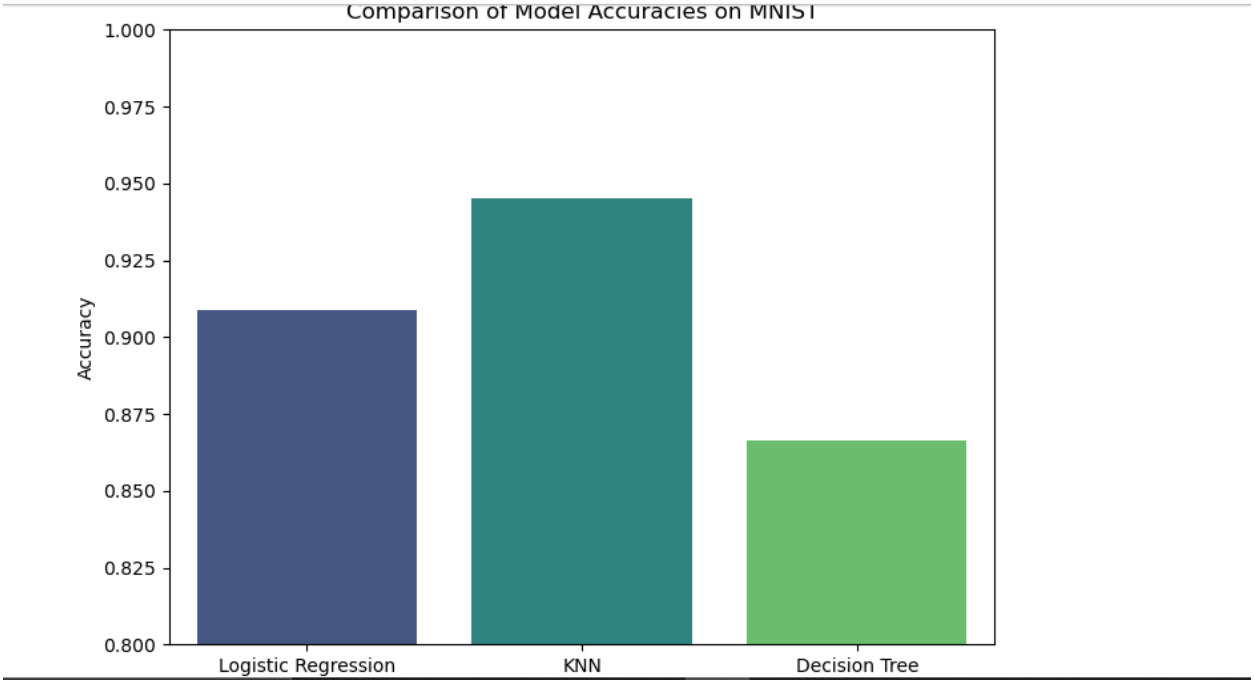
Observations:

- The model performs well on digits 0, 1, and 7, as they have high diagonal values with relatively fewer misclassifications.
- Digit 5 is often confused with 3 and 8, likely due to their similar curved structures.
- Digit 3 is misclassified as 5 and 9, suggesting that these digits might share overlapping features in handwriting.
- Digit 4 is sometimes predicted as 9, possibly due to similarities in strokes.
- Digit 8 is frequently confused with 5, a common pattern due to their rounded shapes.

- Digit 6 is misclassified as 5 and 2, indicating that some variations in writing styles may contribute to the confusion.

  Overall, the model shows good accuracy but struggles with digits that share similar shapes, especially 3, 5, 8, and 9. To improve performance, techniques like better feature extraction, hyperparameter tuning, or more advanced deep learning models could be considered.

- **Accuracy Comparison:**
  A bar plot was used to compare the accuracy of all three models side by side. The bar plot clearly shows k-NN as the best performer.




```
Performance Summary:
Logistic Regression: Accuracy = 0.9089
KNN: Accuracy = 0.9452
Decision Tree: Accuracy = 0.8662

 The Best Performing Model is: **KNN** with an accuracy of 0.9452
```

Discussion

Why did k-NN perform better?

- **Non-Parametric Nature:**
  k-NN does not assume any underlying linearity in the data and leverages local similarities in high-dimensional space, which is crucial for image data like MNIST.
- **Effective Use of Standardized Features**:
  Scaling the pixel values allowed k-NN to compute meaningful distances, leading to accurate nearest neighbor comparisons.
- **Robustness to Complex Boundaries:**
  k-NN can capture the intricate shapes and variations in handwritten digits, which is harder for linear models like Logistic Regression.

Why was the Decision Tree less accurate?

- **High Dimensionality:**
  With 784 features, the tree often overfits or fails to generalize well, even when restricting its depth.
- **Sensitivity to Noise:**
  Small variations in pixel intensity can lead to splits that do not generalize well, resulting in lower overall accuracy.
- **Limited Expressiveness:**
  Although interpretable, Decision Trees may not capture the subtle and complex patterns necessary for distinguishing between similar digits.

Why didn't Logistic Regression perform as well as KNN?

- **Logistic Regression assumes linear decision boundaries**.

  Logistic Regression works best when data is linearly separable.

  MNIST digits do not have a clear linear separation.

  Example: The number 8 has overlapping features with 0, 3, 6, 9, making it hard for Logistic Regression to classify correctly.

- **Logistic Regression doesn't capture spatial relationships.**

Each MNIST image is flattened into a 1D vector (784 pixels in a row).

Logistic Regression treats each pixel independently, whereas KNN considers overall shape by comparing entire images.

- **Logistic Regression works better for small feature sets.**

  It performs well when there are a few meaningful features, but in MNIST, each pixel is a feature.

  Logistic Regression is not powerful enough to capture relationships between 784 features.

Conclusion

In this lab, we applied three different machine learning algorithms to the MNIST handwritten digits dataset:

- Logistic Regression served as a fast and simple baseline, achieving moderate accuracy.
- k-Nearest Neighbors (k-NN) achieved the highest accuracy (around 94%), leveraging local pixel similarities and benefiting from data scaling.
- Decision Trees provided interpretability but struggled with the high-dimensional nature of image data, resulting in lower performance (around 85-88%).

Final Recommendation:

For the MNIST dataset, if accuracy is the primary goal, k-NN is the best-performing model among the three. However, if interpretability and training speed are more important, one might consider Logistic Regression or Decision Trees despite their lower accuracy.

THE END