NAME : Teja Satya Sai Alapati
NJIT UCID: ta473
Email Address: ta473@njit.edu
Date : 22 November 2024
Professor: Yasser Abduallah
CS 634101 Data Mining

Final Project

Exploring the Effectiveness of Random Forest, Logistic Regression, and LSTM on Handwritten Digit Classification

Abstract

The present research assesses the performance of different machine learning and deep learning models using the load_digits dataset for handwritten number classification, a popular benchmark for multiclass classification. There are 1,797 samples available for analysis in this dataset, which is made up of 8x8 greyscale pictures of handwritten numbers 0–9. Three models—Random Forest, Logistic Regression, and Long Short-Term Memory (LSTM) networks—are compared for efficacy in this project. K-Fold cross-validation is used to assess the models in order to prevent overfitting and guarantee reliable performance measures.

As traditional machine learning methods, Random Forest and Logistic Regression are compared to LSTM, a neural network model that can identify sequential dependencies in data. Accuracy, precision, recall, F1 score, and error rate are among the standard performance measures that are calculated and examined fold by fold for every model.

Introduction

Classifying handwritten digits is a basic task in computer vision and machine learning. It entails identifying and classifying handwritten numbers into the appropriate numerical groupings (0–9). Numerous real-world applications, including automated mail sorting, bank cheque processing, form digitisation, and other document recognition tasks, depend on this problem. Handwritten digit classification is a simple and useful benchmark for assessing how well different machine learning and deep learning algorithms perform.

In this context, the **load_digits dataset** from the scikit-learn library is widely used for benchmarking. It consists of 1,797 8x8 grayscale images of digits, making it a compact yet comprehensive dataset for multiclass classification tasks. The dataset provides an opportunity to test both classical machine learning models, such as Random Forests and Logistic Regression, and modern deep learning techniques like Long Short-Term Memory (LSTM) networks, which are capable of modeling sequential dependencies in data.

This project aims to explore and compare the performance of these models on the handwritten digit classification problem using K-Fold cross-validation for robust evaluation. By analyzing the strengths and weaknesses of each model, this study provides insights into their suitability for solving multiclass classification problems and their potential applications in real-world scenarios.

Dataset

In this project, I am using Sklearn's predefined dataset which is load_digit dataset. A popular benchmark dataset for handwritten digit classification is the load_digits dataset, which is made available by the scikit-learn toolkit. It includes 1,797 examples in total of greyscale pictures of handwritten numbers from 0 to 9. An 8x8 matrix is used to represent each image, with each value representing a pixel's intensity, ranging from 0 (black) to 16 (white).

The **load_digits** dataset is a part of the scikit-learn library, making it easy to load and use for machine learning tasks.

```
from sklearn.datasets import load_digits

digits = load_digits( )
```

Features (Images)

The dataset consists of **1,797 samples**, where each sample represents a **grayscale image** of a handwritten digit. The images are **8x8 pixels**, which means each image is a matrix of 8 rows and 8 columns, giving a total of **64 pixel values per image**. These values range from 0 (black) to 16 (white), indicating the intensity of the corresponding pixel.

- **Shape**: (1797, 64) — 1,797 images with 64 pixel values each.
- **Format**: Flattened vector of pixel values for each image (i.e., a 1D array of 64 values).

You can visualize each 8x8 image by reshaping these pixel values back into an 8x8 grid.

Target Labels (Digits)

The **target labels** are the actual digits (0–9) represented by the corresponding images. Each label is an integer in the range 0 to 9.

- **Shape**: (1797,) — A 1D array containing the target digit (label) for each image.
- Each entry corresponds to the correct digit for the image.

Original Image Data (8x8 Arrays)

The **images** attribute contains the **8x8 pixel images** in their original form as 2D arrays.

- **Shape**: (1797, 8, 8) — 1,797 images, each represented as an 8x8 2D array of pixel values.

Each entry in the images array corresponds to one of the handwritten digit images.

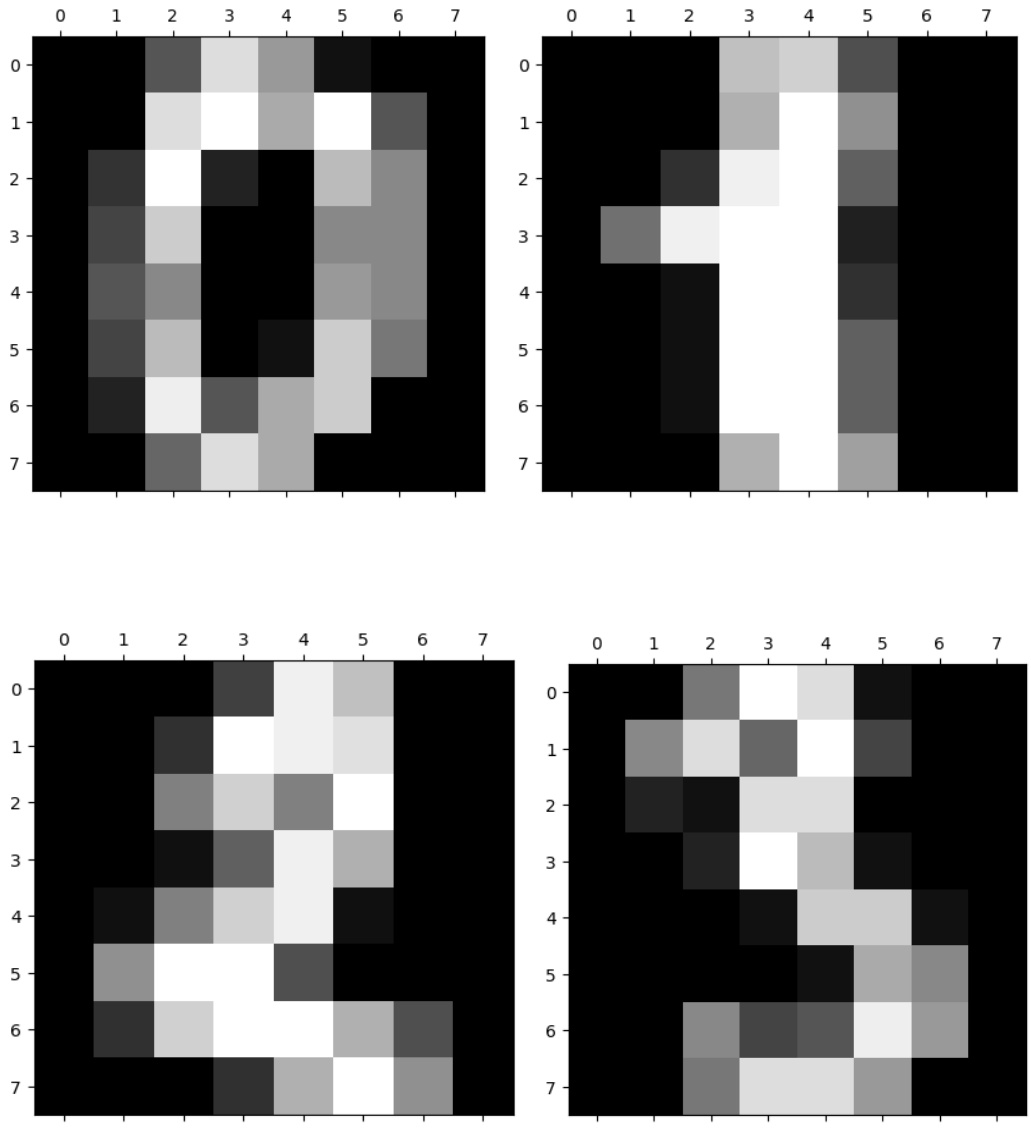
Target Names

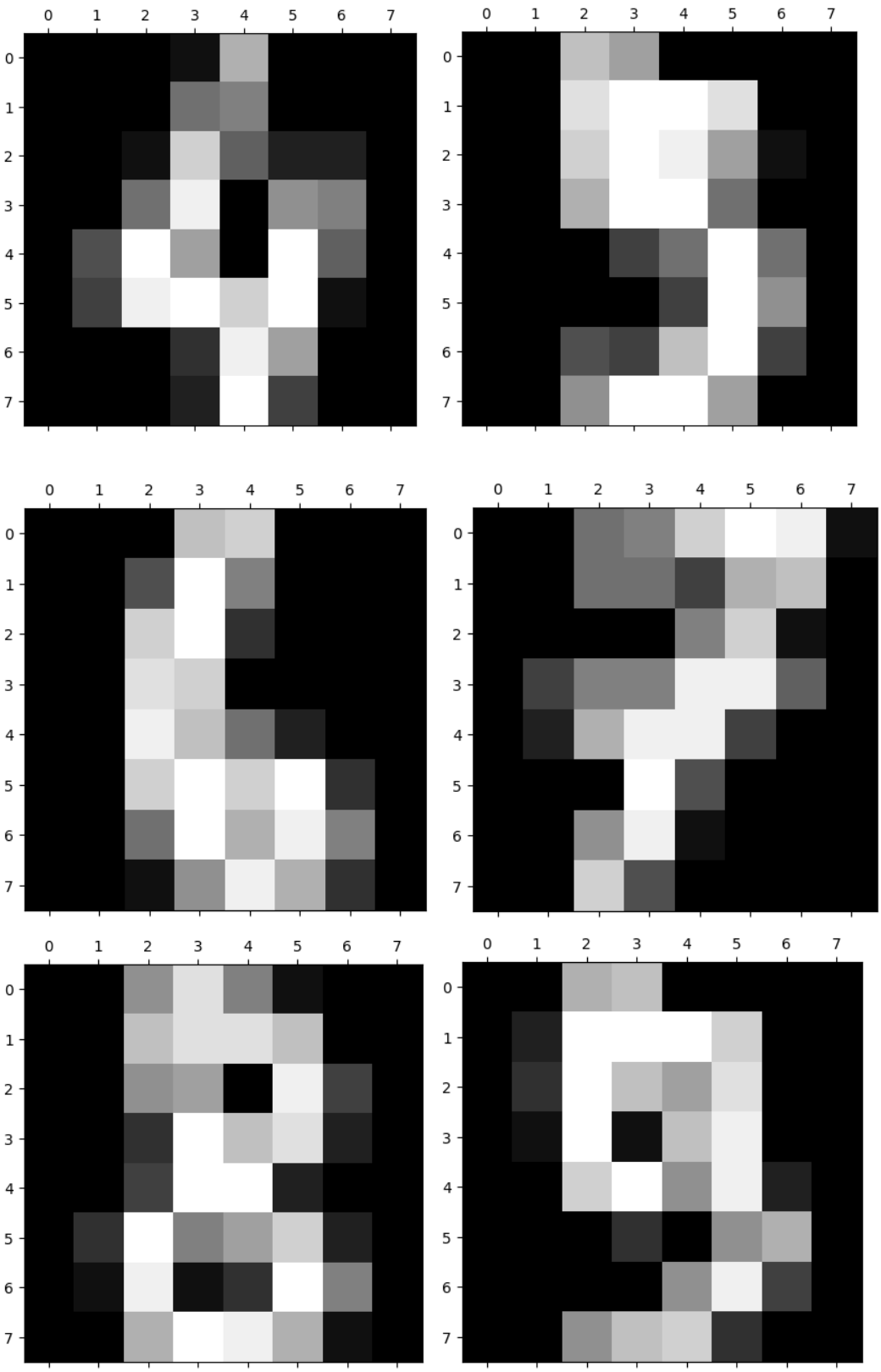
The dataset contains the possible target labels (digits) in the **target_names** attribute.

- **Length**: 10, corresponding to the digits from 0 to 9.

Dataset Description

The **DESCR** attribute provides a textual description of the dataset, including its origin, structure, and other relevant details.





Algorithms

In this project, we explore three different algorithms Random Forest**,** Logistic Regression**,** and Long Short-Term Memory (LSTM) to perform handwritten digit classification using the load_digits dataset**.**

Random Forest

Random Forest is an ensemble learning method based on decision trees. It constructs a set of decision trees during training and outputs the class that is the majority vote of the trees for classification tasks. Random Forest is used to classify the handwritten digits by training on the 64 pixel values for each sample (each 8x8 image flattened into a 64-dimensional feature vector). It is highly effective at handling high-dimensional data like images, as it can capture non-linear patterns and interactions between features.

Logistic Regression

Logistic Regression is a statistical model used for binary and multiclass classification tasks. Despite its name, it is a linear classifier that models the probability of a class label based on a linear combination of input features. In this multiclass classification problem, one-vs-rest (OvR) or one-vs-one strategies are used, where separate binary classifiers are trained for each class to determine which digit the image represents. The logistic regression model will be trained using the flattened pixel values (64 features per sample) to classify digits.

Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) designed to address the vanishing gradient problem, allowing it to learn and remember long-term dependencies in sequential data. LSTM is applied here as an experimental deep learning approach for digit classification. Although traditionally used for sequential data (like text or time series), it can still be applied to images, where each pixel row or column can be treated as a sequence of features. The 8x8 image is treated as a sequence of rows (or columns), and LSTM learns to capture dependencies between pixel values within each row or across rows.

K-Fold Cross-Validation

K-Fold Cross-Validation is a statistical method used to evaluate the performance of a machine learning model. It involves splitting the dataset into K equally sized subsets, or "folds". The model is trained on K-1 of these folds and tested on the remaining fold. This process is repeated K times, with each fold serving as the test set once. The final performance metric is usually the average of the results from all the folds.

In this project, K-Fold Cross-Validation is used as the evaluation method to assess the performance of three different models Random Forest**,** Logistic Regression**,** and Long Short-Term Memory (LSTM) on the load_digits dataset. K-Fold cross-validation is a robust and widely used technique in machine learning to ensure that models are evaluated fairly and accurately, minimizing the risk of overfitting or underfitting.

Performance Metrics

True Positive Rate (TPR)

True Positive Rate (TPR), also known as Recall or Sensitivity, measures how well the model correctly identifies positive cases.

TPR= TP/(TP+FN)

Where:

- TP (True Positives): Correctly predicted positive samples.
- FN (False Negatives): Incorrectly predicted negative samples.

True Negative Rate (TNR)

True Negative Rate (TNR), also known as Specificity, measures how well the model correctly identifies negative cases.

TNR= TN/(TN+FP)

Where:

- TN (True Negatives): Correctly predicted negative samples.
- FP (False Positives): Incorrectly predicted positive samples.

False Positive Rate (FPR)
False Positive Rate (FPR) measures how often the model incorrectly classifies a negative instance as positive.

FPR= FP/(TN+FP)

Where:

- FP (False Positives): Incorrectly predicted positive samples.
- TN (True Negatives): Correctly predicted negative samples.

False Negative Rate (FNR)

False Negative Rate (FNR) measures how often the model incorrectly classifies a positive instance as negative.

FNR= FN/(TP+FN)

Where:

- FN (False Negatives): Incorrectly predicted negative samples.
- TP (True Positives): Correctly predicted positive samples.

Precision

Precision measures the proportion of positive predictions that are actually correct.

Precision = TP/(TP+FP)

Where:

- TP (True Positives): Correctly predicted positive samples.
- FP (False Positives): Incorrectly predicted positive samples.

F1 Score

F1 Score is the harmonic mean of precision and recall (TPR), providing a balance between the two. It is especially useful when the data is imbalanced or when both false positives and false negatives are important.

F1 Score=2×Precision+Recall/(Precision×Recall)

Accuracy

Accuracy measures the proportion of correct predictions made by the model across all classes.

Accuracy = TP + TN /(TP+TN+FP+FN)

Where:

- TP (True Positives): Correctly predicted positive samples.
- TN (True Negatives): Correctly predicted negative samples.
- FP (False Positives): Incorrectly predicted positive samples.
- FN (False Negatives): Incorrectly predicted negative samples

Error Rate

Error Rate is the proportion of incorrect predictions made by the model.

Error Rate = FP + FN /(TP+TN+FP+FN)

Where:

- FP (False Positives): Incorrectly predicted positive samples.
- FN (False Negatives): Incorrectly predicted negative samples.

Balanced Accuracy (BACC)

Balanced Accuracy (BACC) is the average of True Positive Rate (TPR) and True Negative Rate (TNR), and it is used to evaluate models on imbalanced datasets.

BACC = TPR+TNR/2

True Skill Statistic (TSS)

True Skill Statistic (TSS**)** measures the improvement over random chance by combining True Positive Rate (TPR) and False Positive Rate (FPR).

TSS = TPR−FPR

Heidke Skill Score (HSS)

Heidke Skill Score (HSS) is a metric used to evaluate the accuracy of classification models. It measures how well the model performs compared to a random classifier, taking into account both correct and incorrect predictions.

HSS = 2×(TP×TN−FP×FN) / (TP+FN)×(FN+TN)+(TP+FP)×(FP+TN)

Code

Below are the screeshots of of the code from python file

[30]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits # importing pre-defined dataset
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import multilabel_confusion_matrix
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from tensorflow.keras.utils import to_categorical
```

[11]:
```
digits= load_digits()
```

[12]:
```
dir(digits)
```

[12]:
```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

[13]:
```
plt.gray()
for i in range(10):
    plt.matshow(digits.images[i])
```

[13]:
```
plt.gray()
for i in range(10):
    plt.matshow(digits.images[i])
```

[14]:
```
X = digits.data
y = digits.target
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[15]:
```
# Convert labels to categorical for LSTM
y_categorical = to_categorical(y, num_classes=len(np.unique(y)))
```

[16]:
```
kf = KFold(n_splits=3, shuffle=True, random_state=42)
```

[17]:
```
def calc_metrics_multiclass(conf_matrix):
    metrics_list = []
    for cm in conf_matrix:  # Confusion matrix per class
        TP, FN = cm[0][0], cm[0][1]
        FP, TN = cm[1][0], cm[1][1]
        TPR = TP / (TP + FN)
        TNR = TN / (TN + FP)
        FPR = FP / (TN + FP)
        FNR = FN / (TP + FN)
        Precision = TP / (TP + FP)
        F1_measure = 2 * TP / (2 * TP + FP + FN)
        Accuracy = (TP + TN) / (TP + FP + FN + TN)
        Error_rate = (FP + FN) / (TP + FP + FN + TN)
        BACC = (TPR + TNR) / 2
```

```
def calc_metrics_multiclass(conf_matrix):
    metrics_list = []
    for cm in conf_matrix:  # Confusion matrix per class
        TP, FN = cm[0][0], cm[0][1]
        FP, TN = cm[1][0], cm[1][1]
        TPR = TP / (TP + FN)
        TNR = TN / (TN + FP)
        FPR = FP / (TN + FP)
        FNR = FN / (TP + FN)
        Precision = TP / (TP + FP)
        F1_measure = 2 * TP / (2 * TP + FP + FN)
        Accuracy = (TP + TN) / (TP + FP + FN + TN)
        Error_rate = (FP + FN) / (TP + FP + FN + TN)
        BACC = (TPR + TNR) / 2
        TSS = TPR - FPR
        HSS = (
            2 * (TP * TN - FP * FN)
            / ((TP + FN) * (FN + TN) + (TP + FP) * (FP + TN))
            if (TP + FN) * (FN + TN) + (TP + FP) * (FP + TN) != 0
            else 0
        )
        metrics_list.append(
            [TP, TN, FP, FN, TPR, TNR, FPR, FNR, Precision, F1_measure, Accuracy, Error_rate, BACC, TSS, HSS]
        )
    return np.mean(metrics_list, axis=0)
```

```
def get_metrics_multiclass(model, X_train, X_test, y_train, y_test, LSTM_flag=False):
    if LSTM_flag:
        # Reshape for LSTM
        X_train = X_train.reshape(X_train.shape[0], 8, 8)
        X_test = X_test.reshape(X_test.shape[0], 8, 8)
        y_train = to_categorical(y_train)
        y_test = to_categorical(y_test)

        # Train the LSTM model
        lstm_model = Sequential()
        lstm_model.add(LSTM(128, input_shape=(8, 8), activation='relu'))
        lstm_model.add(Dense(64, activation='relu'))
        lstm_model.add(Dense(10, activation='softmax'))
        lstm_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        lstm_model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)

        # Predictions
        y_pred = np.argmax(lstm_model.predict(X_test), axis=1)
        y_true = np.argmax(y_test, axis=1)
    else:
        # Train classical model
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        y_true = y_test

    # Compute confusion matrices per class
    conf_matrix = multilabel_confusion_matrix(y_true, y_pred)
    metrics = calc_metrics_multiclass(conf_matrix)
```

```
        X_train = X_train.reshape(X_train.shape[0], 8, 8)
        X_test = X_test.reshape(X_test.shape[0], 8, 8)
        y_train = to_categorical(y_train)
        y_test = to_categorical(y_test)

        # Train the LSTM model
        lstm_model = Sequential()
        lstm_model.add(LSTM(128, input_shape=(8, 8), activation='relu'))
        lstm_model.add(Dense(64, activation='relu'))
        lstm_model.add(Dense(10, activation='softmax'))
        lstm_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        lstm_model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)

        # Predictions
        y_pred = np.argmax(lstm_model.predict(X_test), axis=1)
        y_true = np.argmax(y_test, axis=1)
    else:
        # Train classical model
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        y_true = y_test

    # Compute confusion matrices per class
    conf_matrix = multilabel_confusion_matrix(y_true, y_pred)
    metrics = calc_metrics_multiclass(conf_matrix)
    return metrics
```

[19]:

```
metric_labels = ["TP", "TN", "FP", "FN", "TPR", "TNR", "FPR", "FNR", "Precision", "F1_measure", "Accuracy", "Error_rate", "BACC", "TSS", "HSS"]

# Store results for each iteration
for fold, (train_index, test_index) in enumerate(kf.split(X_scaled), start=1):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Store metrics for this fold
    rf_metrics = get_metrics_multiclass(RandomForestClassifier(random_state=42, n_jobs=-1), X_train, X_test, y_train, y_test)
    lr_metrics = get_metrics_multiclass(LogisticRegression(max_iter=500, random_state=42, n_jobs=-1), X_train, X_test, y_train, y_test)
    lstm_metrics = get_metrics_multiclass(None, X_train, X_test, y_train, y_test, LSTM_flag=True)


    fold_results = pd.DataFrame({
        "Metric": metric_labels,
        "Random Forest": rf_metrics,
        "Logistic Regression": lr_metrics,
        "LSTM": lstm_metrics,
    })


    print(f"Fold {fold} Results")
    display(fold_results)
    print("\n" + "-" * 50 + "\n")
```

Outputs

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Fold 1 Results
```

	Metric	Random Forest	Logistic Regression	LSTM
0	TP	537.600000	537.500000	535.400000
1	TN	58.400000	58.300000	56.200000
2	FP	1.500000	1.600000	3.700000
3	FN	1.500000	1.600000	3.700000
4	TPR	0.997210	0.997027	0.993145
5	TNR	0.975049	0.974147	0.940899
6	FPR	0.024951	0.025853	0.059101
7	FNR	0.002790	0.002973	0.006855
8	Precision	0.997219	0.997018	0.993133
9	F1_measure	0.997211	0.997021	0.993121
10	Accuracy	0.994992	0.994658	0.987646
11	Error_rate	0.005008	0.005342	0.012354
12	BACC	0.986129	0.985587	0.967022
13	TSS	0.972259	0.971174	0.934045
14	HSS	0.972316	0.970749	0.931881

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Fold 2 Results
```

	Metric	Random Forest	Logistic Regression	LSTM
0	TP	537.300000	536.600000	535.200000
1	TN	58.100000	57.400000	56.000000
2	FP	1.800000	2.500000	3.900000
3	FN	1.800000	2.500000	3.900000
4	TPR	0.996670	0.995377	0.992781
5	TNR	0.969636	0.959277	0.932707
6	FPR	0.030364	0.040723	0.067293
7	FNR	0.003330	0.004623	0.007219

5	TNR	0.969636	0.959277	0.932707
6	FPR	0.030364	0.040723	0.067293
7	FNR	0.003330	0.004623	0.007219
8	Precision	0.996667	0.995357	0.992872
9	F1_measure	0.996665	0.995358	0.992787
10	Accuracy	0.993990	0.991653	0.986978
11	Error_rate	0.006010	0.008347	0.013022
12	BACC	0.983153	0.977327	0.962744
13	TSS	0.966306	0.954654	0.925488
14	HSS	0.965901	0.953730	0.924869

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
Fold 3 Results
```

	Metric	Random Forest	Logistic Regression	LSTM
0	TP	537.600000	536.700000	535.900000
1	TN	58.400000	57.500000	56.700000
2	FP	1.500000	2.400000	3.200000
3	FN	1.500000	2.400000	3.200000
4	TPR	0.997211	0.995540	0.994069
5	TNR	0.975051	0.960743	0.946482
6	FPR	0.024949	0.039257	0.053518
7	FNR	0.002789	0.004460	0.005931

6	FPR	0.024949	0.039257	0.053518
7	FNR	0.002789	0.004460	0.005931

8	Precision	0.997219	0.995538	0.994061
9	F1_measure	0.997213	0.995537	0.994054
10	Accuracy	0.994992	0.991987	0.989316
11	Error_rate	0.005008	0.008013	0.010684
12	BACC	0.986131	0.978142	0.970275
13	TSS	0.972262	0.956284	0.940551
14	HSS	0.972298	0.956116	0.940945

Conclusion

In order to classify handwritten digits, we successfully applied and assessed a number of machine learning methods, including Random Forest, Logistic Regression, and LSTM, on the load_digits dataset. Both traditional machine learning and deep learning methods could be applied thanks to the dataset's extensive collection of pixel data that represented greyscale images of numbers. We obtained generalised performance measures and reduced overfitting by using K-Fold Cross-Validation to ensure robust evaluation. To thoroughly evaluate the models' strengths and shortcomings, a range of performance indicators were used, such as True Positive Rate (TPR), True Negative Rate (TNR), Precision, F1-Score, Accuracy, and others.