Intro to ML (Autumn 2024): Mini-project 1

Akshay Toshniwal (241110005)
Devang Agarwal (241110019)
Prakhar Mandloi (241110051)
Roshan Kumar (241110058)
Suyamoon Pathak (241110091)

1 Exploratory Data Analysis

The Emoticons dataset contains 7,080 rows with no missing values. It has a nearly balanced label distribution, with 3,576 instances of label 0 and 3,504 instances of label 1. So, no class balancing was required. For the Deep Features dataset, no pre-processing was required due to the nature of the features embeddings. The Text Sequence dataset mirrors the emoticons dataset with 7,080 rows and no null values. The most frequent characters are '6' (63,000 occurrences), followed by '2' (62,120 occurrences), and '4' (57,797 occurrences), indicating a distinct pattern in the sequences.

2 Task 1

Task 1 is on training binary classification models on each of the 3 datasets and picking the best possible binary classification model for each of the 3 datasets. We experimented with varying the number of training examples, where we use the first 20%, 40%, 60%, 80%, 100% training examples from the provided training set, and reported how the trained model's accuracy varies on the validation set. The goal of this analysis is to identify the best ML algorithm/model that needs a small amount of training data and generalizes well on the unseen (validation) set.

2.1 Dataset 1: Emoticons as Features Dataset

2.1.1 Preprocessing Step

In the preprocessing steps, we first loaded the dataset and tokenized the input emoticons into numeric sequences. These sequences were then padded to ensure they all have the same length.
After tokenizing the input data, the code calculates the size of the vocabulary. The vocabulary size is the total number of unique tokens (characters in this case) found in the dataset. The `+1` ensures that there's an additional token reserved for padding or unseen characters.
The `embedding_dim = 8` defines the size of the embedding space. An embedding is a way of representing each token (in this case, characters or emoticons) as a dense vector of numbers. Instead of representing each character as a unique number, embeddings allow the model to learn a dense, lower-dimensional representation of each token, which can capture more meaningful relationships between tokens.

2.1.2 Data Modeling

In this dataset, we have used three models: Logistic Regression, Multi-Layer Perceptron, and LSTM. Our problem at hand is Binary classification, so our first intuition was to use a Logistic Regression model. We got decent enough results, but given the outstanding performance of neural networks, we decided to work on MLP and LSTM. MLPs, a type of neural network, are flexible and can handle various kinds of data. and LSTM, being a powerful sequence model, is ideal for tasks involving sequential data.

Out of the three models, Logistic Regression performed the best in terms of both accuracy (97.34%) and macro F1-Score (0.973), indicating that it generalizes better on the validation data. Furthermore, it strikes a balance between precision (0.97), and recall(0.97), which is crucial for classification tasks.

2.1.3 Model Architectures and Hyperparameter Tuning

Logistic Regression Model (using Tensorflow, hence the layered structure)

- **Structure:** Embedding Layer, Flatten Layer, and a Dense Layer (1 unit)
- **Tuned Hyperparameters:**
  - Optimizer: `['adam', 'rmsprop', 'sgd']`
  - Learning rate: `[0.001, 0.0001, 0.01]`
  - Batch size: `[16, 32, 64]`
- **Best Hyperparameters:** Optimizer: `adam`, Learning rate: `0.01`, Batch size: `32`
- **Performance:** Accuracy: 97.34%, Precision: 0.97, Recall: 0.97, F1-Score: 0.97,

Multilayer Perceptron (MLP) Model

- **Structure:** Embedding Layer, Flatten Layer, 2 Dense Layers, Dropout Layers after each dense layer (total 2), Output Layer (1 unit, sigmoid activation).
- **Tuned Hyperparameters:**
  - Number of Dense units: `[16, 8, 4, 2]`
  - Dropout rate: `[0.2, 0.4, 0.5]`
  - Optimizer: `['adam', 'rmsprop', 'sgd']`
  - Learning rate: `[0.001, 0.0001, 0.01]`
  - Batch size: `[16, 32, 64]`
  - Activation function: `['relu', 'sigmoid', 'tanh']`
  - Epochs: `[10, 20, 30]`
- **Best Hyperparameters:** Dense units: `4, 2`, Dropout rate: `0.2`, Optimizer: `adam`, Learning rate: `0.01`, Batch size: `32`, Activation: `relu`, Epochs: `30`
- **Performance:** Accuracy: 96.73%, Precision: 0.97, Recall: 0.97, F1-Score: 0.97,

Long Short-Term Memory (LSTM) Model

- **Structure:** Embedding Layer, LSTM Layer, 2 Dense Layers, Dropout Layer after LSTM and each dense layer (total 3), Output Layer (1 unit, sigmoid activation).
- **Tuned Hyperparameters:**
  - Number of LSTM units: `[16, 32, 64, 128]`
  - Number of Dense units: `[16, 8, 4, 2]`
  - Dropout rate: `[0.2, 0.4, 0.5]`
  - Optimizer: `['adam', 'rmsprop', 'sgd']`
  - Learning rate: `[0.001, 0.0001, 0.01]`
  - Batch size: `[16, 32, 64]`
  - Activation function: `['relu', 'tanh', 'sigmoid']`
  - Epochs: `[10, 20, 30]`
- **Best Hyperparameters:** LSTM units: `8`, Dense units: `4, and 2` repectively, Dropout rate: `0.2`, Optimizer: `adam`, Learning rate: `0.01`, Batch size: `32`, Activation: `relu`, Epochs: `30`
- **Performance:** Accuracy: 97.14%, Precision: 0.96, Recall: 0.97, F1-Score: 0.96,

2.1.4 Validation Accuracy Plots Over Partial Training Sets

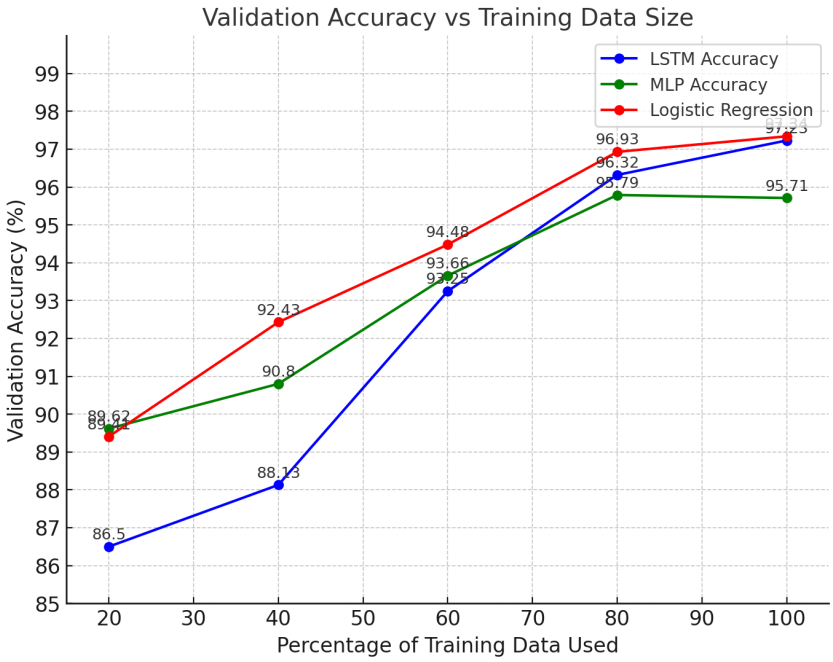

Figure 1: Validation Accuracy vs. Training Set Size for Logistic Regression, MLP, and LSTM Models.

2.2 Dataset 2: Deep Features Dataset

In this dataset we have trained three models using Multi-Layer Perceptron, XGBoost using ensemble of Decision Trees and Logistic Regression algorithm.

2.2.1 Preprocessing Step

In the preprocessing steps, we first reduced the dimensionality of the features using PCA and we found that 194 components were enough for explaining 95% variance in the dataset. So we used 194 components to preserve as much variance as possible from the original data and to eliminate noise and redundant components.
The feature matrices were then flattened to 2D arrays, ensuring the input is suitable for PCA and the other models.

2.2.2 Data Modeling

In this dataset, we have used the following three models: Multi Layer Perceptron (MLP), Logistic Regression, and XGBoost. The feature dataset provided to was genered from a deep learning model, so it was obvious for us to try out the MLP model. Similarly, we tried for Logistic Regression and Decision Trees (DT). We got great results from Logistic Regression, but the DT struggled understanding the relations in the given data. So, we tried using an ensemble of DT- the famous XGBoost, which modeled our data in an amazing way and provided great results.

2.2.3 Model Architectures and Hyperparameter Tuning

Logistic Regression Model (using sk-learn)

- **Tuned Hyperparameters:**
  - **penalty:** `['l1', 'l2', 'elastic']`

  - **c (inverse of reg. strength):** `['0.01', '0.1', '1', '10']`
  - **max_iter:** `['100', '200', '500', '1000']`
- **Best Hyperparameters:** Penalty: `L2`, C: `1`, Max Iterations: `500`
- **Performance:** Accuracy: 98.364%, Precision: 0.9835, Recall: 0.9837, F1-Score: 0.9836

XGBoost Model

- **Tuned Hyperparameters:**
  - **C (Max depth:** `['2', '4', '6', '8']`
  - **n_estimators:** `[`50', `100', `150']`
- **Best Hyperparameters:** Max depth: `6`, n_estimators: `100`
- **Performance:** Accuracy: 97.7505%, Precision: 0.9774, Recall: 0.9776, F1-Score: 0.9775

MLP (Neural Network) Model

- **Tuned Hyperparameters:**
  - **Number of units in Dense layer:** `['16', '32', '64', '128']`
  - **Number of units in the second layer:** `['8', '16', '32', '64']`
  - **Optimizer:** `['Adam', 'RMSprop', 'SGD']`
  - **Batch size:** `['16', '32', '64', '128']`
  - **Dropout rate:** `['0.2', '0.3', '0.5']`
  - **Number of epochs:** `['10', '20', '30', '50']`
  - **Activation function:** `['ReLU', 'Sigmoid', 'Tanh']`
- **Best Hyperparameters:** Number of units in Dense layer: `32`, Number of units in the second dense layer: `16`, Optimizer: `'adam'`, Batch size: `32`, Dropout rate: `0.3`, Number of epochs: `20`, Activation function: `'relu'`
- **Performance:** Accuracy: 98.1595%, Precision: 0.9814, Recall: 0.9817, F1-Score: 0.9816

2.2.4 Validation Accuracy Plots Over Partial Training Sets

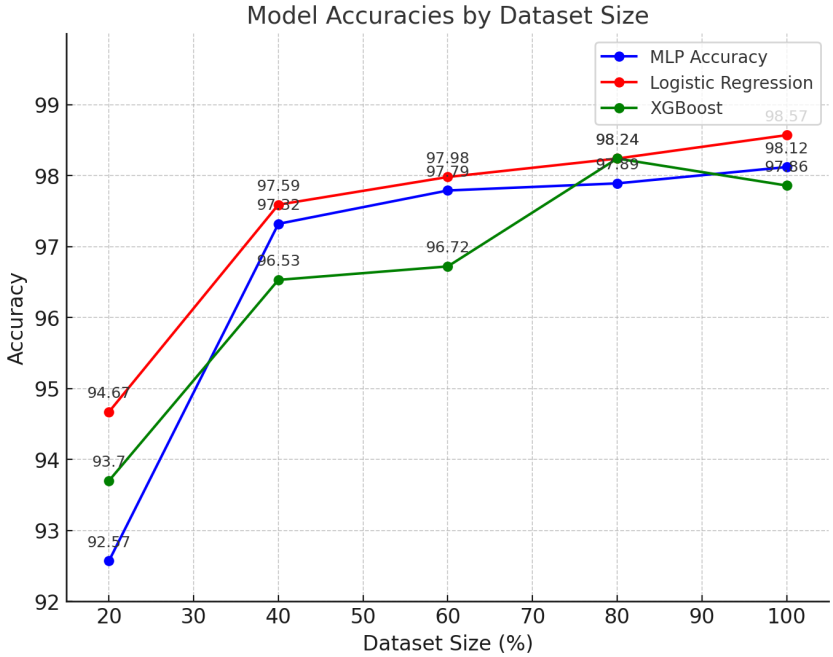

Figure 2: Validation Accuracy vs. Training Set Size for Logistic Regression, XGBoost, and MLP Models.

2.3 Dataset 3: Text Sequence Dataset

2.3.1 Preprocessing Step

For the Text Sequence Dataset, the initial step involved loading the dataset and converting the input strings of digits into numeric sequences. Each string, consisting of 50 digits, was transformed into a list of integers. These sequences were then padded to ensure uniform length, with zero padding applied to sequences shorter than 50 digits. This preprocessing step is essential to maintain consistent input dimensions for the neural network.

The vocabulary size for this dataset is set to 10, representing the digits 0-9. The embedding dimension is defined as 8, allowing for a dense representation of the input sequences.

2.3.2 Data Modeling

Since it was a Text Sequence Dataset, we were sure to go with the Sequential models. Given how better Long Short Term Memory (LSTMs) perform to understand the complex relationships, we first wanted to try LSTMs. LSTMs were not that helpful cause it just gave us an F1 score of 0.55, hence we decided to go with a Convolutional Neural Network (CNN). CNNs performed better than LSTMs, but we were still searching for better ways to model the data. So, we tried making a hybrid model that makes the use of both Gated Recurrent Units (GRU) and CNN to finally give the best model.

2.3.3 Model Architectures and Hyperparameter Tuning

CNN Model

- **Structure:** Embedding Layer, Convolutional Layer (32 filters of size 3), Max Pooling Layer, Dropout Layer, Second Convolutional Layer (16 filters of size 3), Second Max Pooling Layer, Second Dropout Layer, Flatten Layer, Dense Layer (1 unit, sigmoid activation).
- **Tuned Hyperparameters:**
  - **Learning rate:** `[0.1, 0.01, 0.001]`
  - **Batch size:** `[16, 32, 64]`
  - **Number of filters:** `[16, 32, 64]`
  - **Kernel size:** `[2, 3, 4]`
  - **Pool size:** `[2, 3, 4]`
  - **Activation function:** `['relu', 'tanh', 'sigmoid']`
  - **Dropout rate:** `[0.3, 0.5, 0.7]`
  - **Epochs:** `[20, 30, 40]`
- **Best Hyperparameters:** Learning rate: `0.001`, Batch size: `64`, Number of filters: `32`, Kernel size: `3`, Pool size: `2`, Activation function: `'relu'` Dropout rate: `0.5`, Epochs: `30`
- **Performance:** Accuracy: 75.49%, Precision: 0.81, Recall: 0.66, F1-Score: 0.73,

LSTM Model

- **Structure:** Embedding Layer, LSTM Layer, Dropout Layer, Dense Layer (1 unit, sigmoid activation).
- **Tuned Hyperparameters:**
  - **Learning rate:** `[0.1, 0.01, 0.001]`
  - **Batch size:** `[16, 32, 64]`
  - **LSTM units:** `[16, 32, 64]`
  - **Dropout rate:** `[0.3, 0.5, 0.7]`
  - **Activation function:** `['relu', 'tanh', 'sigmoid']`
  - **Epochs:** `[20, 30, 40]`

- **Best Hyperparameters:** Learning rate: `0.001`, Batch size: `64`, LSTM units: `32`, Dropout rate: `0.5`, Activation function: `'relu'`, Epochs: `30`
- **Performance:** Accuracy: 64.55%, Precision: 0.74, Recall: 0.44, F1-Score: 0.55,

CNN + GRU Hybrid Model

- **Structure:** Embedding Layer, Convolutional Layer (45 filters of size 3), Batch Normalization Layer, Max Pooling Layer, GRU Layer, Dropout Layer, 2 Dense Layers (8 units on the first with relu, and 1 unit for output, sigmoid activation).
- **Tuned Hyperparameters:**
  - **Learning rate:** `[0.1, 0.01, 0.001]`
  - **Batch size:** `[16, 32, 64]`
  - **Number of filters:** `[32, 45, 64]`
  - **Kernel size:** `[2, 3, 4]`
  - **Pool size:** `[2, 3, 4]`
  - **Dropout rate:** `[0.3, 0.5, 0.7]`
  - **GRU units:** `[16, 32, 64]`
  - **Epochs:** `[20, 30, 40]`
- **Best Hyperparameters:** Learning rate: `0.001`, Batch size: `64`, Number of filters: `45`, Kernel size: `3`, Pool size: `2`, Dropout rate: `0.5`, GRU units: `32`, Epochs: `30`
- **Performance:** Accuracy: 84.46%, Precision: 0.85, Recall: 0.83, F1-Score: 0.84,

2.3.4 Validation Accuracy Plots Over Partial Training Sets

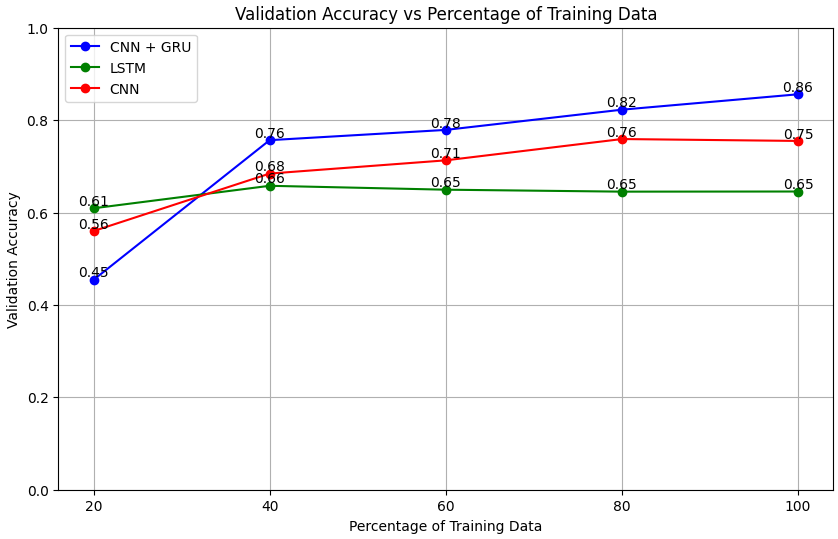

Figure 3: Validation Accuracy vs. Training Set Size for LSTM, CNN and CNN + GRU Hybrid Model.

3 Task 2

3.1 Preprocessing Step

The datasets were individually pre-processed before being combined. For text sequences, each character was converted to an integer, and padding was applied to ensure uniform length. Emoticons were transformed into numbers using their ASCII value modulo 10, followed by padding and embedding into dense vectors. Deep features were pre-extracted and flattened if needed to align

with the other data. Finally, all the pre-processed features were concatenated into a single feature vector per data point, After combining the datasets to address the high dimensionality of the combined dataset, Principal Component Analysis (PCA) is applied, reducing the feature space to 250 components (explains 95% variance).

3.1.1 Data Modeling

In this dataset, we have used the following three models: Logistic Regression, MLP, and Decision Trees. Logistic Regression performed exceedingly well in case of separate datasets, so our first intuition was to go with it for the combined dataset too. Considering the highly complex dataset formed by combining three different datasets with different feature representation, our next go-to-model was the Multi Layer Perceptron (MLP). Similarly, we tried modeling our individual datasets with Decision Trees (DT), but the performance was subpar with respect to the other models. We still wanted to know how DT would perform on the combined dataset, hence we chose DT as our third model.

3.2 Model Architecture and Hyperparameter Tuning

3.2.1 Logistic Regression Model

- **Tuned Hyperparameters:**
  - Penalty: `['l1', 'l2', 'elasticnet', 'none']`
  - C: `[0.01, 0.1, 1, 10, 100]`
  - Solver: `['liblinear', 'saga', 'newton-cg', 'lbfgs']`
  - Max Iterations: `[100, 500, 1000]`
  - Class Weight: `[None, 'balanced']`
  - L1 Ratio: `[None, 0.1, 0.5, 0.7, 0.9]`
- **Best Hyperparameters:** Penalty: `l2`, C: `1`, Solver: `lbfgs`, Max Iterations: `1000`, Class Weight: `None`
- **Performance:** Accuracy: 98.98%, Precision: 0.98, Recall: 0.98, F1-Score: 0.982

3.2.2 Decision Tree Model

- **Tuned Hyperparameters:**
  - Criterion: `['gini', 'entropy', 'log_loss']`
  - Max Depth: `[8, 10, 20, 30, 40, 50]`
  - Min Samples Split: `[2, 5, 10, 20]`
  - Min Samples Leaf: `[1, 2, 5, 10]`
  - Splitter: `['best', 'random']`
  - Min Impurity Decrease: `[0.0, 0.01, 0.05, 0.1]`
  - CCP Alpha: `[0.0, 0.01, 0.05, 0.1]` (for pruning)
- **Best Hyperparameters:** Criterion: `gini`, Max Depth: `8`, Min Samples Split: `10`, Splitter: `best`
- **Performance:** Accuracy: 95.48%, Precision: 0.95, Recall: 0.95, F1-Score: 0.95

3.2.3 Multi-layer Perceptron

- **Structure:** 3 Dense Layers (32, 16, and 16 units each with relu activation), 3 Dropout Layers (one after each Dense layer), Output Layer (1 unit, sigmoid activation).
- **Tuned Hyperparameters:**
  - Batch Size: `[16, 32, 64, 128]`
  - Epochs: `[10, 20, 30, 50]`
  - Optimizer: `['adam', 'rmsprop', 'sgd']`
  - Dropout Rate: `[0.1, 0.2, 0.3]`
  - Dense Layer Units: `[16, 32, 64, 128]`

- Learning rate: `[0.1, 0.01, 0.001]`
- Activation Function: `['relu', 'tanh', 'sigmoid']`
- Regularization: `[None, 'l2', 'l1']`

- **Best Hyperparameters:** Batch Size: `32`, Epochs: `20`, Optimizer: `adam`, Dropout Rate: `0.3`, Dense Layer Units : `32, 16, and 16` respectively, Learning Rate: `0.001`, Activation Function: `relu`
- **Performance:** Accuracy: 96.33%, Precision: 0.96, Recall: 0.96, F1-Score: 0.96

3.3 Validation Accuracy Plots Over Partial Training Sets

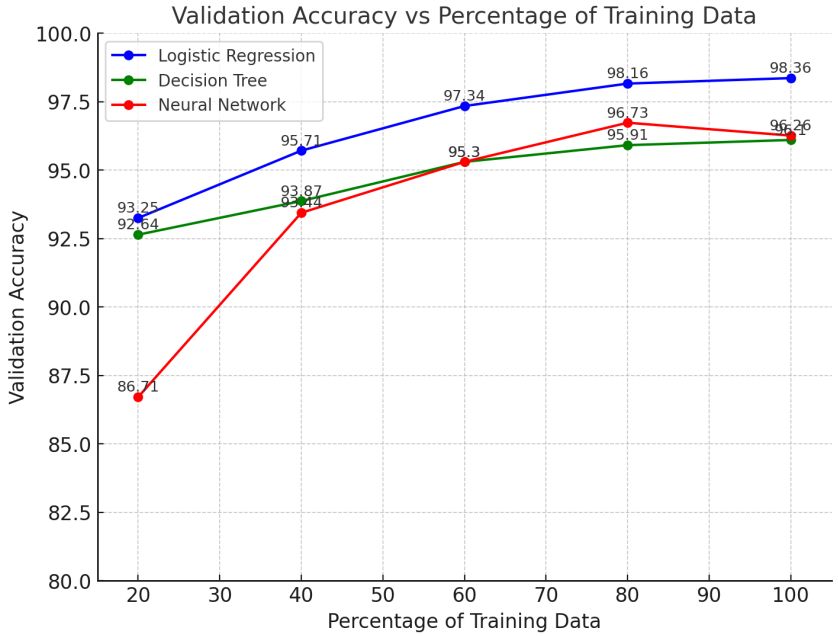

Figure 4: Validation Accuracy vs. Training Set Size for Logistic Regression, Decision Trees, and Deep Neural Network (MLP) Model.

4 Conclusion

Here's a table concluding our work on all the datasets, where the best performing models have been highlighted:

Table 1: Model Performance Metrics

Dataset	Model	Validation Accuracy	Precision	Recall	F1-Score
Emoticons Dataset	**Logistic Regression**	**97.34%**	**0.97**	**0.97**	**0.97**
Emoticons Dataset	MLP	96.73%	0.97	0.97	0.97
Emoticons Dataset	LSTM	97.14%	0.96	0.97	0.96
Deep Features Dataset	**Logistic Regression**	**98.36%**	**0.98**	**0.98**	**0.98**
Deep Features Dataset	XGBoost	97.75%	0.98	0.98	0.98
Deep Features Dataset	MLP	98.12%	0.98	0.98	0.98
Text Sequence Dataset	CNN	75.49%	0.81	0.66	0.73
Text Sequence Dataset	LSTM	64.55%	0.74	0.44	0.55
Text Sequence Dataset	**CNN + GRU Hybrid model**	**84.46%**	**0.85**	**0.83**	**0.84**
Combined Dataset	**Logistic Regression**	**98.98%**	**0.98**	**0.98**	**0.98**
Combined Dataset	Decision Tree	98.48%	0.95	0.95	0.95
Combined Dataset	MLP	96.33%	0.96	0.96	0.96

References

[1] Scikit-learn: Machine learning in Python. (n.d.). Retrieved from https://scikit-learn.org/stable/. Accessed: 2024-10-22.

[2] TensorFlow: API Documentation — TensorFlow v2.16.1. (n.d.). Retrieved from https://www.tensorflow.org/api_docs. Accessed: 2024-10-22.