Name: Anshuman Kumar Yadav

Roll No.: 241030018

Kaggle User ID: anshumankumaryadav

Libraries and Functions:

These libraries and functions are key for building, training, and evaluating a machine learning model,particularly for classification tasks.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
```

User Defined Function:

The function reads data from a CSV-like file, where each line consists of values separated by commas.It splits each line into individual values and appends them to a list.Finally, it converts this list into a pandas DataFrame and returns it.

```
def read_variable_length_data(file_path):
    data = []
    with open(file_path, 'r') as file:
        for line in file:
            values = line.strip().split(',')
            data.append(values)
    return pd.DataFrame(data)
```

Reading Training and Test Data:

This code reads two files:

1. train.txt: Contains the training data.
2. test.txt: Contains the test data.

After reading the data it checks if the training or test data is empty (meaning no data was read),if either is empty, an error message is printed, and the program stops using exit(1). This ensures that the program will not continue without valid training and test data.

```
# Read the training data
train_data = read_variable_length_data('train.txt')
if train_data.empty:
    print("Unable to proceed without train data. Please check the file and try again.")
    exit(1)

# Read the test data
test_data = read_variable_length_data('test.txt')
if test_data.empty:
    print("Unable to proceed without test data. Please check the file and try again.")
    exit(1)
```

Separate Features, Labels for Training Data and prepare Test Data

This segment extracts the features and labels from the training data and prepares the features for the test data.

```
# Separate features and labels for training data
X_train = train_data.iloc[:, 2:].astype(float).values
y_train = train_data.iloc[:, 1].values
X_test = test_data.iloc[:, 1:].astype(float).values
```

User Defined Function:

This function extracts useful statistical features from time series data (like trends or patterns in sequences) and returns an array of the extracted features for each time series.

```
# Feature Extraction Function
def extract_features(time_series):
    features = []
    for ts in time_series:
        ts = ts[~np.isnan(ts)]  # Remove NaN values
        if len(ts) == 0:
            features.append([0] * 10)  # Add default values for empty time series
        else:
            features.append([
                np.mean(ts),
                np.std(ts),
                np.max(ts),
                np.min(ts),
                np.median(ts),
                np.percentile(ts, 25),
                np.percentile(ts, 75),
                np.sum(np.abs(np.diff(ts))),  # Total variation
                np.var(ts),  # Variance
                np.ptp(ts)  # Peak-to-peak range
            ])
    return np.array(features)
```

Extraction and Scaling from Training and Test Data:

Extraction will convert the raw time series data into a structured set of features that can be fed into a machine learning model.And Scaling scales the feature values to have zero mean and unit variance.

```
# Extract Features
X_train_features = extract_features(X_train)
X_test_features = extract_features(X_test)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_features)
X_test_scaled = scaler.transform(X_test_features)
```

Split the Training Data for Validation:

This splits the training data into a new training set and a validation set.

```
# Split training data for validation
X_train_split, X_val, y_train_split, y_val = train_test_split(X_train_scaled, y_train, test_size=0.2, random_state=42)
```

Train the Model (Artificial Neural Network - MLPClassifier):

This part creates and trains a Multi-Layer Perceptron (MLP) classifier.MLPClassifier is a neural network classifier from sklearn, specifically a multi-layer perceptron (ANN).

```
# ANN (MLPClassifier)
model = MLPClassifier(
    hidden_layer_sizes=(128, 64, 32),
    activation='tanh',
    solver='lbfgs',  # Works better for small datasets
    alpha=0.0001,  # Regularization
    max_iter=500,  # Limit iterations to avoid overfitting
    early_stopping=True,
    random_state=42
)

# Train the model
```

```
model.fit(X_train_split, y_train_split)
```

Validation and Prediction:

This part evaluates the model's performance on the validation data and makes predictions on the test data.

```
# Validate the model
y_val_pred = model.predict(X_val)
val_f1_score = f1_score(y_val, y_val_pred, average='weighted')
print(f"Optimized Validation F1-score: {val_f1_score}")

# Predict on test data
y_test_pred = model.predict(X_test_scaled)
```

Creation and Saving of Submission File

This creates a submission DataFrame and saves the submission DataFrame to a CSV file (submission.csv).And then this output CSV file is submitted on Kaggle to check scores.

```
# Create submission file
submission = pd.DataFrame({
    'index': test_data.iloc[:, 0],
    'label': y_test_pred
})

# Save submission file
submission.to_csv('submission.csv', index=False)
print("Optimized submission file 'submission.csv' has been created.")
```