Machine Learning Project Report

Team Members

Student Name	Student ID
Nourhan Mohamed Zaki	20216113
Malak Wael Okasha	20216101
Salma Hazem Salah	20216047
Jana Mohamed Adel	20216031
Nada Yasser Abdelsattar	20216110

<u>Dataset</u>

- **Dataset Name**: Handwritten Alphabets
- **Description**: The dataset contains greyscale images of 26 (A-Z) handwritten alphabets of size 28x28 pixels.
- **Goal**: To classify the alphabets using various machine learning techniques.
- **Dataset link**: [A-Z Handwritten Alphabets in .csv format]

Project Sections

- ❖ **Load the dataset**

```python
# Load the dataset
def load_dataset(path):  1 usage
    """
    Load and concatenate dataset in manageable chunks.
    This function processes a CSV file in chunks, concatenates them,
    and returns the full dataset.
    """
    try:
        notify_loading_process()  # inform the user that the data started the loading process.
        data_chunks = process_chunks(path)  # Process the dataset in chunks.
        complete_dataset = combine_chunks(data_chunks)  # Combine all chunks.
        display_success_message(complete_dataset)  # Display success information.
        return complete_dataset
    except Exception as error:
        handle_loading_error(error)  # Handle any errors that occur.
        return None
```

- ❖ **Notify that data loading is starting**

```python
def notify_loading_process():  1 usage
    """To inform the user that the data started the loading process."""
    print("Loading dataset in chunks (smaller parts) ...")
```

❖ Processing the data into a number of 75 chunks

- *(more efficient because the data set is ways too large)*

```
def process_chunks(file_path):  1 usage
    """
    Divide the dataset into smaller, manageable parts and return them as a list of these parts.
    """
    # Initialize an empty list called chunck_List to store each chunk after being processed.
    chunk_list = []

    try:
        # Create a chunk iterator variable which reads the dataset in chunks.
        chunk_iterator = pd.read_csv(file_path, header=None, engine='python', chunksize=5000)

        # for each chunk do the following
        for index, chunk in enumerate(chunk_iterator):
            # prompt that the chunk is being processed.
            print(f"Processing chunk {index + 1}")
            # add the current chunk to the chunk list.
            chunk_list.append(chunk)
    except Exception as e:
        # if any abnormal thing occured ofr problem faced, print an error message to let the user know.
        print(f"Oops, unfortunately there is an error while processing chunks: {e}")
        # raise the problem to be handled later.
        raise

    # The complete list of chunks are returned after being processed chunk by chunk.
    return chunk_list
```

❖ Combine the chunks into a single DataFrame

```
def combine_chunks(chunks):  1 usage
    """Combine a list of data chunks into a single DataFrame."""
    try:
        # Use pandas' concat method to combine the list of DataFrame chunks into one DataFrame.
        # The ignore_index=True parameter ensures that the resulting DataFrame has a continuous index.
        combined_dataset = pd.concat(chunks, ignore_index=True)
        # Return the combined DataFrame after successful concatenation.
        return combined_dataset
    except Exception as e:
        # Print an error message if an exception occurs during the concatenation process.
        print(f"Error while combining chunks: {e}")
        # Re-raise the exception to inform the caller of the issue.
        raise
```

- **Prompt a message to the user that the data is successfully loaded chunk per chunk**

```
def display_success_message(dataset):  1 usage
    """Display success message and dataset shape."""
    # Print a success message indicating that the dataset was loaded successfully.
    print("Dataset Loaded Successfully")
    # Print the shape of the dataset (rows and columns) for user information.
    print("Dataset Shape:", dataset.shape)
```

- **Telling the user that there is a problem while data loading**

```
def handle_loading_error(error):  1 usage
    """Handle errors during the dataset loading process."""
    # Print an error message describing what went wrong during the loading process.
    print(f"Error loading dataset: {error}")
```

❖ **Explore the dataset and Identify the number of unique classes and show their distribution**

```python
# Explore the dataset
def explore_dataset(dataset):  1 usage
    """Explore the dataset to identify unique classes and their distribution."""
    if dataset is None:
        print("Dataset not loaded. Cannot explore.")
        return

    labels = dataset.iloc[:, 0]
    print("Unique Values in Label Column:", labels.unique())

    # Validate if labels are correct
    unique_classes = labels.unique()
    class_distribution = labels.value_counts()
    print("Number of Unique Classes:", len(unique_classes))
    print("Class Distribution:")
    print(class_distribution)

    # Plot class distribution
    plt.figure(figsize=(10, 6))
    plt.bar(class_distribution.index, class_distribution.values, color='skyblue')
    plt.xlabel("Class (Alphabet)")
    plt.ylabel("Frequency")
    plt.title("Class Distribution")
    plt.xticks(unique_classes, [chr(int(c) + 65) for c in unique_classes])
    plt.show()
```

❖ **Normalize the pixel values of images to range 0,1**

```python
# Normalize the images
def normalize_images(dataset):  1 usage
    """Normalize the pixel values of images to range [0, 1]."""
    if dataset is None:
        print("Dataset not loaded. Cannot normalize.")
        return None, None

    features = dataset.iloc[:, 1:]  # All columns except the first (labels)
    scaler = MinMaxScaler()
    normalized_features = scaler.fit_transform(features)
    print("Images Normalized")
    return pd.DataFrame(normalized_features), dataset.iloc[:, 0]
```

❖ **Display sample images**

```
# Display sample images
def display_sample_images(normalized_features, labels):  1 usage
    """Reshape and display sample images."""
    if normalized_features is None or labels is None:
        print("No data available to display images.")
        return

    num_samples = 10  # Number of samples to display
    sample_indices = np.random.choice(normalized_features.index, num_samples, replace=False)
    plt.figure(figsize=(12, 6))
    for idx, sample_idx in enumerate(sample_indices):
        image_array = normalized_features.iloc[sample_idx].to_numpy().reshape(28, 28)
        plt.subplot( *args: 2, 5, idx + 1)
        plt.imshow(image_array, cmap='gray')
        plt.title(f"Label: {chr(int(labels.iloc[sample_idx]) + 65)}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

Output:

● **Data exploration and preparation:**

✓ **Identify the number of unique classes and show their distribution.**
✓ **Normalize each image.**
✓ **Reshape the flattened vectors to reconstruct and display the corresponding**
✓ **images while testing the models.**

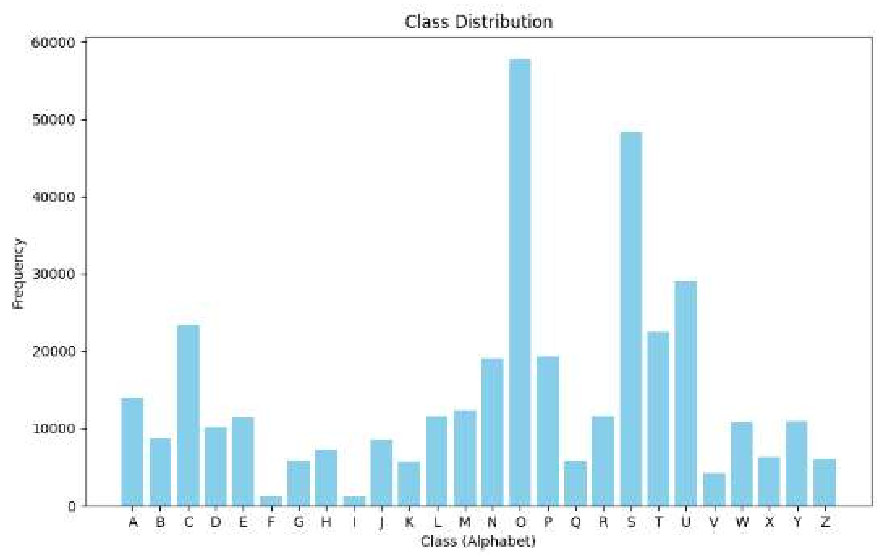
```
F:\ProjectMachineLearning\.venv\Scripts\python.exe F:\ProjectMachineLearning\main2.py
Loading dataset in chunks (smaller parts) ...
Processing chunk 1
Processing chunk 2
Processing chunk 3
Processing chunk 4
Processing chunk 5
Processing chunk 6
Processing chunk 7
Processing chunk 8
Processing chunk 9
Processing chunk 10
Processing chunk 11
Processing chunk 12
Processing chunk 13
Processing chunk 14
Processing chunk 15
Processing chunk 16
Processing chunk 17
Processing chunk 18
Processing chunk 19
Processing chunk 20
Processing chunk 21
Processing chunk 22
Processing chunk 23
Processing chunk 24
Processing chunk 25
Processing chunk 26
Processing chunk 27
```

```
Processing chunk 47
Processing chunk 48
Processing chunk 49
Processing chunk 50
Processing chunk 51
Processing chunk 52
Processing chunk 53
Processing chunk 54
Processing chunk 55
Processing chunk 56
Processing chunk 57
Processing chunk 58
Processing chunk 59
Processing chunk 60
Processing chunk 61
Processing chunk 62
Processing chunk 63
Processing chunk 64
Processing chunk 65
Processing chunk 66
Processing chunk 67
Processing chunk 68
Processing chunk 69
Processing chunk 70
Processing chunk 71
Processing chunk 72
Processing chunk 73
Processing chunk 74
Processing chunk 75
```

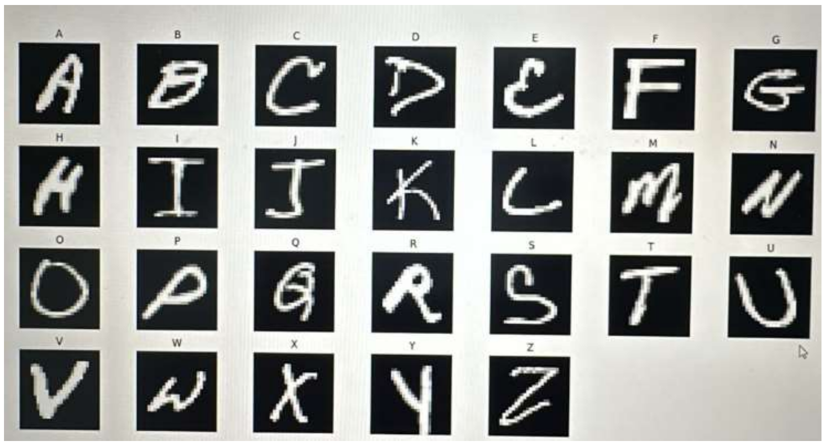
```
Dataset Loaded Successfully
```

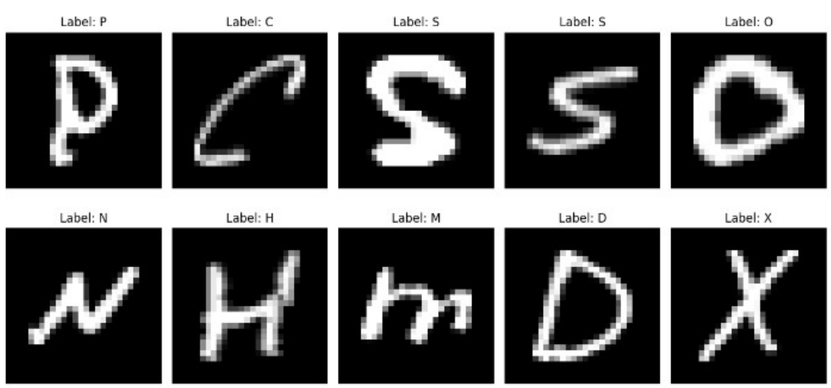
```
Class Distribution:
0
14    57825
18    48419
20    29008
2     23409
19    22495
15    19341
13    19010
0     13870
12    12336
11    11586
17    11566
4     11440
24    10859
22    10784
3     10134
1      8668
9      8493
7      7218
23     6272
25     6076
16     5812
6      5762
10     5603
21     4182
5      1163
8      1120
```

```
Unique Values in Label Column: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25]
Number of Unique Classes: 26
```




```
Images Normalized
```


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z


Label: P
Label: C
Label: S
Label: S
Label: O
Label: N
Label: H
Label: M
Label: D
Label: X

❖ **Select a random subset of 10,000 to be used in training and testing**

```python
def select_random_subset(data_features, data_labels, subset_count=10000):
    """Select a random subset of data."""
    random_indices = np.random.choice(data_features.index, subset_count, replace=False)
    return data_features.iloc[random_indices], data_labels.iloc[random_indices]
```

❖ **Split the data into training and testing datasets**

```python
def partition_data(data_features, data_labels, test_portion=0.4, validation_portion=0.5):
    """Split data into training, validation, and testing sets."""
    features_train, features_temp, labels_train, labels_temp = train_test_split(data_features, data_labels,
                                                                                test_size=test_portion, random_state=42)
    features_val, features_test, labels_val, labels_test = train_test_split(features_temp, labels_temp,
                                                                            test_size=validation_portion,
                                                                            random_state=42)
    return features_train, features_val, features_test, labels_train, labels_val, labels_test
```

❖ **Train and evaluate an SVM model with specified kernel**

```python
def execute_svm_training(features_train, labels_train, features_test, labels_test, kernel_type, class_names):
    """Train and evaluate an SVM model with a specified kernel."""
    print(f"Training SVM with {kernel_type} kernel... Kindly, wait a little.")
    svm_model = SVC(kernel=kernel_type, random_state=42)
    svm_model.fit(features_train, labels_train)

    print(f"Evaluating SVM with {kernel_type} kernel...")
    predicted_labels = svm_model.predict(features_test)
    confusion_mat = confusion_matrix(labels_test, predicted_labels)
    f1_result = f1_score(labels_test, predicted_labels, average='weighted')

    print(f"Confusion Matrix ({kernel_type.capitalize()} Kernel):\n", confusion_mat)
    print(f"F1-Score ({kernel_type.capitalize()} Kernel):", f1_result)

    display_confusion_matrix(confusion_mat, class_names, f"Confusion Matrix - {kernel_type.capitalize()} Kernel")
```

- **Display the confusion matrix**

```
def display_confusion_matrix(confusion_mat, class_names, graph_title):  1 usage
    """Plot confusion matrix as a heatmap."""
    sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='coolwarm',
                xticklabels=class_names, yticklabels=class_names)
    plt.title(graph_title)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

- **Conduct SVM**

```
def svm_experiment_controller(preprocessed_features, target_labels):  1 usage
    """Main function to conduct the SVM experiment."""
    try:
        print("Selecting a subset of samples...")
        subset_features, subset_labels = select_random_subset(preprocessed_features, target_labels)

        print("Splitting the subset into training, validation, and testing sets...")
        features_train, features_val, features_test, labels_train, labels_val, labels_test = partition_data(
            subset_features, subset_labels)

        class_names = [chr(i + 65) for i in range(26)]  # Class labels A-Z

        # Linear Kernel
        execute_svm_training(features_train, labels_train, features_test, labels_test, kernel_type='linear',
                             class_names=class_names)

        # Nonlinear Kernel (RBF)
        execute_svm_training(features_train, labels_train, features_test, labels_test, kernel_type='rbf',
                             class_names=class_names)

    except Exception as error:
        print(f"An issue occurred while running the SVM experiment: {error}")
```

Experiment-1

- ✓ Split the data into training and testing datasets
- ✓ Train 2 SVM models with linear and nonlinear kernels.
- ✓ Test the models and provide the confusion matrix and the average f-1 scores for the testing dataset.

```
Selecting a subset of samples...
Splitting the subset into training, validation, and testing sets...
```

```
Confusion Matrix (Linear Kernel):
 [[ 66   1   0   0   0   0   0   4   0   0   1   0   1   1   0   1   0   0
    1   0   1   0   0   0   1   0]
 [  0  26   1   2   1   0   0   0   0   0   0   0   0   0   1   1   0   0
    1   0   1   0   0   0   0   0]
 [  0   1 110   0   2   0   0   0   0   0   1   1   0   0   4   0   0   3
    5   0   3   0   0   0   0   0]
 [  0   0   2  40   1   0   0   0   0   3   0   0   0   0   5   1   0   0
    0   1   1   0   0   0   1   0]
 [  0   6   5   0  44   2   0   0   0   0   0   1   0   0   1   0   0   1
    2   0   1   0   0   0   0   0]
 [  0   0   0   0   0   6   0   0   0   0   0   0   0   0   0   2   0   0
    0   0   0   0   0   0   0   0]
 [  1   0   1   0   0   0  27   0   0   0   0   0   0   0   0   0   4   0
    2   0   2   0   1   0   0   0]
 [  2   0   0   0   1   0   0  37   0   0   0   0   1   5   0   0   0   0
    0   0   2   0   0   0   1   0]
 [  0   1   0   0   0   0   0   0   9   1   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0]
 [  0   0   0   1   0   0   0   0   0  35   0   0   0   0   0   0   0   0
    2   0   0   0   0   1   1   0]
 [  0   0   0   0   1   0   0   0   0   0  13   1   1   2   0   0   0   3
    0   0   0   0   0   0   0   0]
 [  0   0   3   0   0   0   0   0   0   0   0  64   0   0   0   0   0   0
    0   0   0   0   0   1   0   1]
 [  1   0   0   0   0   0   0   2   0   0   0   0  58   2   0   0   0   0
    0   0   1   0   0   0   1   0]
 [  1   1   0   1   0   0   0   2   0   0   0   0   4  99   0   0   0   0
    1   0   1   0   2   1   0   0]
```

Linear Kernel

```
    2   0   0   0   0   1   1   0]
 [  0   0   0   0   1   0   0   0   0   0  13   1   1   2   0   0   0   3
    0   0   0   0   0   0   0   0]
 [  0   0   3   0   0   0   0   0   0   0   0  64   0   0   0   0   0   0
    0   0   0   0   0   1   0   1]
 [  1   0   0   0   0   0   0   2   0   0   0   0  58   2   0   0   0   0
    0   0   1   0   0   0   1   0]
 [  1   1   0   1   0   0   0   2   0   0   0   0   4  99   0   0   0   0
    1   0   1   0   2   1   0   0]
 [  0   1  11   5   2   0   2   0   0   0   0   0   0   0 279   0   1   0
    1   0   1   0   0   0   0   0]
 [  2   0   0   1   0   2   0   0   0   0   0   0   0   0   0 100   0   0
    0   2   0   0   0   0   3   0]
 [  0   0   1   1   0   0   1   0   0   1   0   0   0   0   3   1  25   0
    0   0   0   0   0   0   0   0]
 [  8   0   0   0   5   0   0   0   0   0   2   0   0   1   0   1   0  45
    0   1   0   0   0   0   0   1]
 [  0   6   2   2   0   0   2   0   0  11   0   2   0   1   0   0   0   0
  233   0   0   0   0   0   0   0]
 [  0   0   0   0   0   2   0   0   0   2   0   0   1   0   0   0   0   0
    0 114   0   0   0   0   0   0]
 [  1   1   1   3   0   0   1   3   0   0   0   0   0   1   0   0   0   1
    1   0 136   0   1   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0  17   0   0   0   0]
 [  0   0   0   0   0   0   1   0   0   0   1   0   1  11   0   0   0   0
    0   0   9   0  34   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   4   1   0   0   0   0   0   0
    0   0   0   1   0  33   2   0]
```

```
    0   0   1   0   0   0   1   0]
 [  1   1   0   1   0   0   0   2   0   0   0   0   4  99   0   0   0   0
    1   0   1   0   2   1   0   0]
 [  0   1  11   5   2   0   2   0   0   0   0   0   0   0 279   0   1   0
    1   0   1   0   0   0   0   0]
 [  2   0   0   1   0   2   0   0   0   0   0   0   0   0   0 100   0   0
    0   2   0   0   0   0   3   0]
 [  0   0   1   1   0   0   1   0   0   1   0   0   0   0   3   1  25   0
    0   0   0   0   0   0   0   0]
 [  8   0   0   0   5   0   0   0   0   0   2   0   0   1   0   1   0  45
    0   1   0   0   0   0   0   1]
 [  0   6   2   2   0   0   2   0   0  11   0   2   0   1   0   0   0   0
  233   0   0   0   0   0   0   0]
 [  0   0   0   0   0   2   0   0   0   2   0   0   1   0   0   0   0   0
    0 114   0   0   0   0   0   0]
 [  1   1   1   3   0   0   1   3   0   0   0   0   0   1   0   0   0   1
    1   0 136   0   1   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0  17   0   0   0   0]
 [  0   0   0   0   0   0   1   0   0   0   1   0   1  11   0   0   0   0
    0   0   9   0  34   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   4   1   0   0   0   0   0   0
    0   0   0   1   0  33   2   0]
 [  0   0   0   0   0   0   0   0   0   2   0   0   1   0   1   3   0   1
    0   4   0   0   0   1  39   0]
 [  0   0   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0  19]]
F1-Score (Linear Kernel): 0.8542733150751141
```

```
F1-Score (Linear Kernel): 0.8542733150751141
```

```
Training SVM with rbf kernel... Kindly, wait a little.
Evaluating SVM with rbf kernel...
```

```
Confusion Matrix (Rbf Kernel):
 [[ 74   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   1
    0   0   1   0   0   0   1   0]
 [  0  29   0   1   1   0   0   0   0   0   0   0   0   0   2   1   0   0
    0   0   0   0   0   0   0   0]
 [  0   1 119   0   0   0   0   0   0   0   0   1   0   1   4   0   0   1
    0   0   3   0   0   0   0   0]
 [  0   0   0  37   0   0   0   0   0   3   0   0   0   0   9   2   0   0
    1   2   1   0   0   0   0   0]
 [  0   2   3   0  53   0   0   0   0   0   0   1   0   0   1   1   0   0
    1   1   0   0   0   0   0   0]
 [  0   0   0   0   0   5   0   0   0   0   0   0   0   0   0   2   0   0
    0   1   0   0   0   0   0   0]
 [  0   1   3   0   0   0  26   0   0   0   0   0   0   0   1   0   2   1
    2   0   1   0   1   0   0   0]
 [  5   0   0   0   1   0   0  36   0   0   0   0   0   5   0   0   0   0
    0   0   2   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   9   2   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  37   0   0   0   0   0   0   0   0
    2   0   0   0   0   0   1   0]
 [  0   0   0   0   0   0   0   0   0   0  16   1   1   2   0   0   0   1
    0   0   0   0   0   0   0   0]
 [  0   0   3   0   0   0   0   0   0   0   0  64   0   0   0   0   0   0
    0   0   1   0   0   0   1   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0  60   3   0   0   0   0
    0   0   1   0   0   0   0   0]
 [  1   0   0   0   0   0   0   1   0   0   0   0   3 105   0   0   0   0
```

```
    0   0   0   0   1   0   2   0]
 [  0   1   5   3   2   0   1   0   0   0   0   0   0   0 291   0   0   0
    0   0   0   0   0   0   0   0]
 [  1   0   0   0   0   0   0   0   0   0   0   0   0   0   0 106   0   0
    0   1   0   0   0   0   2   0]
 [  0   0   0   2   0   0   2   0   0   0   0   0   0   0   5   1  23   0
    0   0   0   0   0   0   0   0]
 [  7   0   0   0   0   0   0   0   0   0   1   0   0   1   0   3   0  52
    0   0   0   0   0   0   0   0]
 [  0   0   1   0   0   0   0   0   0   6   0   1   0   0   0   1   0   0
  250   0   0   0   0   0   0   0]
 [  0   0   0   1   0   0   0   0   0   0   0   0   3   0   0   0   0   0
    0 113   0   0   0   0   2   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   1   2   0   0   0
    0   0 144   0   2   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0  17   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   1   0   1   6   0   0   0   0
    0   0   4   0  45   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0   0   1
    0   0   0   0   0  36   2   0]
 [  0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   1   0   0
    1   1   0   0   0   1  47   0]
 [  0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
    0   0   0   0   0   0   0  19]]
```

```
F1-Score (Rbf Kernel): 0.9051891544870462
```

- **Logistic Regression class (implemented from scratch)**

```python
class LogisticRegression:
    def __init__(self, learning_rate=0.01, max_iter=1000):
        self.learning_rate = learning_rate
        self.max_iter = max_iter
        self.theta = None

    def sigmoid(self, z):
        """Sigmoid activation function."""
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        """Fit logistic regression model using gradient descent."""
        m, n = X.shape
        self.theta = np.zeros(n)
        for _ in range(self.max_iter):
            z = np.dot(X, self.theta)
            h = self.sigmoid(z)
            gradient = np.dot(X.T, (h - y)) / m
            self.theta -= self.learning_rate * gradient

    def predict(self, X):
        """Predict probabilities and return class labels."""
        z = np.dot(X, self.theta)
        probabilities = self.sigmoid(z)
        return np.round(probabilities)  # Binary classification (0 or 1)
```

- **Perform Logistic Regression with One-vs-All Classification**

```python
def experiment_logistic_regression(normalized_features, labels):
    """Perform Experiment 2: Logistic Regression with One-vs-All Classification."""
    try:
        subset_features, subset_labels = select_subset(normalized_features, labels)
        X_train, X_val, X_test, y_train, y_val, y_test = split_data(subset_features, subset_labels)
        X_train, X_val, X_test = add_bias_term(X_train, X_val, X_test)
        classifiers = train_logistic_regression(X_train, y_train)
        y_val_pred, y_test_pred = predict_labels(classifiers, X_val, X_test)
        evaluate_and_plot(y_val, y_val_pred, y_test, y_test_pred, len(classifiers))
    except Exception as e:
        print(f"Error during Logistic Regression experiment: {e}")
```

- **Selecting a subset of 10,000 random data from the csv file**

```
def select_subset(normalized_features, labels):  1 usage
    print("Selecting a subset of 10,000 samples...")
    subset_indices = np.random.choice(normalized_features.index,  size: 10000, replace=False)
    subset_features = normalized_features.iloc[subset_indices]
    subset_labels = labels.iloc[subset_indices]
    return subset_features, subset_labels
```

- **Split the data into training, validation and testing sets**

```
def split_data(subset_features, subset_labels):  1 usage
    print("Splitting the subset into training, validation, and testing sets...")
    X_train, X_temp, y_train, y_temp = train_test_split( *arrays: subset_features, subset_labels, test_size=0.4, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split( *arrays: X_temp, y_temp, test_size=0.5, random_state=42)
    return X_train, X_val, X_test, y_train, y_val, y_test
```

- **Add bias term**

```
def add_bias_term(X_train, X_val, X_test):  1 usage
    X_train = np.hstack((np.ones((X_train.shape[0], 1)), X_train))
    X_val = np.hstack((np.ones((X_val.shape[0], 1)), X_val))
    X_test = np.hstack((np.ones((X_test.shape[0], 1)), X_test))
    return X_train, X_val, X_test
```

- **Train logistic Regression**

```
def train_logistic_regression(X_train, y_train):  1 usage
    num_classes = len(np.unique(y_train))
    y_train_onehot = pd.get_dummies(y_train).to_numpy()
    classifiers = []
    for i in range(num_classes):
        print(f"Training logistic regression for class {chr(i + 65)}...")
        clf = LogisticRegression(learning_rate=0.1, max_iter=500)
        clf.fit(X_train, y_train_onehot[:, i])
        classifiers.append(clf)
    return classifiers
```

❖ **Predicting validation set labels & testing set labels**

```
def predict_labels(classifiers, X_val, X_test):  1 usage
    print("Predicting validation set labels...")
    val_probabilities = np.array([clf.sigmoid(np.dot(X_val, clf.theta)) for clf in classifiers]).T
    y_val_pred = np.argmax(val_probabilities, axis=1)

    print("Predicting test set labels...")
    test_probabilities = np.array([clf.sigmoid(np.dot(X_test, clf.theta)) for clf in classifiers]).T
    y_test_pred = np.argmax(test_probabilities, axis=1)

    return y_val_pred, y_test_pred
```

❖ **Evaluating test set and validation set , then plotting confusion matrix for both**

```
def evaluate_and_plot(y_val, y_val_pred, y_test, y_test_pred, num_classes):  1 usage
    print("Evaluating Logistic Regression on Validation Set...")
    cm_val = confusion_matrix(y_val, y_val_pred)
    f1_val = f1_score(y_val, y_val_pred, average='weighted')
    print("Confusion Matrix (Validation Set):\n", cm_val)
    print("F1-Score (Validation Set):", f1_val)

    print("Evaluating Logistic Regression on Test Set...")
    cm_test = confusion_matrix(y_test, y_test_pred)
    f1_test = f1_score(y_test, y_test_pred, average='weighted')
    print("Confusion Matrix (Test Set):\n", cm_test)
    print("F1-Score (Test Set):", f1_test)

    sns.heatmap(cm_val, annot=True, fmt='d', cmap='coolwarm', xticklabels=[chr(i + 65) for i in range(num_classes)],
                yticklabels=[chr(i + 65) for i in range(num_classes)])
    plt.title("Confusion Matrix - Validation Set (Logistic Regression)")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

    sns.heatmap(cm_test, annot=True, fmt='d', cmap='coolwarm', xticklabels=[chr(i + 65) for i in range(num_classes)],
                yticklabels=[chr(i + 65) for i in range(num_classes)])
    plt.title("Confusion Matrix - Test Set (Logistic Regression)")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

- **Output**

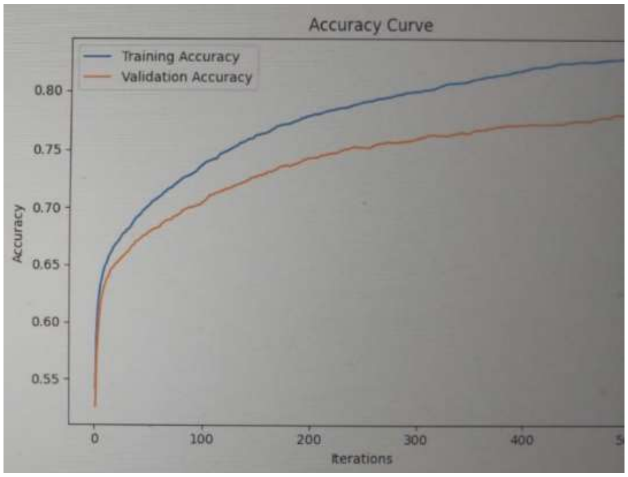
✓ Split the training dataset into training and validation datasets.

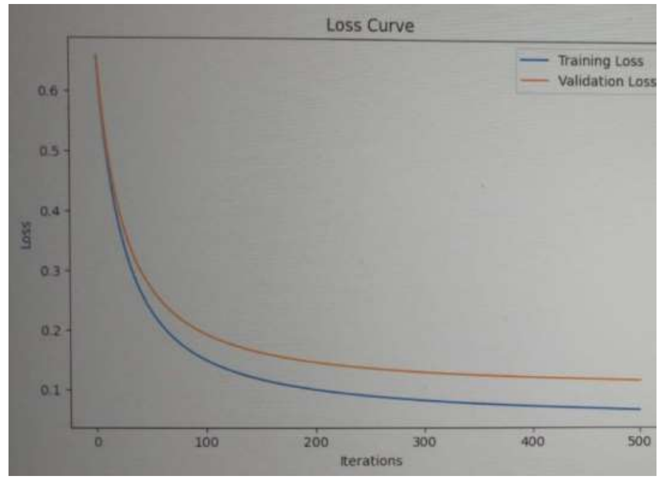
```
Selecting a subset of samples...
Splitting the subset into training, validation, and testing sets...
```

Experiment-2

✓ Implement logistic regression for one-versus-all multi-class
✓ classification.
✓ Train the model and plot the error and accuracy curves for the training
✓ and validation data.
✓ Test the model and provide the confusion matrix and the average f-1
✓ scores for the testing dataset

```
Selecting a subset of 10,000 samples...
Splitting the subset into training, validation, and testing sets...
Training logistic regression for class A...
Training logistic regression for class B...
Training logistic regression for class C...
Training logistic regression for class D...
Training logistic regression for class E...
Training logistic regression for class F...
Training logistic regression for class G...
Training logistic regression for class H...
Training logistic regression for class I...
Training logistic regression for class J...
Training logistic regression for class K...
Training logistic regression for class L...
Training logistic regression for class M...
Training logistic regression for class N...
Training logistic regression for class O...
Training logistic regression for class P...
Training logistic regression for class Q...
Training logistic regression for class R...
Training logistic regression for class S...
Training logistic regression for class T...
Training logistic regression for class U...
Training logistic regression for class V...
Training logistic regression for class W...
Training logistic regression for class X...
Training logistic regression for class Y...
Training logistic regression for class Z...
```






```
Predicting validation set labels...
Predicting test set labels...
Evaluating Logistic Regression on Validation Set...
```

```
Confusion Matrix (Validation Set):
 [[ 57   0   0   0   3   0   1   0   0   0   0   0   2   2   3   0   0   1
    0   0   0   0   0   0   0   0]
 [  2  26   1   1   3   0   0   0   0   0   0   0   1   0   3   1   0   0
   12   0   0   0   0   0   0   2]
 [  0   1 101   0   1   0   0   0   0   0   0   1   0   1   6   0   0   2
    0   0   0   0   0   0   0   0]
 [  1   0   0  25   0   0   0   0   0   0   0   0   1   0  16   1   0   0
    2   0   1   0   0   0   0   0]
 [  0   1   1   0  49   0   0   0   0   0   0   1   0   1   1   0   0   2
    4   0   0   0   0   0   0   0]
 [  0   0   0   0   6   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   1   0   0   0   0   0   0]
 [  0   0   1   0   1   0  10   0   0   0   0   0   0   0   6   0   0   1
   11   0   0   0   0   0   0   0]
 [  7   0   0   0   2   0   0  13   0   0   0   0   1  10   0   0   0   1
    0   0   7   0   1   1   2   0]
 [  0   0   0   1   0   0   0   0   0   1   0   1   0   0   0   0   0   0
    2   1   0   0   0   0   0   0]
 [  0   0   1   1   3   0   0   0   0   9   0   0   1   0   3   0   0   0
   19  15   9   0   0   0   3   0]
 [  1   0   1   0   1   0   0   1   0   0  14   0   0   1   0   0   0   7
    1   0   0   1   1   1   2   0]
 [  0   0   1   0   0   0   0   0   0   0   0  46   0   2   0   0   0   2
    2   0   1   0   0   0   3   0]
 [  4   0   1   0   0   0   0   2   0   0   0   0  63   3   0   0   0   0
    0   0   0   0   0   0   0   0]
 [  3   0   0   0   0   0   0   0   0   0   0   0   3  83   5   0   0   2
    0   0   5   0   4   0   0   0]
```

Validation set


```
    2   0   1   0   0   0   3   0]
 [  4   0   1   0   0   0   0   2   0   0   0   0  63   3   0   0   0   0
    0   0   0   0   0   0   0   0]
 [  3   0   0   0   0   0   0   0   0   0   0   0   3  83   5   0   0   2
    0   0   5   0   4   0   0   0]
```


```
 [  0   0   1   1   1   0   0   0   0   0   0   0   1   0 289   0   0   0
    5   0   2   0   0   0   0   0]
 [  4   0   0   2   1   0   0   0   0   0   0   0   0   2   7  85   0   4
    0   5   0   0   0   0   0   0]
 [  1   0   0   0   0   0   0   0   0   0   0   0   0   0   6   1  21   2
    1   0   2   0   0   0   0   0]
 [  6   0   0   0   2   0   0   0   0   0   0   1   0   1   1   3   1  37
    0   0   1   0   0   1   0   0]
 [  0   0   4   1   1   0   0   0   0   0   0   0   0   0   7   1   2   1
  238   2   2   0   0   1   2   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   2   0   0
    0  95   0   0   0   0   0   0]
 [  2   0   0   2   1   0   1   0   0   0   0   1   0   5  11   0   0   3
    0   0 124   0   9   0   2   0]
 [  0   0   0   0   2   0   0   0   0   0   0   0   0   2   0   0   0   0
    0   0  12   9   0   0   0   0]
 [  1   0   0   0   0   0   0   0   0   0   0   0   0   5   0   0   0   0
    0   0   5   0  43   0   0   0]
 [  3   0   0   0   1   0   0   0   0   0   4   0   0   0   0   0   0   2
    1   0   0   0   0  20   4   0]
 [  1   0   0   0   0   0   0   0   0   0   0   0   1   0   2   3   0   0
    1   7   1   0   0   0  52   0]
 [  1   3   0   0   0   0   0   0   0   0   0   3   0   1   0   1   0   1
    1   2   0   0   0   0   0  26]]
```

```
F1-Score (Validation Set): 0.749510026421582
```

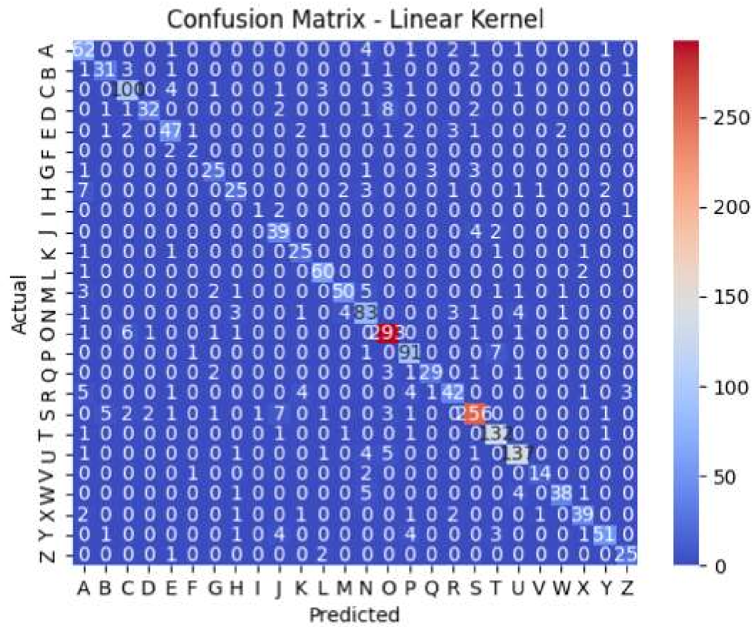
Test Set

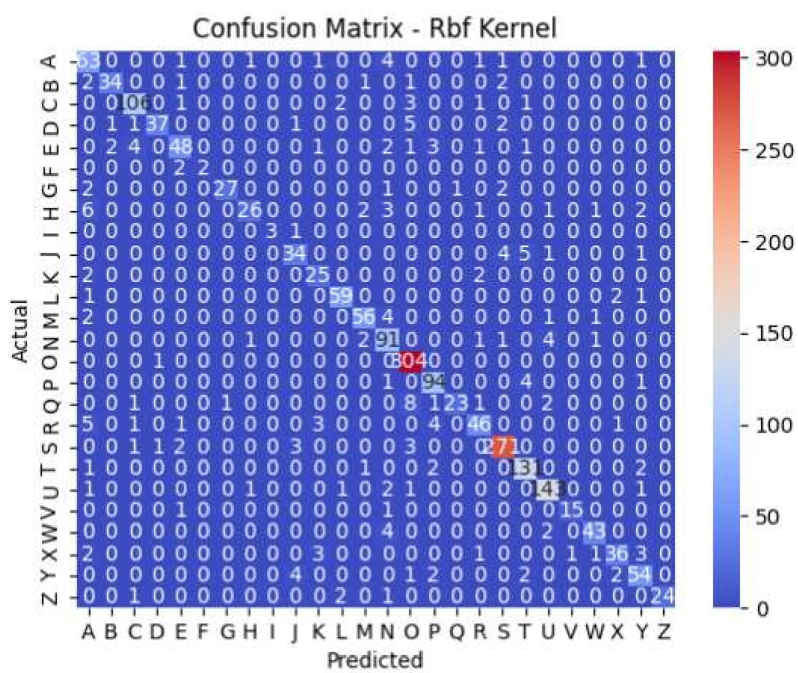
```
Evaluating Logistic Regression on Test Set...
Confusion Matrix (Test Set):
 [[ 66   1   0   0   3   0   1   0   0   0   0   0   4   3   5   1   0   2
    1   0   0   0   3   0   0   0]
 [  2  26   1   0   2   0   0   0   0   0   0   0   0   0   6   0   0   0
   16   0   1   0   0   0   0   1]
 [  0   0 109   0   1   0   0   0   0   0   0   1   0   0  12   0   0   0
    1   0   1   0   0   0   0   0]
 [  0   2   0  30   0   0   0   0   0   0   0   0   1   0  25   0   0   0
    1   1   2   0   0   0   0   1]
 [  1   2   6   0  42   0   0   0   0   0   0   1   0   0   3   0   1   2
    1   0   0   0   0   0   0   0]
 [  0   0   0   0  11   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0]
 [  0   0   4   0   1   0   8   0   0   0   0   0   0   0   2   0   1   0
   11   0   0   0   0   0   0   0]
 [  6   1   0   0   1   0   0  16   0   0   0   0   0  11   0   3   0   1
    0   0   1   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
    4   0   0   0   0   0   0   0]
 [  0   0   3   0   0   0   0   0   0   7   0   0   0   0   0   1   0   0
   15  11   6   0   0   0   1   0]
 [  0   0   1   0   1   0   0   0   0   0  19   1   1   1   0   0   0   6
    0   0   0   0   0   0   1   0]
 [  0   0   0   0   0   0   0   1   0   0   0  55   0   0   0   0   0   0
    0   0   2   0   0   0   1   0]
 [  1   0   0   1   0   0   0   0   0   0   0   0  42   4   1   0   0   0
    0   0   0   0   0   0   0   0]
 [  2   0   0   0   1   0   0   1   0   0   0   0   5  74   1   2   0   0
```

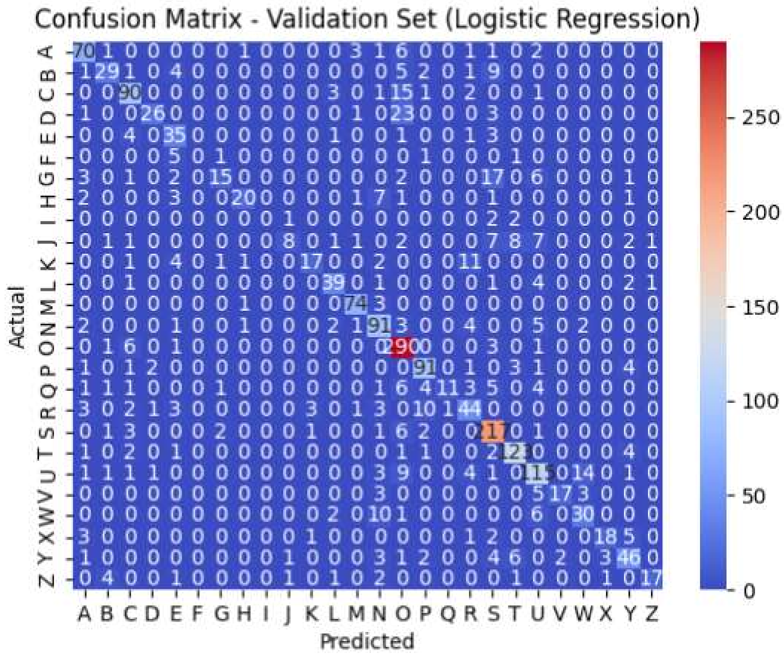
```
    0   0   2   0   0   0   1   0]
 [  1   0   0   1   0   0   0   0   0   0   0   0  42   4   1   0   0   0
    0   0   0   0   0   0   0   0]
 [  2   0   0   0   1   0   0   1   0   0   0   0   5  74   1   2   0   0
    0   0   6   1   3   0   1   0]
 [  0   0   2   1   0   0   0   0   0   0   0   0   0   1 301   0   0   0
    2   0   0   0   0   0   0   0]
 [  2   0   0   3   2   0   0   0   0   0   0   0   0   0   1 100   0   1
    0   5   0   0   0   0   0   0]
 [  0   0   2   0   0   0   0   0   0   0   0   0   1   0   8   3  13   0
    0   0   3   0   0   0   0   0]
 [  2   0   3   0   3   0   0   0   0   0   1   0   3   2   0   3   1  33
    0   2   0   0   1   0   0   0]
 [  0   0   3   0   0   0   0   0   0   0   0   2   0   0   9   1   0   1
  247   1   1   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   1   0   0   6   0   0   3   0   1
    0 110   0   0   0   0   3   0]
 [  1   0   0   1   0   0   0   0   0   0   0   0   0   3   8   1   0   2
    0   1 130   0   7   0   0   1]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
    0   0   4   9   1   0   0   0]
 [  0   0   0   1   1   0   0   0   0   0   0   2   0   5   2   0   0   0
    1   0   7   3  44   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0   1   0   0   0   0
    5   0   0   0   0  15   1   1]
 [  2   1   0   0   0   0   0   0   0   0   1   0   1   1   0   5   0   0
    2   2   1   0   0   2  43   1]
 [  1   1   0   0   1   0   0   0   0   1   0   2   0   1   0   1   0   1
    1   0   0   0   0   0   0  19]]
```

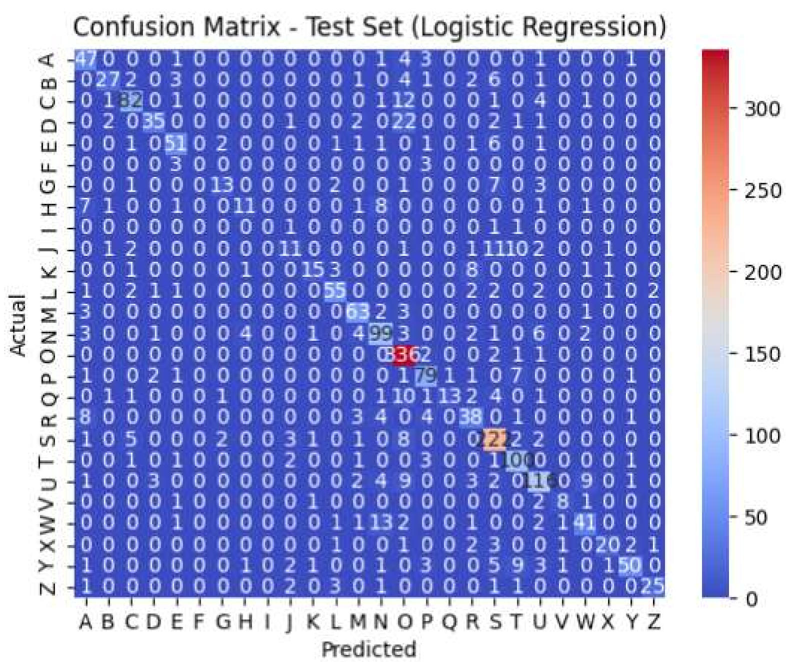
```
F1-Score (Test Set): 0.7622319142100517
```

Additional plots for experiments 1,2 matrices:






Confusion Matrix - Validation Set (Logistic Regression)
Actual
Predicted
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0
50
100
150
200
250


Confusion Matrix - Test Set (Logistic Regression)
Actual
Predicted
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0
50
100
150
200
250
300

Experiment 3 : 2 Neural Network Models

Splitting the data into training , validation and testing sets (60%,20%,20%)

```python
def split_and_one_hot_encode(normalized_features, labels):
    try:

        print("Selecting a subset of 10,000 samples...")
        # Starts by taking only 10k samples from the dataset
        subset_indices = np.random.choice(normalized_features.index, size: 10000, replace=False)
        # Then takes the number of features to be the input neurons
        subset_features = normalized_features.iloc[subset_indices]
        # Takes the labels provided to check if the output was right
        subset_labels = labels.iloc[subset_indices]

        # Splits the dataset we got into training, validation, and testing sets
        print("Splitting the subset into training, validation, and testing sets...")
        # Takes 60% as training and 40% as temp
        X_train, X_temp, y_train, y_temp = train_test_split( *arrays: subset_features, subset_labels, test_size=0.4, random_state=42)
        #Splits the 40% of the temp as 20 % validation and 20 % testing set
        X_val, X_test, y_val, y_test = train_test_split( *arrays: X_temp, y_temp, test_size=0.5, random_state=42)

        # Convert labels to one-hot encoding for each set , this encoding changes the labels to binary vectors
        #The vector size is based on the number of classes , since we have 26 alphabets so the size is 26
        y_train_onehot = tf.keras.utils.to_categorical(y_train, num_classes=26)
        y_val_onehot = tf.keras.utils.to_categorical(y_val, num_classes=26)
        y_test_onehot = tf.keras.utils.to_categorical(y_test, num_classes=26)

        return X_train, X_val, X_test, y_train_onehot, y_val_onehot, y_test_onehot

    except Exception as e:
        print(f"Error during data splitting and encoding: {e}")
```

Trains the data with First Neural Network Model that has 2 hidden layers and relu activation function

```python
def train_neural_network_1(X_train, y_train_onehot, X_val, y_val_onehot):
    # First Neural Network has 2 hidden layers and relu activation function
    try:
        print("Training Neural Network 1...")
        model_1 = Sequential([

            # Takes input neurons based on features
            Flatten(input_shape=(X_train.shape[1],)),
            # First hidden layer's number of neurons along with its activation function
            Dense(128, activation='relu'),
            # Drops 20% of the neurons to avoid overfitting
            Dropout(0.2),
            # Second hidden layer's number of neurons along with its activation function
            Dense(64, activation='relu'),
            # Softmax function to give the probability of each alphabet
            Dense(26, activation='softmax')
        ])

        model_1.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

        #Generates 20 epochs it does this cycle for 20 times , taking 64 by 64 and patience of 3 as if it haven't improved for 3 iterations it stops
        history_1 = model_1.fit(X_train, y_train_onehot, validation_data=(X_val, y_val_onehot), epochs=20, batch_size=64, verbose=1,
                                callbacks=[EarlyStopping(monitor='val_loss', patience=3)])

        # Plots Neural Network 1 performance
        plot_training_curves(history_1, model_name: "Neural Network 1")
        return model_1, history_1
```

Trains the data with Second Neural Network Model using 3 hidden layers and tanh activation function

```python
def train_neural_network_2(X_train, y_train_onehot, X_val, y_val_onehot):
    # Second Neural Network has 3 hidden layers and tanh activation function
    try:
        print("Training Neural Network 2...")
        model_2 = Sequential([
            Flatten(input_shape=(X_train.shape[1],)),
            # First hidden layer's number of neurons along with its activation function
            Dense(256, activation='tanh'),
            # Drops 30% of the neurons to avoid overfitting
            Dropout(0.3),
            # Second hidden layer's number of neurons along with its activation function
            Dense(128, activation='tanh'),
            # Third hidden layer's number of neurons along with its activation function
            Dense(64, activation='tanh'),
            # Softmax function to give the probability of each alphabet
            Dense(26, activation='softmax')
        ])
        model_2.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

        #Generates 20 epochs it does this cycle for 20 times , taking 64 by 64 and patience of 3 as if it haven't improved for 3 iterations it stops
        history_2 = model_2.fit(X_train, y_train_onehot, validation_data=(X_val, y_val_onehot), epochs=20, batch_size=64, verbose=1,
                                callbacks=[EarlyStopping(monitor='val_loss', patience=3)])

        # Plots Neural Network 2 performance
        plot_training_curves(history_2, model_name: "Neural Network 2")
        return model_2, history_2

    except Exception as e:
        print(f"Error during Neural Network 2 training: {e}")
```

Function to evaluate the model

```python
#Function to evaluate the model , test it , and print out a classification report
def evaluate_model(model, X_test, y_test, model_name):

    test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
    print(f"{model_name} - Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")

    y_pred = model.predict(X_test).argmax(axis=1)
    y_true = y_test.argmax(axis=1)
    print(f"{model_name} - Classification Report:")
    print(classification_report(y_true, y_pred, target_names=[chr(i + 65) for i in range(26)]))
```

Preforms the whole neural network experiment along with evaluating both models

```python
#Experiment 3 2 Neural Network models
def experiment_neural_networks(normalized_features, labels):

    try:
        # Step 1: Split the data and apply one-hot encoding
        X_train, X_val, X_test, y_train_onehot, y_val_onehot, y_test_onehot = split_and_one_hot_encode(normalized_features, labels)

        # Step 2: Train Neural Network 1
        model_1, history_1 = train_neural_network_1(X_train, y_train_onehot, X_val, y_val_onehot)

        # Step 3: Train Neural Network 2
        model_2, history_2 = train_neural_network_2(X_train, y_train_onehot, X_val, y_val_onehot)

        # Step 4: Evaluate on Test Set
        print("Evaluating Neural Networks on the Test Set...")
        evaluate_model(model_1, X_test, y_test_onehot, model_name: "Neural Network 1")
        evaluate_model(model_2, X_test, y_test_onehot, model_name: "Neural Network 2")

    except Exception as e:
        print(f"Error during Neural Network experiment: {e}")
```

Functions to select and save the best models

```python
def select_best_model(model_1, history_1, model_2, history_2):
    """Select the best model based on validation accuracy."""
    best_model = model_1 if max(history_1.history['val_accuracy']) > max(history_2.history['val_accuracy']) else model_2
    return best_model


def save_and_load_best_model(best_model, file_path):
    """Save the best model to a file and reload it."""
    # Save the model
    best_model.save(file_path)
    print(f"Model saved to {file_path}")

    # Reload the model
    loaded_model = load_model(file_path)
    print("Model reloaded successfully")
    return loaded_model
```

Function to evaluate the best model

```python
def evaluate_best_model_with_metrics(best_model, X_test, y_test):
    """Evaluate the best model and provide a confusion matrix and average F1 score."""
    y_pred = best_model.predict(X_test).argmax(axis=1)
    y_true = y_test

    # Confusion Matrix
    conf_matrix = confusion_matrix(y_true, y_pred)
    print("Confusion Matrix:")
    print(conf_matrix)

    # Average F1 Score
    avg_f1 = f1_score(y_true, y_pred, average='macro')
    print(f"Average F1 Score: {avg_f1:.4f}")
```

Functions to preprocess the names and gets a random sample for the letter

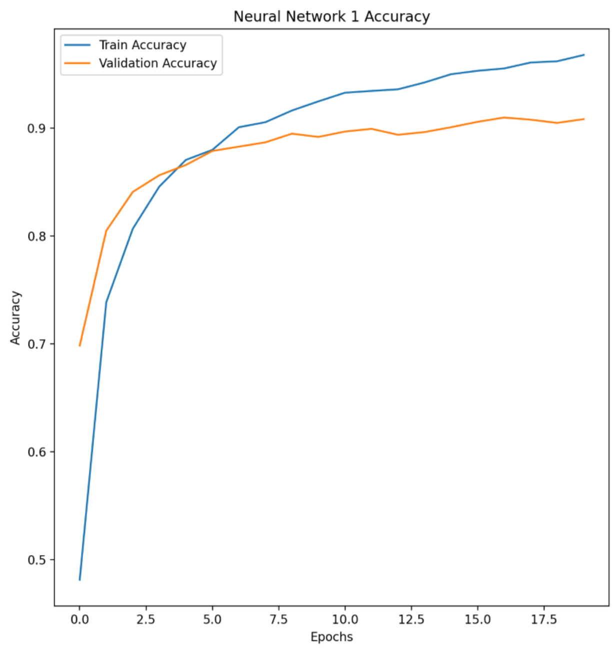
```python
def preprocess_team_names(team_names):
    letters_to_test = set(''.join(team_names).upper())
    return sorted(letters_to_test)

def get_random_sample_for_letter(letter, X_data, y_data):
    # Determine the class number for the given letter
    class_num = ord(letter.upper()) - 65

    # Handle both one-hot encoded and class-label data
    if len(y_data.shape) > 1:  # One-hot encoded
        indices = np.where(np.argmax(y_data, axis=1) == class_num)[0]
    else:  # Class labels
        indices = np.where(y_data == class_num)[0]

    # Check if there are any samples for the given class
    if len(indices) == 0:
        return None, None

    # Choose a random index from the valid indices
    chosen_idx = random.choice(indices)

    # Use iloc for Pandas DataFrame, or direct indexing for NumPy arrays
    if isinstance(X_data, pd.DataFrame):
        selected_feature = X_data.iloc[chosen_idx]
    else:
        selected_feature = X_data[chosen_idx]

    # Return the selected feature and the class number
    return selected_feature, class_num
```

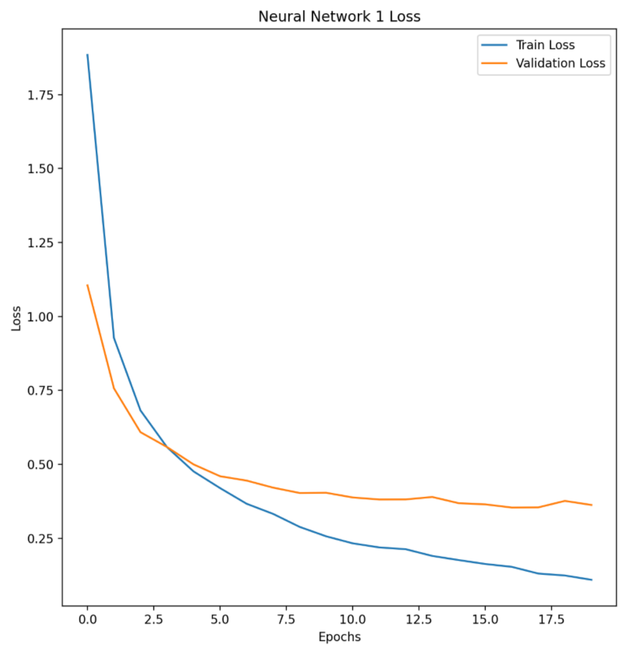
Plotting the team members names

```
def plot_predictions_for_names(team_names, best_model, X_test, y_test):
    for name in team_names:
        letters = list(name.upper())

        # Create subplots for each name based on the number of letters in the name
        fig, axes = plt.subplots( nrows: 1, len(letters), figsize=(len(letters) * 3, 3))
        fig.suptitle( t: f"Predictions for '{name.capitalize()}'", fontsize=16)

        # If there is only one letter in the name, make axes a list for consistent indexing
        if len(letters) == 1:
            axes = [axes]

        # Loop through the letters in the name
        for i, letter in enumerate(letters):
            img, class_num = get_random_sample_for_letter(letter, X_test, y_test)

            if img is not None:
                img_flat = img.to_numpy().reshape(1, -1)  # Flatten the image to match the model input
                pred_proba = best_model.predict(img_flat)  # Predict the probabilities for each class
                pred_class = np.argmax(pred_proba)  # Get the predicted class (the most likely letter)
                predicted_letter = chr(65 + pred_class)  # Convert to the corresponding letter

                # Reshape the image to 28x28 pixels for displaying
                img_reshaped = img_flat.reshape(28, 28)

                # Display the image with the true and predicted letters
                axes[i].imshow(img_reshaped, cmap='gray')
                axes[i].set_title(f"True: {letter}\nPred: {predicted_letter}")
                axes[i].axis('off')
            else:
                axes[i].axis('off')  # If no image found, turn off the axis
```

Experiment 3:

✓ Design 2 Neural Networks (with different number of hidden layers, neurons, activations, etc.) ✓ Train each one of these models and plot the error and accuracy curves for the training data and validation datasets.

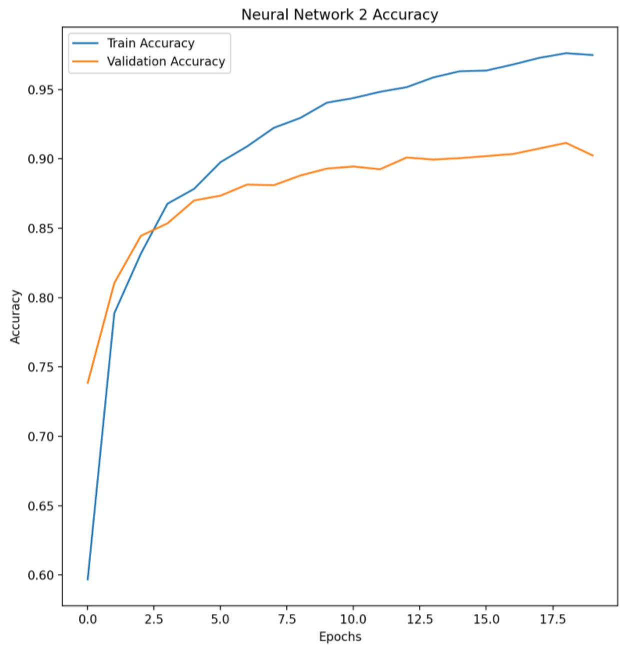


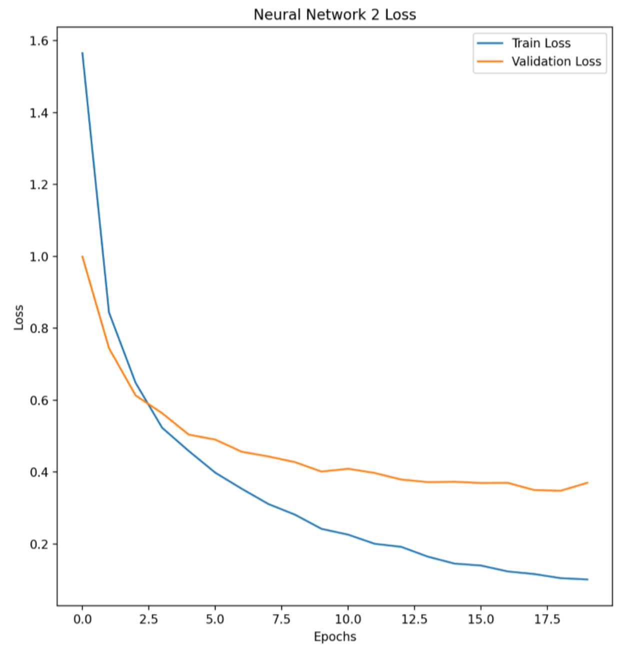


Run MachineLearningNour

Neural Network 1 - Classification Report:

	precision	recall	f1-score	support
A	0.84	0.90	0.87	77
B	0.74	0.77	0.76	44
C	0.95	0.96	0.95	121
D	0.89	0.80	0.84	49
E	0.84	0.87	0.86	62
F	1.00	0.43	0.60	7
G	0.96	0.82	0.88	28
H	0.64	0.68	0.66	34
I	0.75	1.00	0.86	3
J	0.92	0.90	0.91	51
K	0.83	0.74	0.78	27
L	0.85	0.90	0.87	61
M	0.94	0.92	0.93	71
N	0.73	0.86	0.79	83
O	0.96	0.97	0.97	303
P	0.92	0.93	0.92	121
Q	0.89	0.82	0.85	39
R	0.90	0.87	0.88	70
S	0.95	0.95	0.95	263
T	0.96	0.94	0.95	125
U	0.96	0.95	0.96	167
V	0.96	0.92	0.94	24
W	0.91	0.74	0.82	58
X	0.85	0.79	0.81	28
Y	0.86	0.95	0.90	59
Z	0.82	0.92	0.87	25
accuracy			0.91	2000
macro avg	0.88	0.86	0.86	2000
weighted avg	0.91	0.91	0.91	2000


Neural Network 2 Accuracy
Train Accuracy
Validation Accuracy
Accuracy
Epochs
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Neural Network 2 Loss
Train Loss
Validation Loss
Loss
Epochs
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Neural Network 2 - Classification Report:

	precision	recall	f1-score	support
A	0.82	0.86	0.84	77
B	1.00	0.75	0.86	44
C	0.94	0.93	0.94	121
D	0.98	0.86	0.91	49
E	0.82	0.82	0.82	62
F	0.88	1.00	0.93	7
G	0.76	0.79	0.77	28
H	0.71	0.71	0.71	34
I	1.00	1.00	1.00	3
J	0.83	0.88	0.86	51
K	0.83	0.70	0.76	27
L	0.87	0.89	0.88	61
M	0.90	0.92	0.91	71
N	0.79	0.86	0.82	83
O	0.96	0.98	0.97	303
P	0.93	0.93	0.93	121
Q	0.92	0.85	0.88	39
R	0.83	0.93	0.88	70
S	0.96	0.93	0.95	263
T	0.95	0.95	0.95	125
U	0.93	0.94	0.94	167
V	0.95	0.83	0.89	24
W	0.94	0.79	0.86	58
X	0.75	0.86	0.80	28
Y	0.83	0.98	0.90	59
Z	0.87	0.80	0.83	25
accuracy			0.91	2000
macro avg	0.88	0.87	0.88	2000
weighted avg	0.91	0.91	0.91	2000

- ✓ **Save the best model in a separated file, then reload it.**
- ✓ **Test the best model and provide the confusion matrix and the average f-1 scores for the testing data.**
- ✓ **Test the best model with images representing the alphabetical letters for the names of each member of your team.**

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
Model saved to best_model.h5
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be bu
Model reloaded successfully
```

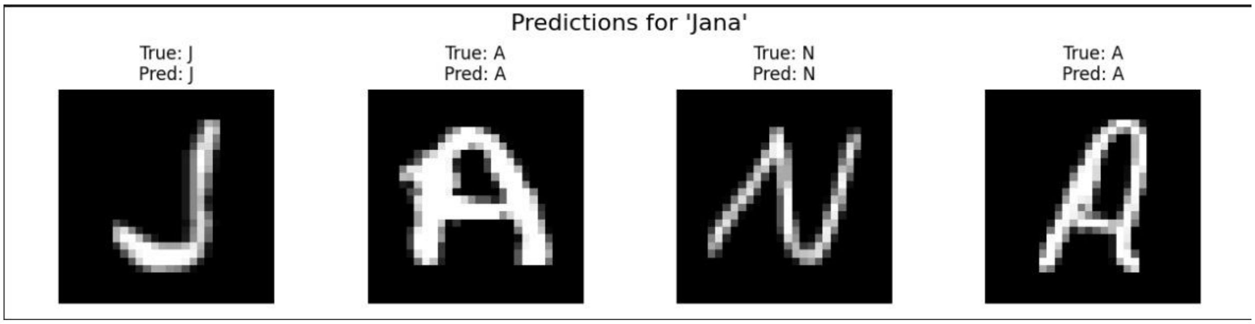
```
2328/2328 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step
Confusion Matrix:
[[ 2481     8     0     8     2     1     2    40     0     0    11     2
     28    54     5    14    20    20     3     0     4     0     6    25
      6     1]
 [   35  1388    14    23    34     0    16    19     0     5     1     1
      8     5    50     5    10    23    18     2     4     0     1     1
      4    17]
 [    2     9  4517     0    20     1     7     3     0     2     4    32
      1     4    60    10     4    27    10     3    13     0     5     0
      2     2]
 [    2    26     1  1686     3     0     1     3     0     7     0     5
      6     6   254    24     7     0    24     0     8     0    11     0
      0     7]
 [   14    33    92     4  1868    12    35     3     0     0    17    20
      1    12     1    12     2    48    19     2    13     1     5     0
      3     8]
 [    3     0     1     0    12   163     0     0     0     0     0     0
      0     0     0    25     0     0     1    11     0     0     0     0
      0     0]
 [   16    15    63     1    25     0   959     4     0     3     0     0
      4     2    10     1    36     0    29     0     4     0     6     0
      0     0]
 [   64     3     1     1     2     0     3  1171     0     0     1     0
     34    94     0     2     1     8     2     0    22     2    25     3
      7     1]
 [    0     0     2     1     7     0     0     0   180    10     0     0
      0     0     0     3     0     0    18     7     0     0     0     1
      5     6]
 [    2     3    17    46     4     0     2     1    12  1179     0     3
```

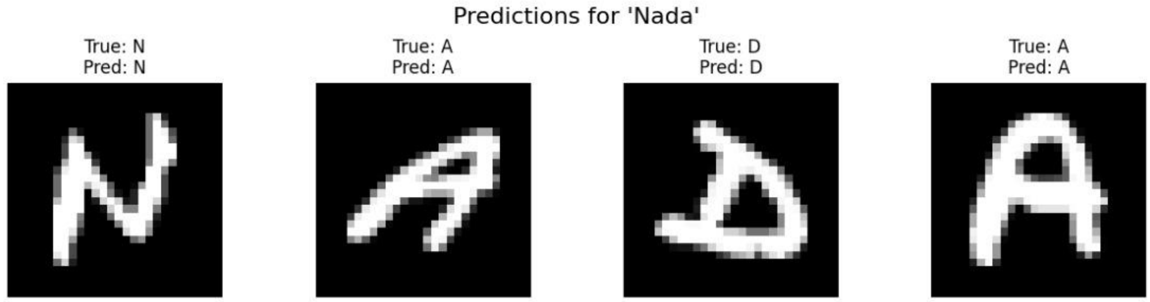
C: > Users > Public > Downloads > MachineLearningNour.py   650:1 CRLF UTF-8 4 spaces Python 3.11 (PythonProject2)

```
      1    12    10     4     5     0   115    87    69     2     4     2
     12     7]
 [    8     3    33     0    14     3     0    30     0     0   775     6
      3    41     0     4     1   103     3     6    16     2    10    11
      8     5]
 [    3     0    61     2    10     0     7     2     9     3     5  2108
      0     7     0     0     0     5     7     1     4     5    14    17
      7     6]
 [   70     1     2     2     1     0     1    70     0     0     1     0
   2161    70    12    12     8     8     2    15    17     0    19     3
      3     0]
 [   51     0     1     6     3     2     1    79     0     1    10     0
     40  3316    10     5    11    10     8     4    29     5    95     1
      3     0]
 [   17    17    51    86     9     0     9     3     0     1     0     2
     22    24 11305    13    30     2    11     2    41     0     2     0
      0     1]
 [   21     2    10    40     4     0     2     2     0     0     3     0
      1     7     8  3693     7    20     0    20     1     0     4     1
     27     0]
 [   15     5     8     5     1     0    36     1     0     0     0     0
      0    10    57    27   983    26     6     0    14     0     2     0
      0     1]
 [  123    22    18     3    46     1     0    15     0     0    31     7
     16    31     2    54    34  1878     1     4    10     0     4    13
      0    15]
 [    9    29    45    28    21     3    57     0     1    46     3     6
      3    25    47     5    18     4  9293     9    21     1    12     8
      3     2]
 [    6     0     6     1     4     3     2    21     1     2     7     0
     15     3     0    49     2     3    16  4328     2     0     0     1
```

```
    0   10   57   27  983   26    6    0   14    0    2    0
    0    1]
 [ 123   22   18    3   46    1    0   15    0    0   31    7
   16   31    2   54   34 1878    1    4   10    0    4   13
    0   15]
 [   9   29   45   28   21    3   57    0    1   46    3    6
    3   25   47    5   18    4 9293    9   21    1   12    8
    3    2]
 [   6    0    6    1    4    3    2   21    1    2    7    0
   15    3    0   49    2    3   16 4328    2    0    0    1
   31    0]
 [   5    4    9   36    5    0   17   63    0    4    2   20
    6   65   70    0    6   21    3    3 5377    7   53    5
    4   13]
 [   0    0    0    0    0    1    0    3    0    2    1    0
    1   21    0    1    1    0    0    0   16  797    6    1
    4    0]
 [  10    6    0    1    3    1    9    4    0    0    1    5
    7  175    1    0    4    4    3    1   53    4 1892    0
    0    0]
 [  22    1    0    2    2    0    2    6    0    0   64    2
    0   36    0    0    0   23    4    2    4   13   11  997
   53   16]
 [   9    7    3    5    0    1    1   40    1   29   18    0
    4   16    7   55    1    2   16   55    3   56    1   37
 1836    3]
 [  21   19    6   20   17    0    4    4    4    9    3   16
    2    8    1    8    5   17    6    4    2    1    2   13
    1 1061]]
Average F1 Score: 0.8655
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
```

Team members names




Predictions for 'Nada'
True: N
Pred: N
True: A
Pred: A
True: D
Pred: D
True: A
Pred: A

Comparison between the 3 models

1. SVM (Support Vector Machine):

Linear Kernel:

- **F1-Score (Linear Kernel): 0.859**

RBF (Radial Basis Function) Kernel:

- **F1-Score (RBF Kernel): 0.904**

2. Logistic Regression:

- **Test Set F1-Score: 0.744**

3. Neural Network 1:

- **Training Accuracy: 0.9661**
- **Training Loss: 0.1095**
- **Validation Accuracy: 0.9050**
- **Validation Loss: 0.3462**

Classification Report (for Neural Network 1):

- **Accuracy**: **0.91**
- **Macro Avg**:
    - **Precision: 0.89**
    - **Recall: 0.85**
    - **F1-Score: 0.86**
- **Weighted Avg**:
    - **Precision: 0.91**
    - **Recall: 0.91**
    - **F1-Score: 0.91**

4. Neural Network 2:

- **Training Accuracy**: **0.9740**
- **Training Loss**: **0.1140**
- **Validation Accuracy**: **0.8960**
- **Validation Loss: 0.3800**

Final Recommendation:

- **Best Model**: **Neural Network 1** is the best overall model, offering a strong combination of high training accuracy, good validation performance, and a balanced classification report.