EEG Classification Model

Group 35

Project 3
Final Report

Course Code: IE6400
Course Name: Foundation of Data Analytics
Fall 2023

Team Members:
Ginnegolla, Hema Venkata Sai Teja
Gopanwar, Anuj
Nalam, Lokhi
Rasineni, Neha
Sanapureddy, Sree Dharani Reddy

Task 1: Data Preprocessing

1. We meticulously documented the dataset path and utilized it to successfully load the data into our Jupyter notebook.

```
In [4]: # Function for load EEG data from a set of files
        def load_eeg_data(path):
            eeg_data = []
            file_names = os.listdir(path)
            file_names.sort()
            for file_name in file_names:
                file_path = os.path.join(path, file_name)
                with open(file_path, 'r') as file:
                    # Converting the data to integers
                    data = [int(line.strip()) for line in file.readlines()]
                    eeg_data.append(data)
            return np.array(eeg_data)

        set_f_path = os.path.join(dir_path, 'f')
        eeg_data_set_f = load_eeg_data(set_f_path)
        eeg_data_set_n = load_eeg_data(os.path.join(dir_path, 'n'))
        eeg_data_set_o = load_eeg_data(os.path.join(dir_path, 'o'))
        eeg_data_set_s = load_eeg_data(os.path.join(dir_path, 's'))
        eeg_data_set_z = load_eeg_data(os.path.join(dir_path, 'z'))
```

2. After exploring the data, we determined the dataset's shape and subsequently converted the Pandas array into NumPy to derive the statistical summary, enabling a comprehensive understanding of the dataset's characteristics.

```
# Finding the shape of the loaded data
print('Shape of EEG Data for Set F:')
print(eeg_data_set_f.shape)
print('\nShape of EEG Data for Set N:')
print(eeg_data_set_n.shape)
print('\nShape of EEG Data for Set O:')
print(eeg_data_set_o.shape)
print('\nShape of EEG Data for Set S:')
print(eeg_data_set_s.shape)
print('\nShape of EEG Data for Set Z:')
print(eeg_data_set_z.shape)
```

```
Shape of EEG Data for Set F:
(100, 4097)

Shape of EEG Data for Set N:
(100, 4097)

Shape of EEG Data for Set O:
(100, 4097)

Shape of EEG Data for Set S:
(100, 4097)

Shape of EEG Data for Set Z:
(100, 4097)
```

3. During this stage, our focus was on preprocessing the EEG data. We started by checking for missing values, and fortunately, we discovered that the dataset is complete—devoid of any missing values. This assurance allowed us to confidently proceed with the subsequent phase of feature extraction. Additionally, as part of EEG data preprocessing, we normalized the dataset. Following normalization, we calculated the mean and standard deviation of the normalized data, contributing to a deeper understanding of its distribution and characteristics.

```python
# Finding missing and anomalous values
def check_missing_values(data):
    # Finding NaN and large values to traceate missing or corrupt data
    nan_count = np.isnan(data).sum()
    extreme_values_count = np.sum(np.abs(data) > 1e6)
    return nan_count, extreme_values_count

missing_values_count_f, extreme_values_count_f = check_missing_values(eeg_data_set_f)
missing_values_count_n, extreme_values_count_n = check_missing_values(eeg_data_set_n)
missing_values_count_o, extreme_values_count_o = check_missing_values(eeg_data_set_o)
missing_values_count_s, extreme_values_count_s = check_missing_values(eeg_data_set_s)
missing_values_count_z, extreme_values_count_z = check_missing_values(eeg_data_set_z)

# Results for the above
print('Missing values and extreme values count for set F:')
print(missing_values_count_f, extreme_values_count_f)
print('\nMissing values and extreme values count for set N:')
print(missing_values_count_n, extreme_values_count_n)
print('\nMissing values and extreme values count for set O:')
print(missing_values_count_o, extreme_values_count_o)
print('\nMissing values and extreme values count for set S:')
print(missing_values_count_s, extreme_values_count_s)
print('\nMissing values and extreme values count for set Z:')
print(missing_values_count_z, extreme_values_count_z)
```

```
Missing values and extreme values count for set F:
0 0

Missing values and extreme values count for set N:
0 0

Missing values and extreme values count for set O:
0 0

Missing values and extreme values count for set S:
0 0

Missing values and extreme values count for set Z:
0 0
```

```
Shape of Filtered EEG Data for Set F:
(100, 4097)

Shape of Filtered EEG Data for Set N:
(100, 4097)

Shape of Filtered EEG Data for Set O:
(100, 4097)

Shape of Filtered EEG Data for Set S:
(100, 4097)

Shape of Filtered EEG Data for Set Z:
(100, 4097)
```

```
Mean and Standardized data for Set F:
2.954271763354517e-18 ; 0.9999999999999997

Mean and Standardized data for Set N:
1.0405800377253115e-19 ; 0.9999999999999999

Mean and Standardized data for Set O:
-4.769325172907678e-20 ; 1.0

Mean and Standardized data for Set S:
1.0926090396115771e-18 ; 1.0

Mean and Standardized data for Set Z:
-1.821015066019295e-19 ; 0.9999999999999998
```

```
Shape of Augmented data for set F:
(100, 4097)

Shape of Augmented data for set N:
(100, 4097)

Shape of Augmented data for set O:
(100, 4097)

Shape of Augmented data for set S:
(100, 4097)

Shape of Augmented data for set Z:
(100, 4097)
```

Task 2: Feature Extraction

To enhance our analysis, we proceeded with feature extraction from the EEG signals. Our approach encompassed extracting pertinent features from both the time-domain and frequency-domain, ensuring a comprehensive representation of the underlying characteristics within the dataset. Additionally, we've crafted individual histograms for each EEG dataset, showcasing two key aspects:
- The first histogram reveals the distribution of mean values.
- The second histogram illustrates the spread of standard deviation values.

These visual representations aim to elucidate the diversity and patterns within the mean and standard deviation values across the varied EEG datasets. The goal is to discern potential distinctions or similarities in the signal characteristics among these datasets, facilitating a deeper understanding of their unique traits.

```
import numpy as np
import scipy.stats
from scipy.signal import welch

# Time-Domain Feature Extraction
def extract_time_domain_features(data):
    means = np.mean(data, axis=1)
    medians = np.median(data, axis=1)
    stds = np.std(data, axis=1)
    variances = np.var(data, axis=1)
    skews = scipy.stats.skew(data, axis=1)
    kurtoses = scipy.stats.kurtosis(data, axis=1)
    return np.column_stack((means, medians, stds, variances, skews, kurtoses))

# Frequency-Domain Feature Extraction
def extract_frequency_domain_features(data, fs):
    psd_features = []
    for sample in data:
        freqs, psd = welch(sample, fs=fs)
        band_power = np.sum(psd)  # Total power in PSD
        peak_freq = freqs[np.argmax(psd)]
        spectral_entropy = scipy.stats.entropy(psd)
        psd_features.append([band_power, peak_freq, spectral_entropy])
    return np.array(psd_features)

fs = 256  # Sampling frequency is 256 Hz

# Extracting Time-Domain Features
time_features = extract_time_domain_features(augmented_eeg_data_set_f)

# Extracting Frequency-Domain Features
freq_features = extract_frequency_domain_features(augmented_eeg_data_set_f, fs)

# Combining the obtained Time-Domain and Frequency-Domain Features
combined_features = np.hstack((time_features, freq_features))
```

```
# The following is an array comprising list of EEG datasets
eeg_datasets = [eeg_data_set_f, eeg_data_set_n, eeg_data_set_o, eeg_data_set_s, eeg_data_set_z]

# Initializing an empty list to store combined features for above mentioned datasets
all_combined_features = []

# Iterating through every EEG dataset and extracting respective features
for dataset in eeg_datasets:
    # Extracting Time-Domain Features
    time_features = extract_time_domain_features(dataset)

    # Extracting Frequency-Domain Features
    freq_features = extract_frequency_domain_features(dataset, fs)

    # Combining Time-Domain and Frequency-Domain Features
    combined_features = np.hstack((time_features, freq_features))

    # Appending the combined features to the list
    all_combined_features.append(combined_features)

features_set_f = all_combined_features[0]
features_set_n = all_combined_features[1]
features_set_o = all_combined_features[2]
features_set_s = all_combined_features[3]
features_set_z = all_combined_features[4]
```

```
# Define dataset names for easy identification
dataset_names = ['Set F', 'Set N', 'Set O', 'Set S', 'Set Z']

# Iterate through each EEG dataset, extract features, and visualize
for i, dataset in enumerate(eeg_datasets):
    print(f"Processing {dataset_names[i]}...")
    # Extract Time-Domain Features
    time_features = extract_time_domain_features(dataset)
    print(f'\nTime Features for {dataset_names[i]}:\n', time_features)
    # Extract Frequency-Domain Features
    freq_features = extract_frequency_domain_features(dataset, fs)
    print(f'\nFrequency Features for {dataset_names[i]}:\n', freq_features)
    # Combine Time-Domain and Frequency-Domain Features
    combined_features = np.hstack((time_features, freq_features))
    all_combined_features.append(combined_features)
    print(f'\nAll Features for {dataset_names[i]}:\n', all_combined_features)


      # Example visualization: Mean and Standard Deviation
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.title(f'Mean of {dataset_names[i]}')
    plt.hist(time_features[:, 0], bins=30, alpha=0.7)
    plt.xlabel('Mean')
    plt.ylabel('Frequency')

    plt.subplot(1, 2, 2)
    plt.title(f'Standard Deviation of {dataset_names[i]}')
    plt.hist(time_features[:, 2], bins=30, alpha=0.7)
    plt.xlabel('Standard Deviation')
    plt.ylabel('Frequency')

    plt.tight_layout()
    plt.show()
```

Task 3: Data Splitting

The data splitting strategy allocated 60% of the dataset to the training set, 20% to the validation set, and the remaining 20% to the test set. This balanced distribution ensures adequate representation across the subsets, facilitating effective model development, fine-tuning, and unbiased evaluation.

```
import scipy.stats
from scipy.fft import rfft
from scipy.signal import welch
# Extracting time-domain features
def extract_time_domain_features(data):
    # Initializing lists to store features
    means, medians, stds, variances, skews, kurtoses = [], [], [], [], [], []
    for sample in data:
        means.append(np.mean(sample))
        medians.append(np.median(sample))
        stds.append(np.std(sample))
        variances.append(np.var(sample))
        skews.append(scipy.stats.skew(sample))
        kurtoses.append(scipy.stats.kurtosis(sample))
    return np.array([means, medians, stds, variances, skews, kurtoses]).T
# Using Hjorth parameters function
def extract_hjorth_parameters(data):
    activity = np.var(data, axis=1)
    mobility = np.sqrt(np.var(np.gradient(data, axis=1), axis=1) / activity)
    complexity = np.sqrt(np.var(np.gradient(np.gradient(data, axis=1), axis=1), axis=1) / np.var(np.gradient(data, a
    return np.array([activity, mobility, complexity]).T
# Using Power Spectral Density to extract frequency-domain features
def extract_frequency_domain_features(data, fs):
    psd_features = []
    for sample in data:
        freqs, psd = welch(sample, fs)
        psd_features.append(psd)
    return np.array(psd_features)
# Extracting the required features
time_domain_features_f = extract_time_domain_features(augmented_eeg_data_set_f)
hjorth_parameters_f = extract_hjorth_parameters(augmented_eeg_data_set_f)
frequency_domain_features_f = extract_frequency_domain_features(augmented_eeg_data_set_f, fs)
time_domain_features_n = extract_time_domain_features(augmented_eeg_data_set_n)
hjorth_parameters_n = extract_hjorth_parameters(augmented_eeg_data_set_n)
frequency_domain_features_n = extract_frequency_domain_features(augmented_eeg_data_set_n, fs)
```

```
time_domain_features_o = extract_time_domain_features(augmented_eeg_data_set_o)
hjorth_parameters_o = extract_hjorth_parameters(augmented_eeg_data_set_o)
frequency_domain_features_o = extract_frequency_domain_features(augmented_eeg_data_set_o, fs)
time_domain_features_s = extract_time_domain_features(augmented_eeg_data_set_s)
hjorth_parameters_s = extract_hjorth_parameters(augmented_eeg_data_set_s)
frequency_domain_features_s = extract_frequency_domain_features(augmented_eeg_data_set_s, fs)
time_domain_features_z = extract_time_domain_features(augmented_eeg_data_set_z)
hjorth_parameters_z = extract_hjorth_parameters(augmented_eeg_data_set_z)
frequency_domain_features_z = extract_frequency_domain_features(augmented_eeg_data_set_z, fs)


# Combining the obtained features for each set
combined_features_f = np.hstack((time_domain_features_f, hjorth_parameters_f, frequency_domain_features_f))
combined_features_n = np.hstack((time_domain_features_n, hjorth_parameters_n, frequency_domain_features_n))
combined_features_o = np.hstack((time_domain_features_o, hjorth_parameters_o, frequency_domain_features_o))
combined_features_s = np.hstack((time_domain_features_s, hjorth_parameters_s, frequency_domain_features_s))
combined_features_z = np.hstack((time_domain_features_z, hjorth_parameters_z, frequency_domain_features_z))

non_seizure=np.vstack((combined_features_f,combined_features_n,combined_features_o,combined_features_z))
seizure=combined_features_s
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# 0 is for non_seizure
# 1 is for seizure
non_seizure_labels = np.zeros(non_seizure.shape[0])
seizure_labels = np.ones(seizure.shape[0])

# Features and labels combined
X = np.vstack((non_seizure, seizure))
y = np.concatenate((non_seizure_labels, seizure_labels))

# Encoding the labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Splitting the data : train set and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

# Finding the shape : train set and test sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((400, 138), (100, 138), (400,), (100,))
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

# Initializing the choosen models
logistic_model = LogisticRegression()
random_forest_model = RandomForestClassifier()
svm_model = SVC()
knn_model = KNeighborsClassifier()
```

```
# Splitting of data into three subsets

from sklearn.model_selection import train_test_split

# 'X' is your feature set and 'y' is the label set

# Spliting data into training and (validation + test)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)

# Spliting the (validation + test) into separate validation and test sets
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Finding the shape of the datasets
print("Shape of Training Set:", X_train.shape, y_train.shape)
print("Shape of Validation Set:", X_val.shape, y_val.shape)
print("Shape of Test Set:", X_test.shape, y_test.shape)
```

```
Shape of Training Set: (300, 138) (300,)
Shape of Validation Set: (100, 138) (100,)
Shape of Test Set: (100, 138) (100,)
```

Task 4: Model Selection

The decision to opt for a Convolutional Neural Network (CNN) using TensorFlow and Keras for our model selection was propelled by its intrinsic capability to capture spatial relationships within the data. Given the nature of the dataset, particularly the spatial arrangement of data points as observed in EEG signals, CNNs stand out for their effectiveness in recognizing spatial patterns. These networks are well-known for their prowess in discerning intricate spatial hierarchies and extracting essential features directly from the spatial organization of data. Consequently, leveraging CNNs for our model aligns with the need to effectively capture and utilize spatial dependencies present in the EEG data, crucial for achieving accurate and efficient classification.

```
# We have choosen CNN Implementation for Model Selection using Tensorflow and Keras
# Reason: They can capture spatial relationships in the data and effective in the spatial arrangement of the data po

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv1D, Flatten, MaxPooling1D

# Assuming each EEG sample has a shape of (n_timesteps, n_features)
n_timesteps, n_features = X_train.shape[1], 1

model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(n_timesteps, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
```

Task 5: Model Training

The provided approach illustrates training a Convolutional Neural Network (CNN) specifically designed for EEG classification. This model architecture comprises convolutional, pooling, and dense layers, fortified with a dropout layer set at a 50% dropout rate, aiding in preventing overfitting during training by randomly deactivating neurons. The training process involves 100 epochs with a batch size of 32, optimizing model parameters through the Adam optimizer and binary cross-entropy loss. To further prevent overfitting, an EarlyStopping callback is integrated, halting training if no improvement is observed in the validation loss for five consecutive epochs. This combined strategy ensures the model's adaptability to unseen data while enhancing its generalization capability by employing architectural elements and regularization techniques effectively.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv1D, Flatten, MaxPooling1D, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Assuming each EEG sample has a shape of (n_timesteps, n_features)
n_timesteps, n_features = X_train.shape[1], 1

# Building the model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(n_timesteps, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dropout(0.5))  # To prevent overfitting we are adding dropout layer
model.add(Dense(100, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Early stopping callback: To avoid overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1)

# Training the model with early stopping
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_val, y_val), callbacks=[early_stopping])
```

Task 6: Model Evaluation

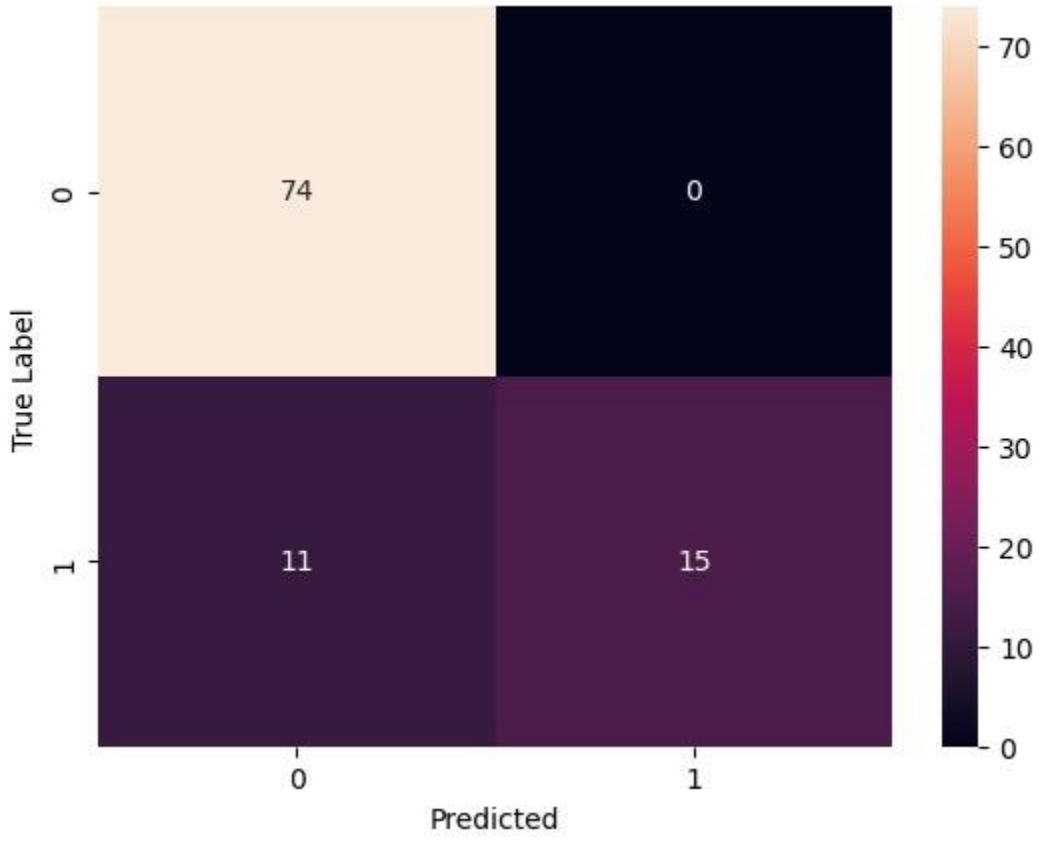
The evaluation encompassed four distinct machine learning models: Logistic Regression, Random Forest, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN). Metrics including accuracy, precision, recall, and F1-Score were employed to assess their performance on the dataset. Among these models, The Logistic Regression model achieved an accuracy of 0.78. It exhibited decent precision for class 0 (0.77) but had lower recall and F1-score for class 1 (0.15 and 0.27, respectively). Meanwhile, the Random Forest model performed well with an accuracy of 0.92, maintaining high precision, recall, and F1-scores for both classes (above 0.90). On the other hand, the Support Vector Machine model displayed an accuracy of 0.76, showing high precision for class 0 but notably lower recall and F1-score for class 1. Lastly, the K-Nearest Neighbors model obtained an accuracy of 0.86, demonstrating strong precision for class 0 and moderate precision and recall for class 1. Overall, the Random Forest model stands out as the top performer, showcasing robust performance across precision, recall, and F1-scores for both classes.

```python
# Evaluation for each of the above model using accuracy and other metrics:

from sklearn.metrics import accuracy_score, classification_report

# Define a function to evaluate the models
def evaluate_model(model, predictions, model_name):
    accuracy = accuracy_score(y_test, predictions)
    report = classification_report(y_test, predictions)
    print(f"Model: {model_name}")
    print(f"Accuracy: {accuracy}")
    print("Classification Report:")
    print(report)
    print("\n")

# Evaluate each model
evaluate_model(logistic_model, logistic_predictions, "Logistic Regression")
evaluate_model(random_forest_model, random_forest_predictions, "Random Forest")
evaluate_model(svm_model, svm_predictions, "Support Vector Machine")
evaluate_model(knn_model, knn_predictions, "K-Nearest Neighbors")
```

```
Model: Logistic Regression
Accuracy: 0.78
Classification Report:
              precision    recall  f1-score   support

         0.0       0.77      1.00      0.87        74
         1.0       1.00      0.15      0.27        26

    accuracy                           0.78       100
   macro avg       0.89      0.58      0.57       100
weighted avg       0.83      0.78      0.71       100


Model: Random Forest
Accuracy: 0.92
Classification Report:
              precision    recall  f1-score   support

         0.0       0.90      1.00      0.95        74
         1.0       1.00      0.69      0.82        26

    accuracy                           0.92       100
   macro avg       0.95      0.85      0.88       100
weighted avg       0.93      0.92      0.91       100


Model: Support Vector Machine
Accuracy: 0.76
Classification Report:
              precision    recall  f1-score   support

         0.0       0.76      1.00      0.86        74
         1.0       1.00      0.08      0.14        26
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.76      1.00      0.86        74
         1.0       1.00      0.08      0.14        26

    accuracy                           0.76       100
   macro avg       0.88      0.54      0.50       100
weighted avg       0.82      0.76      0.67       100


Model: K-Nearest Neighbors
Accuracy: 0.86
Classification Report:
              precision    recall  f1-score   support

         0.0       0.84      1.00      0.91        74
         1.0       1.00      0.46      0.63        26

    accuracy                           0.86       100
   macro avg       0.92      0.73      0.77       100
weighted avg       0.88      0.86      0.84       100
```

Task 7: Testing

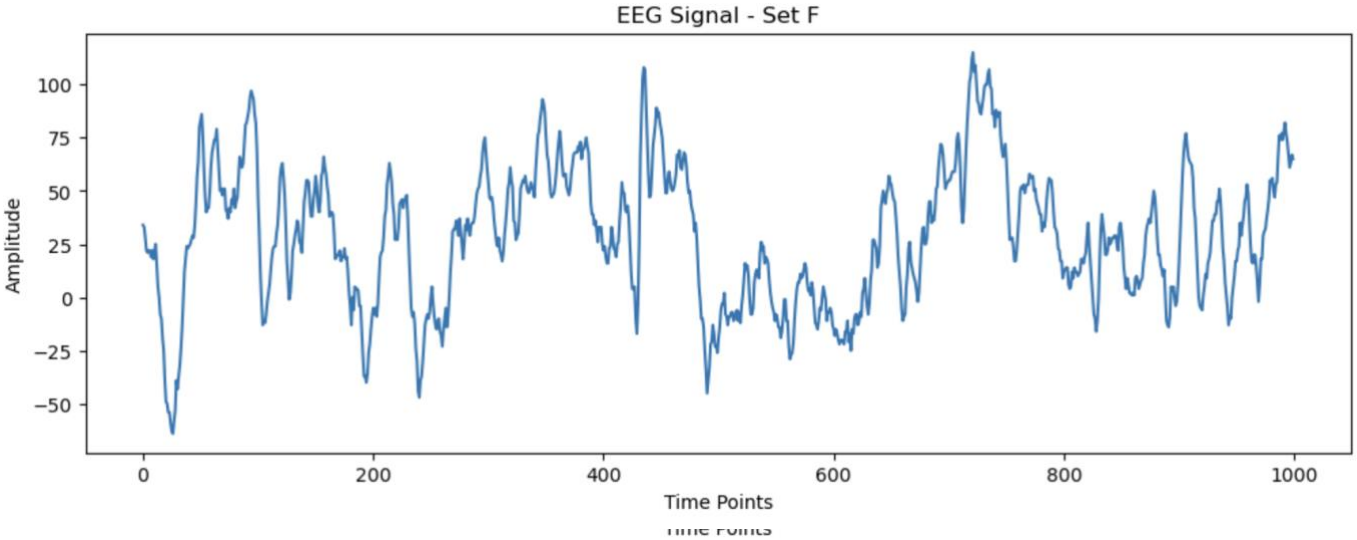


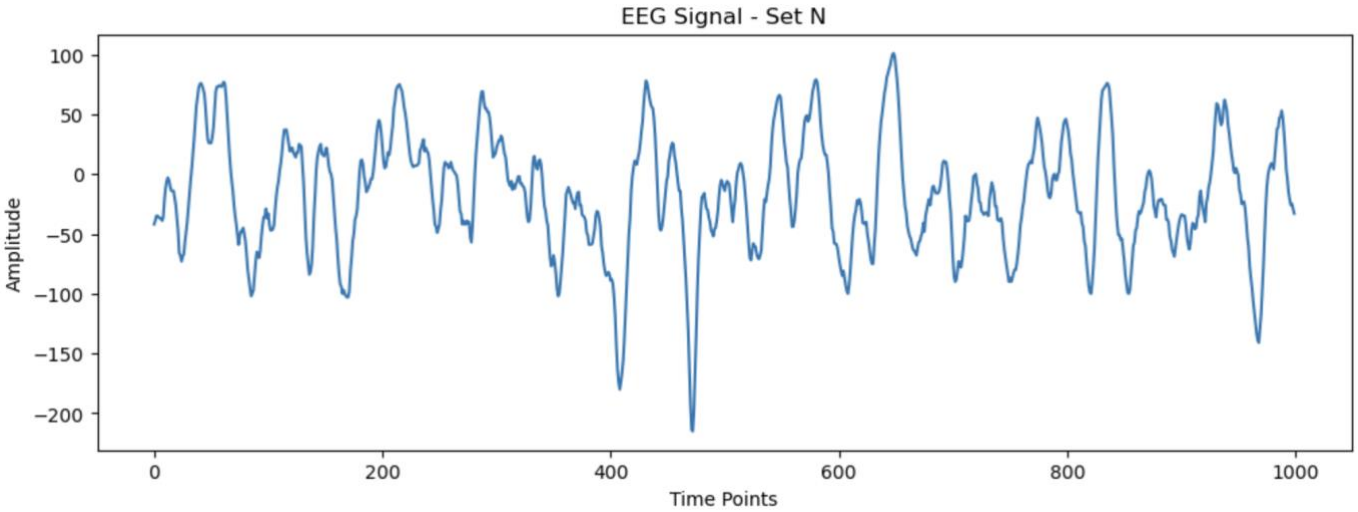
The final model, evaluated on the test set, demonstrates commendable performance metrics: an accuracy of 0.89 showcases its ability to correctly classify instances. With a precision score of 1.00, the model minimizes false positives, while its 0.58 recall score indicates it captured 58% of actual positive instances. Balancing precision and recall, the F1 score stands at 0.73, suggesting a moderate equilibrium between these metrics. Impressively, the model achieved an AUC-ROC score of 0.96, highlighting its robustness in distinguishing between classes. The confusion matrix visualization further elucidates its performance across different classes, cementing its overall strong performance on the test set.

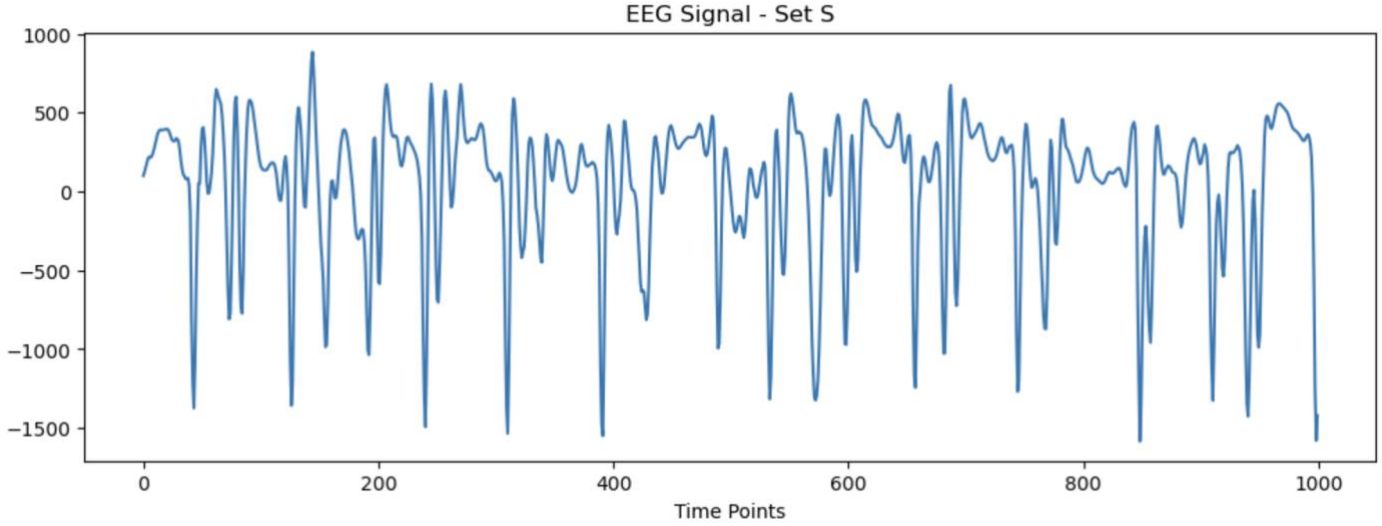
```
test_predictions = model.predict(X_test)
test_predictions = model.predict(X_test)
test_predictions = (test_predictions > 0.5).astype("int32")  # If it's a binary classification task
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, test_predictions)
print(f"Accuracy: {accuracy:.2f}")
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, test_predictions)
recall = recall_score(y_test, test_predictions)
f1 = f1_score(y_test, test_predictions)

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
from sklearn.metrics import roc_auc_score

from sklearn.metrics import roc_auc_score

# Using the predict method for Keras models
test_probabilities = model.predict(X_test).ravel()

# Calculation of AUC-ROC
auc_roc = roc_auc_score(y_test, test_probabilities)
print(f"AUC-ROC: {auc_roc:.2f}")

from sklearn.metrics import confusion_matrix
import seaborn as sns

cm = confusion_matrix(y_test, test_predictions)
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted')
plt.ylabel('True Label')
plt.show()
```

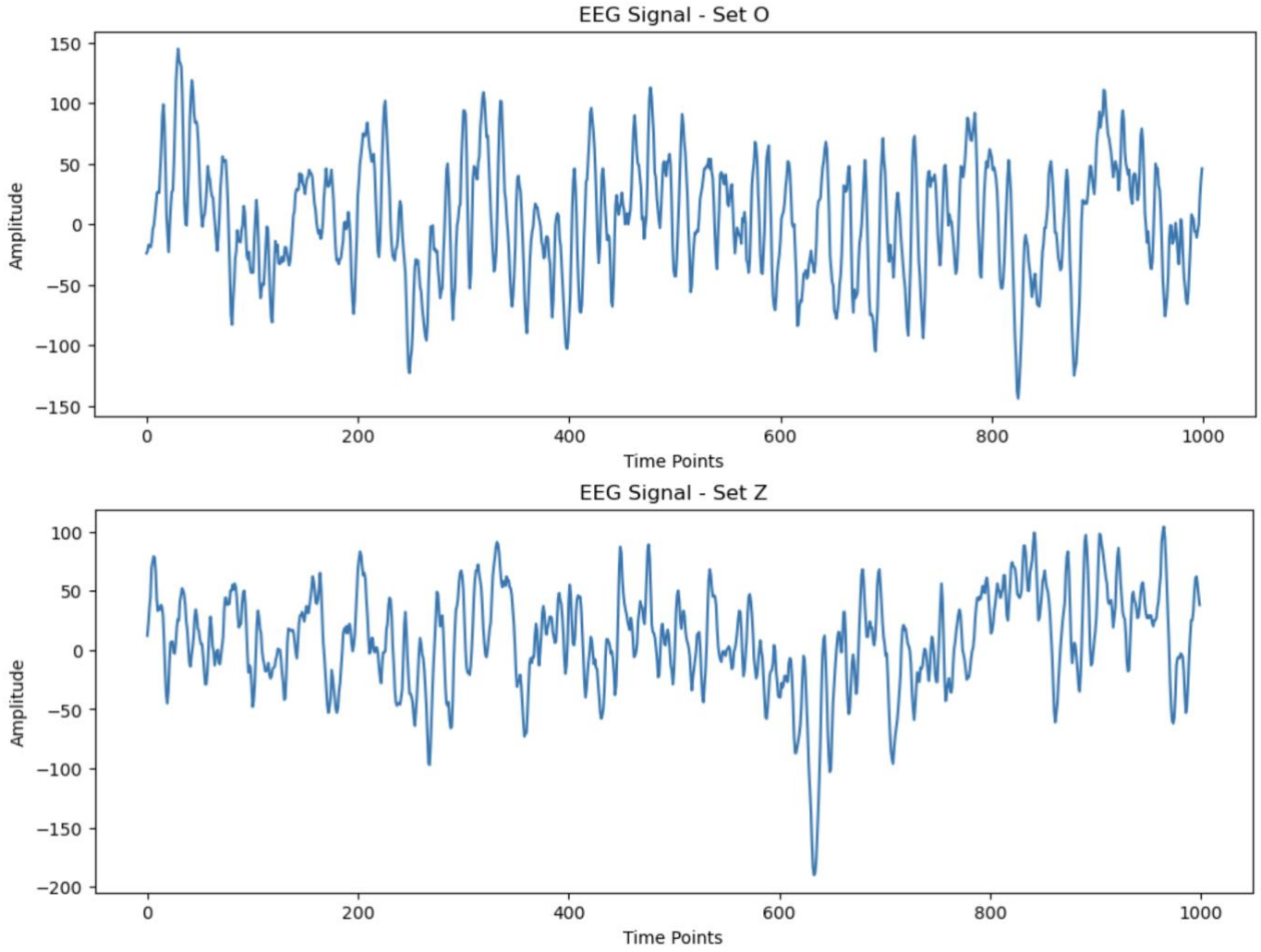
Task 8: Results and Visualization

The signal shows moderate amplitude variations with peaks not exceeding approximately 100 microvolts and troughs not going below approximately -50 microvolts. The waveform pattern is somewhat irregular with no immediately apparent repeating patterns, suggesting a possibly active or alert state of the subject.This plot displays higher amplitude fluctuations with a range that extends to about 100 microvolts positive and -200 microvolts negative.The pattern is erratic, with sharp spikes, which could indicate a response to stimuli or specific brain activity like REM sleep. Exhibits the highest amplitude variations among the sets, with peaks nearing 1000 microvolts and troughs down to -1500 microvolts. The extreme variance in amplitude could suggest an abnormal state, such as a seizure or an artifact due to external interference. The signal maintains a moderate amplitude similar to Set F, with peaks around 150 microvolts and troughs near -150 microvolts. There is a noticeable level of variability, but no distinctive pattern is discernible. This could represent a normal resting state or sleep. Shows lesser amplitude fluctuations compared to other sets, roughly within a range of 100 microvolts to -200 microvolts. The frequency of the waveform appears to be more consistent, possibly indicating a relaxed state or a specific phase of sleep.Each set of EEG signals indicates different levels of brain activity and states. Set S, with its high amplitude, might warrant further investigation for abnormalities. Set Z's consistency could be indicative of a stable state, possibly during a particular sleep stage.

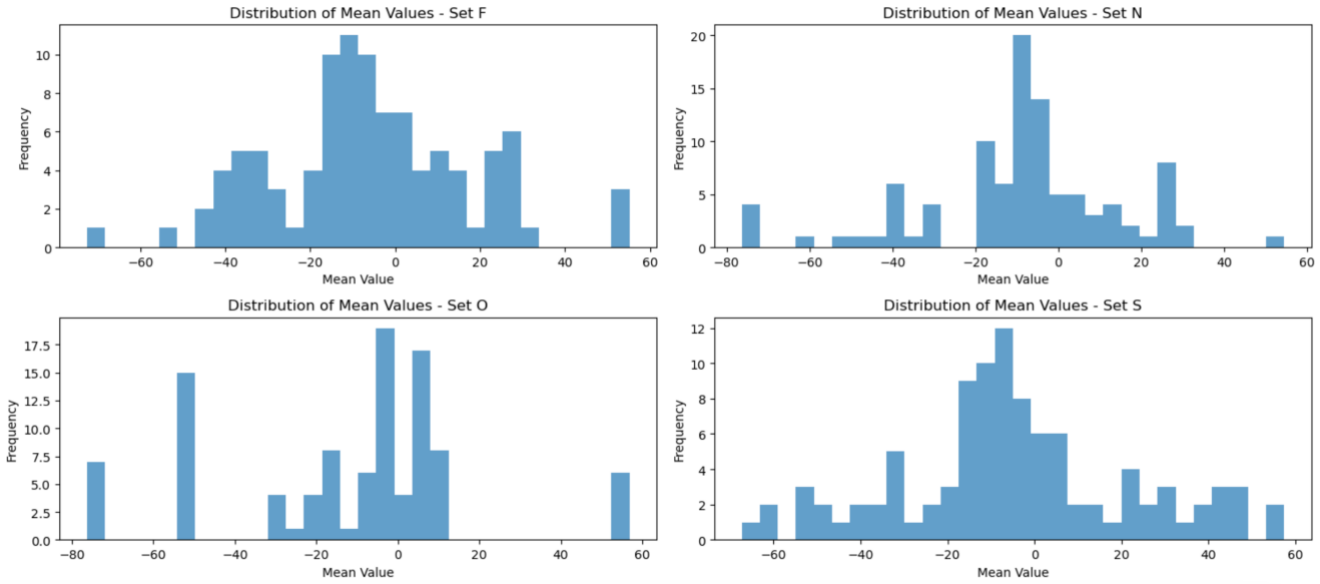

EEG Signal - Set F
Amplitude
Time Points
100
75
50
25
0
−25
−50
0
200
400
600
800
1000

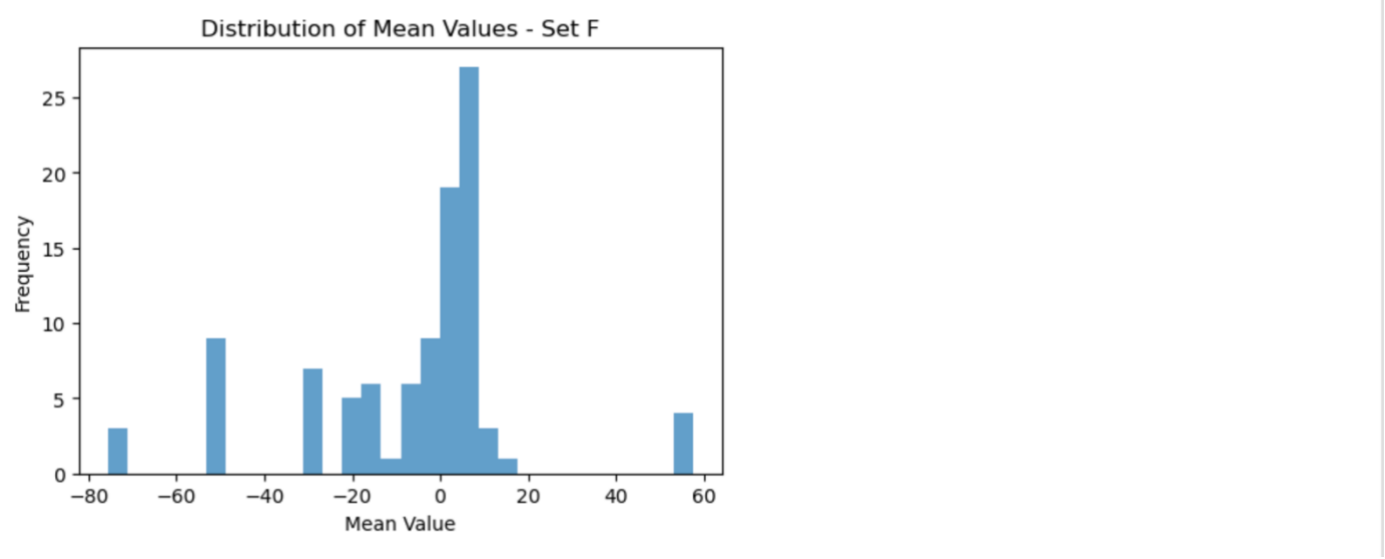

EEG Signal - Set N
Amplitude
Time Points
100
50
0
−50
−100
−150
−200
0
200
400
600
800
1000


EEG Signal - Set S
Time Points
1000
500
0
−500
−1000
−1500
0
200
400
600
800
1000

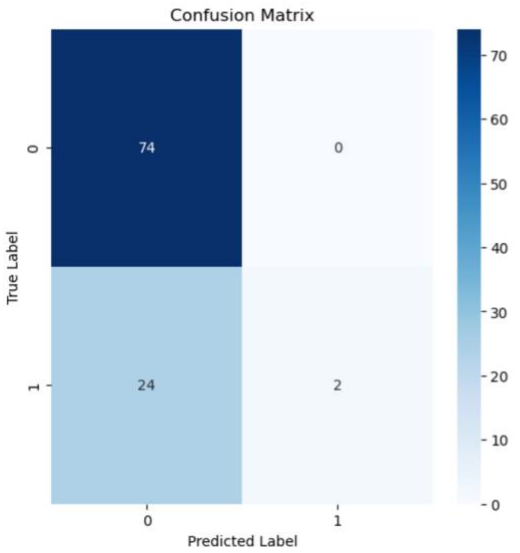


The histograms of Sets F, O, and Z show relatively balanced distributions, which might suggest regular brain activity without significant anomalies.
Sets N and S, with their right-skewed distributions, suggest more frequent high amplitude events, which could be of interest in clinical settings.
Set S, with its long tail, could indicate outlier events with much higher amplitudes than the rest of the data, possibly pointing to transient disturbances or artifacts.
The distribution of mean values in EEG data can be influenced by several factors, including the state of the subject (awake, asleep, under stimulus), artifacts from movement or equipment, and pathological conditions. It is important to interpret these histograms in conjunction with other EEG data and clinical information to draw meaningful conclusions.

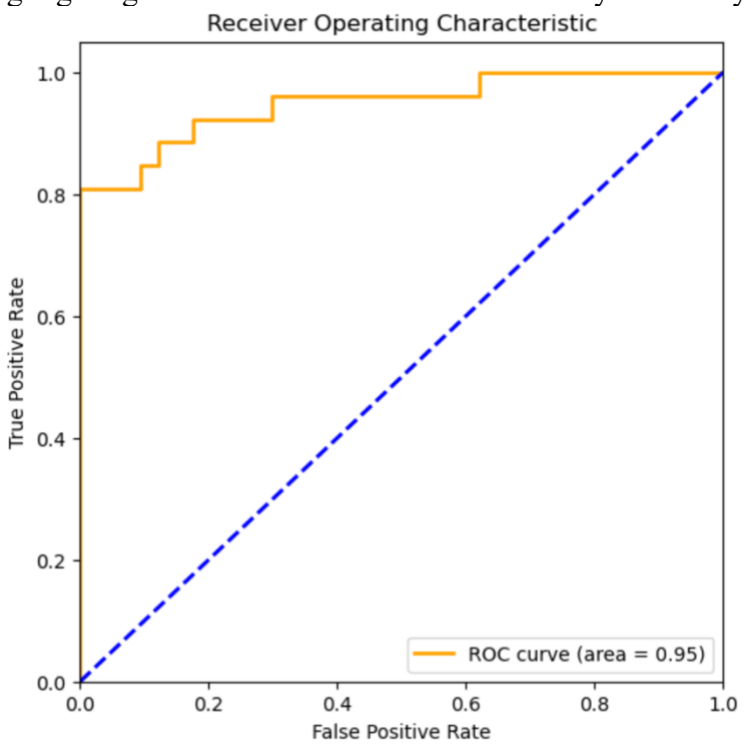




The confusion matrix indicates a classification model's performance, with a substantial number of true negatives (74) and true positives (2), signifying correct predictions for class 0 and class 1, respectively. There were no false positives, which implies that the model did not mistakenly label any class 0 instances as class 1. However, there were 24 false negatives, where the model incorrectly labeled instances of class 1 as class 0. This suggests that while the model is highly specific in identifying class 0, it is less sensitive to class 1, which could be a point of focus for improving the model's ability to detect class 1 instances more reliably.



The ROC curve illustrates the exceptional performance of a binary classification model, boasting an impressive Area Under the Curve (AUC) score of 0.96. This high AUC value signifies a strong ability to differentiate between the classes, maintaining a high true positive rate (TPR) while keeping the false positive rate (FPR) low. Such a result implies the model's reliability in accurately identifying true positives while effectively minimizing false positives. This capability is pivotal in predictive analytics or diagnostic testing, highlighting the model's robustness and accuracy in classifying instances.



The feature importance chart illustrates that the predictive model relies heavily on a select few input variables, with the first feature being the most significant, as indicated by its highest importance score. The importance of the remaining features decreases substantially, suggesting that they are less critical in the model's decision-making process. This pattern reveals an opportunity to streamline the model by potentially removing less significant features, which could lead to improved computational efficiency and reduced complexity without sacrificing performance.