Development of a Digital Biomarker using a Neural Network Model to Differentiate Between Long and Short EEG Blinks

**Author:** Hsaine El-Ali

**Date:** 08 Jan 2024

Contents

[1. Abstract: 3](#_Toc155530405)

[2. Introduction 3](#_Toc155530406)

[The Emergence of EEG in Biomedical Applications 3](#_Toc155530407)

[The Challenge of Blink Detection in EEG 3](#_Toc155530408)

[Project Objective 3](#_Toc155530409)

[3. Data Preprocessing 4](#_Toc155530410)

[Initial Data Handling 4](#_Toc155530411)

[Noise Filtering 4](#_Toc155530412)

[4. Feature Extraction 5](#_Toc155530413)

[Time-Domain Features 5](#_Toc155530414)

[Frequency-Domain Features 5](#_Toc155530415)

[5. Model Development 6](#_Toc155530416)

[Introduction 6](#_Toc155530417)

[Data Preparation for Training 6](#_Toc155530418)

[Model Description and Training 7](#_Toc155530419)

[Performance on Test Data 8](#_Toc155530420)

[Results and Analysis 8](#_Toc155530421)

[6. Decision Tree Classifier 8](#_Toc155530422)

[7. Conclusion 10](#_Toc155530423)

[Achievements and Implications 10](#_Toc155530424)

[Advancements in EEG Analysis 10](#_Toc155530425)

# Abstract:

This report documents the process and outcomes of a project aimed at distinguishing between long and short blinks using EEG data. The project involved data preprocessing, feature extraction, model building, and evaluation. The final model, a neural network, demonstrated high accuracy on new, unseen data, indicating its potential for real-world applications.

# Introduction

## The Emergence of EEG in Biomedical Applications

Electroencephalography (EEG) is a cornerstone in the field of biomedical research and diagnostics. This non-invasive technique captures electrical activity of the brain, offering invaluable insights into neurological functions and disorders. Its applications span from clinical diagnostics, particularly in epilepsy and sleep disorders, to advanced research in cognitive neuroscience and brain-computer interfaces.

EEG data, characterized by its complex and dynamic nature, encodes rich information about brain activity. The analysis of these signals provides a window into the brain's functioning, allowing researchers and clinicians to interpret various cognitive states and neural responses.

## The Challenge of Blink Detection in EEG

Among the myriad of phenomena observable through EEG, the distinction between different types of eye blinks stands out as a subtle yet significant challenge. Eye blinks, particularly long and short blinks, are not only common artifacts in EEG recordings but also carry potential diagnostic value. Long blinks, for instance, might be associated with certain neurological conditions or fatigue levels.

## Project Objective

This project aims to develop a computational model based on a digital biomarker extracted from EEG data, to differentiate effectively between long and short blink.

# Data Preprocessing

## Initial Data Handling

The EEG data provided comprised two primary datasets: long blinks and short blinks. Each dataset was a collection of EEG signals across multiple channels. The initial step involved flattening the raw data and segmenting it into sessions, each containing a sequence of 510 data points. This segmentation was vital in order to properly display each channel.

A screenshot of a graph

Description automatically generatedA screenshot of a graph

Description automatically generated

## Noise Filtering

A critical preprocessing step was the application of a bandpass filter. With a lower frequency of 0.1 Hz and a higher frequency of 5.0 Hz, this filter was instrumental in removing noise and irrelevant frequency components from the EEG signals. It ensured the isolation of frequencies most relevant to blink activities, enhancing signal quality for subsequent analysis.

A line graph of a graph

Description automatically generated with medium confidenceA screenshot of a graph

Description automatically generated

# **Feature Extraction**

## Time-Domain Features

In the time domain, three features were extracted from each session:

1. Peak-to-Peak Amplitude: This provided insights into the signal's overall range, crucial for distinguishing blink types.
2. Mean: It offered a measure of the central tendency of the EEG signal values.
3. Variance: This feature represented the spread of the EEG signal values.

## Frequency-Domain Features

Frequency-domain analysis was performed using Welch’s method to compute the Power Spectral Density (PSD). From the PSD, two features were derived:

1. Dominant Frequency: The frequency with the maximum power in the PSD.
2. Bandwidth: The range of frequencies where the signal’s power was above half of its maximum.

All extracted features per channel were summed up and an average feature function was computed.

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

# Model Development

## Introduction

In this project, we developed a machine learning model to differentiate between long and short blinks in electroencephalogram (EEG) data. The approach involved data preprocessing, feature extraction, and the use of a neural network model. Key to the model's success was ensuring that the training and testing data were handled correctly to avoid label leakage and ensure an unbiased evaluation of the model's performance.

## Data Preparation for Training

1. **Merging and Labeling Data:**
   * We combined features from both long and short blink data for each of the four EEG channels (**Channel\_1** to **Channel\_4**). This was done using **np.vstack** to vertically stack the features of each channel type, ensuring data from both blink types were included.
   * Labels were created with **0** representing short blinks and **1** for long blinks. These labels corresponded to the stacked feature data of each blink type.
2. **Splitting the Dataset:**
   * The combined feature set and labels were divided into training and testing datasets using **train\_test\_split** with a 70/30 split. This ensured that 30% of the data was held out for testing the model's performance on unseen data.
   * The labels for the training set were converted to one-hot encoding format using **to\_categorical**, a necessary step for classification in neural networks.

## Model Description and Training

1. **Building the Neural Network:**
   * A Sequential model was constructed with dense layers. The input layer had 64 neurons, and subsequent layers (including hidden layers) were added with 32 neurons each.
   * To mitigate overfitting, dropout layers with a dropout rate of 0.5 were introduced. These layers randomly set input units to 0 at each step during training, which helps prevent overfitting.
   * The final layer was a dense layer with 2 neurons corresponding to our two classes (short and long blinks), using a 'softmax' activation function for classification.
   * The model was compiled with the 'adam' optimizer and 'categorical\_crossentropy' as the loss function.
2. **Cross-Validation and Training:**
   * Stratified K-Fold cross-validation with 5 splits was employed to validate the model's effectiveness. This approach ensured that each fold was a good representative of the whole dataset.
   * For each fold, the model was trained and validated, allowing us to observe the variation in model performance across different subsets of the training data.
3. **Model Evaluation and Saving:**
   * After training, the model's performance was evaluated on the test set. The accuracy, confusion matrix, and a classification report were generated, providing insights into the model's predictive capabilities on unseen data.
   * Finally, the trained model was saved to a specified path for future use in making predictions on new data.
4. **Model Evaluation and Results**

## Performance on Test Data

The model was evaluated on a separate test set, accounting for 30% of the total data. This approach ensured unbiased evaluation on unseen data.

## Results and Analysis

The model achieved a remarkable 96.67% accuracy, with the confusion matrix showing only one misclassification in the test set. The classification report indicated high precision, recall, and F1-scores for both classes, demonstrating the model's capability to effectively distinguish A screenshot of a computer

Description automatically generated

# Decision Tree Classifier

A Decision Tree Classifier was implemented using Scikit-learn's **DecisionTreeClassifier** class. The classifier was trained on a set of labeled features extracted from the EEG data. The features for both long and short blinks were combined and used as input variables, while the labels were encoded as binary outputs (0 for short blinks, 1 for long blinks).

The training process involved splitting the data into training and test sets, with a 70-30 split ratio, ensuring that 30% of the data was reserved for model evaluation. The classifier was then fitted to the training data, learning to associate the input features with the correct blink type.

**Model Evaluation**

The performance of the trained Decision Tree Classifier was evaluated using the reserved test set. The evaluation metrics included:

* Accuracy: The proportion of correctly predicted instances out of all predictions.
* Confusion Matrix: A table used to describe the performance of a classification model on a set of test data for which the true values are known. It allows visualization of the model's ability to correctly or incorrectly classify instances.
* Classification Report: A summary of the precision, recall, and F1-score for each class.

The model achieved an accuracy of 96.67% on the test data, indicating a high level of predictive performance. The confusion matrix and classification report provided further insight into the model's classification capabilities for each blink type.

**A screen shot of a computer

Description automatically generated**

# Conclusion

## Achievements and Implications

This project successfully demonstrated the feasibility of using machine learning algorithms to differentiate between long and short blinks in EEG data. Through meticulous data preprocessing, feature extraction, and the development of a robust neural network model, we have paved the way for more accurate EEG analysis in both clinical and research settings.

## Advancements in EEG Analysis

The ability to distinguish between various types of blinks represents a significant advancement in EEG data interpretation. This capability enhances the accuracy of neurological assessments and potentially aids in diagnosing specific conditions.

**8. References**

1. Niedermeyer, E., & da Silva, F. L. (Eds.). (2005). *Electroencephalography: Basic Principles, Clinical Applications, and Related Fields*. Lippincott Williams & Wilkins.
2. Teplan, M. (2002). *Fundamentals of EEG measurement*. Measurement Science Review, 2(2), 1-11.
3. Sanei, S., & Chambers, J. A. (2013). *EEG Signal Processing*. John Wiley & Sons.
4. Shambroom, J. R., Fábregas, S. E., & Johnstone, J. (2012). *Validation of an automated wireless system to monitor sleep in healthy adults*. Journal of Sleep Research, 21(2), 221-230.
5. Fatourechi, M., Bashashati, A., Ward, R. K., & Birch, G. E. (2007). *EMG and EOG artifacts in brain computer interface systems: A survey*. Clinical Neurophysiology, 118(3), 480-494.
6. Croft, R. J., & Barry, R. J. (2000). *Removal of ocular artifact from the EEG: A review*. Neurophysiologie Clinique/Clinical Neurophysiology, 30(1), 5-19.
7. Schlögl, A., Keinrath, C., Zimmermann, D., Scherer, R., Leeb, R., & Pfurtscheller, G. (2007). *A fully automated correction method of EOG artifacts in EEG recordings*. Clinical Neurophysiology, 118(1), 98-104.
8. Lindsley, D. B. (1939). *A longitudinal study of the occipital alpha rhythm in normal children: Frequency and amplitude standards*. Journal of Genetic Psychology, 55(1), 197-213.

**9. Appendix**

*Code:*

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from tabulate import tabulate

import ast  # For safely evaluating the string

import mne

from scipy.signal import find\_peaks

from scipy.signal import welch

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from keras.models import Sequential

from keras.layers import Dense, Dropout

from keras.utils import to\_categorical

from keras.callbacks import History

from sklearn.model\_selection import StratifiedKFold

from keras.regularizers import l2

plt.ion()  # Turn on interactive mode

# Load the CSV files

long\_blink\_data = pd.read\_csv('C:/dev/digitalbiomarkers/EEG-data/LongBlink.csv')

short\_blink\_data = pd.read\_csv('C:/dev/digitalbiomarkers/EEG-data/ShortBlink.csv')

# Display data tables

print("\nTable for LongBlink.csv:")

print(tabulate(long\_blink\_data.head(), headers='keys', tablefmt='grid'))

print("\nTable for ShortBlink.csv:")

print(tabulate(short\_blink\_data.head(), headers='keys', tablefmt='grid'))

# Convert string representations to actual lists

long\_blink\_data['data'] = long\_blink\_data['data'].apply(ast.literal\_eval)

short\_blink\_data['data'] = short\_blink\_data['data'].apply(ast.literal\_eval)

# Flatten the data

long\_blink\_values = [value for sublist in long\_blink\_data['data'].tolist() for value in sublist]

short\_blink\_values = [value for sublist in short\_blink\_data['data'].tolist() for value in sublist]

# Chunk the data

long\_blink\_sessions = [long\_blink\_values[i:i + 510] for i in range(0, len(long\_blink\_values), 510)]

short\_blink\_sessions = [short\_blink\_values[i:i + 510] for i in range(0, len(short\_blink\_values), 510)]

# Convert to arrays

long\_blink\_array = np.array(long\_blink\_sessions)

short\_blink\_array = np.array(short\_blink\_sessions)

# Apply bandpass filter

def apply\_bandpass\_filter(data, l\_freq=0.1, h\_freq=5.0):

    sfreq = 215.0

    data\_filtered = mne.filter.filter\_data(data, sfreq, l\_freq, h\_freq, method='iir', verbose=False)

    return data\_filtered

long\_blink\_array\_filtered = apply\_bandpass\_filter(long\_blink\_array)

short\_blink\_array\_filtered = apply\_bandpass\_filter(short\_blink\_array)

# Plot sessions function

def plot\_linked\_sessions(data\_array, title):

    fig, axs = plt.subplots(4, 1, figsize=(8, 12))

    for channel\_idx in range(4):

        blink\_data = np.concatenate([data\_array[session\_idx \* 4 + channel\_idx] for session\_idx in range(20)])

        axs[channel\_idx].plot(blink\_data)

        axs[channel\_idx].set\_title(f"Channel {channel\_idx + 1}")

        axs[channel\_idx].set\_xticks([])

    plt.tight\_layout()

    plt.subplots\_adjust(top=0.95)

    fig.suptitle(title)

    plt.draw()

# Plot the data

plot\_linked\_sessions(long\_blink\_array, "EEG Data Linked Sessions - Long Blink (Non-filtered)")

plot\_linked\_sessions(short\_blink\_array, "EEG Data Linked Sessions - Short Blink (Non-filtered)")

plot\_linked\_sessions(long\_blink\_array\_filtered, "EEG Data Linked Sessions - Long Blink (Filtered)")

plot\_linked\_sessions(short\_blink\_array\_filtered, "EEG Data Linked Sessions - Short Blink (Filtered)")

plt.show(block=True)

# Feature Extraction function

def extract\_features\_per\_channel(data\_array):

    # Initialize feature lists

    features\_per\_channel = {f"Channel\_{i+1}": [] for i in range(4)}

    for session\_idx in range(len(data\_array) // 4):  # 4 channels per session

        for channel\_idx in range(4):

            session = data\_array[session\_idx \* 4 + channel\_idx]

            # Time domain features

            # 1. Peak-to-peak amplitude

            positive\_peaks, \_ = find\_peaks(session)

            negative\_peaks, \_ = find\_peaks(-session)

            if positive\_peaks.size and negative\_peaks.size:

                max\_peak = max(session[positive\_peaks])

                min\_peak = min(session[negative\_peaks])

                peak\_to\_peak\_amplitude = max\_peak - min\_peak

            else:

                peak\_to\_peak\_amplitude = 0

            # 2. Mean

            mean\_val = np.mean(session)

            # 3. Variance

            variance\_val = np.var(session)

            # Frequency domain features

            # Compute the Power Spectral Density

            f, Pxx = welch(session, fs=250.0, nperseg=256)

            # 4. Dominant frequency

            dominant\_frequency = f[np.argmax(Pxx)]

            # 5. Bandwidth

            half\_power = np.max(Pxx) / 2

            indices = np.where(Pxx > half\_power)

            first\_index = indices[0][0]

            last\_index = indices[0][-1]

            bandwidth = f[last\_index] - f[first\_index]

            # Append features to the channel's feature list

            features\_per\_channel[f"Channel\_{channel\_idx+1}"].append([peak\_to\_peak\_amplitude, mean\_val, variance\_val, dominant\_frequency, bandwidth])

    # Convert lists to arrays

    for key in features\_per\_channel:

        features\_per\_channel[key] = np.array(features\_per\_channel[key])

    return features\_per\_channel

long\_blink\_features\_per\_channel = extract\_features\_per\_channel(long\_blink\_array\_filtered)

short\_blink\_features\_per\_channel = extract\_features\_per\_channel(short\_blink\_array\_filtered)

# Compute average features function

def compute\_average\_features\_per\_channel(features\_dict):

    average\_features = {}

    for key in features\_dict:

        avg\_peak\_to\_peak\_amplitude = np.mean(features\_dict[key][:, 0])

        avg\_mean\_val = np.mean(features\_dict[key][:, 1])

        avg\_variance\_val = np.mean(features\_dict[key][:, 2])

        avg\_dominant\_frequency = np.mean(features\_dict[key][:, 3])

        avg\_bandwidth = np.mean(features\_dict[key][:, 4])

        average\_features[key] = [avg\_peak\_to\_peak\_amplitude, avg\_mean\_val, avg\_variance\_val, avg\_dominant\_frequency, avg\_bandwidth]

    return average\_features

# Calculate average features for long and short blinks

long\_blink\_avg\_features\_per\_channel = compute\_average\_features\_per\_channel(long\_blink\_features\_per\_channel)

short\_blink\_avg\_features\_per\_channel = compute\_average\_features\_per\_channel(short\_blink\_features\_per\_channel)

# Print results

print("\nAverage Features for Long Blink:")

for key in long\_blink\_avg\_features\_per\_channel:

    print(f"{key}:")

    print(f"Peak-to-Peak Amplitude: {long\_blink\_avg\_features\_per\_channel[key][0]}")

    print(f"Mean: {long\_blink\_avg\_features\_per\_channel[key][1]}")

    print(f"Variance: {long\_blink\_avg\_features\_per\_channel[key][2]}")

    print(f"Dominant Frequency: {long\_blink\_avg\_features\_per\_channel[key][3]} Hz")

    print(f"Bandwidth: {long\_blink\_avg\_features\_per\_channel[key][4]} Hz\n")

print("Average Features for Short Blink:")

for key in short\_blink\_avg\_features\_per\_channel:

    print(f"{key}:")

    print(f"Peak-to-Peak Amplitude: {short\_blink\_avg\_features\_per\_channel[key][0]}")

    print(f"Mean: {short\_blink\_avg\_features\_per\_channel[key][1]}")

    print(f"Variance: {short\_blink\_avg\_features\_per\_channel[key][2]}")

    print(f"Dominant Frequency: {short\_blink\_avg\_features\_per\_channel[key][3]} Hz")

    print(f"Bandwidth: {short\_blink\_avg\_features\_per\_channel[key][4]} Hz\n")

   # Preparing Dataset

# 1. Merge data from both blink types and label them

all\_features = []

for channel in ["Channel\_1", "Channel\_2", "Channel\_3", "Channel\_4"]:

    all\_features\_channel = np.vstack([short\_blink\_features\_per\_channel[channel], long\_blink\_features\_per\_channel[channel]])

    all\_features.append(all\_features\_channel)

all\_features = np.hstack(all\_features)  # Combining features from all channels

labels\_short = np.zeros(short\_blink\_features\_per\_channel["Channel\_1"].shape[0])

labels\_long = np.ones(long\_blink\_features\_per\_channel["Channel\_1"].shape[0])

all\_labels = np.hstack([labels\_short, labels\_long])

# 2. Split the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(all\_features, all\_labels, test\_size=0.3)

# Convert labels to one-hot encoding for training

y\_train\_categorical = to\_categorical(y\_train)

# 3. Build the Neural Network with modifications

model = Sequential()

model.add(Dense(64, input\_dim=X\_train.shape[1], activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(32, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Cross-validation

kfold = StratifiedKFold(n\_splits=5, shuffle=True)

cv\_scores = []

for train, val in kfold.split(X\_train, y\_train):

    y\_train\_k\_categorical = to\_categorical(y\_train[train])

    history: History = model.fit(X\_train[train], y\_train\_k\_categorical, epochs=50, batch\_size=10, validation\_data=(X\_train[val], to\_categorical(y\_train[val])))

    scores = model.evaluate(X\_train[val], to\_categorical(y\_train[val]), verbose=0)

    cv\_scores.append(scores[1] \* 100)

    print(f"Fold Accuracy: {scores[1]\*100:.2f}%")

print(f"Mean Accuracy: {np.mean(cv\_scores):.2f}%, Standard Deviation: {np.std(cv\_scores):.2f}%")

# 4. Plot the learning curve

plt.figure(figsize=(10, 5))

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(loc='lower right')

plt.show(block=True)

# 5. Use the trained model to predict on the test set

y\_pred\_probabilities = model.predict(X\_test)

y\_pred = np.argmax(y\_pred\_probabilities, axis=1)

# Compare the predictions with the true labels of the test set

accuracy = accuracy\_score(y\_test, y\_pred)

# Print results

print("\nModel Performance on Test Data:")

print(f"Accuracy: {accuracy \* 100:.2f}%")

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# Save the model

model\_save\_path = 'C:/dev/digitalbiomarkers/blink\_model.keras'

model.save(model\_save\_path)

print("Model saved at", model\_save\_path)

DECISION TREE CALISSIFIER CODE

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from tabulate import tabulate

import ast  # For safely evaluating the string

import mne

from scipy.signal import find\_peaks

from scipy.signal import welch

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from keras.models import Sequential

from keras.layers import Dense, Dropout

from keras.utils import to\_categorical

from keras.callbacks import History

from sklearn.model\_selection import StratifiedKFold

from keras.regularizers import l2

from sklearn.tree import DecisionTreeClassifier

plt.ion()  # Turn on interactive mode

# Load the CSV files

long\_blink\_data = pd.read\_csv('C:/dev/digitalbiomarkers/EEG-data/LongBlink.csv')

short\_blink\_data = pd.read\_csv('C:/dev/digitalbiomarkers/EEG-data/ShortBlink.csv')

# Display data tables

print("\nTable for LongBlink.csv:")

print(tabulate(long\_blink\_data.head(), headers='keys', tablefmt='grid'))

print("\nTable for ShortBlink.csv:")

print(tabulate(short\_blink\_data.head(), headers='keys', tablefmt='grid'))

# Convert string representations to actual lists

long\_blink\_data['data'] = long\_blink\_data['data'].apply(ast.literal\_eval)

short\_blink\_data['data'] = short\_blink\_data['data'].apply(ast.literal\_eval)

# Flatten the data

long\_blink\_values = [value for sublist in long\_blink\_data['data'].tolist() for value in sublist]

short\_blink\_values = [value for sublist in short\_blink\_data['data'].tolist() for value in sublist]

# Chunk the data

long\_blink\_sessions = [long\_blink\_values[i:i + 510] for i in range(0, len(long\_blink\_values), 510)]

short\_blink\_sessions = [short\_blink\_values[i:i + 510] for i in range(0, len(short\_blink\_values), 510)]

# Convert to arrays

long\_blink\_array = np.array(long\_blink\_sessions)

short\_blink\_array = np.array(short\_blink\_sessions)

# Apply bandpass filter

def apply\_bandpass\_filter(data, l\_freq=0.1, h\_freq=5.0):

    sfreq = 215.0

    data\_filtered = mne.filter.filter\_data(data, sfreq, l\_freq, h\_freq, method='iir', verbose=False)

    return data\_filtered

long\_blink\_array\_filtered = apply\_bandpass\_filter(long\_blink\_array)

short\_blink\_array\_filtered = apply\_bandpass\_filter(short\_blink\_array)

# Plot sessions function

def plot\_linked\_sessions(data\_array, title):

    fig, axs = plt.subplots(4, 1, figsize=(8, 12))

    for channel\_idx in range(4):

        blink\_data = np.concatenate([data\_array[session\_idx \* 4 + channel\_idx] for session\_idx in range(20)])

        axs[channel\_idx].plot(blink\_data)

        axs[channel\_idx].set\_title(f"Channel {channel\_idx + 1}")

        axs[channel\_idx].set\_xticks([])

    plt.tight\_layout()

    plt.subplots\_adjust(top=0.95)

    fig.suptitle(title)

    plt.draw()

# Plot the data

plot\_linked\_sessions(long\_blink\_array, "EEG Data Linked Sessions - Long Blink (Non-filtered)")

plot\_linked\_sessions(short\_blink\_array, "EEG Data Linked Sessions - Short Blink (Non-filtered)")

plot\_linked\_sessions(long\_blink\_array\_filtered, "EEG Data Linked Sessions - Long Blink (Filtered)")

plot\_linked\_sessions(short\_blink\_array\_filtered, "EEG Data Linked Sessions - Short Blink (Filtered)")

plt.show(block=True)

# Feature Extraction function

def extract\_features\_per\_channel(data\_array):

    # Initialize feature lists

    features\_per\_channel = {f"Channel\_{i+1}": [] for i in range(4)}

    for session\_idx in range(len(data\_array) // 4):  # 4 channels per session

        for channel\_idx in range(4):

            session = data\_array[session\_idx \* 4 + channel\_idx]

            # Time domain features

            # 1. Peak-to-peak amplitude

            positive\_peaks, \_ = find\_peaks(session)

            negative\_peaks, \_ = find\_peaks(-session)

            if positive\_peaks.size and negative\_peaks.size:

                max\_peak = max(session[positive\_peaks])

                min\_peak = min(session[negative\_peaks])

                peak\_to\_peak\_amplitude = max\_peak - min\_peak

            else:

                peak\_to\_peak\_amplitude = 0

            # 2. Mean

            mean\_val = np.mean(session)

            # 3. Variance

            variance\_val = np.var(session)

            # Frequency domain features

            # Compute the Power Spectral Density

            f, Pxx = welch(session, fs=250.0, nperseg=256)

            # 4. Dominant frequency

            dominant\_frequency = f[np.argmax(Pxx)]

            # 5. Bandwidth

            half\_power = np.max(Pxx) / 2

            indices = np.where(Pxx > half\_power)

            first\_index = indices[0][0]

            last\_index = indices[0][-1]

            bandwidth = f[last\_index] - f[first\_index]

            # Append features to the channel's feature list

            features\_per\_channel[f"Channel\_{channel\_idx+1}"].append([peak\_to\_peak\_amplitude, mean\_val, variance\_val, dominant\_frequency, bandwidth])

    # Convert lists to arrays

    for key in features\_per\_channel:

        features\_per\_channel[key] = np.array(features\_per\_channel[key])

    return features\_per\_channel

long\_blink\_features\_per\_channel = extract\_features\_per\_channel(long\_blink\_array\_filtered)

short\_blink\_features\_per\_channel = extract\_features\_per\_channel(short\_blink\_array\_filtered)

# Compute average features function

def compute\_average\_features\_per\_channel(features\_dict):

    average\_features = {}

    for key in features\_dict:

        avg\_peak\_to\_peak\_amplitude = np.mean(features\_dict[key][:, 0])

        avg\_mean\_val = np.mean(features\_dict[key][:, 1])

        avg\_variance\_val = np.mean(features\_dict[key][:, 2])

        avg\_dominant\_frequency = np.mean(features\_dict[key][:, 3])

        avg\_bandwidth = np.mean(features\_dict[key][:, 4])

        average\_features[key] = [avg\_peak\_to\_peak\_amplitude, avg\_mean\_val, avg\_variance\_val, avg\_dominant\_frequency, avg\_bandwidth]

    return average\_features

# Calculate average features for long and short blinks

long\_blink\_avg\_features\_per\_channel = compute\_average\_features\_per\_channel(long\_blink\_features\_per\_channel)

short\_blink\_avg\_features\_per\_channel = compute\_average\_features\_per\_channel(short\_blink\_features\_per\_channel)

# Print results

print("\nAverage Features for Long Blink:")

for key in long\_blink\_avg\_features\_per\_channel:

    print(f"{key}:")

    print(f"Peak-to-Peak Amplitude: {long\_blink\_avg\_features\_per\_channel[key][0]}")

    print(f"Mean: {long\_blink\_avg\_features\_per\_channel[key][1]}")

    print(f"Variance: {long\_blink\_avg\_features\_per\_channel[key][2]}")

    print(f"Dominant Frequency: {long\_blink\_avg\_features\_per\_channel[key][3]} Hz")

    print(f"Bandwidth: {long\_blink\_avg\_features\_per\_channel[key][4]} Hz\n")

print("Average Features for Short Blink:")

for key in short\_blink\_avg\_features\_per\_channel:

    print(f"{key}:")

    print(f"Peak-to-Peak Amplitude: {short\_blink\_avg\_features\_per\_channel[key][0]}")

    print(f"Mean: {short\_blink\_avg\_features\_per\_channel[key][1]}")

    print(f"Variance: {short\_blink\_avg\_features\_per\_channel[key][2]}")

    print(f"Dominant Frequency: {short\_blink\_avg\_features\_per\_channel[key][3]} Hz")

    print(f"Bandwidth: {short\_blink\_avg\_features\_per\_channel[key][4]} Hz\n")

   # Preparing Dataset

# 1. Merge data from both blink types and label them

all\_features = []

for channel in ["Channel\_1", "Channel\_2", "Channel\_3", "Channel\_4"]:

    all\_features\_channel = np.vstack([short\_blink\_features\_per\_channel[channel], long\_blink\_features\_per\_channel[channel]])

    all\_features.append(all\_features\_channel)

all\_features = np.hstack(all\_features)  # Combining features from all channels

labels\_short = np.zeros(short\_blink\_features\_per\_channel["Channel\_1"].shape[0])

labels\_long = np.ones(long\_blink\_features\_per\_channel["Channel\_1"].shape[0])

all\_labels = np.hstack([labels\_short, labels\_long])

# 2. Split the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(all\_features, all\_labels, test\_size=0.3)

# Convert labels to one-hot encoding for training

# y\_train\_categorical = to\_categorical(y\_train)

# Build and train the Decision Tree model

model = DecisionTreeClassifier(random\_state=42)

model.fit(X\_train, y\_train)

# Use the trained model to predict on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred)

# Print results

print("\nModel Performance on Test Data:")

print(f"Accuracy: {accuracy \* 100:.2f}%")

print("\nConfusion Matrix:")

print(conf\_matrix)

print("\nClassification Report:")

print(classification\_rep)

# Save the model

# DecisionTree models are typically saved using joblib or pickle

import joblib

model\_save\_path = 'C:/dev/digitalbiomarkers/decision\_tree\_model.joblib'

joblib.dump(model, model\_save\_path)

print("Model saved at", model\_save\_path)