Machine Learning-based Prediction of Gait Events from Dual-Layer EEG Datasets

The project's central hypothesis is that the motion artifact layer of dual-layer EEG datasets contains more information related to motion and thus can provide more accurate predictions of gait events during walking. This hypothesis will be tested through the development and evaluation of machine learning models using different combinations of data sources, including the motion artifact layer alone and in combination with scalp channel data and source signals after cleaning.

Overview:

The objective of this project is to develop a machine learning-based approach for predicting four gait events (ULHS, ULTO, URHS, URTO) from dual-layer EEG datasets. Electroencephalography (EEG) is a widely used technique for studying brain activity during walking and dynamic movements. However, EEG also records electrical signals from other sources such as muscles, eyes, and motion, which can be considered as artifact signals. These artifact signals can provide useful information about other ongoing processes occurring in the body. The aim of this project is to leverage these EEG artifact signals to predict gait events during walking.

Methods:

The project will involve five datasets, which include:

1) motion artifact layer alone (78/128 channels)

2) source signals after cleaning (78 sources)

3) scalp channel data alone (78/128 channels)

For each dataset, we will extract data for specific time windows and gait events add preprocessing and feature extraction if required. We will then train and evaluate various machine learning models, including Random Forest, Support Vector Machines (SVM), and Logistic Regression, to predict the four gait events.

Expected Outcomes:

The project is expected to result in the development of a machine learning-based approach for predicting gait events from dual-layer EEG datasets. The study will contribute to the development of novel methods for using EEG artifact signals to predict gait events during walking. This project will also provide insights into the potential of EEG to be used for understanding more than just brain dynamics and lay the foundation for the field to develop additional technologies for multimodal neuromechanics.

In the long run the expected outcome of this project is to establish that EEG alone can provide new insights about neuromechanics from a comprehensive perspective that integrates brain, kinematic, visual, and metabolic measures in a single experiment.

Impact:

The project will advance our understanding of the interplay between neural processes and biomechanics of walking, which may provide greater insight regarding the underlying deficits in mobility and cognition that occur with aging and disease. The outcomes of the project will also provide researchers and clinicians with a non-invasive and cost-effective method for predicting gait events during walking.

Future works:

Future work includes extending the approach to predict other metrics such as metabolic cost and eye gaze fixation. The successful development of such a predictive model would have significant potential for real-time monitoring of gait events and associated metrics during daily activities, providing valuable insights for both clinical and research applications.

(Predicting live gait events would involve real-time processing of EEG signals and predicting gait events as they occur. This would require the development of an online machine learning model that can continuously process the incoming EEG signals and provide accurate predictions of gait events. The real-time processing would also require efficient and fast algorithms that can handle large amounts of data in real-time. Additionally, the system would need to be validated in a real-world setting to ensure its accuracy and effectiveness in predicting gait events. Overall, the future work would involve the development of a robust and reliable system for predicting live gait events using EEG signals.)

Details

To predict these four gait events from dual-layer EEG datasets, machine learning techniques will be used. The datasets that will be compared are:

1) motion artifact layer alone (78/128 channels)

2) source signals after cleaning (78 sources)

3) scalp channel data alone (78/128 channels)

The idea is to determine which dataset provides the best estimate or largest contributor to a good estimate of the four gait events. It is hypothesized that the motion artifact layer contains the most information related to motion and will be the most useful dataset for prediction. To achieve this, the following steps will be taken:

1.**Data extraction and Data preprocessing:** EEG data will be loaded from the .set file into an MNE raw object. Annotations will be converted to events, and a list of event information will be created.

preprocess_eeg_data.py:

Preprocess EEG Data module

This code loads the EEG data from .set files and preprocesses it. It also concatenates the preprocessed data from all subjects and saves the epoched data for each event type. The preprocessed EEG data is used as input for machine learning-based gait event prediction.

Dependencies

scipy

numpy

pandas

mne

Function signature

def preprocess_eeg_data(subject_ids, file_path, event_id, save_dir, use_78_channels=False):

Input

subject_ids (list of ints) - List of subject IDs.

file_path (str) - File path for the .set files.

event_id (dict) - Dictionary containing event names as keys and event IDs as values.

save_dir (str) - Directory to save the epoched data.

use_78_channels (bool, optional) - Whether to use only 78 good channels or all channels. Default is False.

Output

all_data (MNE Epochs object) - Epoched and preprocessed EEG data for all subjects and all event types.

Example usage

```
subject_ids = [1, 2, 3]
file_path = 'data/sub{sub_id}/allwalk_EEG.set'
event_id = {'ULHS': 1, 'ULTO': 2, 'URHS': 3, 'URTO': 4}
save_dir = 'preprocessed_data'
use_78_channels = True

preprocessed_data = preprocess_eeg_data(subject_ids, file_path, event_id, save_dir, use_78_channels)
```

Read_data.py:

This code defines a function **load_data** that loads the epoched data previously saved for each event type using the mne package.

The function takes two arguments:

event_id - is a list of integers specifying the event types to load

file_path - is a string specifying the directory path where the epoched data files are located.

Inside the function, a list all_epochs is created to store the epoched data for each event type.

The function loops through each event type in event_id and uses the mne.read_epochs function to load the data from the corresponding epoched data file. The loaded data is appended to the all_epochs list.After all the data for each event type is loaded, the function

uses mne.concatenate_epochs to concatenate the data for all event types into a single mne.Epochs object called all_data. Finally, the function returns the all_data object. This function can be used in subsequent code to load the preprocessed EEG data for further analysis, such as machine learning modeling or statistical analysis.

```
#Read the data

#define the event_ids
event_id = ('ULHS', 'ULTO', 'URHS', 'URTO')
file_path = 'DATA\Event_Epoched_Data_Artifact_Layer_128'
#file_path = 'DATA\Event_Epoched_Data_Artifact_Layer_78'
#file_path = 'DATA\Event_Epoched_Data_Inner_Layer_128'
#file_path = 'DATA\Event_Epoched_Data_Inner_Layer_78'
#store the concatenate in all_epochs
all_data = load_data(event_id, file_path)
```

2.Feature extraction: Feature extraction: The EEG data is processed to extract three types of features, as described below. These features are extracted using two distinct approaches:

a. Averaged across channels: In this approach, the features are computed for each epoch and then averaged across all channels, disregarding the spatial aspect of the brain/electrodes. This results in a more compact representation of both the artifact layer (motion artifacts) and the scalp layer (inner brain activity) but may sacrifice some information about the spatial distribution of the signals. (Feature_Extraction_Parallel.py)

*The functions used to extract features are as follows:

extract_psd_features(epoch): This function computes the power spectral density (PSD) of the EEG data in each epoch and then calculates the mean power in several frequency bands defined in the bands dictionary. The function returns a dictionary containing the mean power in each frequency band.

extract_time_domain_features(epoch): This function computes several statistical measures of the EEG data in each epoch, including the mean, standard deviation, skewness, and kurtosis. The function returns a list of these statistical measures.

extract_wavelet_features(epoch): This function applies the discrete wavelet transform to the EEG data in each epoch and then calculates several statistics of the resulting wavelet coefficients. The function returns a dictionary containing the mean of each statistic across all channels.

process_epoch(i, epoch): This function combines the features extracted by the above functions into a single dictionary and returns the dictionary along with an index i indicating which epoch the features were extracted from.

extract_features(all_data): This function takes a list of epochs (all_data) and applies the process_epoch() function to each epoch in parallel using the joblib library. The resulting feature dictionaries are then combined into a single pandas DataFrame and returned.

Overall, this code is used to extract a variety of features from EEG data, including frequency domain features, time domain features, and wavelet features

#calling the function where all_data is the preprocessed EEG data

```
#feature extraction averaged across all channels

now = datetime.now()
current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)

feature_df = extract_features(all_data)
# Save the combined feature DataFrame to a CSV file
feature_df.to_csv('feature_Event_Epoched_Data_Inner_Layer_128_0.05.csv', index=False)


now = datetime.now()
current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)
```

b. Individual channels: Preliminary results indicate that averaging feature values may reduce the accuracy of the artifact detection. To address this issue, an alternative approach is adopted where features are extracted for each of the 128 channels separately. This method preserves the spatial information of both the artifact and scalp layers. To manage the increased dimensionality of the dataset, feature selection algorithms can be employed to reduce the number of features while retaining the most relevant information for training the model. (Feature_extraction_all_channels_Parallel.py)

By comparing these two approaches, it is possible to determine which method best balances the trade-off between dimensionality reduction and preserving the spatial information necessary for accurate differentiation between artifact and scalp layers.

1) PSD features: The power is calculated for each frequency band (delta, theta, alpha, and beta) by averaging the PSD values within the specified frequency range. These features are then aggregated by computing the mean power value across all channels for each frequency band.

2) Time-domain features: Four statistical measures (mean, standard deviation, skewness, and kurtosis) are computed for each epoch. These values are then averaged across all channels to obtain a single value for each feature.

3) Wavelet features: The EEG signals are decomposed using the Daubechies 4 wavelet, resulting in approximation (cA) and detail (cD) coefficients. Various features are calculated for both cA and cD, and these values are then averaged across all channels.

*The functions used to extract features are as follows:

extract_psd_features function computes the PSD for each channel in the epoch (a segment of EEG data) and averages the power within each frequency band.

extract_time_domain_features function calculates several time-domain features for each channel in the epoch, including mean, standard deviation, skewness, and kurtosis.

extract_wavelet_features function performs a wavelet transform on each channel in the epoch using the Daubechies 4 wavelet. It then calculates several features from the wavelet coefficients, such as mean, standard deviation, energy, and entropy.

extract_combined_features function takes a list of EEG epochs as input, extracts feature from each epoch using the previously defined functions and combines the features into a single pandas DataFrame. The feature extraction process is parallelized using Joblib to speed up the computation.

#calling the function where all_data is the preprocessed EEG data

```
# feature Extraction across all_chanells
now = datetime.now()
current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)
combined_feature_df = extract_combined_features(all_data)
# Save the combined feature DataFrame to a CSV file
combined_feature_df.to_csv('combined_feature.csv', index=False)
now = datetime.now()
current_time = now.strftime("%H:%M:%S")
print("Current Time =", current_time)
```

Note: Initial results of the machine learning models trained on the extracted features showed a decrease in classification accuracy when using the random forest algorithm. For the model trained on the averaged features from the artifact layer's 128 channels, the accuracy was 52%. This could be attributed to a loss of information in the process of averaging the features.

On the other hand, the model trained on the features for individual channels from the artifact layer's 128 channels achieved an accuracy of 72%. However, this result could be affected by the small epoch size, which might have led to imprecise feature extraction.

In summary, the initial results suggest that there might be some challenges in accurately extracting features from the data, which could affect the overall performance of the machine learning models.

Create_ICA_Data_Epochs.py

load_ica_data(subject_code): This function loads ICA data for a given subject from a .mat file, which contains the ICA weights, a sphere matrix, and information about good channels. It returns the ICA weights, sphere matrix, and indices of the good channels.

epoch_ica_data(ica_data, events, event_id, tmin=-0.1, tmax=0.1): This function epochs the ICA activations around specific events of interest. It takes as input the ICA activations, the event markers, the event types, and the epoch window (default -0.1 to 0.1 seconds around the event). It returns the epoched data.

apply_ica_weights(raw, icaweights, icasphere, good_channels, num_ics=78): This function applies the ICA weights to the raw EEG data to remove artifacts. It takes as input the raw EEG data, the ICA weights, the sphere matrix, and the indices of the good channels. It selects the top 78 ICA components based on the sum of the absolute weights and applies them to the good channel data to remove artifacts.

mne.io.read_raw_eeglab(set_file, preload=True): This function loads the raw EEG data in. set format using MNE. It takes the path to the .set file as input and returns a Raw object.

mne.events_from_annotations(raw): This function extracts the event markers from the raw EEG data using the annotations. It takes input the raw EEG data and returns a tuple containing the event markers and event types.

mne.Epochs(ica_raw, events, event_id, tmin=tmin, tmax=tmax): This function epochs the ICA activations around specific events of interest using MNE. It takes as input the ICA activations, the event markers, the event types, and the epoch window (default -0.1 to 0.1 seconds around the event). It returns the epoched data.

mne.concatenate_epochs(all_epochs[event]): This function concatenates the epoched data for each event type using MNE. It takes as input a list of epoched data and returns a single Epochs object.

concatenated_epochs.save(save_path, overwrite=True): This method saves the epoched data to disk in .fif format. It takes input the path to the output file and a boolean flag to indicate whether to overwrite existing files.

3.. Model training: The extracted features will be used to train various machine learning models such as Random Forest, SVM, and Logistic Regression. The models will be trained on

1) motion artifact layer alone (78/128 channels)

2) source signals after cleaning (128)

3) scalp channel data alone (78/128 channels)

The accuracy of each model will be evaluated using classification reports and confusion matrices.

Traning_Models2.py

This function **train_models_and_save** takes in the preprocessed data, trains machine learning models (Random Forest, SVM, Logistic Regression), evaluates their performance, and saves the models, classification reports, and confusion matrices to a specified directory.

Here is a breakdown of the function:

The function takes in three arguments: all_data, save_dir, and random_seed.

all_data is the preprocessed EEG data. The function extracts the data and labels from the all_data object and standardizes the data.

The function then splits the data into train and test sets using train_test_split() method.

Next, it trains three different machine learning models: Random Forest, SVM, and Logistic Regression.

For each model, the function computes and prints out the model's accuracy, classification report, and confusion matrix.

Finally, the function saves the models, classification reports, and confusion matrices to the specified directory using pickle.dump() method.

```
#train the Models

save_dir = 'Trained_Models/Test'
train_models_and_save(all_data, save_dir)
```

4.Final prediction: The best-performing model will be used to predict the four gait events on raw EEG data.
Through this process, it is expected that a machine learning model will be developed that can accurately predict the ULHS, ULTO, URHS, and URTO gait events from dual-layer EEG datasets.

Final_Predictions_on_raw_data.py

This code is designed to predict events from raw EEG data using a pre-trained machine learning model. The **predict_events** function takes as input a **raw_data** object, which is an MNE Raw object containing the EEG data to be analyzed. The function also requires the **model_path** argument, which is a string indicating the path to the trained machine learning model file.

The **predict_events** function then applies the trained model to the input data by sliding a window over the data and predicting the probability of each trial belonging to each class. The function applies the trained model in parallel using the **ThreadPoolExecutor** from the **concurrent.futures** module to speed up the predictions. The function outputs a **DataFrame** containing the predicted events and their corresponding times.

The **save_event_times** function takes as input the **DataFrame** containing the predicted events and their times and outputs a CSV file containing the times at which each event occurred. The output CSV file is saved in the specified output directory.

```
# Load the raw data
raw_data = mne.io.read_raw_eeglab('Data\Data_Artifact_Layer\PTW01_allwalk_artifact.set')

model_path='Trained_Models/Trained_models_Artifact_Layer_128/SVM Accuracy.pkl'

# Assuming you have imported the necessary libraries and have loaded your raw data and set the model path

# Call the function and pass the required parameters
#def predict_events(raw_data, model_path, window_size, step_size, threshold=0.5, tmin=None, tmax=None):
event_preds_times = predict_events(raw_data, model_path, window_size=0.201171875, step_size=0.001,t_min=40,t_max=41, threshol

# Print the predicted events and their times
#print(event_preds_times)
```

```
# call the save_event_times function
output_dir = 'Final_Prediction/Test'
save_event_times(event_preds_times, output_dir)
```

Extract_Window_Data.py (to analyze/Validate the final prediction)

The data will be split into windows of a specific time frame, and data in a specific time window will be extracted. Data between two indices will be extracted, and window data will be saved to a CSV file.

Input:

file_path: path to the EEG file

event_dict: dictionary of event names and corresponding event IDs

first_event: name of the first event in the window

last_event: name of the last event in the window

window_times: list of tuples specifying the start and end times for each window

save_dir: directory to save the event information and window data CSV files (optional)

Output:

window_data: a list of dataframes containing the event information and data for each time window

```
file_path = 'Data\Data_Artifact_Layer\PTW01_allwalk_artifact.set'

event_dict = {'CLHS': 1, 'CLP_On': 2, 'CLTO': 3, 'CRHS': 4, 'CRTO': 5,
              'PLHS': 6, 'PLP_On': 7, 'PLTO': 8, 'PRHS': 9, 'PRTO': 10,
              'ULHS': 11, 'ULP_On': 12, 'ULTO': 13, 'URHS': 14, 'URTO': 15,
              'boundary': 16}

window_times = [(0, 250), (500, 750)]

save_dir = "DATA\Windowed_data"

window_data = extract_window_data(file_path, event_dict, 'ULHS', 'ULTO', window_times, save_dir)
```

EEG_gait_prediction_pipleline

The code is a pipeline for processing EEG data and predicting events from it. The pipeline consists of several modules for preprocessing the data, training machine learning models, and making predictions on new raw data.

The first module **preprocess_eeg_data ipynb** is used to concatenate the epochs data for multiple subjects into a single file for training the models.

The second module **Read_data.ipynb** is used to load the preprocessed data from the first module and prepare it for training the models.

The Feature Extraction Module (**Feature_Extraction_Parallel /Feature_extraction_all_channels_Parallel.py**)is used to extract the features as mentioned in the feature extraction section.

a) #feature extraction averaged across all channels
b) # feature Extraction across all_channels

The third module **Traning_Models2.ipynb** is used to train machine learning models such as Random Forest, Decision Trees, Linear Regression, Support Vector Machines (SVM), and Neural Networks. The models are trained in preprocessed data and their accuracy is checked.

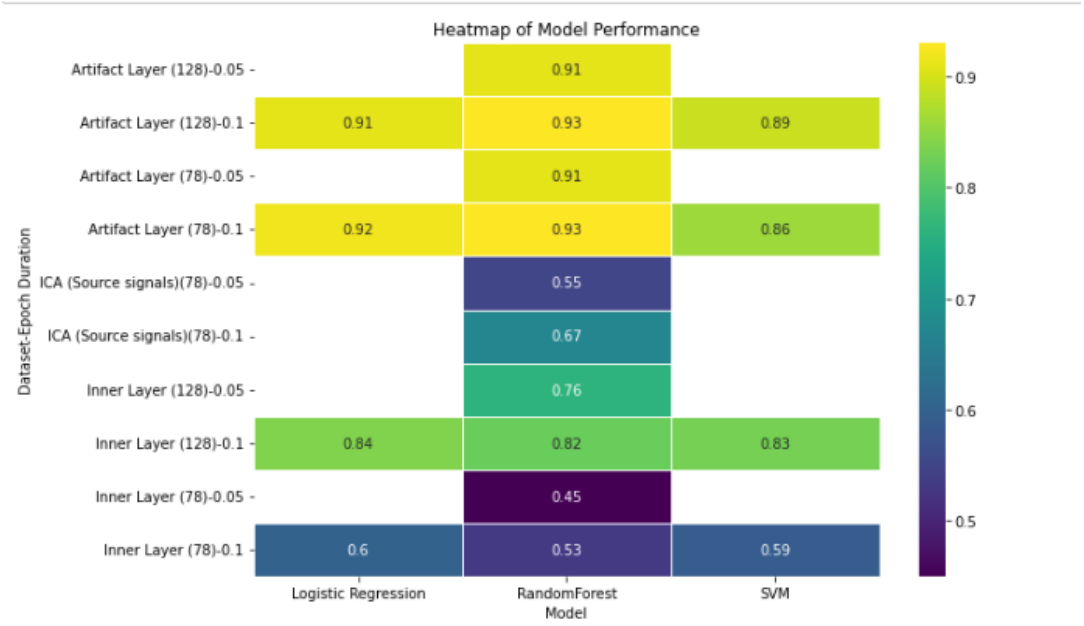
The fourth module **Extract_Window_Data.ipynb** is used to extract the required windowed data for making predictions on the raw data.

The fifth module **Final_Predictions_on_raw_data.ipynb** is used to make predictions on the new raw data using the trained machine learning models. The predicted events and their times are saved to a CSV file.

The output of the pipeline is a CSV file containing the **predicted events and their times** for the **new raw data.**

Current Results description:

Number of events	44136
Events	ULHS: 11034 ULTO: 11034 URHS: 11034 URTO: 11034
Time range	-0.100 – 0.100 sec
Baseline	-0.100 – 0.000 sec



The pipeline was run for four datasets mentioned below:

1) **motion artifact layer alone (78 channels)**

Random Forest

a) Epoch 0.1 sec

```
Random Forest Accuracy: 0.9339261944546716
              precision    recall  f1-score   support

          11       0.94      0.94      0.94      8799
          13       0.91      0.93      0.92      8787
          14       0.95      0.94      0.94      8836
          15       0.93      0.92      0.93      8887

    accuracy                           0.93     35309
   macro avg       0.93      0.93      0.93     35309
weighted avg       0.93      0.93      0.93     35309

[[8302   97  367   33]
 [  82 8207   26  472]
 [ 382   76 8299   79]
 [  50  617   52 8168]]
```

b) Epoch 0.05 sec

```
Random Forest Accuracy: 0.9135810460901087
              precision    recall  f1-score   support

          11       0.93      0.92      0.92      7705
          13       0.90      0.90      0.90      7693
          14       0.92      0.93      0.93      7721
          15       0.91      0.90      0.90      7777

    accuracy                           0.91     30896
   macro avg       0.91      0.91      0.91     30896
weighted avg       0.91      0.91      0.91     30896

[[7103  129  427   46]
 [ 134 6924   53  582]
 [ 366   75 7201   79]
 [  53  597  129 6998]]
```

Logistic Regression

```
Logistic Regression Accuracy: 0.9209549973094678
              precision    recall  f1-score   support

          11       0.91      0.93      0.92      8799
          13       0.93      0.92      0.92      8787
          14       0.93      0.91      0.92      8836
          15       0.92      0.91      0.92      8887

    accuracy                           0.92     35309
   macro avg       0.92      0.92      0.92     35309
weighted avg       0.92      0.92      0.92     35309

[[8223  106  402   68]
 [ 101 8118   57  511]
 [ 583   72 8053  128]
 [ 118  474  171 8124]]
```

Support vector machine

```
SVM with SGD Accuracy: 0.8694667082047071
              precision    recall  f1-score   support

          11       0.88      0.86      0.87      8799
          13       0.82      0.91      0.86      8787
          14       0.91      0.83      0.87      8836
          15       0.87      0.88      0.87      8887

    accuracy                           0.87     35309
   macro avg       0.87      0.87      0.87     35309
weighted avg       0.87      0.87      0.87     35309

[[7554  576  460  209]
 [ 183 7961   53  590]
 [ 628  432 7377  399]
 [ 176  702  201 7808]]
```

RNN-LSTM

```
              precision    recall  f1-score   support

           0       0.90      0.91      0.91      2224
           1       0.91      0.89      0.90      2205
           2       0.92      0.91      0.91      2207
           3       0.89      0.92      0.91      2192

    accuracy                           0.91      8828
   macro avg       0.91      0.91      0.91      8828
weighted avg       0.91      0.91      0.91      8828
```

```
[[2034   32  135   23]
 [  47 1957   19  182]
 [ 146   23 2001   37]
 [  21  128   30 2013]]
```

2) motion artifact layer alone (128 channels)

Random Forest

a) Epoch 0.1 sec

```
Random Forest Accuracy: 0.9348041575802203
              precision    recall  f1-score   support

          11       0.94      0.94      0.94      8799
          13       0.91      0.94      0.93      8787
          14       0.95      0.94      0.94      8836
          15       0.94      0.92      0.93      8887

    accuracy                           0.93     35309
   macro avg       0.93      0.93      0.93     35309
weighted avg       0.94      0.93      0.93     35309

[[8301  103  363   32]
 [  81 8251   25  430]
 [ 394   84 8270   88]
 [  37  604   61 8185]]
```

b) Epoch 0.05 sec

```
Random Forest Accuracy: 0.9147786121180735
              precision    recall  f1-score   support

          11       0.93      0.92      0.93      7705
          13       0.89      0.91      0.90      7693
          14       0.92      0.93      0.93      7721
          15       0.91      0.90      0.91      7777

    accuracy                           0.91     30896
   macro avg       0.91      0.91      0.91     30896
weighted avg       0.91      0.91      0.91     30896

[[7102  119  443   41]
 [ 120 6965   57  551]
 [ 346   91 7213   71]
 [  56  615  123 6983]]
```

Logistic Regression

```
Logistic Regression Accuracy: 0.918604321844289
              precision    recall  f1-score   support

          11       0.91      0.92      0.92      8799
          13       0.93      0.93      0.93      8787
          14       0.91      0.90      0.91      8836
          15       0.93      0.92      0.92      8887

    accuracy                           0.92     35309
   macro avg       0.92      0.92      0.92     35309
weighted avg       0.92      0.92      0.92     35309

[[8112  183  384  120]
 [ 124 8165  128  370]
 [ 610   70 7986  170]
 [  73  366  276 8172]]
```

Support Vector Machine

```
SVM with SGD Accuracy: 0.897278314310799
              precision    recall  f1-score   support

          11       0.90      0.88      0.89      8799
          13       0.89      0.91      0.90      8787
          14       0.91      0.88      0.90      8836
          15       0.90      0.92      0.91      8887

    accuracy                           0.90     35309
   macro avg       0.90      0.90      0.90     35309
weighted avg       0.90      0.90      0.90     35309

[[7740  389  494  176]
 [ 179 8007  106  495]
 [ 619  166 7792  259]
 [  98  462  184 8143]]
```

3) scalp layer (78 randomly selected good channels)

Random Forest

a) Epoch 0.1 sec

```
Random Forest Accuracy: 0.5366903622305927
              precision    recall  f1-score   support

          11       0.48      0.54      0.51      8799
          13       0.53      0.58      0.56      8787
          14       0.52      0.47      0.49      8836
          15       0.63      0.55      0.59      8887

    accuracy                           0.54     35309
   macro avg       0.54      0.54      0.54     35309
weighted avg       0.54      0.54      0.54     35309

[[4788 1273 1954  784]
 [1280 5120  968 1419]
 [2591 1412 4156  677]
 [1295 1826  880 4886]]
```

b) Epoch 0.05 sec

```
Random Forest Accuracy: 0.4564021232522009
              precision    recall  f1-score   support

          11       0.44      0.45      0.45      7705
          13       0.42      0.45      0.43      7693
          14       0.46      0.44      0.45      7721
          15       0.51      0.49      0.50      7777

    accuracy                           0.46     30896
   macro avg       0.46      0.46      0.46     30896
weighted avg       0.46      0.46      0.46     30896

[[3504 1462 1763  976]
 [1306 3446 1134 1807]
 [1973 1462 3367  919]
 [1160 1808 1025 3784]]
```

Logistic Regression

```
Logistic Regression Accuracy: 0.5960803194652922
              precision    recall  f1-score   support

          11       0.57      0.55      0.56      8799
          13       0.62      0.64      0.63      8787
          14       0.55      0.55      0.55      8836
          15       0.64      0.65      0.64      8887

    accuracy                           0.60     35309
   macro avg       0.60      0.60      0.60     35309
weighted avg       0.60      0.60      0.60     35309

[[4873  781 2466  679]
 [ 721 5610  741 1715]
 [2309  894 4819  814]
 [ 662 1759  721 5745]]
```

Support Vector Machine

```
SVM with SGD Accuracy: 0.600979920133677
              precision    recall  f1-score   support

          11       0.57      0.57      0.57      8799
          13       0.64      0.63      0.64      8787
          14       0.53      0.58      0.55      8836
          15       0.67      0.63      0.65      8887

    accuracy                           0.60     35309
   macro avg       0.60      0.60      0.60     35309
weighted avg       0.60      0.60      0.60     35309

[[5033  687 2540  539]
 [ 764 5505 1021 1497]
 [2272  789 5123  652]
 [ 783 1569  976 5559]]
```

RNN-LSTM

```
              precision    recall  f1-score   support

           0       0.39      0.41      0.40      2224
           1       0.40      0.39      0.40      2205
           2       0.38      0.37      0.37      2207
           3       0.45      0.44      0.44      2192

    accuracy                           0.40      8828
   macro avg       0.40      0.40      0.40      8828
weighted avg       0.40      0.40      0.40      8828
```

```
[[919 369 588 348]
 [377 868 438 522]
 [679 405 808 315]
 [399 531 305 957]]
```

4) scalp layer (128 channels)

a) Epoch 0.1 sec

Random Forest

```
Random Forest Accuracy: 0.8206123084765924
              precision    recall  f1-score   support

          11       0.77      0.79      0.78      8799
          13       0.84      0.86      0.85      8787
          14       0.80      0.79      0.80      8836
          15       0.87      0.84      0.85      8887

    accuracy                           0.82     35309
   macro avg       0.82      0.82      0.82     35309
weighted avg       0.82      0.82      0.82     35309

[[6971  257 1423  148]
 [ 309 7573   89  816]
 [1528  163 7004  141]
 [ 259 1016  185 7427]]
```

b) Epoch 0.05 sec

```
Random Forest Accuracy: 0.7662156913516313
              precision    recall  f1-score   support

          11       0.79      0.73      0.76      7705
          13       0.72      0.80      0.76      7693
          14       0.78      0.75      0.77      7721
          15       0.78      0.77      0.78      7777

    accuracy                           0.77     30896
   macro avg       0.77      0.77      0.77     30896
weighted avg       0.77      0.77      0.77     30896

[[5663  565 1226  251]
 [ 250 6177  119 1147]
 [1110  502 5821  288]
 [ 140 1356  269 6012]]
```

Logistic Regression

```
Logistic Regression Accuracy: 0.8440907417372341
              precision    recall  f1-score   support

          11       0.80      0.80      0.80      8799
          13       0.89      0.89      0.89      8787
          14       0.80      0.80      0.80      8836
          15       0.89      0.89      0.89      8887

    accuracy                           0.84     35309
   macro avg       0.84      0.84      0.84     35309
weighted avg       0.84      0.84      0.84     35309

[[7030  160 1520   89]
 [ 142 7807  105  733]
 [1504  121 7075  136]
 [ 107  727  161 7892]]
```

Support Vector Machine

```
SVM with SGD Accuracy: 0.8348862896145459
              precision    recall  f1-score   support

          11       0.78      0.80      0.79      8799
          13       0.88      0.88      0.88      8787
          14       0.79      0.79      0.79      8836
          15       0.89      0.87      0.88      8887

    accuracy                           0.83     35309
   macro avg       0.84      0.83      0.84     35309
weighted avg       0.84      0.83      0.84     35309

[[7064  144 1512   79]
 [ 247 7721  131  688]
 [1600  102 6979  155]
 [ 166  835  171 7715]]
```

RNN-LSTM

```
              precision    recall  f1-score   support

           0       0.61      0.63      0.62      2224
           1       0.60      0.68      0.64      2205
           2       0.62      0.58      0.60      2207
           3       0.71      0.65      0.68      2192

    accuracy                           0.63      8828
   macro avg       0.64      0.63      0.63      8828
weighted avg       0.64      0.63      0.63      8828


[[1398  205  536   85]
 [ 152 1500  133  420]
 [ 657  199 1280   71]
 [  95  579  103 1415]]
```

5) ICA with 78 channels

a) Epoch 0.1 sec

```
Random Forest Accuracy: 0.67180217503884
              precision    recall  f1-score   support

          11       0.66      0.67      0.66      7705
          13       0.67      0.71      0.69      7693
          14       0.68      0.63      0.66      7721
          15       0.68      0.67      0.67      7777

    accuracy                           0.67     30896
   macro avg       0.67      0.67      0.67     30896
weighted avg       0.67      0.67      0.67     30896

[[5136  519 1498  552]
 [ 475 5497  387 1334]
 [1669  536 4894  622]
 [ 495 1613  440 5229]]
```

b) Epoch 0.05 sec

```
Random Forest Accuracy: 0.5553469704816157
              precision    recall  f1-score   support

          11       0.60      0.58      0.59      7705
          13       0.51      0.58      0.54      7693
          14       0.59      0.55      0.57      7721
          15       0.53      0.52      0.52      7777

    accuracy                           0.56     30896
   macro avg       0.56      0.56      0.56     30896
weighted avg       0.56      0.56      0.56     30896

[[4433  892 1648  732]
 [ 679 4447  603 1964]
 [1685  926 4249  861]
 [ 625 2439  684 4029]]
```

Conclusion

In conclusion, the results demonstrate that the performance of the machine learning models was poorest for the scalp layer with only good channels. This can be attributed to the fact that the bad channels contained the necessary information about motion artifacts, which enabled the models to accurately predict gait events. In contrast, the scalp layer with only good channels lacked this information, leading to suboptimal model performance.

Furthermore, it is worth noting that the ICA source signals (top 78) performed better than the scalp layer but were still significantly outperformed by the artifact layer. This finding supports the hypothesis that the presence of motion artifact information is crucial for accurate gait event prediction. Therefore, future research should consider incorporating such information when developing machine learning models for gait analysis, as this approach may lead to more reliable and accurate predictions.

Next in line

1) Reducing epoch duration:

Reducing the epoch duration may help to capture more precise information about the EEG signal around the events. However, it may also increase the computational load and potentially lead to overfitting if the model is too complex. It is worth experimenting with different epoch durations and assessing the impact on the model's accuracy and generalization performance.

//done (the classification accuracy is not affected if considered an epoch of 100 ms)

In future we can try using asymmetric time duration around the event

2) Extracting relevant features:

Extracting relevant features from the EEG signal can help to reduce the dimensionality of the data and capture more informative features. This can be achieved through various techniques such as wavelet transforms, Fourier transforms, and time-frequency analysis. Feature extraction may improve the model's accuracy and reduce the training time required.

//Initial stage done although there needs to be more experimentation done to improve the classification accuracy

3)Training model on ICA data:

When trained using ICA with 78 source signals, the Random Forest classification model achieved an accuracy of 55%. This performance was 10% higher compared to the model trained on the scalp layer with 78 channels, demonstrating an improvement in predictive accuracy. However, the model's performance was still 36% lower than when trained on the artifact layer with 78 channels. This disparity highlights the importance of motion artifact information in accurately predicting gait events and suggests that incorporating such information can significantly enhance model performance.

4) Hyperparameter tuning:

Hyperparameters are set before training the model and cannot be learned from the data. They include parameters such as learning rate, batch size, number of layers, and activation functions. Tuning these hyperparameters can significantly improve the performance of the model. We will try using techniques such as grid search or random search to find the optimal combination of hyperparameters.

(On going process)

6) Training on Neural Networks:

Neural networks have shown promising results in EEG signal processing and classification tasks. Using neural networks for real-time prediction of EEG gait events may be a good approach as they can handle large amounts of data and capture complex patterns in the signal. However, neural networks can also be computationally expensive to train and may require careful optimization of the model's architecture and hyperparameters.

Neural Network	Advantages	Disadvantages	Suitability for Gait Event Prediction
Feedforward NN	•Simple architecture •Easy to train •Can model non-linear relationships	•Cannot handle temporal dependencies •Prone to overfitting	•Not well-suited for this task due to the lack of handling temporal dependencies in EEG data
CNN (Convolutional Neural Network)	•Can learn spatial features •Good for image and time-series data •Translation invariant	•Limited in handling temporal dependencies •Complex architecture, may require large datasets	•Moderately suitable for this task; can handle spatial features in EEG data but might struggle with temporal dependencies
RNN (Recurrent Neural Network)	•Can handle temporal dependencies •Good for time-series and sequential data	•Vanishing gradient problem •Difficult to train •Limited in learning long-term dependencies	•Suitable for this task, but LSTM might be a better choice due to its ability to learn long-term dependencies
LSTM (Long Short-Term Memory)	•Can handle temporal dependencies •Good for time-series and sequential data •Can learn long-term dependencies •Overcomes the vanishing gradient problem	•Complex architecture •May require more training time and computational resources	•Highly suitable for this task due to its ability to handle temporal dependencies and learn long-term patterns in EEG data

Based on this comparison, an RNN with LSTM architecture is the most suitable choice for predicting gait events from dual-layer EEG datasets. LSTMs (Long Short-Term Memory) can handle temporal dependencies and learn long-term patterns in the data, which are crucial for accurately predicting gait events. While CNNs (Convolutional Neural Networks) can also be used for this task, they may struggle with capturing temporal dependencies present in the data.

LSTM's ability to overcome the vanishing gradient problem makes it a better choice than traditional RNNs (Recurrent Neural Network) (Recurrent Neural Network) for this specific problem.

An RNN with LSTM architecture is an ideal choice for predicting gait events from dual-layer EEG datasets for several reasons. Here, I will provide a more detailed explanation of the key advantages of LSTM and how they relate to the problem at hand.

Handling temporal dependencies and sequential data:

EEG data is a form of time-series data, characterized by its sequential structure, as signals are recorded over time. RNNs, including LSTMs, are specifically designed to manage temporal dependencies and sequential data. They can learn patterns across time steps, which is vital for accurately predicting gait events based on the underlying neural activity. This feature gives LSTMs a significant advantage over feedforward neural networks and CNNs, which do not inherently handle temporal dependencies.

Addressing the vanishing gradient problem:

Traditional RNNs face the vanishing gradient problem, making it challenging for them to learn long-term dependencies in the data. This issue arises when the gradients of the loss function concerning the model parameters become minuscule during backpropagation, leading to minor weight updates and slow convergence. LSTMs resolve this problem through their distinct cell state and gating mechanisms, enabling them to capture and learn long-term dependencies in the data. This capability is particularly relevant for gait event prediction, as understanding the relationship between past and current neural activities can enhance prediction accuracy.

Gating mechanisms for selective memory:

LSTMs utilize gating mechanisms (input, forget, and output gates) to control the flow of information through the network. These gates help LSTMs determine what information to keep or discard at each time step, allowing them to preserve relevant information while forgetting irrelevant or outdated information. This selective memory feature is beneficial for gait event prediction, as it enables the model to adapt to varying walking patterns and maintain only the most pertinent information for precise predictions.

Versatility and adaptability:

LSTMs can be integrated with other neural network architectures to develop more complex and adaptable models. For example, a combination of CNNs and LSTMs (ConvLSTM) can be used to capture both spatial features and temporal dependencies in the data. This adaptability allows for the creation of more advanced models that can better handle the complexities of gait event prediction from dual-layer EEG datasets.

In conclusion, an RNN with LSTM architecture is highly suitable for predicting gait events from dual-layer EEG datasets because of its ability to manage temporal dependencies, learn long-term patterns, and selectively maintain relevant information. Their versatility allows for the creation of adaptable models that can better capture the complexities of gait event prediction.

We will train RRN LSTM model and explore other models for comparison in the future.

After training our models, we obtained the following classification accuracy results:

1. Motion artifact layer alone (78 channels) - 91%
2. Scalp layer (78 randomly selected good channels) - 40%
3. Scalp layer (128 channels) - 61%

While the results for the classical machine learning models are comparable, we aim to improve the accuracy by performing parameter tuning for the deep learning models. By fine-tuning the model parameters, we anticipate achieving better accuracy and potentially surpassing the performance of the classical machine learning models.

Source code:

https://github.com/RushikeshKankar/Predicting_Gait_Events_Using_Dual_layer_EEG_dataset