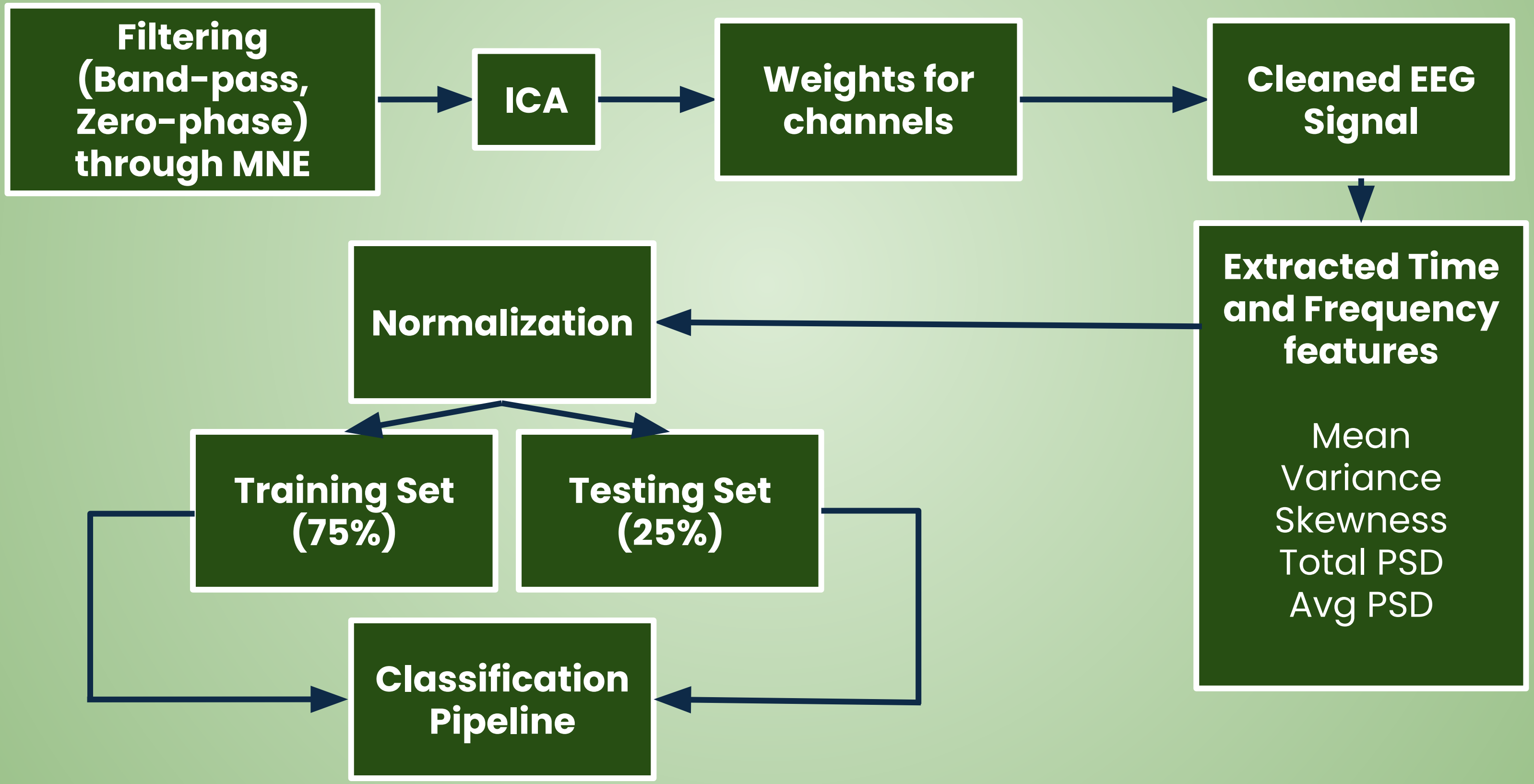
Signal Modeling Final Project

Nikhil Kuppa, Susanna Baek, Yash Bhambhani

Flowchart


Filtering (Band-pass, Zero-phase) through MNE
ICA
Weights for channels
Cleaned EEG Signal
Extracted Time and Frequency features
Mean
Variance
Skewness
Total PSD
Avg PSD
Normalization
Training Set (75%)
Testing Set (25%)
Classification Pipeline

Classification Pipeline

Train following classification models

Logistic Regression
Support Vector Machine
Decision Tree
Random Forest
Naive Bayes
K-Nearest Neighbors
Neural Network

Choose the model with the best test accuracy

Predict Y_TEST labels

01

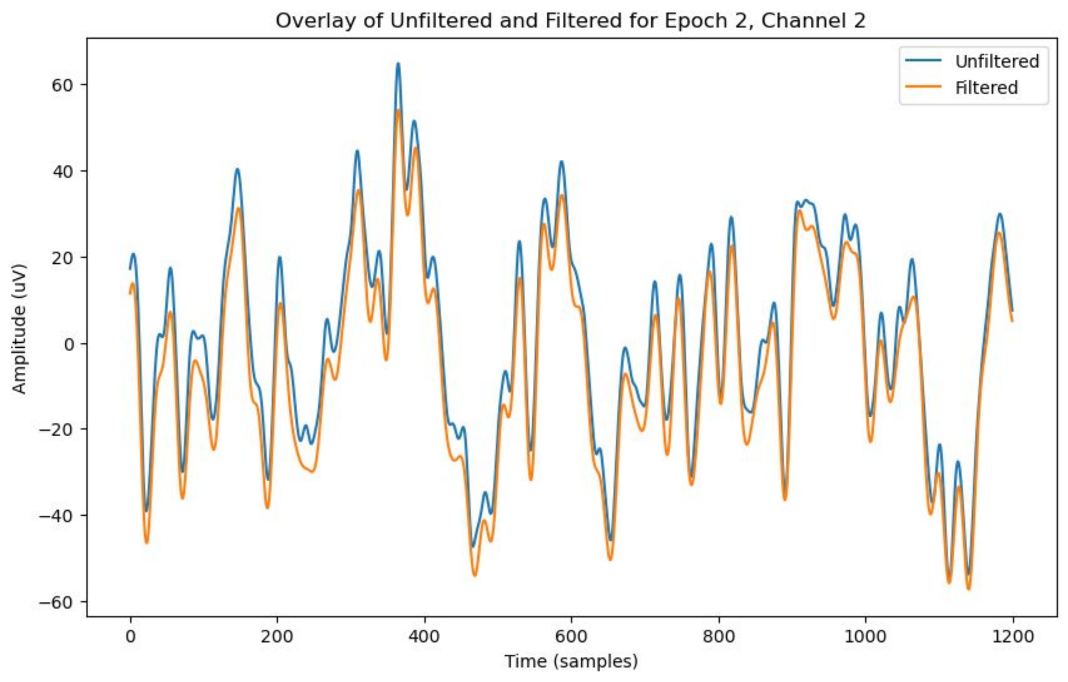
EEG Data Pre-Processing

Bandpass Filtering, Artefact Rejection, Feature Selection, Feature Normalization, Feature Reduction

Bandpass Filtering

1. Zero-phase bandpass filter between 0.5-40 Hz on each channel using **MNE Library:**

```
raw = mne.EpochsArray(eeg, info)

raw.filter(0.5,40)
```



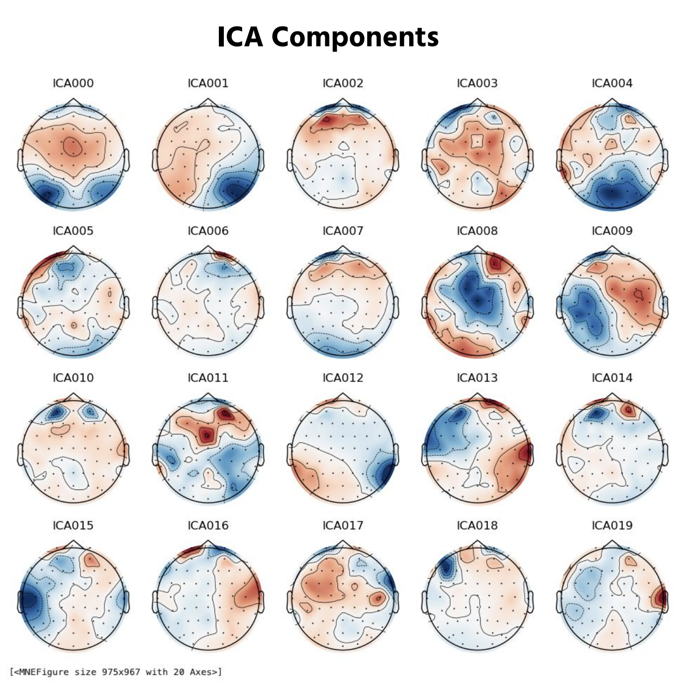
Artefact Removal

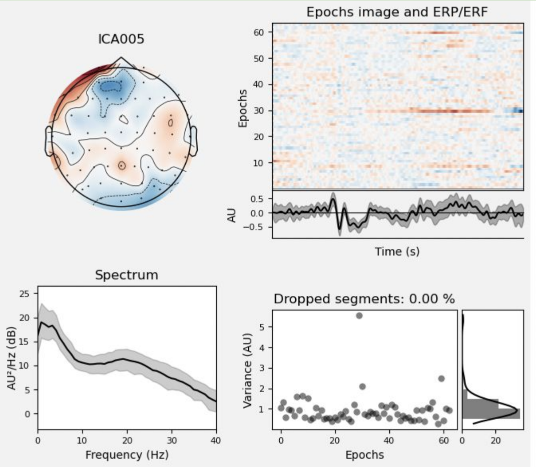
```python
ica = mne.preprocessing.ICA(n_components=20, random_state=42)
ica.fit(raw)

icalabel_df = pd.DataFrame(icalabel.label_components(raw, ica, 'iclabel'))
indices = icalabel_df.loc[icalabel_df['labels'] != 'brain'].index

ica.exclude = indices

eeg_cleaned = ica.apply(raw)

ica_weights = ica.get_components()
ica_weights = ica_weights.T
weighted_data = np.dot(ica_weights, eeg_cleaned)
weighted_data = weighted_data.T
```

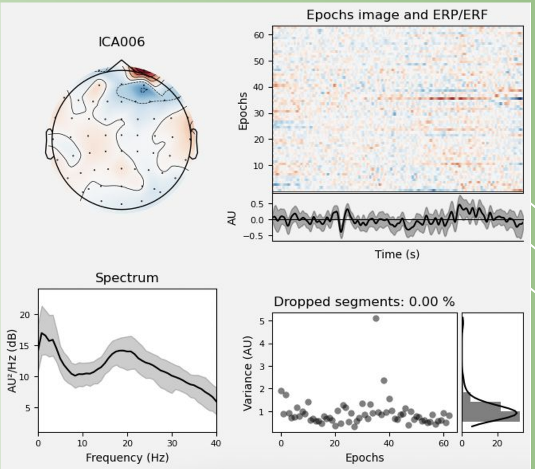
Independent Component Analysis (ICA):

- Identify and separate 20 components from zero-phase bandpass filtered data (0.5Hz-40Hz)
- Predict labels of those components
- Discard component if not predicted as ‘brain’ data
- Apply the ‘brain’ components on the filtered EEG to derive cleaned EEG brain signals
- Get the mixing matrix of brain ICA, which represents the weights learned during ICA decomposition
- Multiply weights with the clean EEG signals to get the channel weighted data of dimensions (20,1200,74)

Artefact Removal







ARTEFACTS

Feature Selection, Reduction, and Normalization

```python
# define function for extracting features
def extract_features(eeg_data, fs = 1000):
    num_channels, num_samples, num_epochs = eeg_data.shape
    feature_matrix = np.zeros((num_epochs, num_channels*5)) # 5 features per channel
#     print(feature_matrix.shape)

    for epoch_idx in range(num_epochs):
        epoch_data = eeg_data[:, :, epoch_idx]

        for channel_idx in range(num_channels):
            channel_data = epoch_data[channel_idx, :]

            # compute power spectral density
            freqs, psd = welch(channel_data, fs=fs, nperseg=fs*2)

            # compute temporal features
            mean = np.mean(channel_data)
            skewness = scipy.stats.skew(channel_data)
            variance = np.var(channel_data)

            # compute frequency domain features
            total_power = np.sum(psd)
            mean_power = np.mean(psd)

            # add features to feature matrix
            feature_matrix[epoch_idx, channel_idx*5] = mean
            feature_matrix[epoch_idx, channel_idx*5+1] = skewness
            feature_matrix[epoch_idx, channel_idx*5+2] = variance
            feature_matrix[epoch_idx, channel_idx*5+3] = total_power
            feature_matrix[epoch_idx, channel_idx*5+4] = mean_power

    return feature_matrix
```

With the pre-processed, weighted EEG data, we extract the following features:

1. **Temporal Features**: Mean, Variance and Skewness

2. **Frequency Features:** PSD and Average PSD

- These features (5) are extracted for each channel (20), over all trials
- Results in a feature matrix would have the dimensions **(n_trials, 100)**

Our algorithm then uses StandardScaler() to normalize the feature matrix

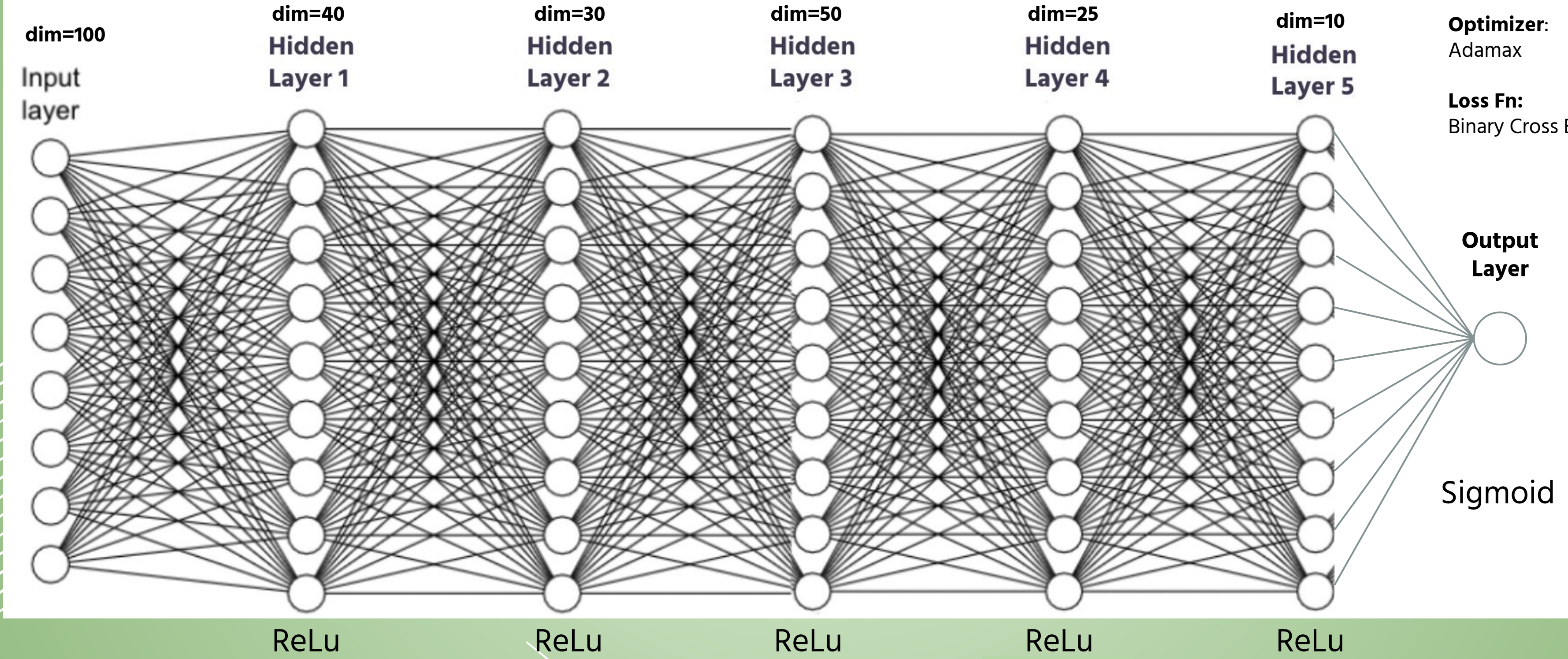
02

Classification Models

Classification Models

Logistic Regression	Since it assumes a linear relationship between features and target variable, showed low validation accuracy
Support Vector Machine (SVM)	Analyzes data and identifies the optimal boundary, or hyperplane, that separates different classes of data points with the maximum margin of separation.
Decision tree	Uses a tree-like model to make decisions and predictions by recursively partitioning the data into subsets based on the most informative features until a decision is reached.
Random Forest	Can handle high-dimensional data, and can capture non-linear relationships
Naive Bayes	Calculates the likelihood of a new data point belonging to a particular class based on the prior probability of that class and the conditional probabilities of the features, assuming independence between them.
K-Nearest Neighbors (KNN)	Classifies a new data point by comparing it to the K closest data points in the training set, where K is a user-defined parameter, and assigns it to the class that is most prevalent among its nearest neighbors. Works well if EEG data has clear boundaries between the two classes
Neural Networks	Learning algorithm that mimics the structure and function of the human brain by using interconnected nodes, or artificial neurons, to process and transform input data through multiple layers to generate output predictions or decisions.

Neural Network Architecture



Optimizer: Adamax

Loss Fn: Binary Cross Entropy

03

Accuracies

Accuracies

Machine Learning Models

	Logistic Regression	Support Vector Machines	Decision Trees	Random Forest	Naive Bayes	K-Nearest Neighbor
Accuracy	0.526316	0.473684	0.473684	0.315789	0.368421	0.368421
Precision	0.666667	0.6	0.75	0.4	0.5	0.5
Recall	0.5	0.5	0.25	0.166667	0.166667	0.416667
Accuracy	0.625	0.5	0.4375	0.8125	0.75	0.5
Precision	0.571429	0.444444	0.375	0.75	1	0.4
Recall	0.571429	0.571429	0.428571	0.857143	0.428571	0.285714
Accuracy	0.578947	0.473684	0.684211	0.684211	0.631579	0.631579
Precision	0.75	0.625	0.8	0.8	1	0.777778
Recall	0.5	0.416667	0.666667	0.666667	0.416667	0.583333
Accuracy	0.555556	0.555556	0.666667	0.555556	0.666667	0.666667
Precision	0.8	0.714286	0.777778	0.8	0.727273	0.727273
Recall	0.363636	0.454545	0.636364	0.363636	0.727273	0.727273
Accuracy	0.722222	0.722222	0.277778	0.444444	0.5	0.5
Precision	0.888889	0.888889	0.4	0.75	0.8	1
Recall	0.666667	0.666667	0.166667	0.25	0.333333	0.25
Accuracy	0.631579	0.578947	0.631579	0.736842	0.789474	0.684211
Precision	0.727273	0.7	0.727273	0.769231	0.833333	0.75
Recall	0.666667	0.583333	0.666667	0.833333	0.833333	0.75
Accuracy	0.473684	0.473684	0.631579	0.526316	0.578947	0.631579
Precision	0.666667	0.666667	0.857143	0.666667	0.642857	0.857143
Recall	0.333333	0.333333	0.5	0.5	0.75	0.5
Accuracy	0.526316	0.526316	0.578947	0.684211	0.631579	0.631579
Precision	0.636364	0.615385	0.75	0.75	0.777778	0.727273
Recall	0.583333	0.666667	0.5	0.75	0.583333	0.666667

Selected Models

	NN Acc	ML Acc
Subject_1	0.578947	0.526316
Subject_2	0.625	0.8125 - RF
Subject_3	0.578947	0.684211 - DT
Subject_4	0.833333	0.666667
Subject_5	0.666667	0.722222 - LR
Subject_6	0.789474	0.789474 - NB
Subject_7	0.631579	0.631579 - DT
Subject_8	0.736842	0.684211

Overall Test Acc	0.7236

04

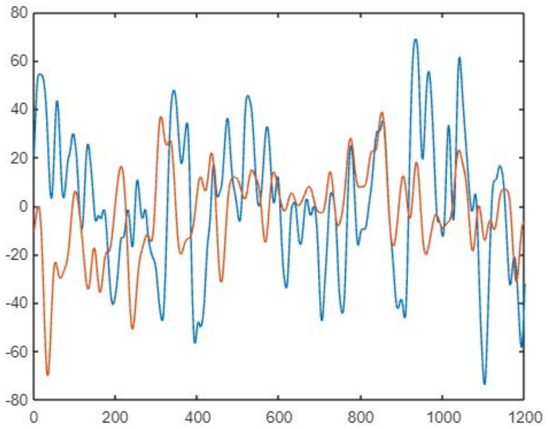
Alternate Approaches

Bandpass Filtering

1. Zero-phase filter between 0.5-40 Hz on each channel using **butterworth:**

```
fs = 1000; % Sampling rate
fpass = [0.5 40]; % Passband frequency range, test that the filter is working by changing the frequency range
order = 4; %

% Create the filterlength
[b,a] = butter(order,fpass/(fs/2),'bandpass');

% Save filtered data train
data_filtered_save = cell(1, length(files));

for i = 1:length(files)
    load(files(i).name, '-mat');
    data_filtered = zeros(size(X_EEG_TRAIN));
    for j = 1:size(X_EEG_TRAIN, 3)
        % Apply the filter to each trial
        for k = 1:size(X_EEG_TRAIN, 1)
            % Apply the filter to each channel
            data_filtered(k, :, j) = filtfilt(b, a, X_EEG_TRAIN(k, :, j)');
        end
    end
    data_filtered_save{i} = data_filtered;
    % % Save the filtered data to a separate file
    % filename = sprintf('filtered_train_data_%d.mat', i);
    % save(filename, 'data_filtered');
end
% save('filtered_train_data.mat', 'data_filtered_save');
```

```
% test that code above is only filtering one dimension, the trials
figure(1);
plot(X_EEG_TRAIN(1,:,1));
hold on;
plot(data_filtered_save{1}(1,:,1));
hold off;
```



Feature Selection, Normalization, Reduction

1. **LassoGLM** to select features from both time points and channel: feature*samples

nel*time points)*trials

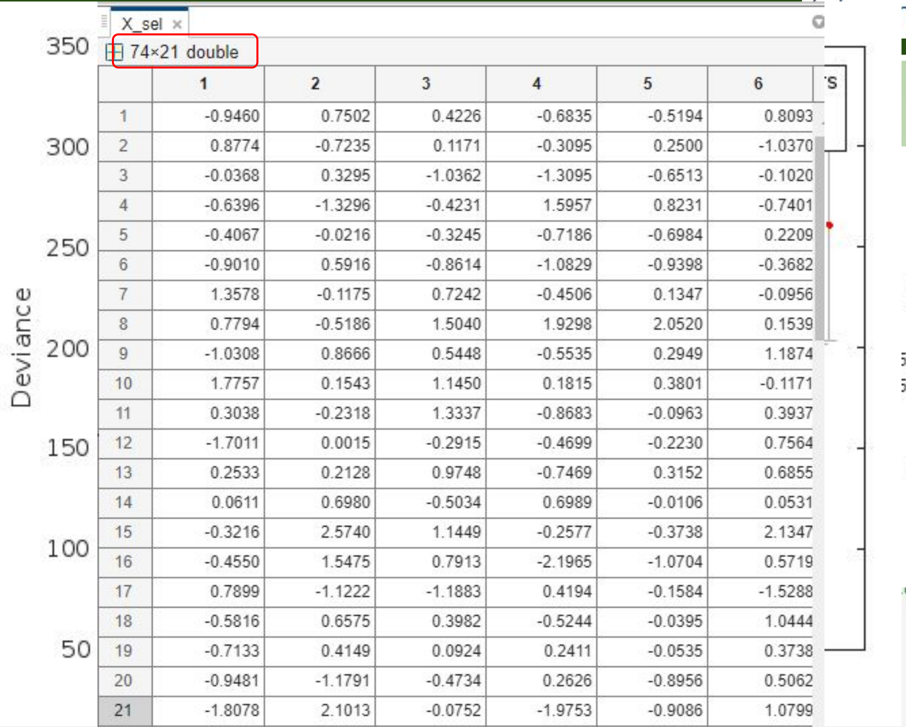
```
FitInfo =

  struct with fields:

           Intercept: [-0.2608 -0.2          -0.2377 -0.2343 -0.2310 … ]
              Lambda: [1.6306e-04 1          3.4322e-04 3.7668e-04 … ]
               Alpha: 1
                  DF: [58 54 57 57          55 55 56 56 53 53 52 49 54 … ]
            Deviance: [260.9535 258          51 235.1252 232.4592 … ]
      PredictorNames: {}
       UseCovariance: 0
                  SE: [57.6261 56.8          50.8536 50.3217 49.7599 … ]
   LambdaMinDeviance: 0.0912
           Lambda1SE: 0.2311
    IndexMinDeviance: 69
            Index1SE: 79
```

```
    mincoefs
    % Create
    X_sel =
    lasso_sa
end
```

X_sel

74×21 double

	1	2	3	4	5	6
1	-0.9460	0.7502	0.4226	-0.6835	-0.5194	0.8093
2	0.8774	-0.7235	0.1171	-0.3095	0.2500	-1.0370
3	-0.0368	0.3295	-1.0362	-1.3095	-0.6513	-0.1020
4	-0.6396	-1.3296	-0.4231	1.5957	0.8231	-0.7401
5	-0.4067	-0.0216	-0.3245	-0.7186	-0.6984	0.2209
6	-0.9010	0.5916	-0.8614	-1.0829	-0.9398	-0.3682
7	1.3578	-0.1175	0.7242	-0.4506	0.1347	-0.0956
8	0.7794	-0.5186	1.5040	1.9298	2.0520	0.1539
9	-1.0308	0.8666	0.5448	-0.5535	0.2949	1.1874
10	1.7757	0.1543	1.1450	0.1815	0.3801	-0.1171
11	0.3038	-0.2318	1.3337	-0.8683	-0.0963	0.3937
12	-1.7011	0.0015	-0.2915	-0.4699	-0.2230	0.7564
13	0.2533	0.2128	0.9748	-0.7469	0.3152	0.6855
14	0.0611	0.6980	-0.5034	0.6989	-0.0106	0.0531
15	-0.3216	2.5740	1.1449	-0.2577	-0.3738	2.1347
16	-0.4550	1.5475	0.7913	-2.1965	-1.0704	0.5719
17	0.7899	-1.1222	-1.1883	0.4194	-0.1584	-1.5288
18	-0.5816	0.6575	0.3982	-0.5244	-0.0395	1.0444
19	-0.7133	0.4149	0.0924	0.2411	-0.0535	0.3738
20	-0.9481	-1.1791	-0.4734	0.2626	-0.8956	0.5062
21	-1.8078	2.1013	-0.0752	-1.9753	-0.9086	1.0799
22	1.1106	0.0665	-1.0859	-0.2763	-0.7713	-1.4944
23	0.2690	-0.2198	0.2345	1.1327	0.9118	0.0598
24	-1.1424	0.0378	0.4415	-1.0978	-0.2886	1.1352



Kernel SVM Accuracies

Kernel SVM for binary classification of face vs. trial, **CNN**, **Logistic Regression**

a
accuracy_train_hol
accuracy_train_kfol
accuracy_train_leav

```
    %% Predict the labels for the test data
    % Y_pred_leaveout_label = predict(SVMModel,X_EEG_TEST);

    %% Evaluate the classification performance
    % accuracy_leaveout = sum(Y_pred_leaveout == Y_EEG_TEST) / numel(Y_EEG_TEST);

    %% Predict the labels for the test data
    % Y_pred_kfold_label = predict(SVMModel,X_EEG_TEST);

    %% Evaluate the classification performance
    % accuracy_kfold= sum(Y_pred_kfold == Y_EEG_TEST) / numel(Y_EEG_TEST);

    %% Predict the labels for the test data
    % Y_pred_holdout_label = predict(SVMModel,X_EEG_TEST);

    %% Evaluate the classification performance
    % accuracy_holdout= sum(Y_pred_holdout_label == Y_EEG_TEST) / numel(Y_EEG_TEST);

    % % Choose highest predicted test method and save into 1x8 cells variable
    % highest_predicted_test = max([Y_pred_leaveout_label, Y_pred_kfold_label, Y_pred_holdout_label]);
    % highest_predicted_test_save{p} = highest_predicted_test;
    % save('Predicted_Test_Subjects','highest_predicted_test_save');

    % % Choose highest accuracy test method and save into separate .mat file
    % highest_accuracy_test = max([accuracy_kfold, accuracy_leaveout, accuracy_holdout]);
    % highest_accuracy_test_save{p} = highest_accuracy_test;
end
```

References

1. Sajda, P., Philiastides, M.G. and Parra, L.C. (2009) “Single-trial analysis of neuroimaging data: Inferring neural networks underlying perceptual decision-making in the human brain,” *IEEE Reviews in Biomedical Engineering*, 2, pp. 97–109.

THANK YOU!

Classification Pipeline

Train following classification models

Logistic Regression
Support Vector Machine
Decision Tree
Random Forest
Naive Bayes
K-Nearest Neighbors
Neural Network

→

Choose the model with the best test accuracy

→

Predict Y_TEST labels

Bandpass Filtering

Zero-phase filter between 0.5-40 Hz on each channel using **MNE Python library window FIR followed by artifact removal**

Designing a one-pass, zero-phase, non-causal bandpass filter:

- Windowed time-domain design (firwin) method
- Hamming window with 0.0194 passband ripple and 53 dB stopband attenuation
- Lower passband edge: 0.50
- Lower transition bandwidth: 0.50 Hz (-6 dB cutoff frequency: 0.25 Hz)
- Upper passband edge: 40.00 Hz
- Upper transition bandwidth: 10.00 Hz (-6 dB cutoff frequency: 45.00 Hz)
- Filter length: 6601 samples (6.601 sec)

Appendix: Features

- **LassoGLM:** Helps in reducing overfitting by shrinking the coefficients of irrelevant features to zero, can handle a large number of features effectively, can identify important features by setting the coefficients of unimportant features to zero.
  - May result in underfitting if the regularization parameter is too high. May be sensitive to the choice of the regularization parameter. Assumes that the relationship between the response variable and the predictors is linear.

- **Temporal Features:** Simple and easy to compute, may be useful in capturing temporal patterns in the data.
  - May not be sufficient to capture complex relationships in the data, may not be effective in distinguishing between classes if the temporal patterns are similar.

- **Frequency Features:** May be useful in capturing frequency-specific information, may be effective in distinguishing between classes if there are frequency differences between them.
  - May not be sufficient to capture complex relationships in the data, may be affected by noise.

- **Linear Discriminant Analysis (LDA):** Can effectively reduce the dimensionality of the data, can be useful in identifying the features that are most effective in distinguishing between classes, can handle small sample sizes.
  - Assumes that the covariance matrices of the classes are equal, may not be effective in distinguishing between classes if the within-class variance is large.

- **Principal Component Analysis (PCA):** Can effectively reduce the dimensionality of the data, can be useful in identifying the features that explain the most variance in the data.
  - Does not consider the class labels, which may result in suboptimal feature selection, may not be effective in capturing non-linear relationships in the data.

- **Independent Component Analysis (ICA):** Can effectively separate signals from artifacts, can be useful in identifying the features that are most effective in distinguishing between classes.
  - Requires a large dataset to estimate the mixing matrix, may be sensitive to the choice of the algorithm and parameters used for estimation.

Appendix : Models

- **Logistic Regression:** Easy to implement and interpret, can handle both binary and multiclass classification problems, can estimate the probabilities of each class.
  - Assumes a linear relationship between the features and the target variable, May not perform well when there are non-linear relationships between the features and the target variable.

- **SVM:** Can handle both linear and non-linear relationships between the features and the target variable, Can handle high-dimensional data effectively, Can be effective in separating classes that are not linearly separable in the original feature space.
  - Can be computationally expensive for large datasets, May be sensitive to the choice of kernel and hyperparameters.

- **Decision Tree:** Easy to interpret and explain, can handle both numerical and categorical data. Can handle nonlinear relationships between the features and the target variable.
  - Can be prone to overfitting if the tree is too deep, Can be sensitive to small changes in the data.

- **Random Forest Classifier:** Can handle both numerical and categorical data, can handle high-dimensional data effectively. Can be effective in reducing overfitting by combining multiple decision trees.
  - Can be computationally expensive for large datasets, May be sensitive to the choice of hyperparameters.

- **Naive Bayes:** Can handle high-dimensional data effectively, can be effective in classifying data with many features and few observations. Can be trained quickly and with less data.
  - Assumes that the features are conditionally independent given the target variable, May not perform well if this assumption is violated.

- **KNN:** Does not make any assumptions about the underlying distribution of the data. Can handle both binary and multiclass classification problems, can be effective in capturing local structure in the data.
  - Can be sensitive to the choice of k, can be computationally expensive for large datasets.

- **Neural Networks:** Can handle both linear and nonlinear relationships between the features and the target variable, Can handle high-dimensional data effectively. Can be effective in capturing complex relationships in the data.
  - Can be computationally expensive for large datasets, May be prone to overfitting if the network is too complex.