Weekly Progress

Nikhil Anand

Week 1

30th May, 2023

Nikhil Anand

Progress

- Accessed the data from the server (space_lin1) and went over all the datasets/looked at the file structure.
- Read up on the Human Connectome Project.
- Started working on the first research paper on ML analysis of EEG for brain age determination.

First Paper

- Gasser et al.
  - Relative power **increases with age** for **fast bands** and **decreases with age for slow bands**.
- Many other previous studies were done with these kinds of analyses.

First Paper

- 468 individuals (297 females) - divided between:
  - mood/anxiety
  - eating disorder
  - substance use disorder
  - healthy
- 5 sets of preprocessed EEG features across channels/freq bands
- They used stacked ensemble learning (meta classifiers) and nested cross val to train from EEG features to predict age.

Week 2

6th June, 2023

Nikhil Anand

My Progress

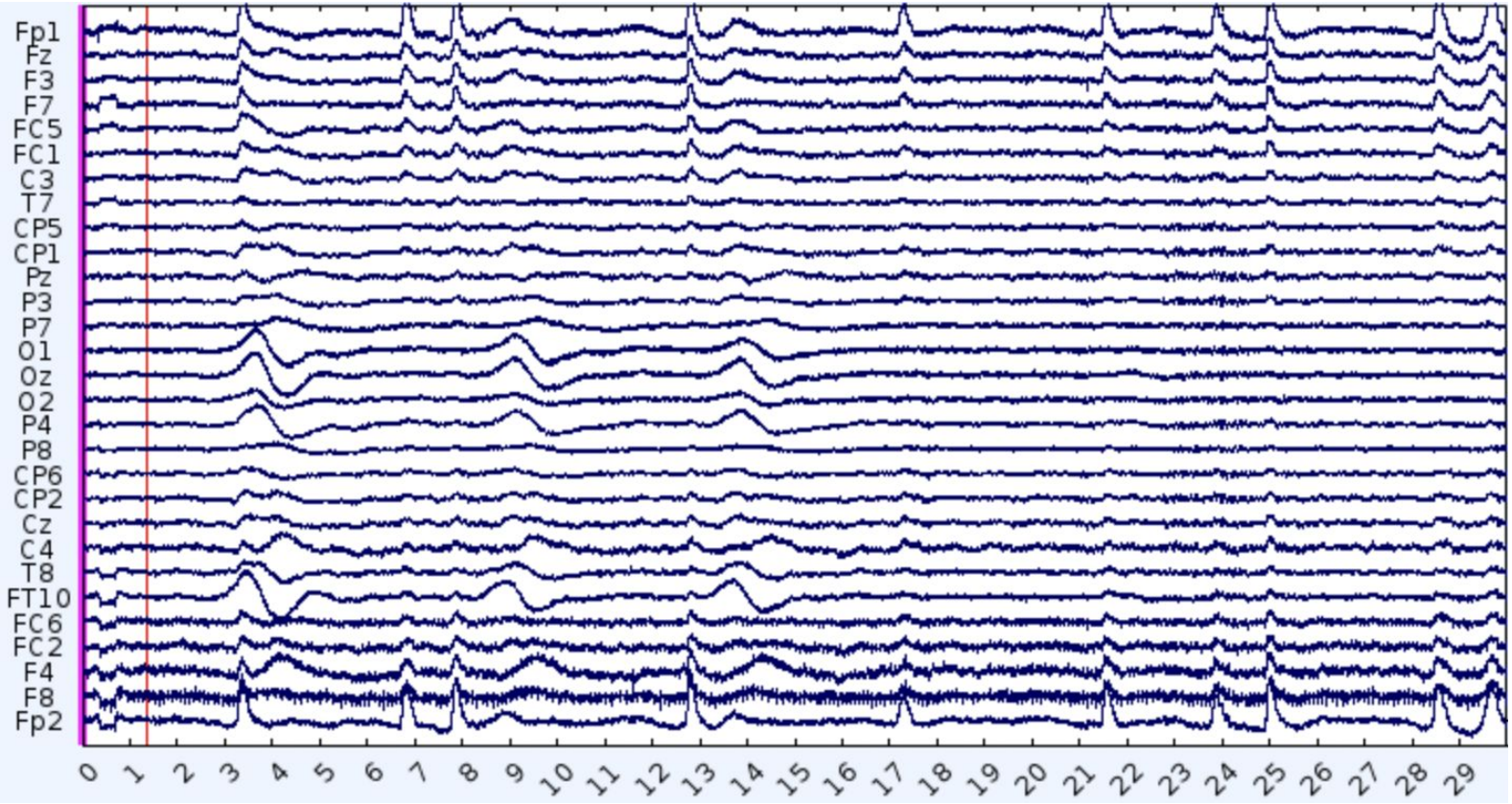
- Prepared for the chapter 3 (fMRI book) for Wednesday's book club presentation
- Worked on EEG preprocessing tasks and completed preprocessing of subjects 0001, 0002, 0332, 0333, 0334.
- Completed a literature review of 6 important papers for predicting brain age from EEG.

EEG Tasks

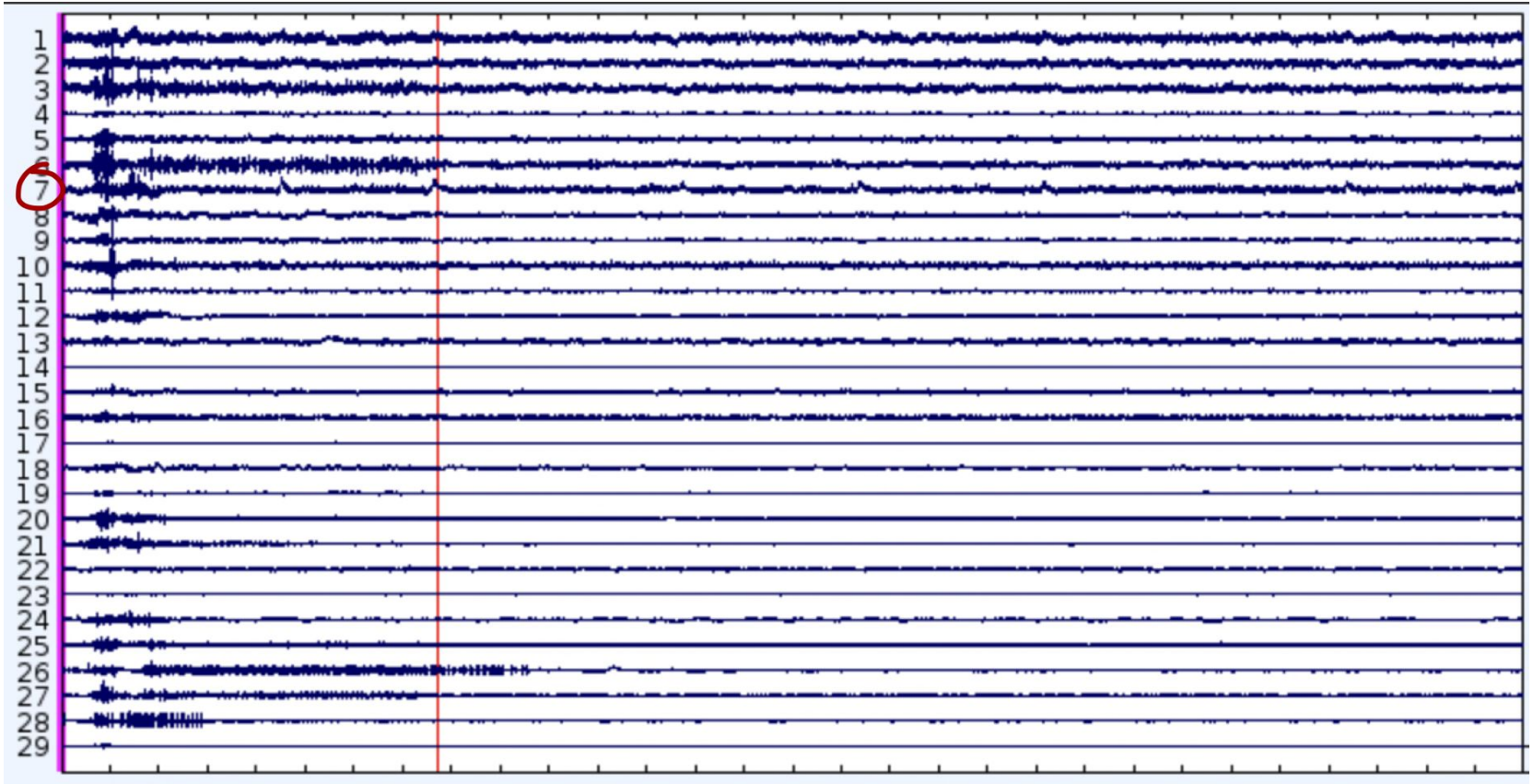
Steps summarised:

- Applying notch filter around 60 Hz, and band pass filter above 1 Hz.
- Re-referencing the data against TP9 and TP10.
- Decomposing the data by ICA.
- Viewing the component maps.
- Removing typically the first component (eye artifacts).

Data from Subject 0001



Components after ICA



Component Map

Issues

- The eye artifact isn’t the first or most prominent component in this case.
- Not sure why?

Some Common Sections Among EEG Papers

- Description of the dataset
- Age range
- Diseases of the demographic
- Sampling rate, epoch length
- Preprocessing strategies
- ML Architecture used
- Findings and Metrics

Given this, the literature review table has columns according to these columns.

Lit Review

Paper	Modality	Year	Author	N	Age range	Disease	Dataset	EEG Sampling rate and Epoch length	Preprocessing	Architecture	Unique Features/Findings	Metrics	Important References
Predicting BrainAGE from brain EE(	EEG	2018	Zoubi et al	468	20 - 50 years	Healthy Mood/anxiety Substance use Eating disorders	Tulsa-1000	5000 Hz and 60 s	After feature extracti	*Nested cross validation* Overall cross validation of an inne **Models used:** ENET, SVR, XgbTr R repeats of each model used in a Ensemble of outputs from all RF n	best accuracy = SVR with rt R2 = 0.34 MAE = 7.01 years RMSE = 8.7 years best performance = stack e R2 = 0.37 MAE = 6.87 years RMSE = 8.46 years	R2 MAE	https://www.front
Brain age from the electroencephal(	EEG	2019	H Sun et al	*MGH*: 2532 (167 *SHHS*: 1974	MGH: 18-80 SHHS: 40-80	Healthy Significant neuro	MGH and SHHS visit 1	200 Hz and 30 s	Notch-filtered at 60H Bandpass 0.5-20Hz 1 Hz harmonic (ECG Ampl > 500 uV	**Softplus regression** yi = BA = softplus(wTxi) Minimise MSE of this, with regular So BA is close to CA but the varia	Prediction using macrostruc MAE = 23.3 years Using 510 EEG microstructu MAE = 7.8 years	MAE	
Excess brain age in the sleep electr	EEG	2020	L Paixao et al	4887	40-80 years		SHHS Visit 1	200 Hz and 30 s	*Like above*: 102 'mic So 102x5=510 featu *Additional:* LE compi	Same as above	individuals with higher BAI (BA - CA) have lower LE	MAE	
Brain Age Prediction/Classification t	EEG	2022	K Jusseaume et	564	2-88 years	Seizure patients Stroke (12%)	Temple Uni Abn(	87% is 250 Hz, remaining Epoch length 8 min	Resampling done to Band pass filter 0.5 - individual freq bands	**LSTM-based model (RNN)** (LSTM -> dropout)x2 -> Dense **BLSTM = bidirectional LSTM** BLSTM -> norm -> (LSTM -> norm Optimiser: ADAM	**LSTM** MAE = 12.5, RMSE = 15.1 **BLSTM** MAE = 7.3, RMSE = 10.07	MSE, MAE, RMSE	
Brain age prediction based on EEG	EEG	2022	M Klymenko et a	7048	0-100 years	Mix of inpatients and outpatients (	Greater Vancouv	500 Hz or 512 Hz	Resampling done to Flat (no signal) inter Band pass 1-55 Hz Special procedures PAA done to reduce BPE: symbol encodi	BPE algorithm (symbols are comb Commonly occuring groups are tol we also tokenize relative distances then we count number of occurren *Model*: RF (RAPIDS for faster com	MAE = 15.9 years if we use and relative tokenising featu low? due to nature of datase	MAE	
NeurML : From an automated EEG	EEG	2020	T Segré et al	1371	6-18 years	healthy (underst	Healthy Brain Network (Child Mind Institute)		High pass filter 0.5 H Notch filter around 6 Rejects signal above ICA applied (infomax features extracted fr FOOOF and NICE li Correlations observe	*Age regression:* lasso regression a *Sex classification:* KNN, DT, MLP, GaussNB, NN, RF, Adaboost, logi	*Sex:* Best = VotingClassifier MinMaxScaler, KNN + RF + **Accuracy = 78.02%** *Age:* regular regression **MA** with lasso, **MAE = 2.88**	MAE	

Next Week's Plans

- Complete the preprocessing of Quanta dataset on server through script.
- Set up PyTorch environment on server.
- Begin implementation. Try to implement one paper and tune it over time. That's the best strategy.

Week 3

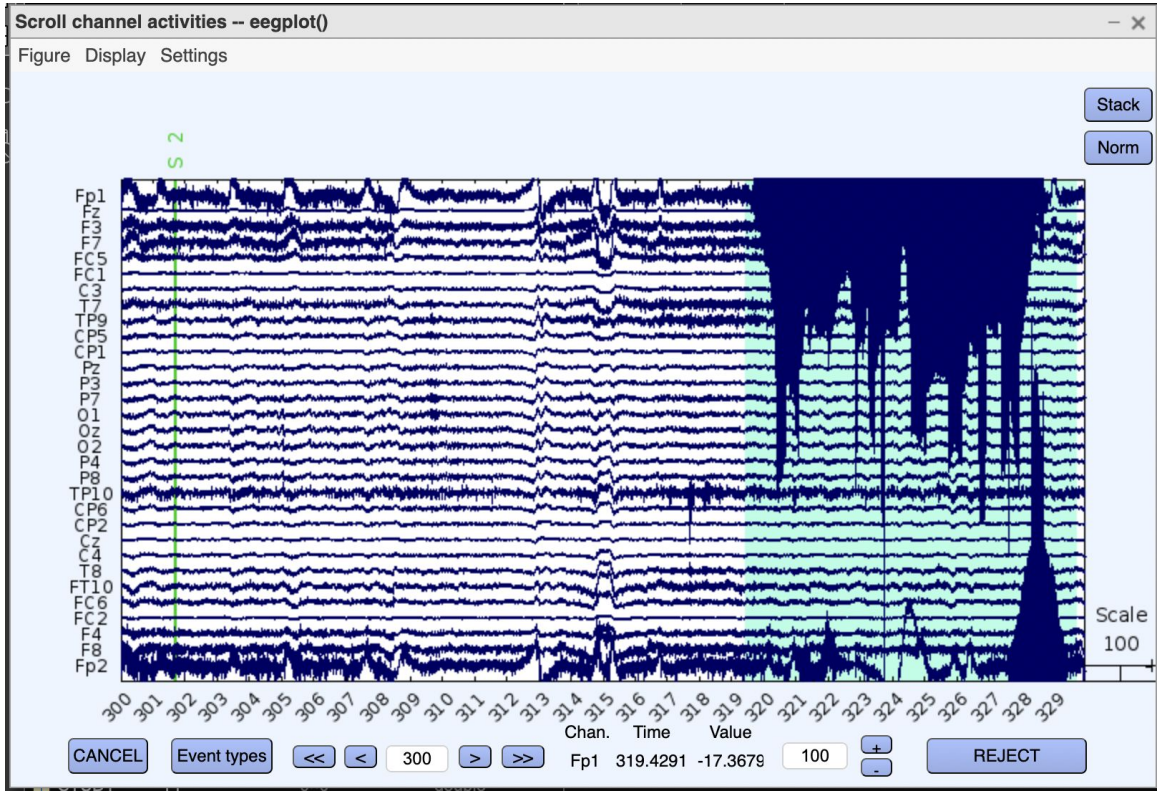
13th June, 2023

Nikhil Anand

Progress

- Did an in-depth review of methods of the first paper, for incorporation into the pipeline.
- Detailed the pipeline that I will be following.
- Completed EEG Tasks for Preprocessing (completed the rejection of noise and decomposed it again).
- Set up PyTorch locally. Completed some tutorials.
- Wrote code for feature extraction using MNE library.

EEG Analysis Noise Rejection



Pipeline

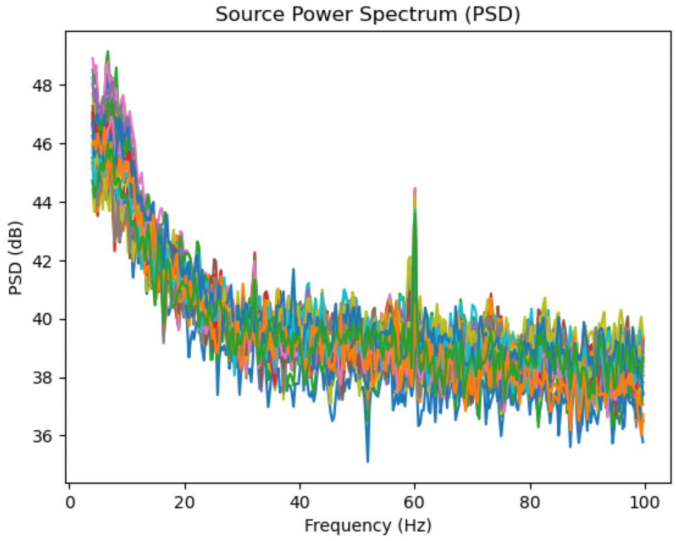
- ***Preprocessing:***
  - Make a MATLAB script to automate this for the entire quanta dataset. Save the preprocessed data onto another folder in my directory in the server.
  - The script should run the following preprocessing steps:
    - Filtering the data (notch at 60 Hz and bandpass above 1 Hz)
    - Re-referencing the data (to TP9 and TP10)
    - Doing an ICA decomposition and removing the eye blink artifacts
    - Storing the signal data into a new file for all the files
- ***Feature extraction:***
  - I can perform this in python after the datasets are preprocessed. Libraries such as FOOOF and NICE can be used for this. MNE can be used to import the data and also for feature extraction.
  - First filter into different frequency bands and then extract various features from those bands.
    - Signal power
    - Signal envelope
    - Peak-to-peak
    - Power spectral density
    - Edge frequency
    - Entropy

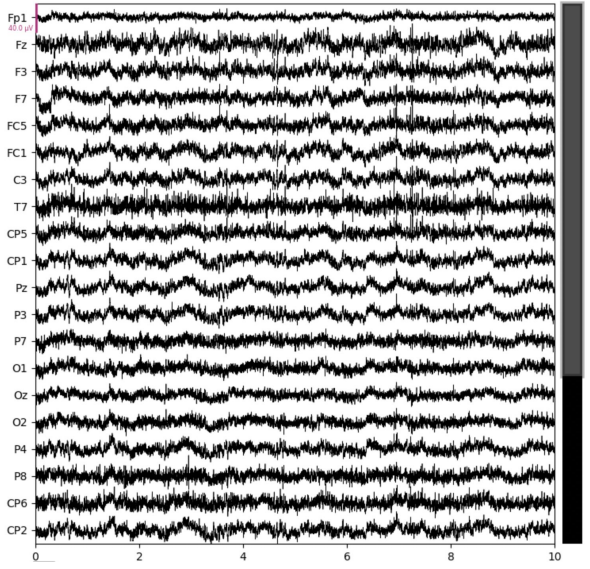
Pipeline

- ***Model:***
  - Implement the following:
    - ENET
    - SVR with rbf Kernel
    - RF
    - XgbTree
    - gaussprPoly
  - Each model should be repeated R times with a randomised K-fold subset of the data and averaged to get the net prediction of each model.
  - Then all models are ensembled using the model with the best accuracy.
- ***Analysis:***
  - Show the results (highest BrainAGE accuracy), and show the most accurate model for the given data.
  - Show which features were best correlated with BrainAGE.

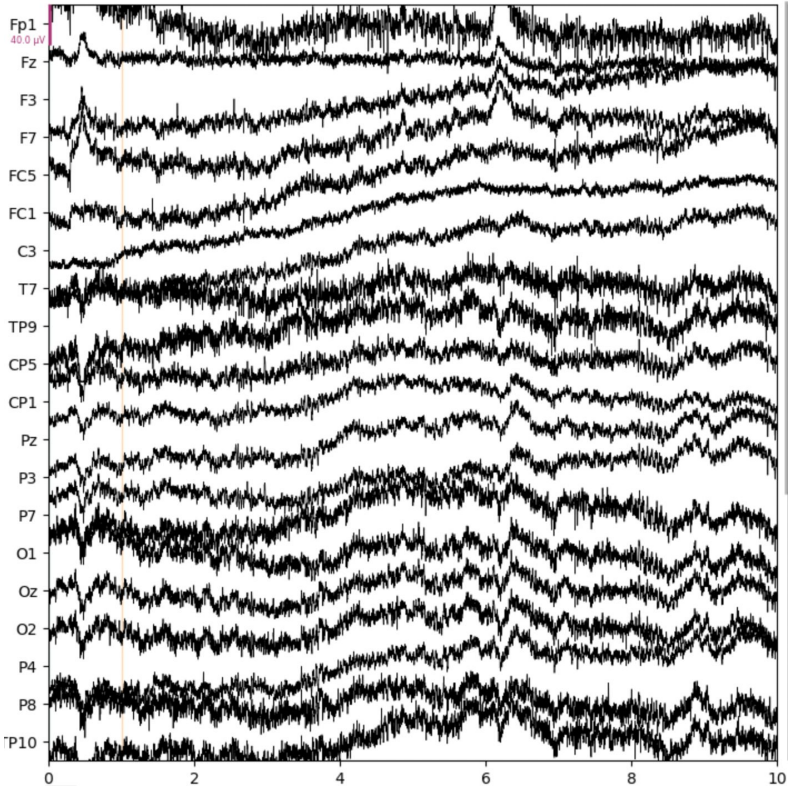
Feature Extraction – Coding

- Imported sub-0334 preprocessed data and visualised in Python.
- Implemented Power Spectral Density calculation for public dataset.

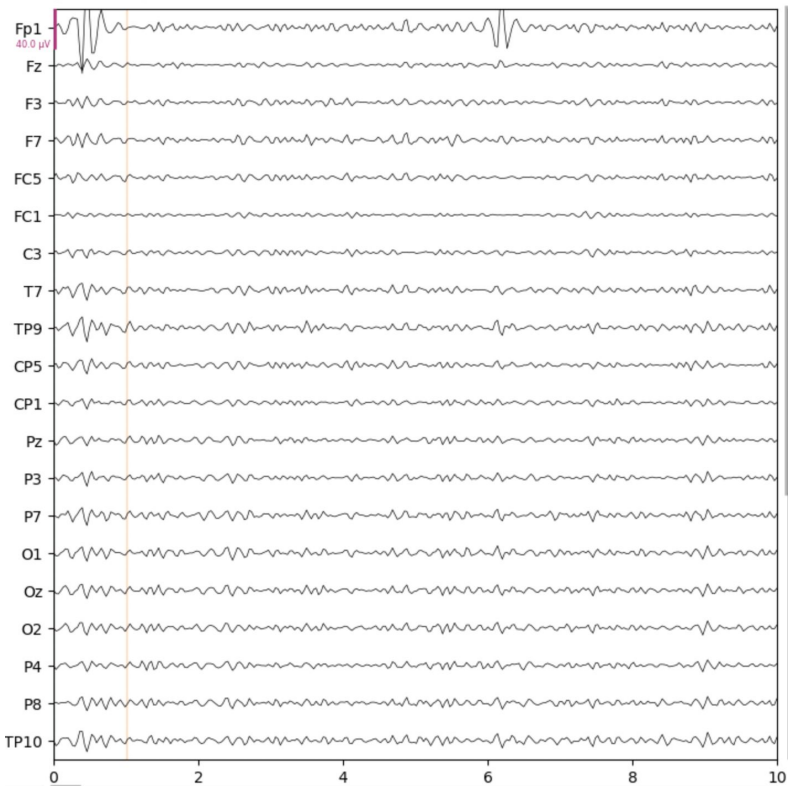


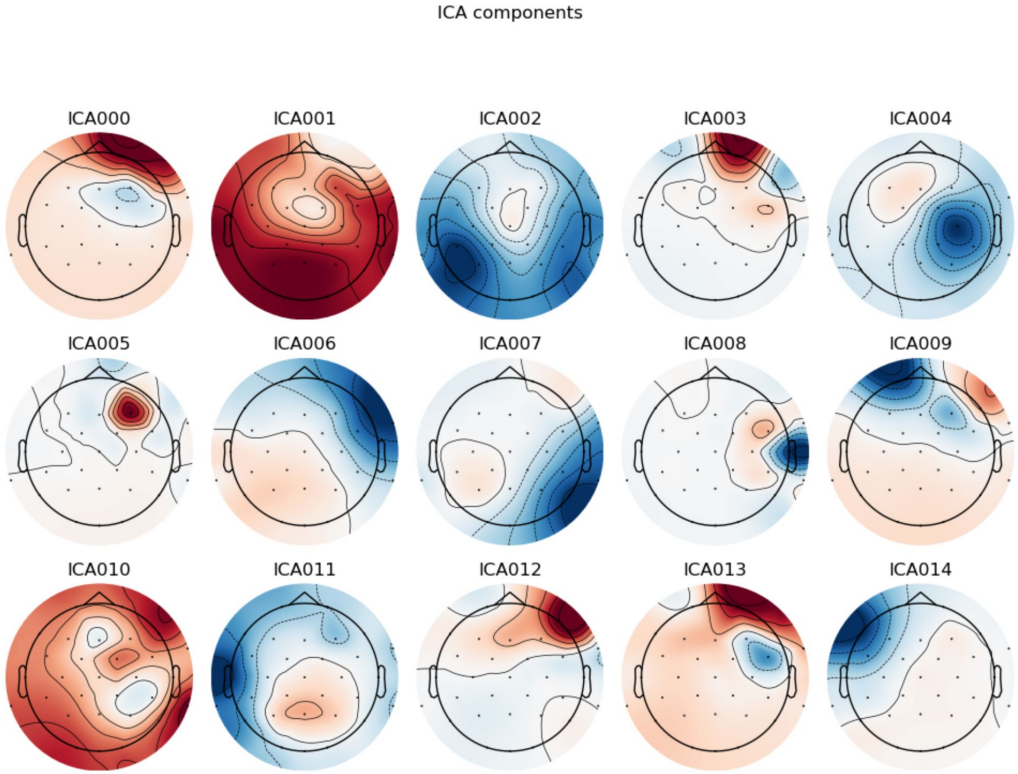


Feature Extraction – Coding




```
raw.filter(5,10)
```




ICA components
ICA000
ICA001
ICA002
ICA003
ICA004
ICA005
ICA006
ICA007
ICA008
ICA009
ICA010
ICA011
ICA012
ICA013
ICA014

Next Week

- Integrate Jupyter with server and develop MATLAB preprocessing scripts to run on server.
- Move the code to the server.
- Complete the code development of the pipeline in Python.

Week 4

20th June, 2023

Nikhil Anand

Progress

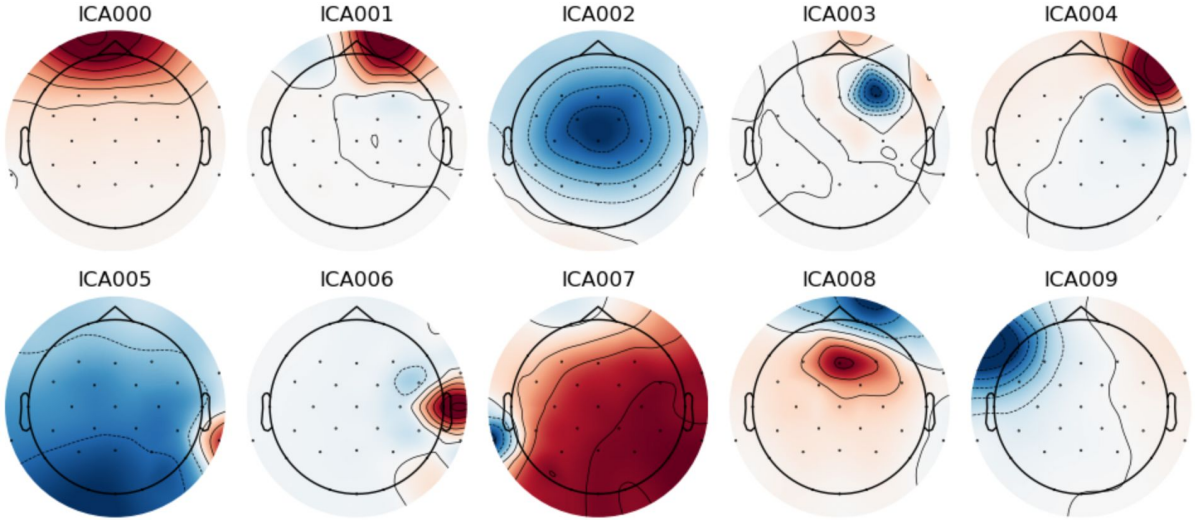
- Completed server environment setup.
- Wrote Python script for complete preprocessing. Ran this in the server and preprocessed 150 data-points of quanta dataset, and exported all their topographies.
- Wrote feature extraction code.

Preprocessing Script

Repeat 150 times

1. Read the raw Brainvision file
2. Filter, Notch filter
3. Re-reference
4. Run ICA and plot components
5. Write topography to PDF
6. Exclude the first component
7. Save to a .set file

Preprocessing Script



Sub #:	Notes
1	Check - first component not eyeblink
2	Good
3	Good - but component first component is blue
7	Good - two components are eyeblink
8	Good - but first component is blue and second component is red
9	Good
10	Good
13	Good - but component first component is blue
14	Good
15	Good
16	Good
25	Good - but component first component is blue
26	Good - but first component is blue and second component is red
27	Good - but component first component is blue
28	Check - first two components are ambiguous
29	Good - but component first component is blue

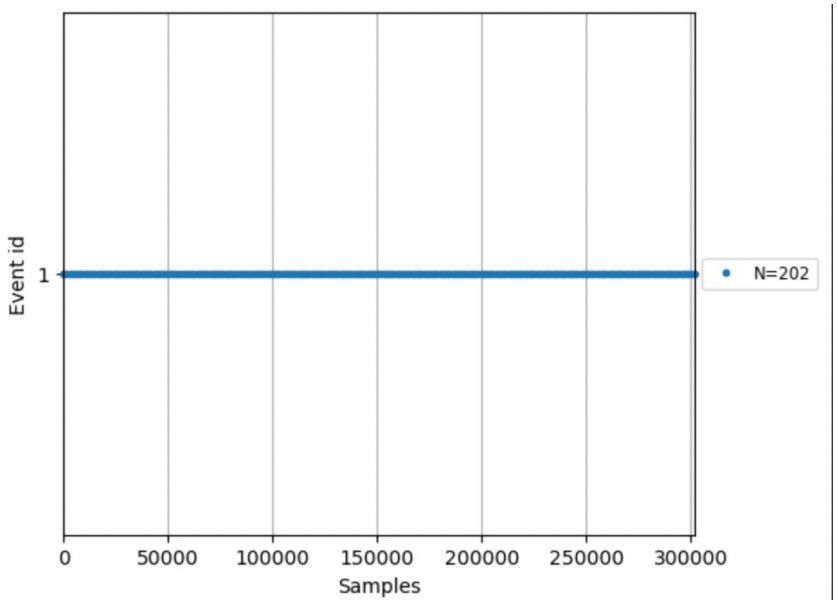
Preprocessing Script – To Be Done

- Need to export all the overlay plots for analysis.
- Analyse the problematic .set files and manually preprocess them.

Feature Extraction

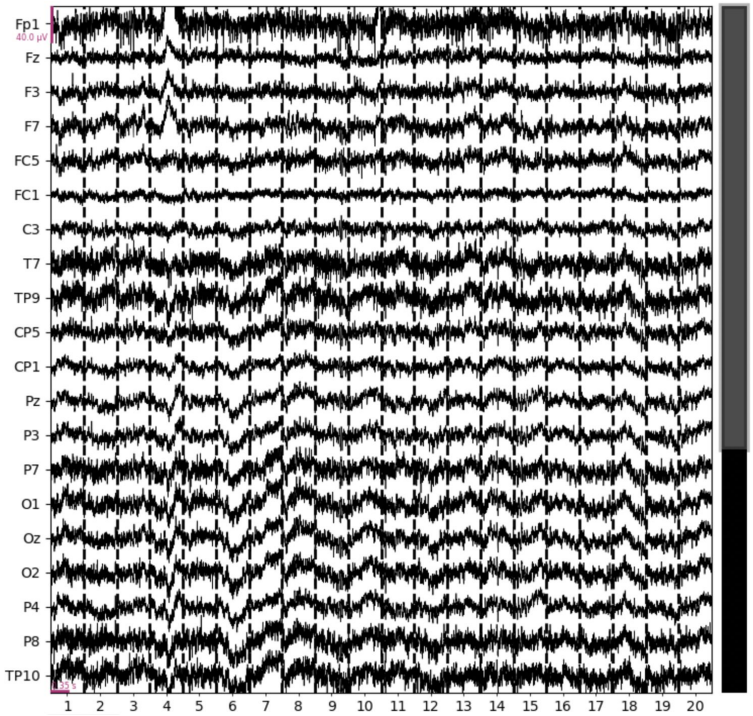
NOTE: We are using subject 334's data for the testing

- Divided data into epochs based on annotations
- Divided data into time-based epochs using artificial events
- Plotted events



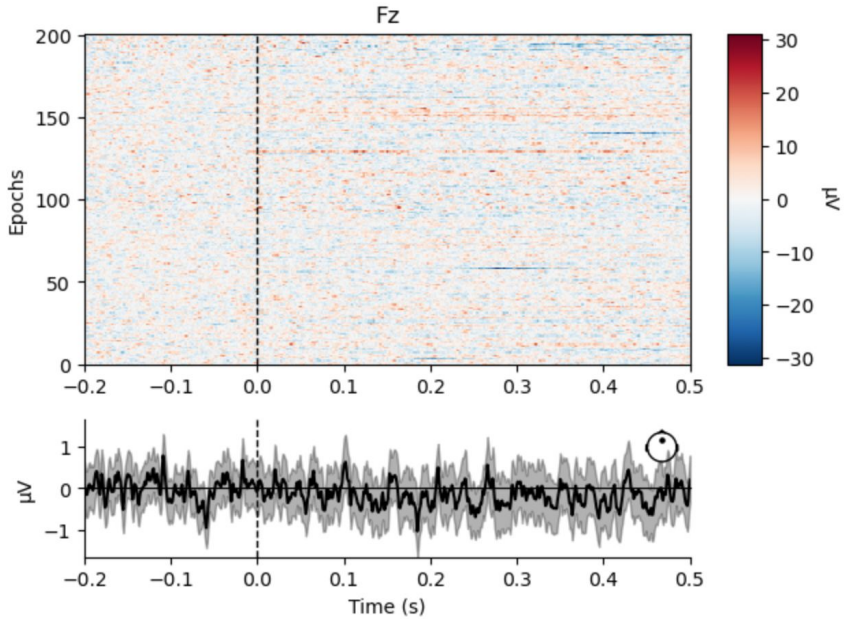
Feature Extraction

- Made an epochs object and plotted it.



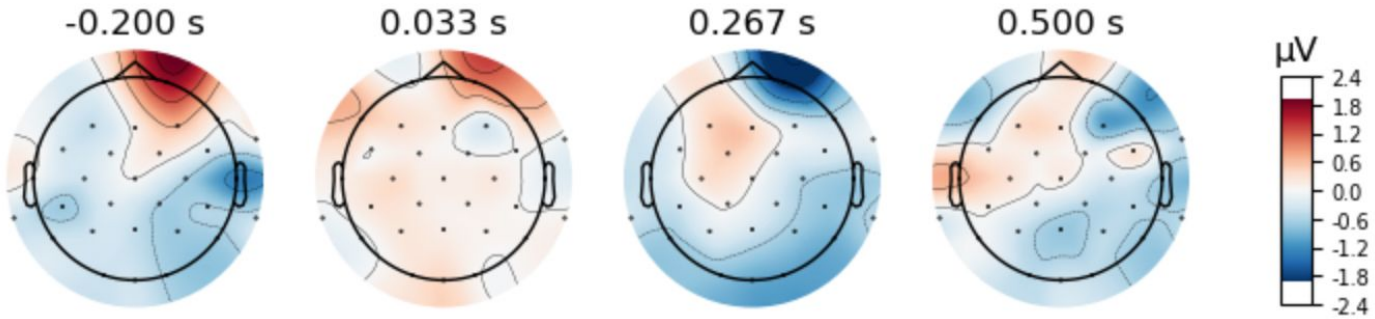
Feature Extraction

- Picture of signals at all epochs for a given time range, for the Fz channel:

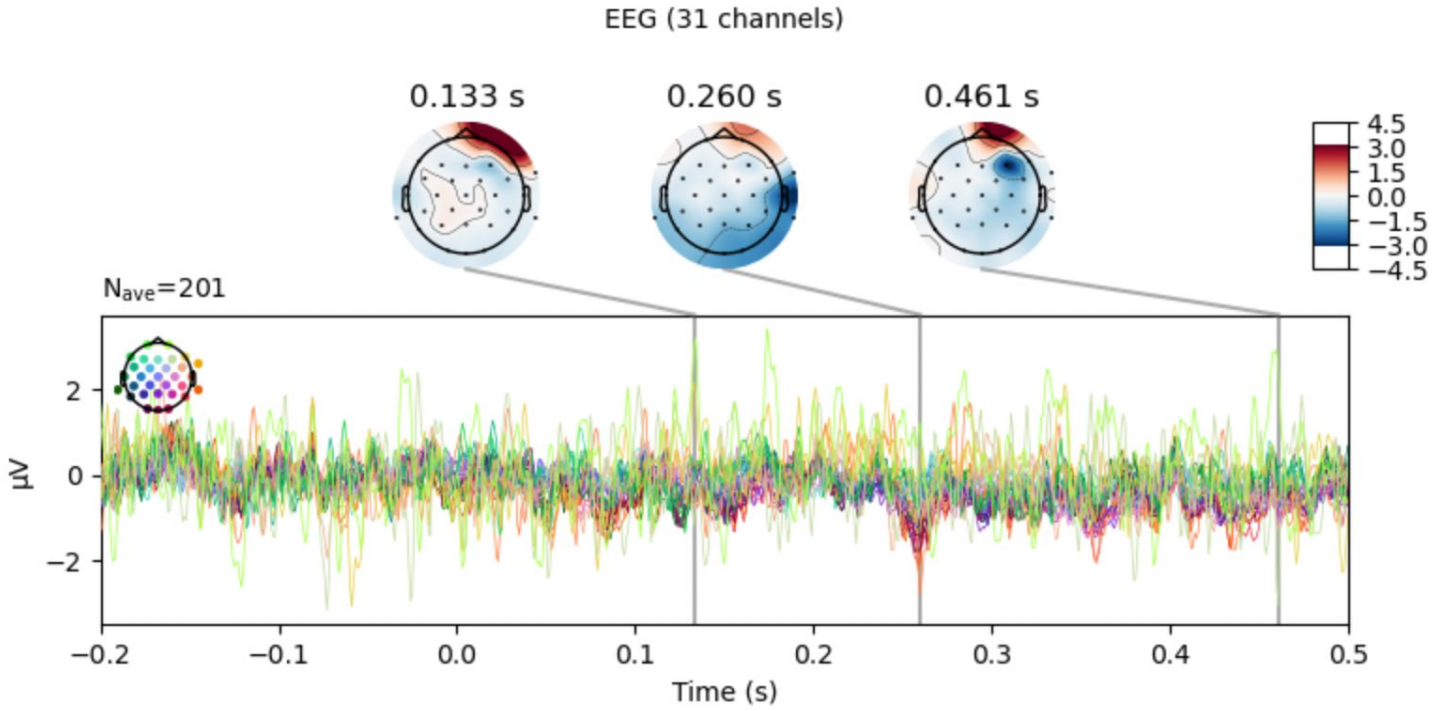


Feature Extraction

- Averaged all epochs to get a single ‘mean’ epoch.
- Its topomap:

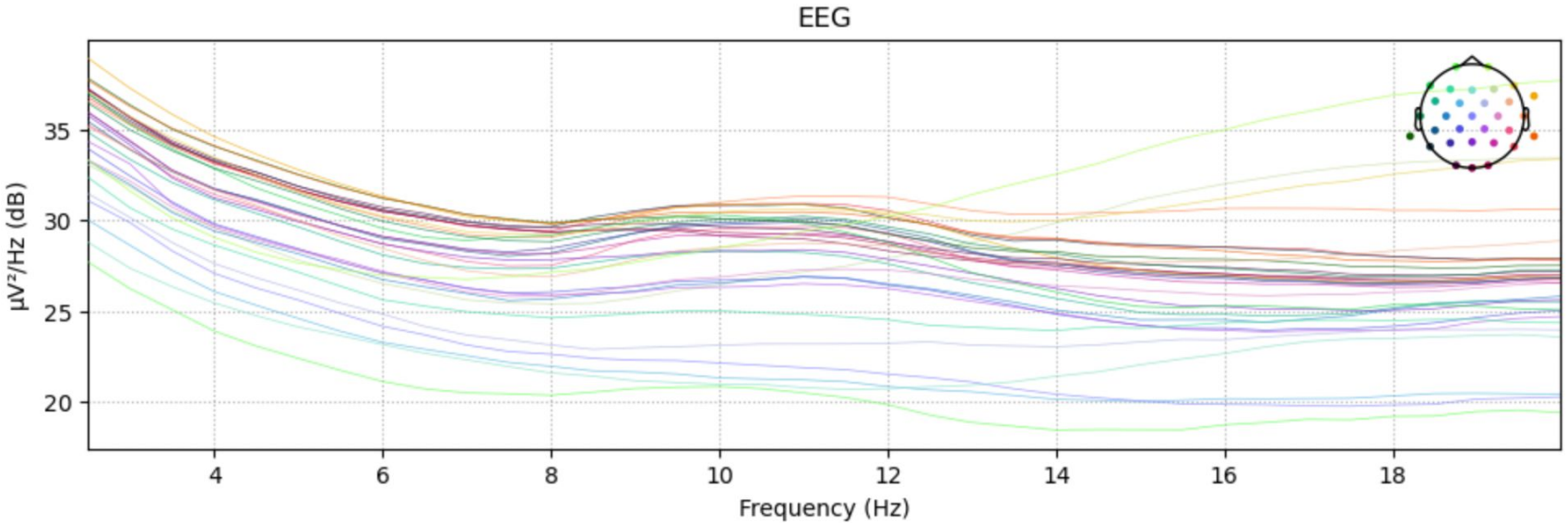


Feature Extraction

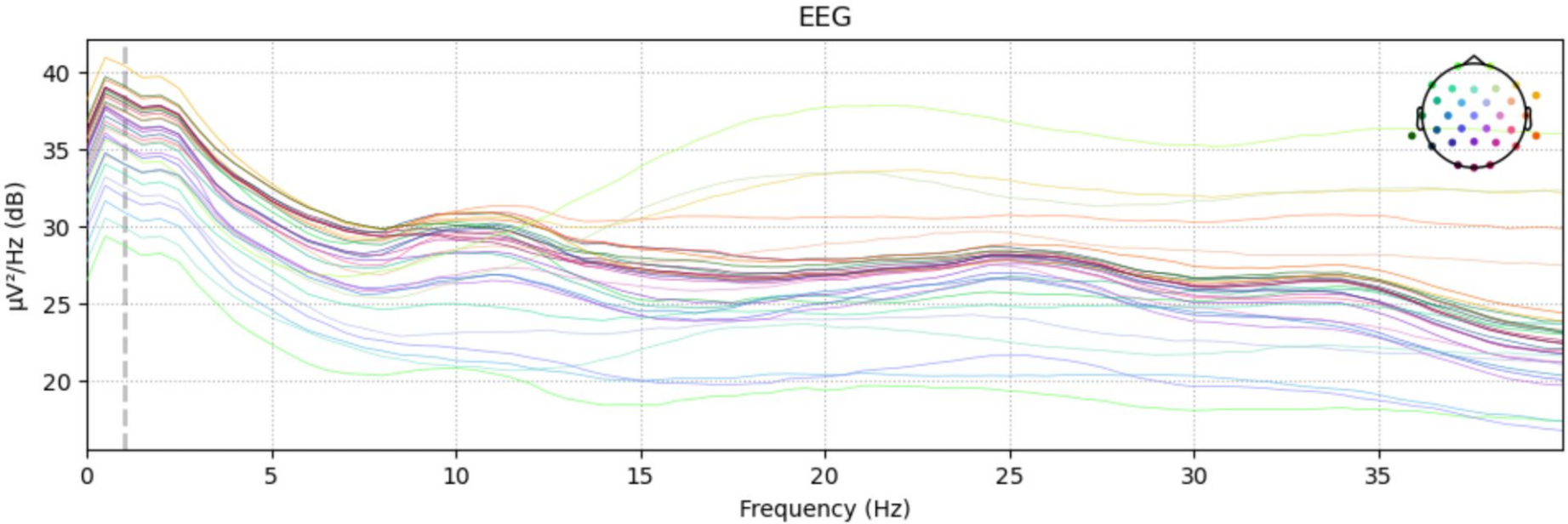


Feature Extraction

- Plot of PSD for the averaged epoch (each channel is coloured differently):



Feature Extraction



Feature Extraction

- Also learnt about spatial and spectral methods of feature extraction, and frequency-domain analysis features.

Feature Extraction - To Be Done

- Perform an analysis of all the features that are of importance to brain age.
- Make the script for extracting these and run it on the entire Quanta dataset

Next Week – Overall

Preprocessing

- Need to export all the overlay plots for analysis.
- Analyse the problematic .set files and manually preprocess them.

Feature extraction

- Perform an analysis of all the features that are of importance to brain age.
- Make the script for extracting these and run it on the entire Quanta dataset

Other

- Make a basic model on these features.

Week 5

27th June, 2023

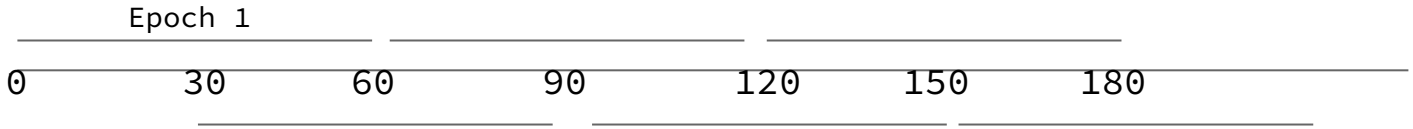
Nikhil Anand

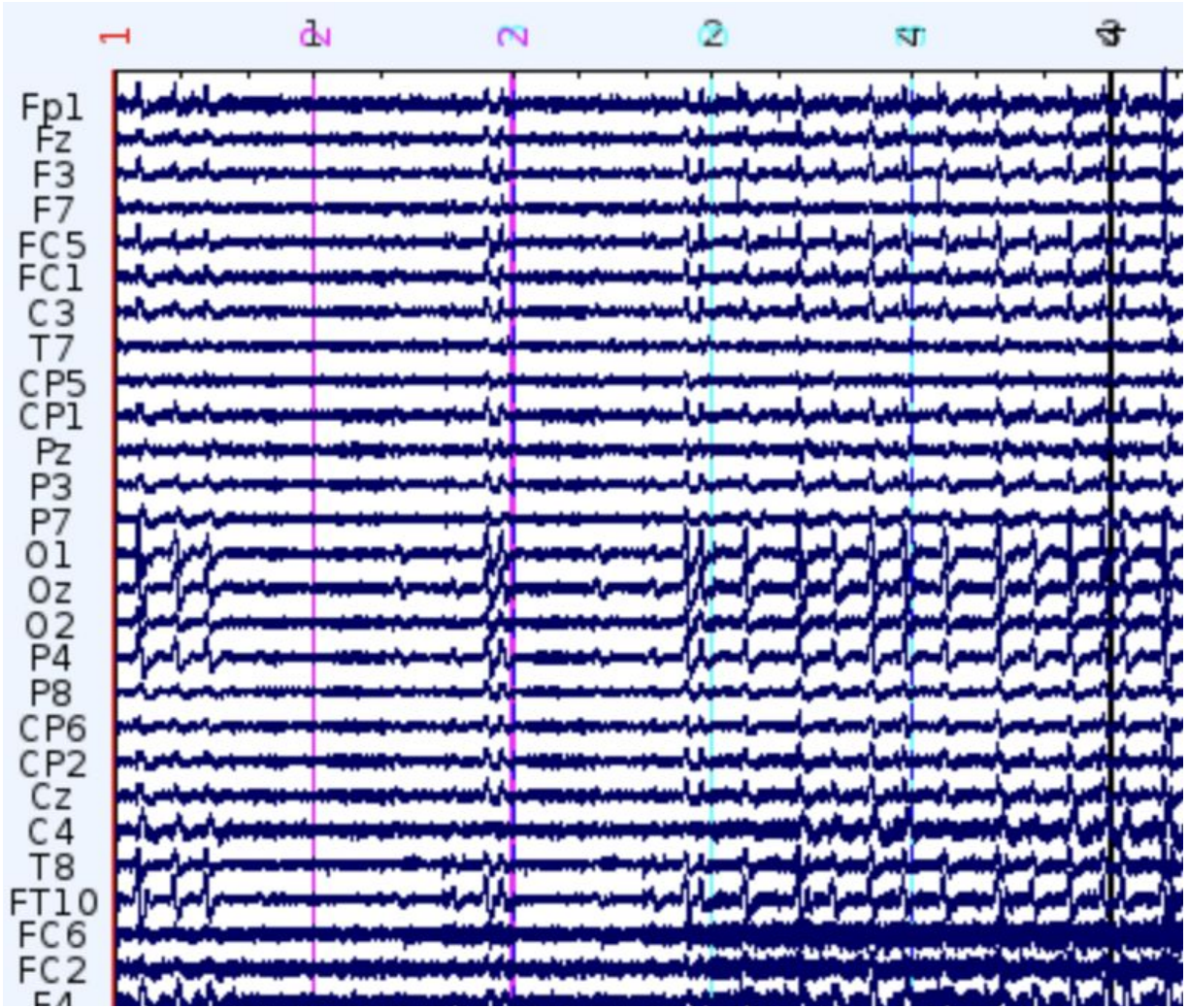
Progress

- Decided to shift feature extraction pipeline to MATLAB due to better support.
- Completed PSD feature extraction on MATLAB and completed feature reduction in Python.

Step 1: Dividing Into Epochs

- Wrote script on MATLAB (EEGLAB) that inputs the .set file containing the preprocessed data and divides it into epochs as mentioned in the paper.
- 60s epochs with 50% overlap.
- Epoched data is then stored in a folder called epoched_data.



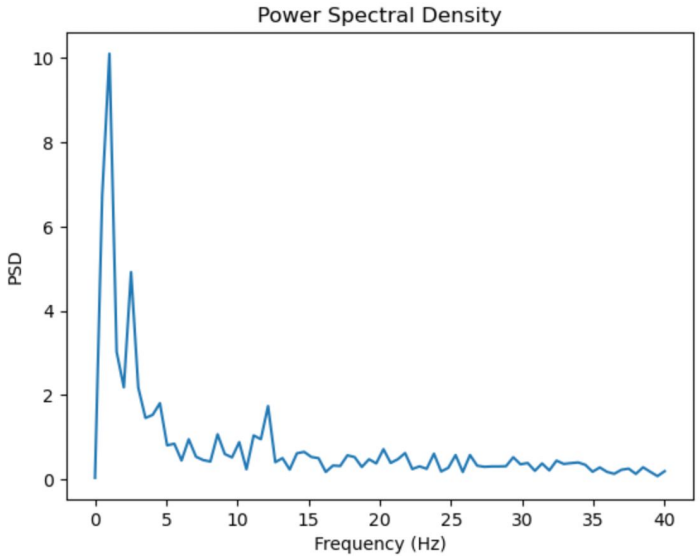

Fp1
Fz
F3
F7
FC5
FC1
C3
T7
CP5
CP1
Pz
P3
P7
O1
Oz
O2
P4
P8
CP6
CP2
Cz
C4
T8
FT10
FC6
FC2

Step 2: Extracting PSD

- Wrote script on MATLAB (EEGLAB) that inputs the .set file containing the epoched data, and runs Welch PSD for a 2s hamming window and overlap of 50%.
- It then generates a PSD Tensor of shape (200x9x29) based on the number of bins (200) to calculate for each epoch (9) and each channel (29). This is saved to a .mat file.

Step 3: Import to Python and Visualise

- Visualised the PSD graphs obtained for each **subject**, for each **epoch** and for each **channel.**




```
loadmat('psd_export/sub-0334_psd_data')
```

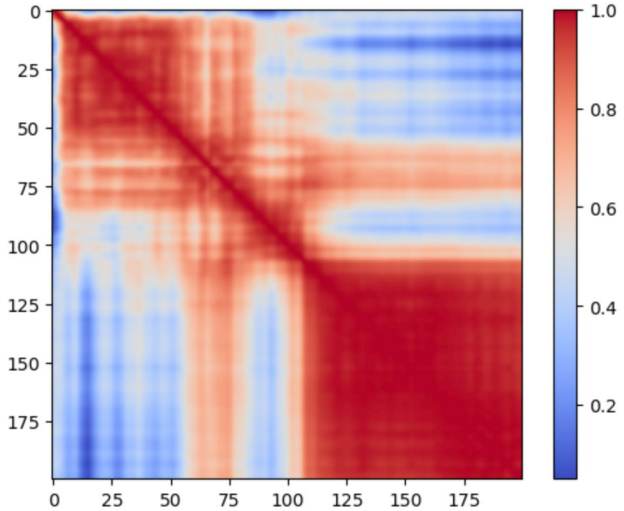
```
epoch_index = 8
channel_index = 4
```

Step 4: Perform Averaging

- Currently we have a 200x29x9 length tensor as a feature for each subject.
- We average the tensor over the 3rd axis (epochs).
- Thus, our tensor is now 200x29.

Step 5: Plot The Feature Correlation Map

- Currently we have a 200x29 shape tensor.
- We now want to reduce the number of features using correlations. We plot the feature correlation map among these 200 features as follows.




```
loadmat('../sub-0332_psd_data')
```

Step 5: Perform Feature Reduction

- Currently we have a 200x29 shape tensor.
- Wrote a script on Python that takes in each subject's data, finds correlations between features in each subject, and finds all the features crossing a 99% threshold of correlation. All features across all subjects that cross this are combined into an array and these are omitted from all subjects to maintain uniformity among subjects.

Step 4: Perform Feature Reduction

- Considering only the five subjects downloaded (subject 1, 2, 3, 332, 333, 334), the reduced data shape turns out to be (696,) for each subject (5 of them).

```
print(reducedData.shape)
(5, 696)
```

Plan For Preprocessing

- Find correlations between component removed and Fp1, and use that to confirm if the ICA went well.
- Plot the topography maps and export.

Week 6

4th July, 2023

Nikhil Anand

Preprocessing (What I Did)

- Created “preprocess_self.m” script which runs ICA through “pop_ica.m” and rejects components by “pop_icflag.m”.
- Created a folder “preprocess_exports” to store the exported matrices.
- Rejected components are stored in a .mat file.
- Another .mat file stores the correlations of the removed component with all the components.
- The ica and oculomotor removed versions of the data are all saved as .set files.

Results for Subject 2

```
>> size(eeg_vem)

ans =

           1      304700
```

```
>> size(seeg_ic)

ans =

          29      304700
```

```
CM = zeros(1, size(seeg_ic,1));
for i = 1: size(seeg_ic,1)
    disp(i);
    R = corrcoef(eeg_vem,seeg_ic(i,:)); %
    CM(i) = R(2,1);
end
```

Results for Subject 2

Correlation Values

```
Columns 1 through 9

  0.0483    0.1818   -0.0898   -0.1702   -0.0708   -0.0247    0.0736    0.1781   -0.1671

Columns 10 through 18

 -0.0854    0.1206   -0.0446    0.0893    0.0314    0.1291    0.0555    0.1247    0.0353

Columns 19 through 27

 -0.0954   -0.1439   -0.1951   -0.1380   -0.1554   -0.1654   -0.1826   -0.1895   -0.1628

Columns 28 through 29

 -0.1123   -0.0987
```

Results for Subject 2

```
>> eeg_allcomps = Uf_eeg*EEG.data;
CM2 = zeros(1, size(eeg_allcomps,1));
    for i = 1: size(eeg_allcomps,1)
        disp(i);
        R = corrcoef(eeg_vem,eeg_allcomps(i,:)); %
        CM2(i) = R(2,1);
    end
```

Results for Subject 2

Correlation Values (slight errors due to numerical precision errors in calculation of the two variables)

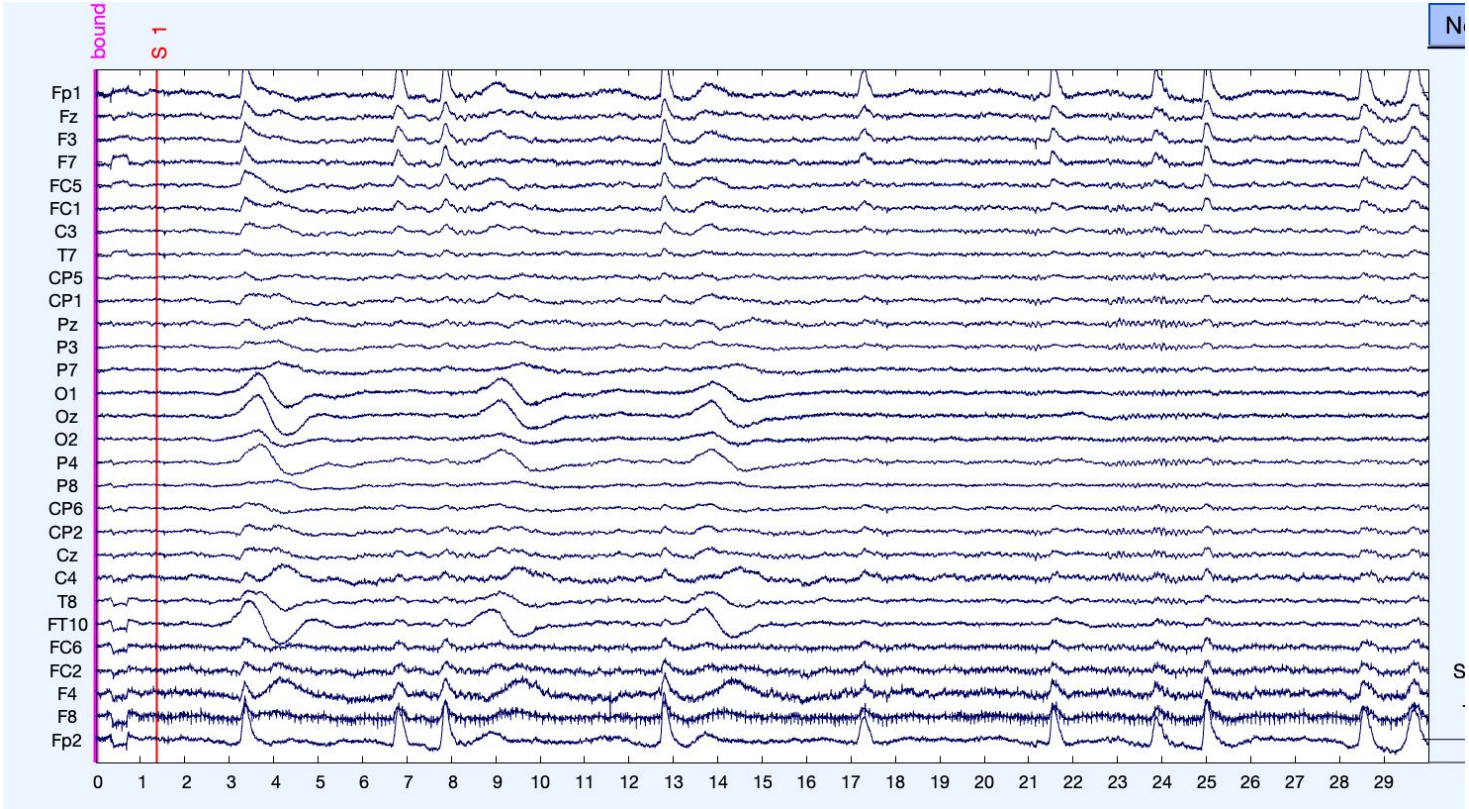
```
CM2 =

  Columns 1 through 9

    0.8011    0.1575    0.0164   -0.0657   -0.0124    0.0796    0.0050    0.0910   -0.0027

  Columns 10 through 18

   -0.0025    0.0484    0.0377    0.0169   -0.0423    0.0382    0.0294    0.0287    0.0594

  Columns 19 through 27

   -0.0024   -0.0041   -0.0507   -0.0462    0.0373   -0.0223    0.0194   -0.0229    0.0137

  Columns 28 through 29

    0.0108    0.0094
```

Why am I not seeing maximum correlation with Fp1?

Tried with a different method and got this:

```
CM =

  Columns 1 through 9

    0.6678    0.0224   -0.0632    0.0300   -0.0035   -0.0074   -0.0139    0.0209    0.0241

  Columns 10 through 18

    0.0061    0.0396    0.0147    0.0060    0.0090    0.0128    0.0309    0.0544    0.0552

  Columns 19 through 27

    0.0854    0.0307   -0.0024   -0.0011    0.0967    0.0993    0.1078   -0.0053   -0.0323

  Columns 28 through 29

    0.0244    0.3656
```

But it still doesn’t match with what’s expected from the plots.



To Be DOne

- Analyse these results further, understand the algorithm properly, and fix this.

Pipeline To Be Run on The Server

Set up EEGLAB on the server

Replace pop_icflag with my custom code and update paths

Create folder for preprocess_exports

Replace paths in preprocess_self.m script and run the script

Wait for 50 hours

Done

We either run this on a lab computer, or split the work and not preprocess all 300 subjects at a stretch on a single system. Each subject takes up to 10 mins to run ICA.

10 mins/subject x 300 subjects = 50 hours

Power Spectral Density (To Be Done)

- Now I plan to go back to my old PSD code on MATLAB, visualise it, plot the correlation map, and properly explain it.
- After that, I'll incorporate more features and get more possible correlation plots, and finalise the feature vector to be used for modelling.

Week 7

11th July, 2023

Nikhil Anand

Progress Summary

- In-depth review of Zoubi et al paper and presentation.
- Read up some other papers regarding various features that can be used.
- Implemented feature extraction of amplitude/envelope features (mean, median, variance, kurtosis, skewness) and spectral features (total power, mean power, edge frequency, relative band powers).
- Extracted graph features from functional connectivity matrices and trained an SVR model.

Features Used Commonly

- **Time domain features:**
  - Statistical features (mean, median, variance, kurtosis, skewness)
  - ZCR (Number of times the signal crosses the X-axis)
  - Period-amplitude analysis (num of waves, wave duration, peak amplitude, instantaneous frequency)
  - Hjorth parameters (Mobility, Activity, Complexity are the 1st, 2nd and 3rd derivatives respectively)
  - Energy (sum of squares of amplitude)
- **Frequency domain features:**
  - Power Spectral Density (PSD) is the basis
  - Parametric (Autoregression/Multivariate AR)
  - Non-parametric (FFT, Welch's method, Thompson multitaper)
  - Statistical features are used here
  - Relative powers - delta, theta, alpha, beta, gamma, theta/alpha, beta/alpha, (theta+alpha)/beta, theta/beta, (theta+alpha)/(alpha+beta), gamma/delta, (gamma+beta)/(delta+alpha)

Features Used Commonly

- **Time-frequency domain features:**
  - Spectrogram (apply STFT)
  - Wavelet transform
- **Non-linear features:**
  - Fractal dimension D0, D1, D2 are extracted by Higuchi method
- **Entropies**
  - Shannon entropy, Tsallis entropy, Rényi's entropy
- **Complex networks**
  - Network analysis (graph-theory-based approach)
  - Connectivity matrix (pairwise connections between channels)
  - Features - number of vertices, edges, degree, mean degree, degree distribution, transitivity, efficiency, centrality

MATLAB Feature Extraction

- The preprocessed, epoched data (60s, 50% overlap) is the input to the script.
- We first filter the data into the 5 bands as:

```
deltaBand = [0.5, 4];
thetaBand = [4, 8];
alphaBand = [8, 14];
betaBand = [14, 30];
gammaBand = [30, Inf];
```

- We now have shape (channels x samples/epoch x epochs x bands).
- For each (channel,epoch,band) triplet, we consider the signal vector and compute the Hilbert transform.
- The magnitude of the complex vector is the signal envelope.

MATLAB Feature Extraction

- Features extracted from the envelope are:

```
epochStats = [mean(data,2); median(data,2); mode(data,2);
    std(data,0,2); kurtosis(data,1,2); skewness(data,1,2)];
```

- This is then averaged over the epochs.
- Thus, the final feature vector shape is (channels x bands x 6).

```
magFeatures(:,:,5) =

   1.0e+03 *

    0.0478    0.0178    0.0229    0.0179    0.3800
    0.0216    0.0187    0.0215    0.0241    0.0223
    0.0251    0.0155    0.0250    0.0218    0.5157
    0.0890    0.0578    0.0249    0.0345    1.1533
    0.0495    0.0152    0.0384    0.0346    0.5727
    0.0167    0.0285    0.0337    0.0237    0.0489
    0.0275    0.0214    0.0288    0.0399    0.0724
    0.0500    0.0186    0.0773    0.0184    0.4658
```

MATLAB Feature Extraction

- Next, the PSD Data vector is computed and the features extracted are totalPower, meanPower, edgeFreq.

```
totalPower = sum(psd);
meanPower = totalPower/(top-arr(b)); % arr(b+1) - arr(b) is
totalPower = mean(totalPower);
meanPower = mean(meanPower);

cumulativePower = cumsum(psd);
edgeFreqIdx = find(cumulativePower >= 0.95*totalPower, 1);
edgeFreq = (edgeFreqIdx - 1)*(top-arr(b))/500;

psdFeatures(ch, :, b) = [totalPower, meanPower, edgeFreq];
```

```
psdFeatures(:,:,5) =

   1.0e+03 *

    0.0175    0.0003    0.2786
    0.1039    0.0015    0.0543
    0.0912    0.0013    0.3401
    0.0664    0.0009    0.4039
    0.0723    0.0010    0.4019
    0.0563    0.0008    0.2748
    0.0451    0.0006    0.2724
    0.0679    0.0010    0.0522
    0.0400    0.0006    0.0515
    0.0369    0.0005    0.0689
```

MATLAB Feature Extraction

- Finally, relative powers are saved in a separate .mat file.
- Order of relative powers: theta/alpha, beta/alpha, (theta+alpha)/beta, theta/beta, (theta+alpha)/(alpha+beta), gamma/delta, (gamma+beta)/(delta+alpha)
- 7 features. Shape = channels x 7

```
relPowFeatures =

    1.0930    1.1344    1.8451    0.9636    0.9806    1.0041    1.0653
    1.3032    1.2694    1.8143    1.0266    1.0149    0.8890    1.0422
    1.4422    1.2221    1.9984    1.1801    1.0991    0.9695    1.0762
    1.8754    1.1295    2.5458    1.6604    1.3503    0.8146    0.9418
    1.4633    1.2713    1.9376    1.1510    1.0845    1.0113    1.1206
    1.2190    1.2986    1.7089    0.9388    0.9654    0.9674    1.1064
    1.2267    1.2579    1.7703    0.9753    0.9862    1.0661    1.1484
    1.5293    1.1859    2.1328    1.2896    1.1571    0.9367    1.0416
    1.2550    1.1408    1.9766    1.1001    1.0533    0.9546    1.0360
    1.0865    1.1841    1.7621    0.9176    0.9553    1.0580    1.1141
    0.9756    1.0971    1.8007    0.8893    0.9421    1.0594    1.0775
    0.9926    1.1672    1.7072    0.8504    0.9194    1.0010    1.0802
    1.2403    1.0249    2.1858    1.2101    1.1063    0.7712    0.8753
    1.1513    1.0858    1.9813    1.0603    1.0314    0.7637    0.9081
    1.0086    1.0783    1.8629    0.9354    0.9665    0.9402    1.0057
    1.0414    1.0823    1.8861    0.9622    0.9804    0.8225    0.9457
    1.0514    1.1430    1.7948    0.9199    0.9573    0.9440    1.0357
    1.0425    1.1629    1.7565    0.8965    0.9444    0.9372    1.0437
    1.0389    1.1665    1.7479    0.8906    0.9411    1.1539    1.1601
    1.0999    1.2069    1.7400    0.9114    0.9515    1.1232    1.1625
    1.2138    1.1814    1.8739    1.0274    1.0149    1.0018    1.0813
    1.2112    1.1301    1.9565    1.0717    1.0380    1.0305    1.0746
    1.2282    1.0910    2.0423    1.1258    1.0656    0.8746    0.9737
    1.1022    1.1676    1.8004    0.9439    0.9698    2.0503    1.5894
    1.1765    1.1503    1.8921    1.0228    1.0122    0.9655    1.0482
    1.2230    1.1694    1.9009    1.0458    1.0247    0.8644    0.9980
    1.3096    1.1307    2.0427    1.1582    1.0840    0.8697    0.9883
    2.2424    1.1763    2.7564    1.9063    1.4898    0.5659    0.7605
    1.7717    1.2383    2.2383    1.4308    1.2383    0.7339    0.9228
```

Extracting Features From FCONN

```
(30, 30, 17)
```

```
mat_data["freqVec"]
```

✓ 0.7s Python

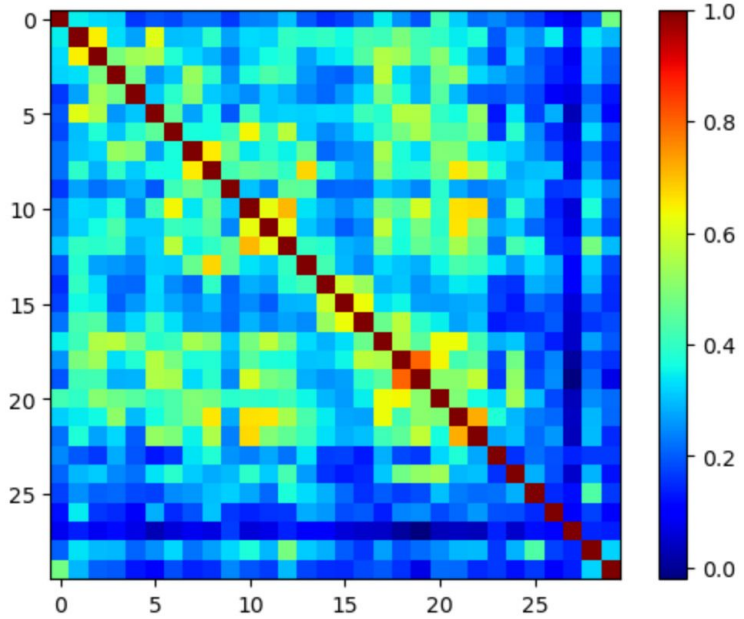
```
array([[  4,   5,   6,   8,  10,  12,  15,  20,  25,  30,  35,  40,  50,
         60,  70,  80, 100]], dtype=uint8)
```

```
conn_matrix = mat_data["fconn"][:,:,3]
for i in range(5):
    print(conn_matrix[i][i])

print("Shows that highest connectivity value is 1, similar to a correlation matrix")
```

✓ 0.5s Python

```
1.0
1.0
1.0
1.0
1.0
Shows that highest connectivity value is 1, similar to a correlation matrix
```

Extracting Features From FCONN

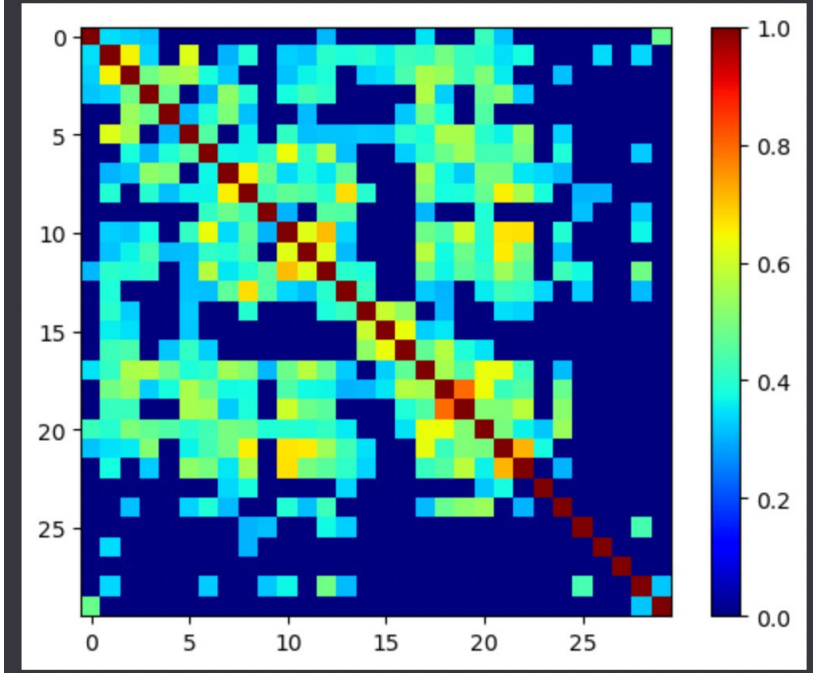


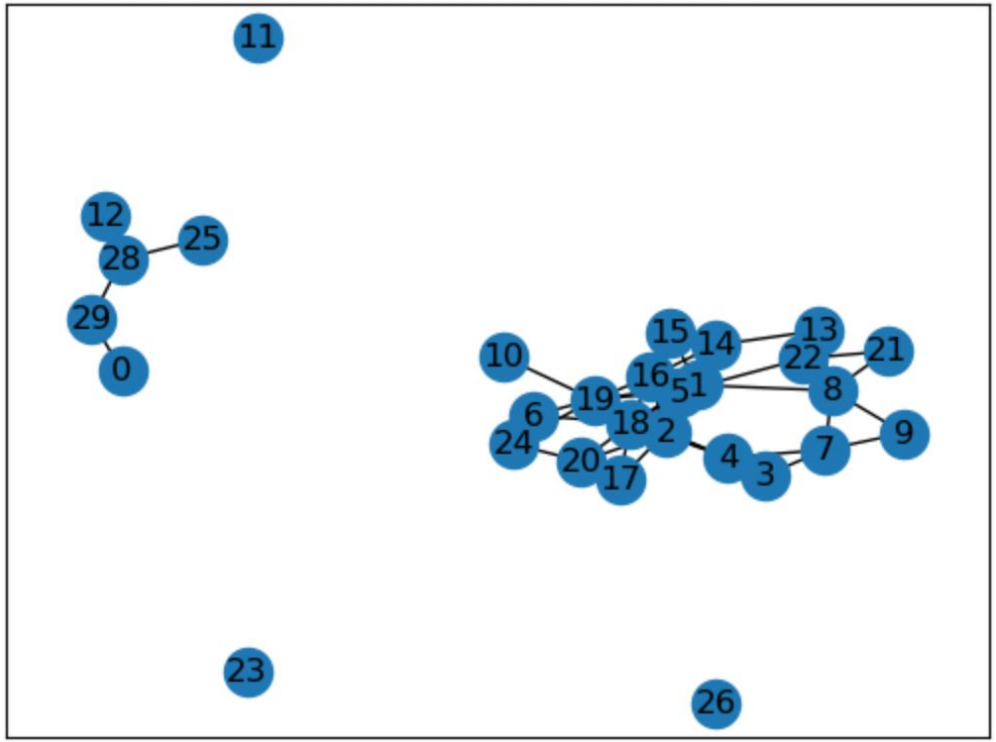

```
Connectivity in range (0, 0.1): 42
Connectivity in range (0.1, 0.2): 156
Connectivity in range (0.2, 0.3): 250
Connectivity in range (0.3, 0.4): 212
Connectivity in range (0.4, 0.5): 114
Connectivity in range (0.5, 0.6): 56
Connectivity in range (0.6, 0.7): 32
Connectivity in range (0.7, 0.8): 6
```

Extracting Features From fCONN

```
# Setting threshold of 0.3
conn_thresh = np.where(conn_matrix < 0.3, 0,conn_matrix)
plt.imshow(conn_thresh, cmap=cmap)
plt.colorbar()
plt.show()
```

✓ 0.1s



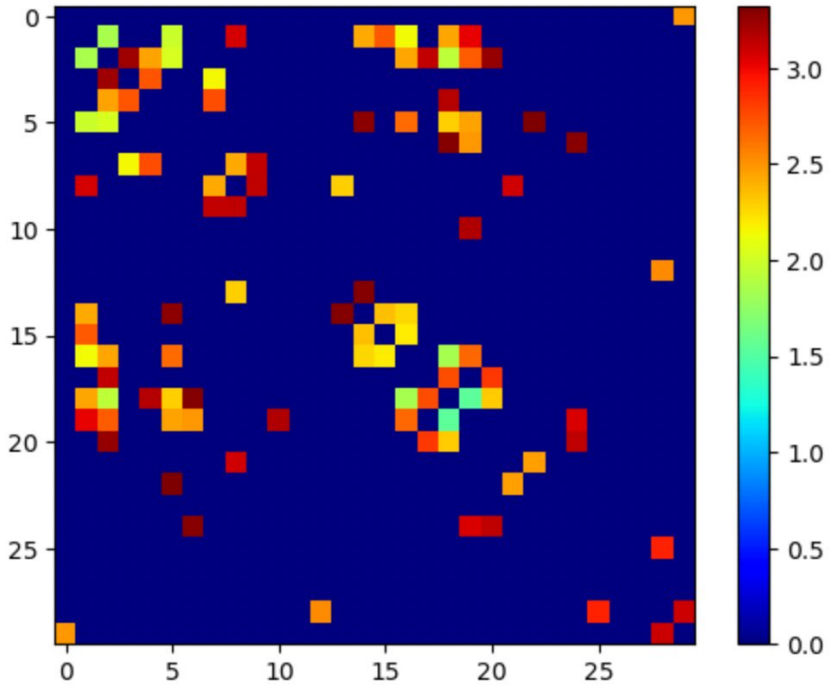


Extracting Features (Degree Centrality)

```
feats = np.zeros((17,30,3))

degrees = np.zeros((17,30))
degrees[:] = -1

# Computing degree of nodes
for i in range(17):
    graph = mat_data["fconn"][:,:,i]
    graph = np.where(graph < 0.3, 0,graph)
    for j in range(30):
        jgraph = graph[j,:]
        for k in range(30):
            conn = jgraph[k]
            degrees[i,j] += conn

print(degrees[3])
feats[:,:,0] = degrees
```

✓ 0.1s

```
[ 2.86470258  8.37481725  8.12501317  6.26279283  4.47354168  7.83567357
  8.43357176  7.92804706  9.2267268   3.85245481  8.87537432  7.81589696
  9.69175971  6.86751306  4.36177713  2.93896839  4.86780062 10.16019762
  9.5730972   7.92817071  9.83433726  9.25838292  7.81437001  1.75459054
  4.61807173  1.75653124  0.64750534  0.          2.91198781  0.79859534]
```



Plot after taking inverse of all non-zero values, and setting 1s to 0s

Extracting Features (Betweenness Centrality)

```
bcents = np.zeros((17,30))

# Computing BCs
for i in range(17):
    graph = mat_data["fconn"][:,:,1]
    graph = np.where(graph < 0.3, 0,graph)
    graph = np.where(graph == 0, 0, 1/graph)
    graph = np.where(graph == 1, 0, graph)
    G = nx.from_numpy_array(graph)
    bc = nx.betweenness_centrality(G,normalized=False,weight='weight')
    bc_arr = list(bc.values())
    bcents[i,:] = bc_arr

print(bcents[i])
feats[:,:,1] = bcents
```

✓ 0.1s

```
[ 0. 38. 26.  0. 10. 22.  0.  8. 34.  0.  0.  0.  0.  0. 10.  0. 21.  0.
 37. 42.  1.  4.  9.  0.  0.  0.  0.  0.  5.  3.]
```

Extracting Features (EigenVector Centrality)

```
ecents = np.zeros((17,30))

# Computing BCs
for i in range(17):
    graph = mat_data["fconn"][:,:,1]
    graph = np.where(graph < 0.3, 0,graph)
    graph = np.where(graph == 1, 0, graph)
    G = nx.from_numpy_array(graph)
    ec = nx.eigenvector_centrality(G,weight='weight')
    ec_arr = list(ec.values())
    ecents[i,:] = ec_arr

print(ecents[i])
feats[:,:,2] = ecents
```

✓ 0.7s

```
[6.32923169e-08 3.80141250e-01 3.92244886e-01 7.09749244e-02
 1.27201229e-01 3.49611795e-01 1.09284104e-01 4.12172861e-02
 6.17720625e-02 1.24715689e-02 4.11090662e-02 6.29495167e-12
 8.09124414e-08 3.19690562e-02 1.89569654e-01 1.48176019e-01
 3.72388303e-01 1.25025009e-01 4.23968055e-01 3.44713564e-01
 1.41060592e-01 1.40493411e-02 4.20042967e-02 6.29495167e-12
 7.23334007e-02 7.07283486e-08 6.29495167e-12 6.29495167e-12
 1.35872513e-07 1.04488921e-07]
```

Extracting Features (Standardisation)

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled = scaler.fit_transform(feats.reshape(-1,3))

print(scaled.shape)

print(scaled.reshape(17,30,3).shape)

scaled = scaled.flatten()
print(scaled.shape)
```

✓ 0.4s

```
(510, 3)
(17, 30, 3)
(1530,)
```

Features have different scales, so they need to be standardised

```
array([[[7.85488218e-01, 0.00000000e+00, 6.32923169e-08],
        [3.36712295e+00, 3.80000000e+01, 3.80141250e-01],
        [3.85056296e+00, 2.60000000e+01, 3.92244886e-01],
        ...,
        [0.00000000e+00, 0.00000000e+00, 6.29495167e-12],
        [1.34455672e+00, 5.00000000e+00, 1.35872513e-07],
        [7.66633451e-01, 3.00000000e+00, 1.04488921e-07]],

       [[4.01010990e-01, 0.00000000e+00, 6.32923169e-08],
        [3.35895923e+00, 3.80000000e+01, 3.80141250e-01],
        [3.67818499e+00, 2.60000000e+01, 3.92244886e-01],
```

$$z = \frac{x - \mu}{\sigma}$$

Modelling (Analysis of Variance of Features Across Subjects)

```
data = np.loadtxt('features.csv', delimiter=',')

print(data.shape)
```

✓ ✓ 0.1s

```
(152, 1530)
```

```
variances = np.var(data, axis=0)

indices_of_least_variance = np.argsort(variances)[:10]
print("Features with the least variance:")
print(indices_of_least_variance,variances[indices_of_least_variance])

indices_of_most_variance = np.argsort(variances)[-10:]
print("Features with the most variance:")
print(indices_of_most_variance,variances[indices_of_most_variance])
```

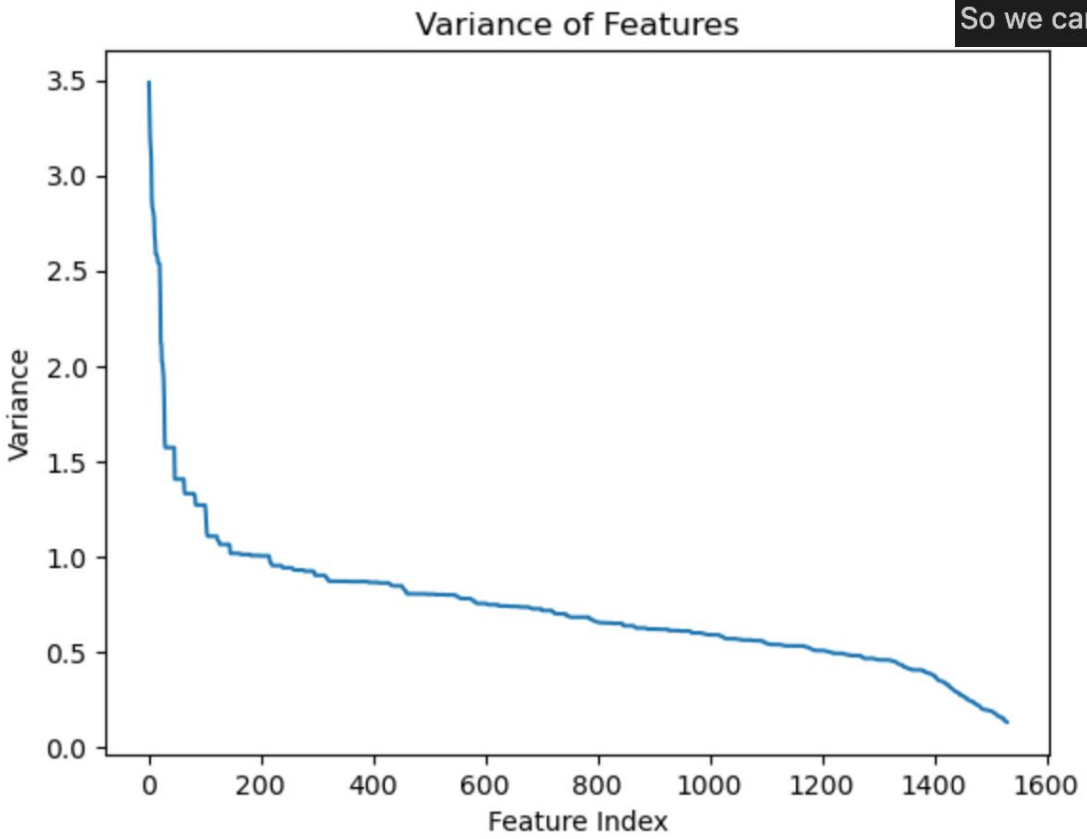
✓ 0.4s

```
Features with the least variance:
[771 774 744 780 843 777 861 750 753 741] [0.1345037  0.1349112  0.13849206 0.14017222 0.14302973 0.14445151
 0.15311575 0.15657468 0.15671253 0.15709589]
Features with the most variance:
[1503 1482 1443 1509 1521 1455 1464 1518 1497 1506] [2.77532686 2.80105321 2.81938194 2.82861597 2.87798357 3.09420954
 3.14772192 3.19453374 3.30973006 3.48601461]
```

Modelling (PCA)

From the graph, it appears that the knee point is around the 210th mark.

So we can consider the top 200 features alone in our model.



Modelling (The Top 200 Features)

Channel, band and feature indices are labelled on the features

```
['ch1b1f3', 'ch1b2f2', 'ch1b3f2', 'ch1b6f2', 'ch1b8f2', 'ch1b15f3', 'ch1b16f3', 'ch2b10f2', 'ch2b11f2', 'ch2b13f2', 'ch2b14f3', 'ch2b15f2',
'ch2b16f2', 'ch3b2f2', 'ch3b4f2', 'ch3b12f3', 'ch4b6f2', 'ch4b7f2', 'ch4b9f2', 'ch4b10f3', 'ch4b11f2', 'ch4b12f2', 'ch4b15f2', 'ch4b17f2',
'ch5b8f3', 'ch6b2f2', 'ch6b3f2', 'ch6b5f2', 'ch6b6f3', 'ch6b7f2', 'ch6b8f2', 'ch6b11f2', 'ch6b13f2', 'ch7b3f3', 'ch7b4f3', 'ch7b15f2',
'ch7b16f2', 'ch8b1f2', 'ch8b2f1', 'ch8b2f3', 'ch8b3f2', 'ch8b4f1', 'ch8b4f2', 'ch8b5f1', 'ch8b7f2', 'ch8b9f2', 'ch8b13f1', 'ch8b14f1',
'ch8b16f1', 'ch8b16f3', 'ch8b17f1', 'ch8b17f3', 'ch9b1f1', 'ch9b2f1', 'ch9b7f1', 'ch9b11f2', 'ch9b12f1', 'ch9b12f2', 'ch9b13f1', 'ch9b14f1',
'ch9b14f2', 'ch9b15f3', 'ch9b16f2', 'ch9b17f2', 'ch10b3f2', 'ch10b5f2', 'ch10b9f1', 'ch10b12f3', 'ch10b13f3', 'ch11b7f2', 'ch11b8f2',
'ch11b10f1', 'ch11b10f2', 'ch11b11f3', 'ch11b12f2', 'ch11b13f2', 'ch11b16f2', 'ch12b1f2', 'ch12b9f3', 'ch13b3f2', 'ch13b4f2', 'ch13b6f2',
'ch13b7f3', 'ch13b8f2', 'ch13b9f2', 'ch13b12f2', 'ch13b14f2', 'ch14b4f3', 'ch14b5f3', 'ch14b16f2', 'ch14b17f2', 'ch15b2f2', 'ch15b3f3',
'ch15b4f2', 'ch15b5f2', 'ch15b8f2', 'ch15b10f2', 'ch15b17f3', 'ch16b1f3', 'ch16b12f2', 'ch16b13f2', 'ch16b15f2', 'ch16b16f3', 'ch16b17f2',
'ch17b1f2', 'ch17b4f2', 'ch17b6f2', 'ch17b13f3', 'ch17b14f3', 'ch18b8f2', 'ch18b9f2', 'ch18b11f2', 'ch18b12f3', 'ch18b13f2', 'ch18b14f2',
'ch18b17f2', 'ch19b2f2', 'ch19b9f3', 'ch19b10f3', 'ch20b4f2', 'ch20b5f2', 'ch20b7f2', 'ch20b8f3', 'ch20b9f2', 'ch20b10f2', 'ch20b13f2',
'ch20b15f2', 'ch21b5f3', 'ch21b6f3', 'ch21b17f2', 'ch22b1f2', 'ch22b3f2', 'ch22b4f3', 'ch22b5f2', 'ch22b6f2', 'ch22b9f2', 'ch22b11f2',
'ch23b1f3', 'ch23b2f3', 'ch23b13f2', 'ch23b14f2', 'ch23b16f2', 'ch23b17f3', 'ch24b1f2', 'ch24b2f2', 'ch24b5f2', 'ch24b7f2', 'ch24b14f3',
'ch24b15f3', 'ch25b9f2', 'ch25b10f2', 'ch25b12f2', 'ch25b13f3', 'ch25b14f2', 'ch25b15f2', 'ch26b1f2', 'ch26b3f2', 'ch26b10f3', 'ch26b11f3',
'ch27b5f2', 'ch27b6f2', 'ch27b8f2', 'ch27b9f3', 'ch27b10f2', 'ch27b11f2', 'ch27b14f2', 'ch27b16f2', 'ch28b7f3', 'ch29b1f2', 'ch29b2f2',
'ch29b4f2', 'ch29b5f1', 'ch29b5f3', 'ch29b6f1', 'ch29b6f2', 'ch29b7f1', 'ch29b7f2', 'ch29b8f1', 'ch29b9f1', 'ch29b10f1', 'ch29b10f2',
'ch29b11f1', 'ch29b12f1', 'ch29b12f2', 'ch29b13f1', 'ch29b14f1', 'ch29b15f1', 'ch29b16f1', 'ch29b17f1', 'ch30b1f1', 'ch30b2f1', 'ch30b3f1',
'ch30b3f3', 'ch30b4f1', 'ch30b5f1', 'ch30b6f1', 'ch30b7f1', 'ch30b8f1', 'ch30b9f1', 'ch30b10f1', 'ch30b11f1', 'ch30b12f1', 'ch30b13f1',
'ch30b14f1', 'ch30b14f2', 'ch30b15f1', 'ch30b15f2', 'ch30b16f1', 'ch30b17f1', 'ch30b17f2']
```

Modelling (Augmented the Age Labels)

```
aug_labels = np.array(labels) + np.random.normal(0, 1, len(labels))
aug_labels
```

✓ 0.1s

```
array([41.26547106549365, 26.632833099615393, 27.532006937158666,
       21.93487170007416, 22.622136591350806, 23.963407388401713,
       30.092125994546183, 29.09128729369801, 22.0709703244881,
       23.338112832936257, 23.774194465515308, 46.630868015723905,
       25.7412402861552, 21.97965732296544, 23.279878747455314,
       29.120993894860455, 53.50668828819571, 31.58015419193194,
       36.57985518302252, 35.30309211407207, 30.117024443474396,
       35.872821332491256, 37.40226723181941, 23.64379202771985,
       46.5584500890084, 21.724678601300212, 39.2552200798992,
       57.27774865730446, 56.042537526153815, 57.99628616328316,
       46.81899741524977, 47.971150077584234, 55.63481792964702,
       49.34742800001581, 28.671138986369165, 25.19757748241087,
```

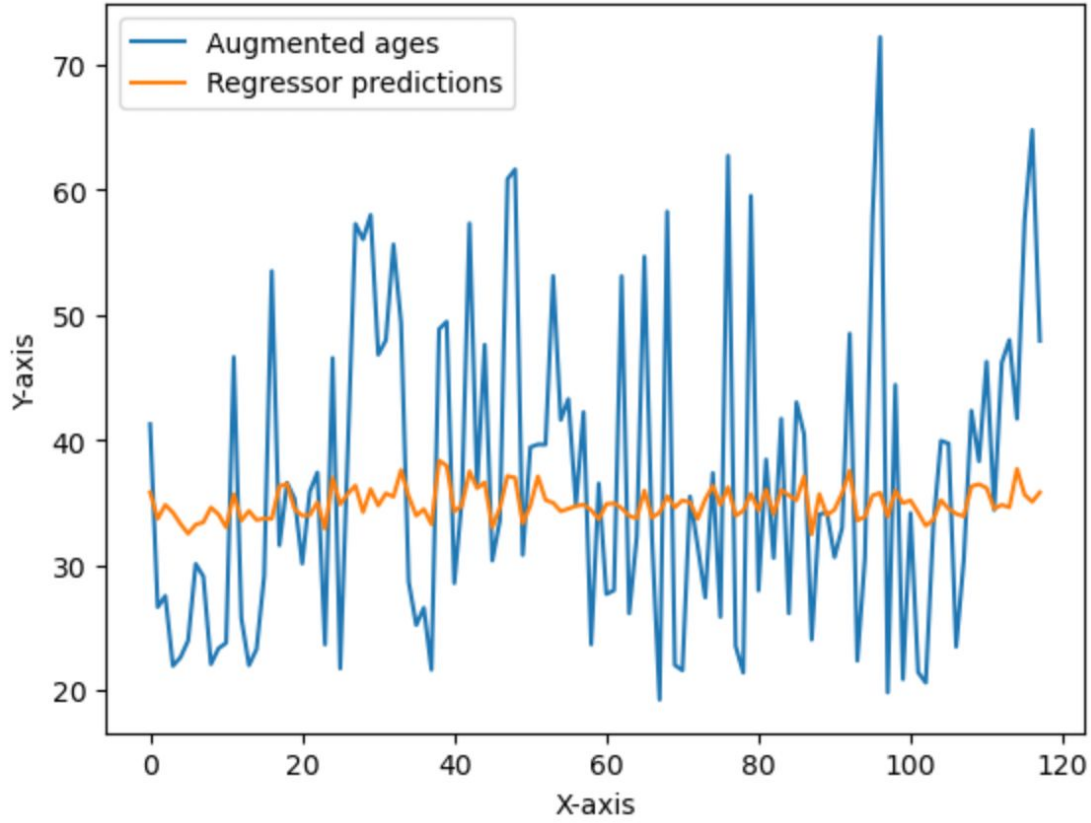
Modelling (SVR Model With RBF Kernel)

```
from sklearn.svm import SVR
regressor = SVR(kernel='rbf')

regressor.fit(X,aug_labels)
```

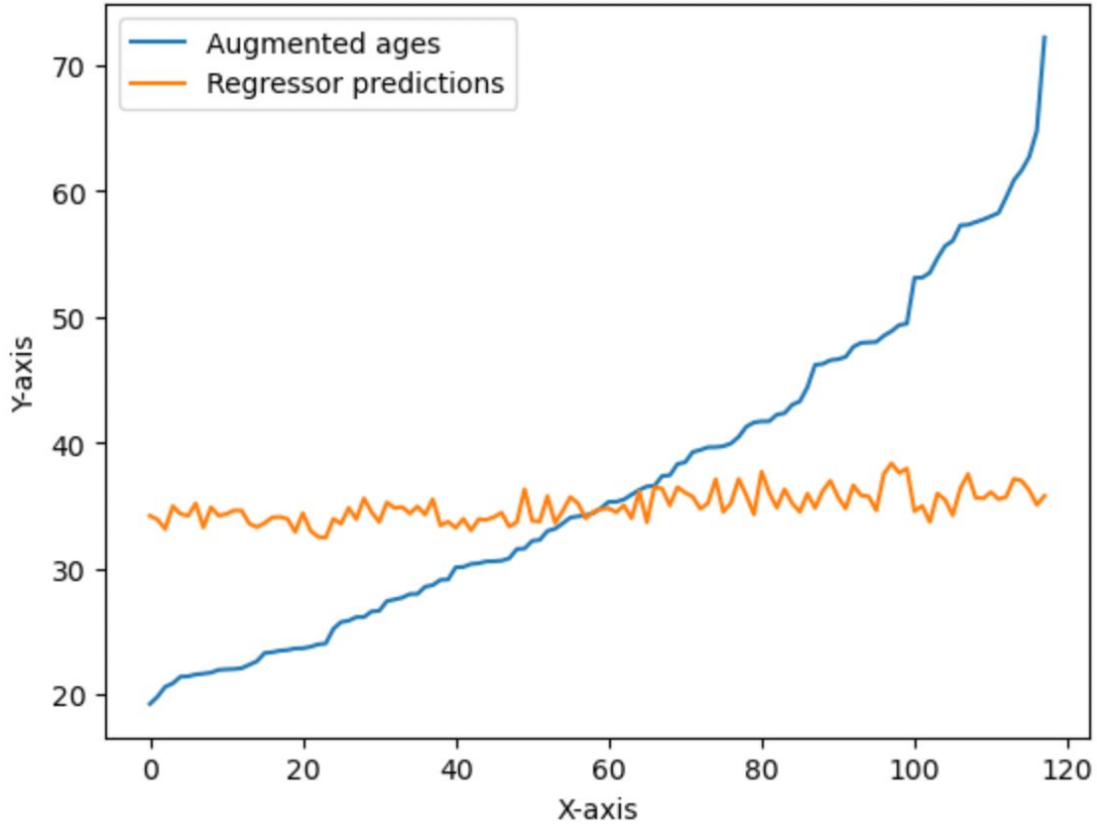
MAE

```
sum(abs(regressor.predict(X) - aug_labels))/len(aug_labels)
✓ 0.3s
9.332388742617875
```

Modelling (Results)



Modelling (Results)



Modelling (K-Fold Cross Validation with RBF Kernel)

```
Iteration 1: Validation Error = 10.946847298910384
Iteration 2: Validation Error = 7.211041192776622
Iteration 3: Validation Error = 9.780095080901775
Iteration 4: Validation Error = 8.353806328231213
Iteration 5: Validation Error = 8.627618067565978
Iteration 6: Validation Error = 12.556922788578111
Iteration 7: Validation Error = 10.52305543245855
Iteration 8: Validation Error = 11.453799279606264
Iteration 9: Validation Error = 13.380676148560855
Iteration 10: Validation Error = 10.078126078418894
Average validation error:  10.078126078418894
```

Week 8

17th July, 2023

Nikhil Anand

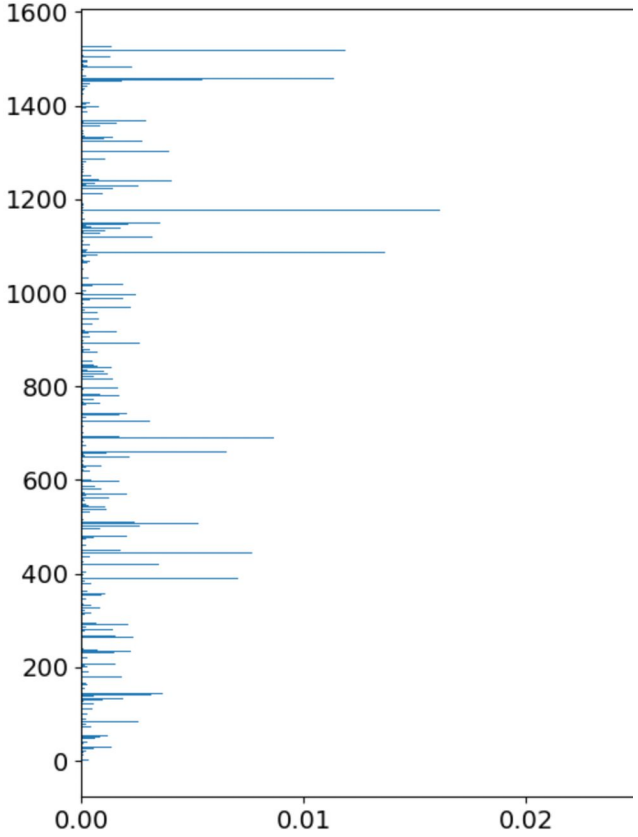
Progress Summary

- Experimented with hyperparameters of the PCA+SVR model. (number of features extracted, augmentation std dev, regression kernel)
- Tried a RandomForestRegressor and got much higher train accuracy (but it overfit and test accuracy low)
- Documented code and wiki page on Github.
- Tried augmenting data, and performed K-fold cross-validation while doing a grid search over all parameters to get the optimum hyperparameters. (MAE = 9.3 years)
- Performed linear correction and increased train MAE to 3.9, decreased test MAE to 8.64. Did Cross-Validation on this.

Methods – Random Forest Regressor Model

- Knowing that PCA is probably extracting the ‘most important’ features without any information from the labels, it might be useful to use an approach that utilised the labels.
- The Random Forest Regressor was trained with 100 estimators, and feature importances were computed and plotted.

Results – Random Forest Feature Importances

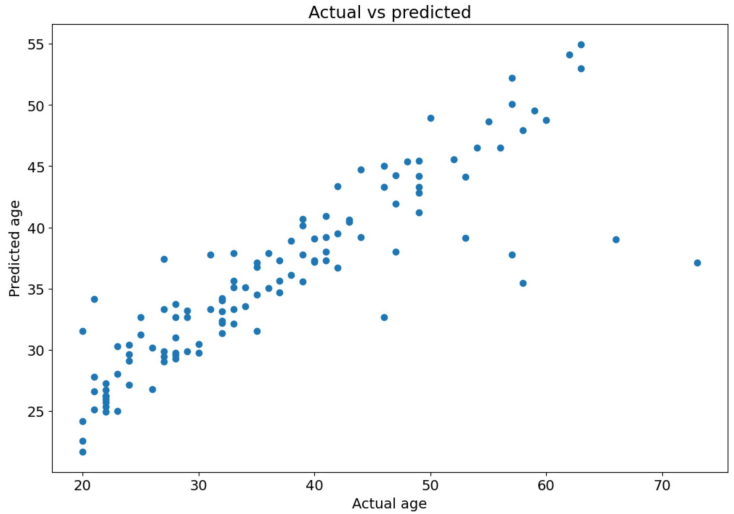


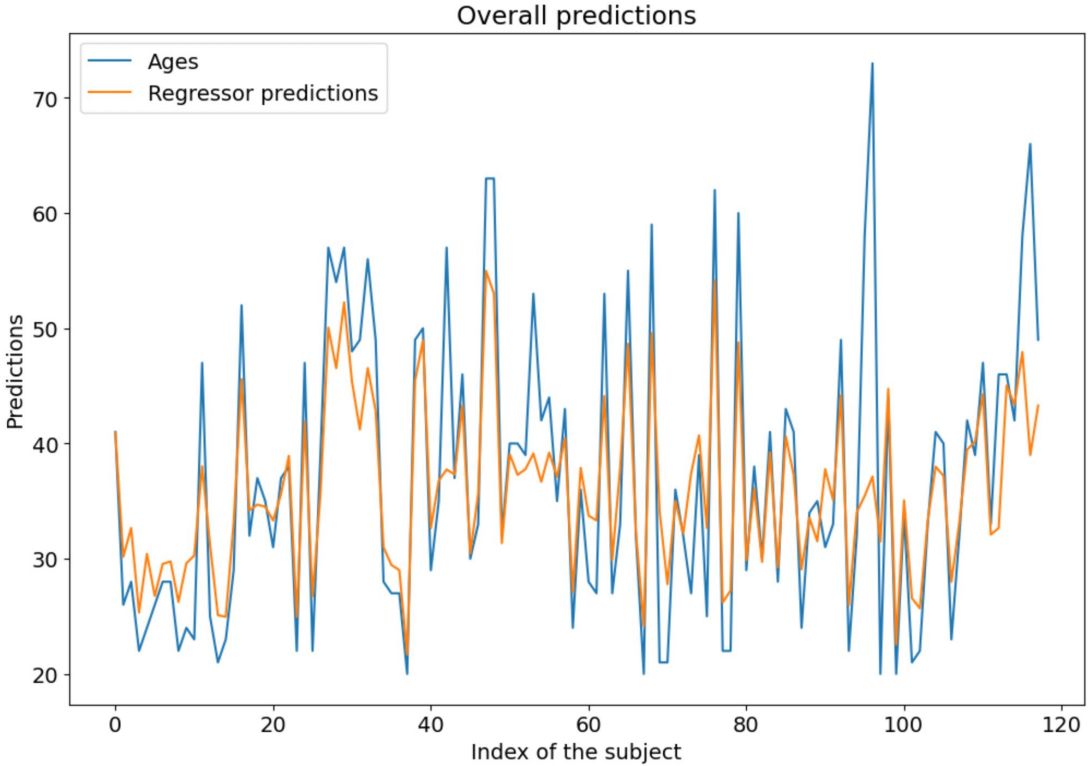
Methods – Metrics

- After training, the entire dataset was passed through the model and predictions were obtained.
- Metrics studied were Pearson's Correlation, MAE, R^2, and graphs were plotted.
- The same metrics were computed on the train data and test data individually.
- Note that the model was trained using the augmented labels, but the metrics were computed using the non-augmented labels.

Results – Metrics (Overall)

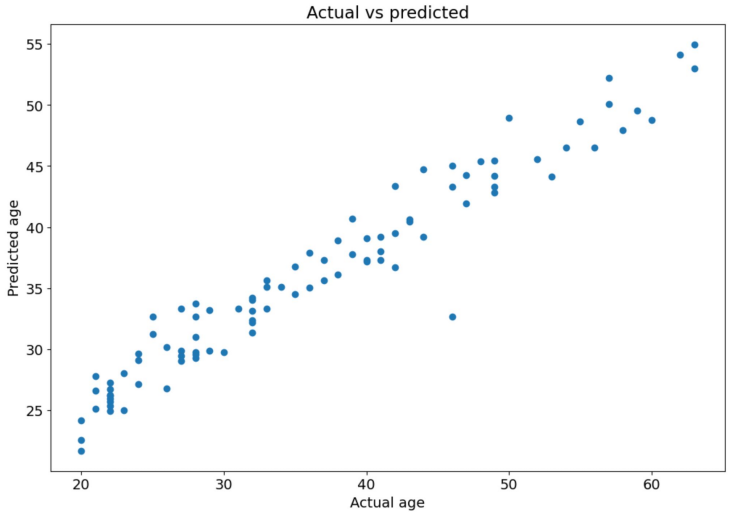
```
Overall measures:
Correlation: 0.8738383442089874
MAE:  4.7885209720193735
R2:  0.6874963958743534
```

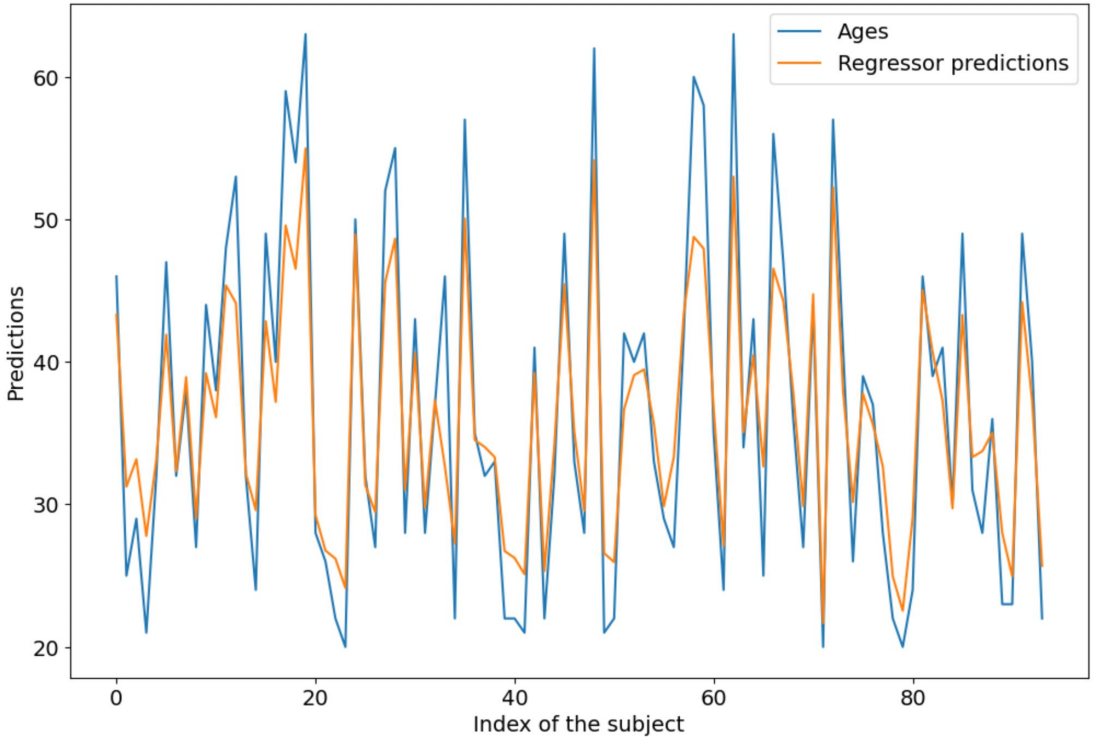




Results – Metrics (Train)

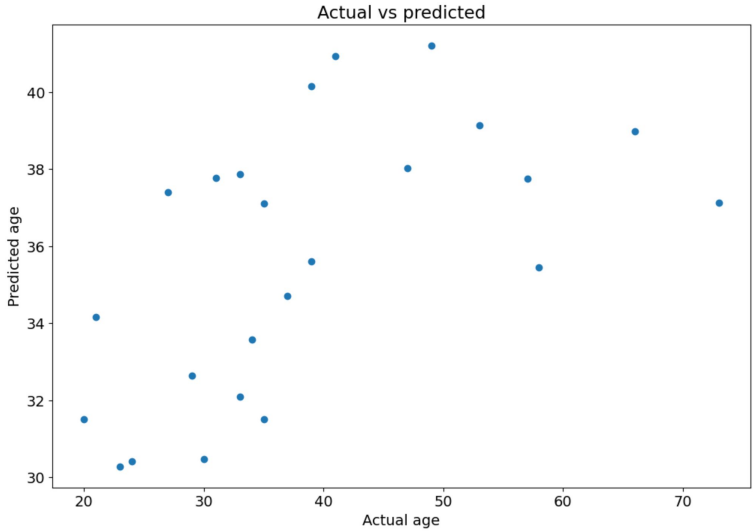
```
Train measures:
Correlation: 0.969296244086625
MAE:  3.7378855546938414
R2:  0.8495923048174515
```

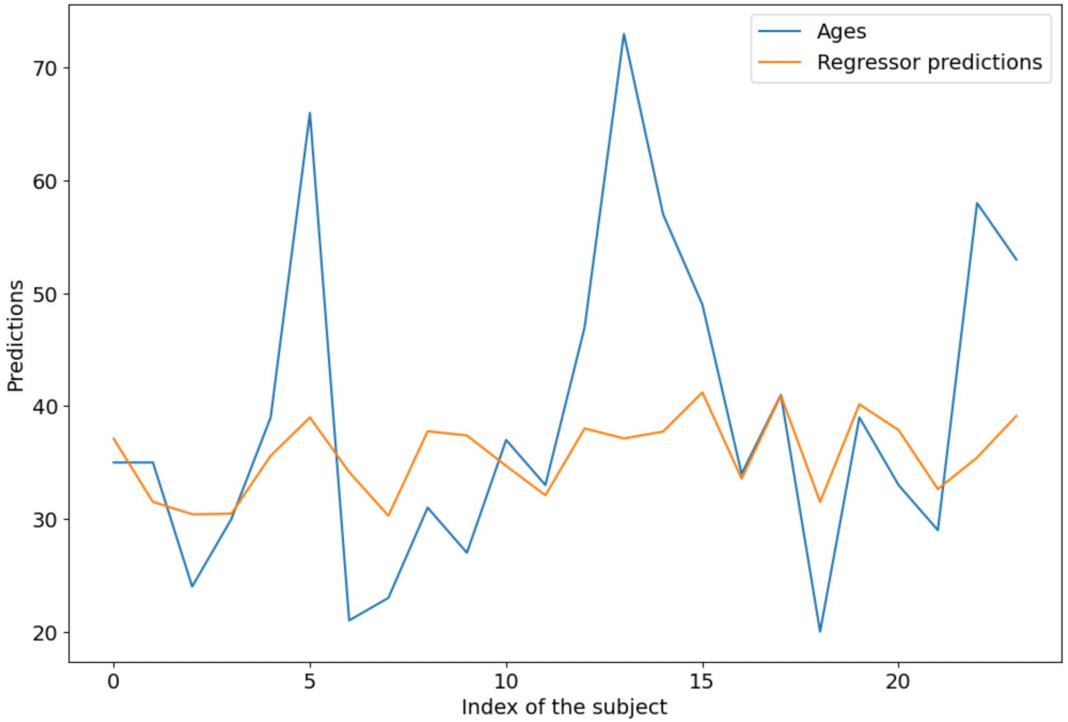




Results – Metrics (Test)

```
Test measures:
Correlation: 0.5738754975776376
MAE:  8.903509689877694
R2:  0.16578956282973234
```





Clearly Overfitting!

Methods – Cross-Validation

- Although training and testing could be done as above, it is natural that the values of errors would change with time based on the stochasticity of the process. So, a better estimate of test accuracy would be the validation accuracy through cross-validation.
- 10-fold cross-validation was performed on the data.

Results – Cross-Validation

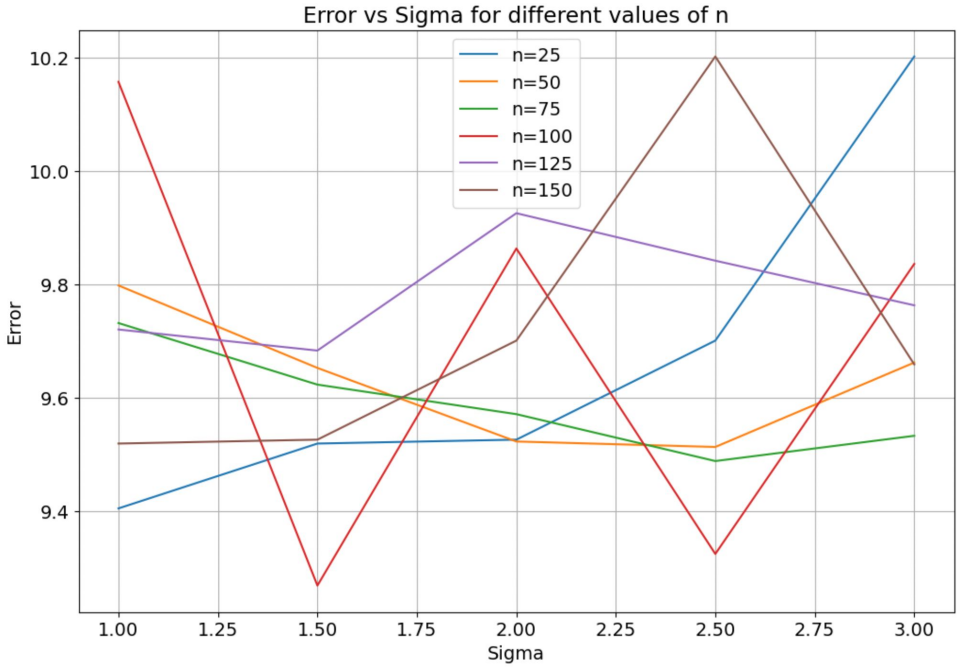
- The average validation MAE was 9.685 years

```
Iteration 1: Validation Error = 8.569850629256143
Iteration 2: Validation Error = 8.505061923530725
Iteration 3: Validation Error = 7.046444299877884
Iteration 4: Validation Error = 11.363765220225682
Iteration 5: Validation Error = 10.463647299219263
Iteration 6: Validation Error = 9.454924062169818
Iteration 7: Validation Error = 9.441967053843173
Iteration 8: Validation Error = 12.485720867303266
Iteration 9: Validation Error = 8.752154519206995
Iteration 10: Validation Error = 10.774753711677677
Average validation error:  9.685828958631062
```

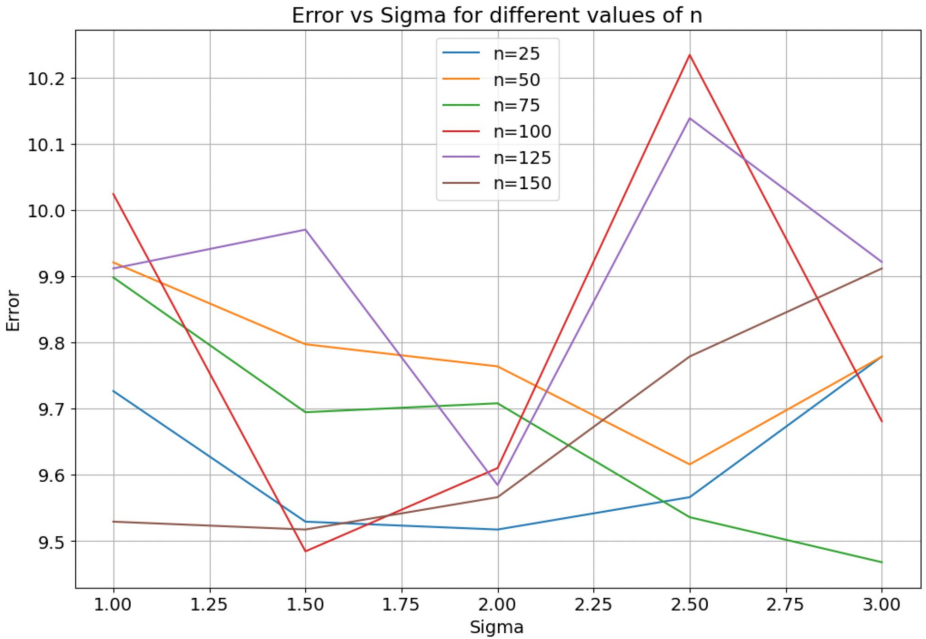
Methods – Grid Search

- A Grid Search was carried out with sigma (standard deviation of the augmentation of labels) as 1, 1.5, 2, 2.5, or 3, and the number of estimators of the Random Forest model as 25, 50, 75, 100, 125, 150.
- Each (sigma, n_estimators) pair was traversed and the cross-validation cycle was carried out, with the mean validation error outputted. The validation errors for different combinations were plotted.
- This code was run twice, and the plots of validation errors were recorded twice, to ensure there was no significant variation from run to run.

Results – Grid Search (Run 1)



Results – Grid Search (Run 2)



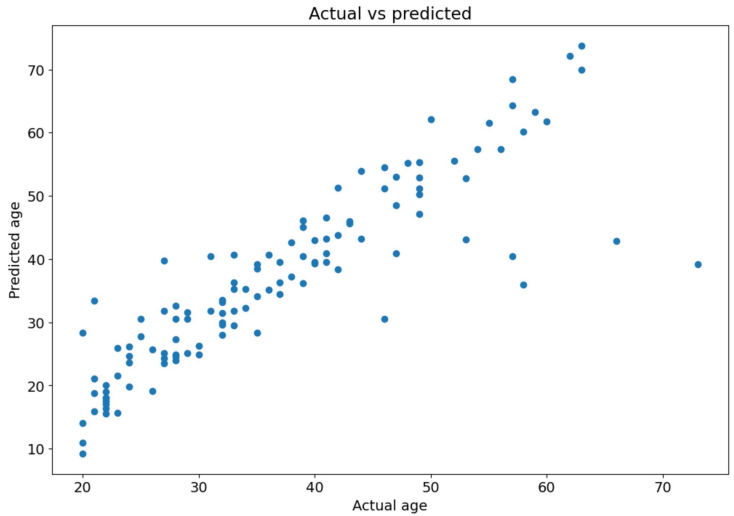
Between the runs, the best combination of parameters seems to be at n = 100 and sigma = 1.5

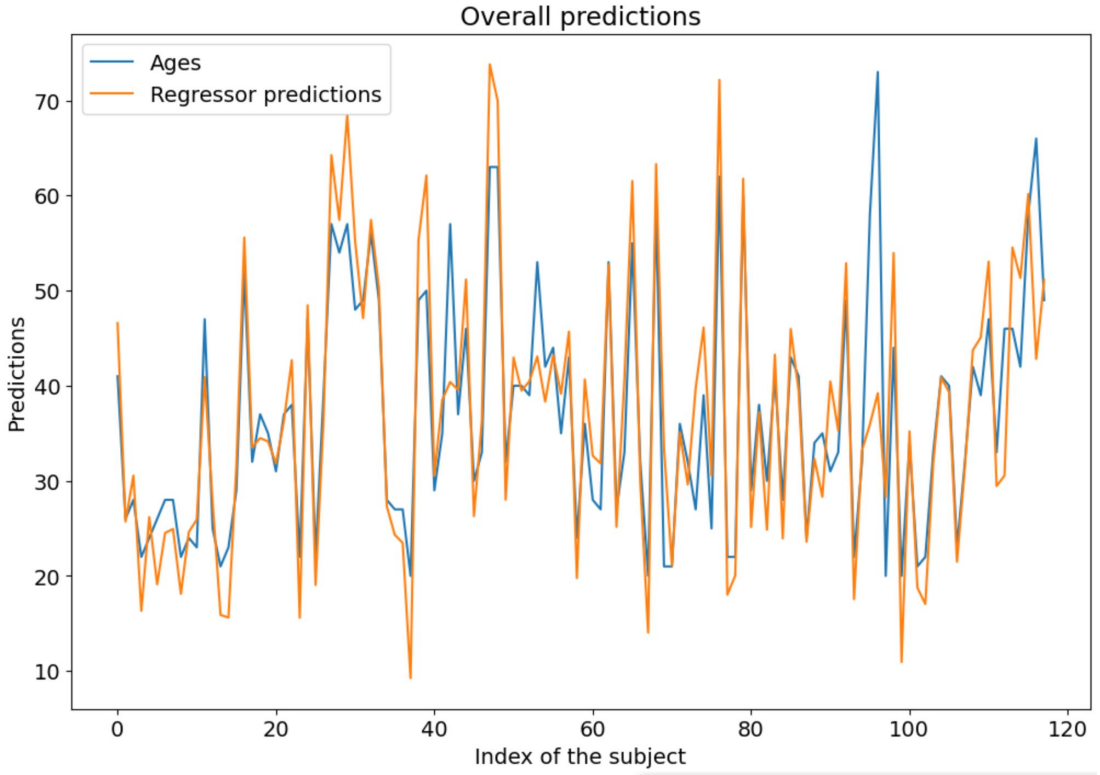
Methods – Linear Correction

- The slope and intercept are computed from the correlation graph of the train data alone.
- This is then used to correct the test and train data prediction and the results are analysed.
- Further, Cross-validation is carried out where each fold consists of training the data, finding the slope and intercept of the predictions of train data, and using that to transform validation predictions.
- The validation error is then analysed after implementing this linear correction.

Results – Linear Correction (Overall)

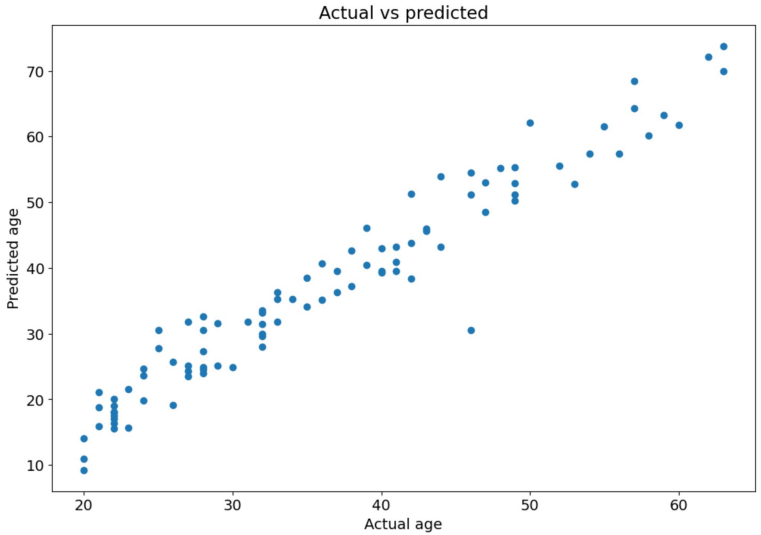
```
Overall measures:
Correlation: 0.8738383442089876
MAE:  4.865576575486653
R2:  0.6874963958743534
```

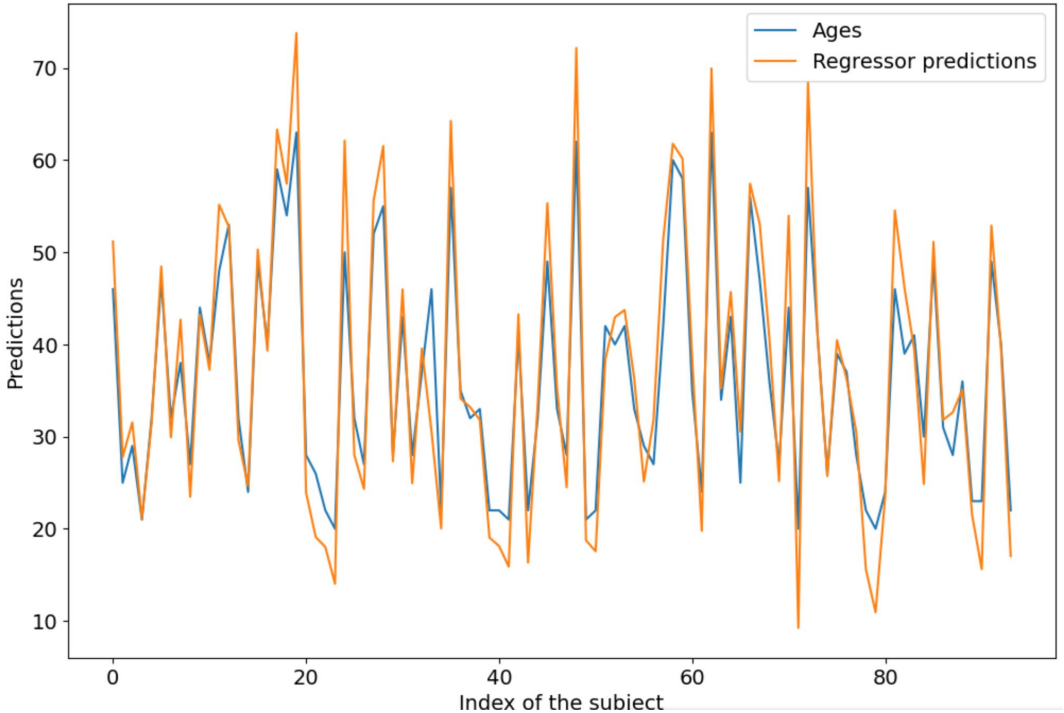




Results – Linear Correction (Train)

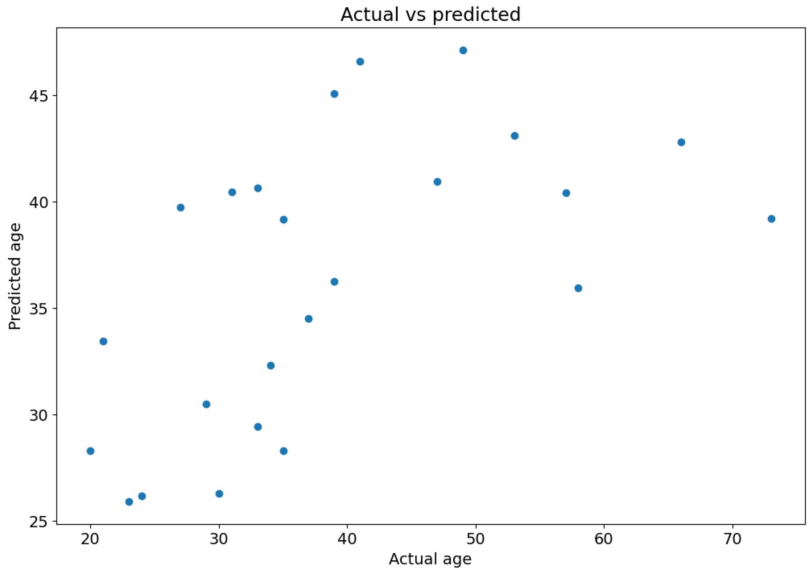
```
Correlation: 0.9692962440866247
MAE:  3.900971801467854
R2:  0.8495923048174515
```

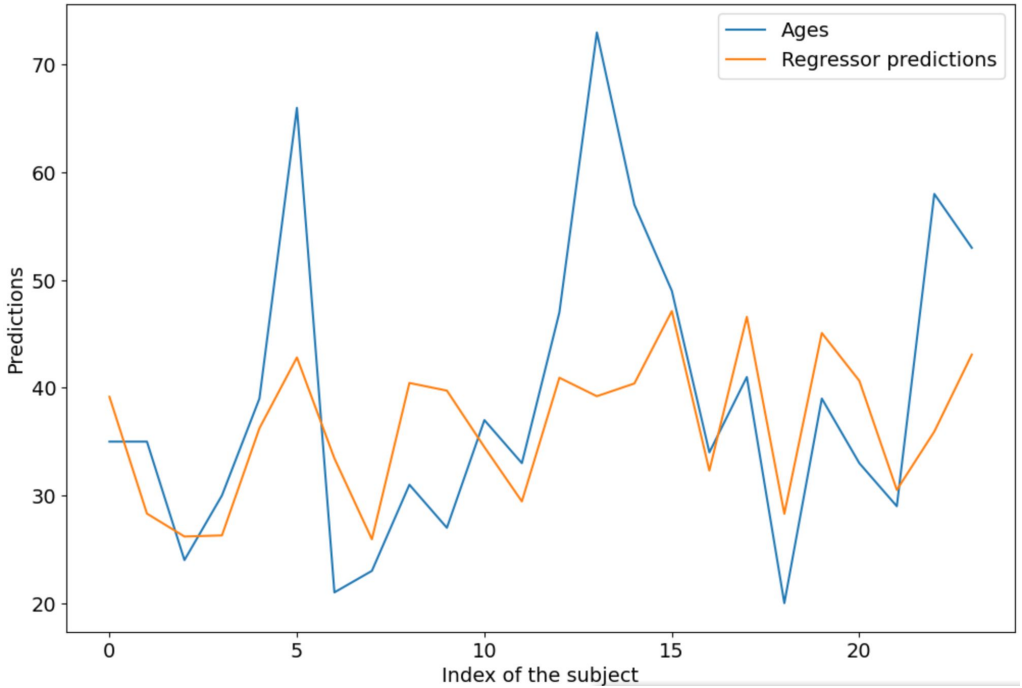




Results – Linear Correction (Test)

```
Correlation: 0.5738754975776377
MAE:  8.643611940393603
R2:  0.16578956282973234
```





Results – Linear Correction

- The average validation error is 9.816 after applying the linear correction.
- Without the correction, the validation error was around the same, so it appears that the linear correction did not really have an overall effect on the validation of the dataset.

```
Average validation error:  9.816883564843826
```

Next Steps?

- Trying a Graph Convolutional Network to input connectivity graphs directly and perhaps learn something through that. It will be able to make use of more features rather than just centrality.
- Try reducing features based on importance assigned by the random forest to reduce overfitting.
- Incorporate amplitude and spectral features, since many papers have claimed a relation between relative band powers and aging (so there could be some information there too).

GitHub Wiki

Move to the GitHub wiki.
I’ve updated the code and some wiki articles to go over the code.

Week 9

25th July, 2023

Nikhil Anand

Progress Summary

- Implemented augmentation of data points before cross-validation and performed grid search on the parameters – got an excellent MAE
- Figured out that there was leakage of augmented data into validation and test sets which led to this MAE.
- Learnt about feature importance methods in Random Forest.
- Realised that feature importances were not consistent across folds, likely due to small amount of data and overfitting on the specific data present in each fold.
- Experimented with other hyperparameters on the Random Forest to reduce complexity of the model and prevent overfitting.
- Ran the divide_epochs code in the server as well as extract_features code to extract amplitude and spectral features.

Methods – Data Augmentation and Feature Importance

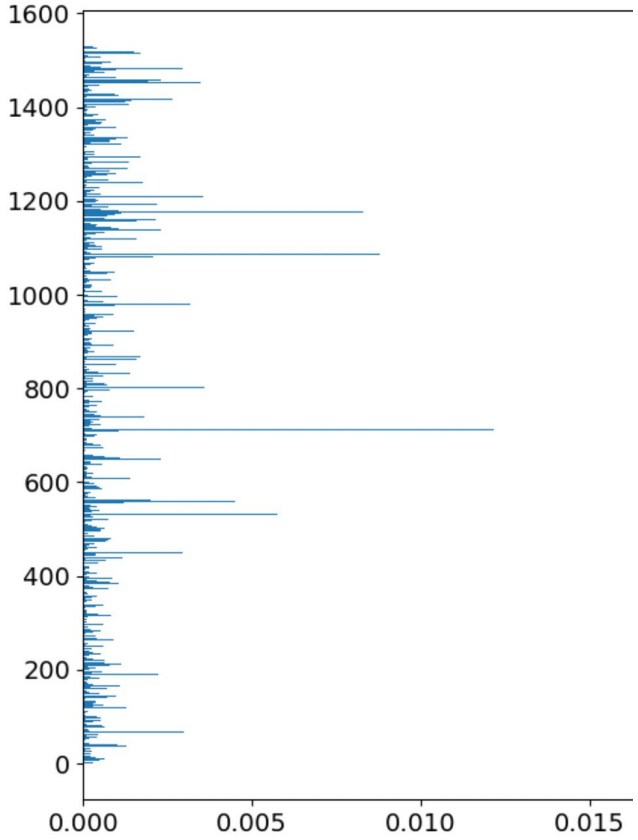
- Augmented X and y (feature and label sets) with noise that follows a normal distribution centred at 0 with a standard deviation of sigma (this hyperparameter was initially set to 0.5).
- Repeated this k times (initially, 10) and added it to the dataset, thus making the dataset 11 times larger.
- Divided the data into train and test sets and ran it through a RandomForestRegressor with n estimators (100 initially).
- The feature importance bar-plot was plotted after this.

Results – Data Augmentation And Feature Importance

```
X_aug = []
y_aug = []

X_aug.append(X)
y_aug.append(y)

for _ in range(10):
    noisex = np.random.normal(loc=0, scale=0.5, size=X.shape)
    noisey = np.random.normal(loc=0, scale=0.5, size=y.shape)
    X_aug.append(X + noisex)
    y_aug.append(y + noisey)

X_aug = np.concatenate(X_aug, axis=0)
y_aug = np.concatenate(y_aug, axis=0)

np.array(X_aug).shape,np.array(y_aug).shape
```

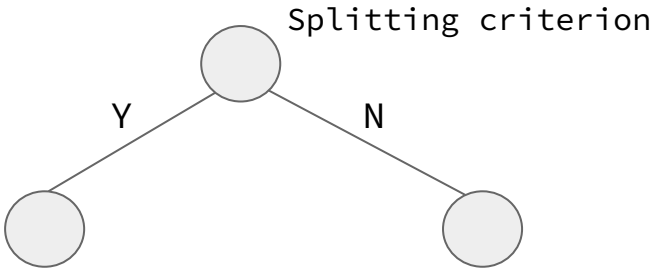
✓ 0.1s

```
((1298, 1530), (1298,))
```



Side Note: Feature Importance

- RFRegressor package uses the “mean decrease in impurity” measure for feature importance.
- Mean decrease in impurity is basically the “information gain” of a specific node, that is, the impurity of the node is computed before and after the split of the data. The difference is the mean decrease in impurity.
- For each feature and for each base learner (decision tree) of the RF, we find all nodes that use the given feature in that tree and compute the decrease in impurity, called the splitting criterion.
- All these gains are added up across all nodes of that feature to get that feature’s importance in a tree. This is then repeated across all trees and averaged to get that feature’s overall importance.

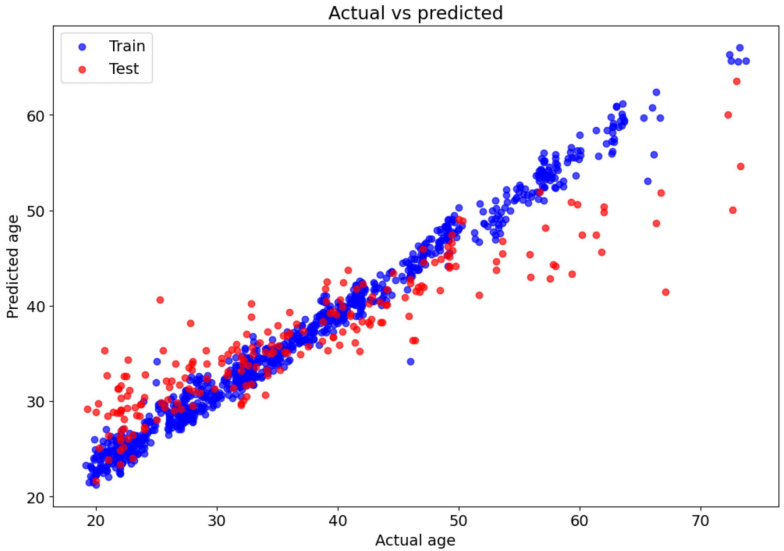


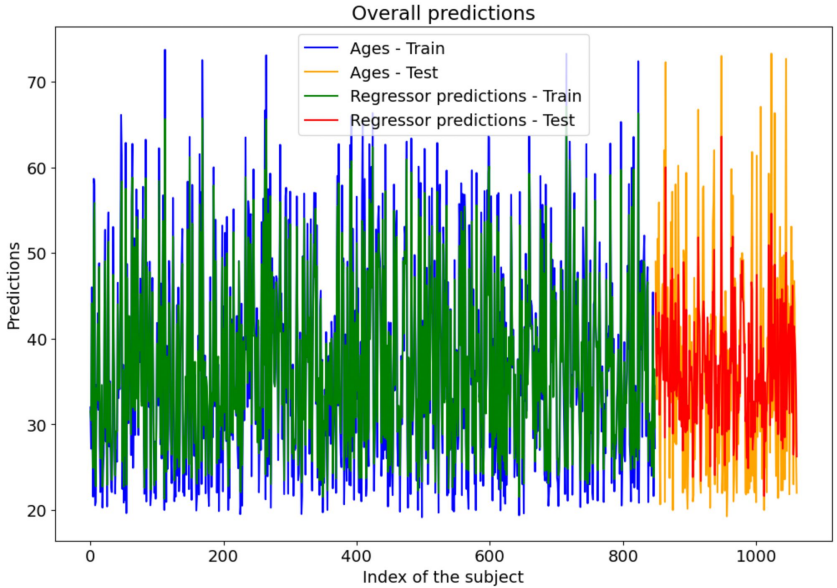
Methods – Metrics

- After training, the entire dataset was passed through the model and predictions were obtained.
- Metrics studied were Pearson's Correlation, MAE, R^2, and graphs were plotted.
- The same metrics were computed on the train data and test data individually and plotted on the same graph.

Results – Metrics (Train and Test)

```
Overall measures:
Correlation - Train: 0.9924640381938907, Test: 0.9194572680437152
MAE - Train:  2.0832338803160653
MAE - Test:  5.202763683638315
R2 - Train:  0.9543738482535076
R2 - Test:  0.7170712344771071
```





Methods – Cross-Validation

- Although training and testing could be done as above, it is natural that the values of errors would change with time based on the stochasticity of the process. So, a better estimate of test accuracy would be the validation accuracy through cross-validation.
- 10-fold cross-validation was performed on the data.

Results – Cross-Validation

- The average validation MAE was 5.02 years

```
Iteration 1: Validation Error = 5.241218907898747
Iteration 2: Validation Error = 4.3405758474731675
Iteration 3: Validation Error = 4.9771036114021765
Iteration 4: Validation Error = 4.778119273770197
Iteration 5: Validation Error = 5.14212831854521
Iteration 6: Validation Error = 5.211712420470219
Iteration 7: Validation Error = 4.3300497319777485
Iteration 8: Validation Error = 5.185063555456855
Iteration 9: Validation Error = 5.431942923172634
Iteration 10: Validation Error = 5.656821909521988
Average validation error:  5.029473649968894
```

Methods – Grid Search

- A Grid Search was carried out with sigma in [0.1, 0.3, 0.5, 1], the number of estimators of the Random Forest model in [50, 75, 100, 125], and k (multiple of extra augmented points) in [6,8,10,12,15].
- Each (sigma, n_estimators, k) triplet was traversed and the cross-validation cycle was carried out, with the mean validation error outputted. The validation errors for different combinations were plotted.
- This took 66000s to run (18.3 hours) – safe to say it was a bit of a waste of time.
- Both the error and the time it took to run was saved for each triplet.

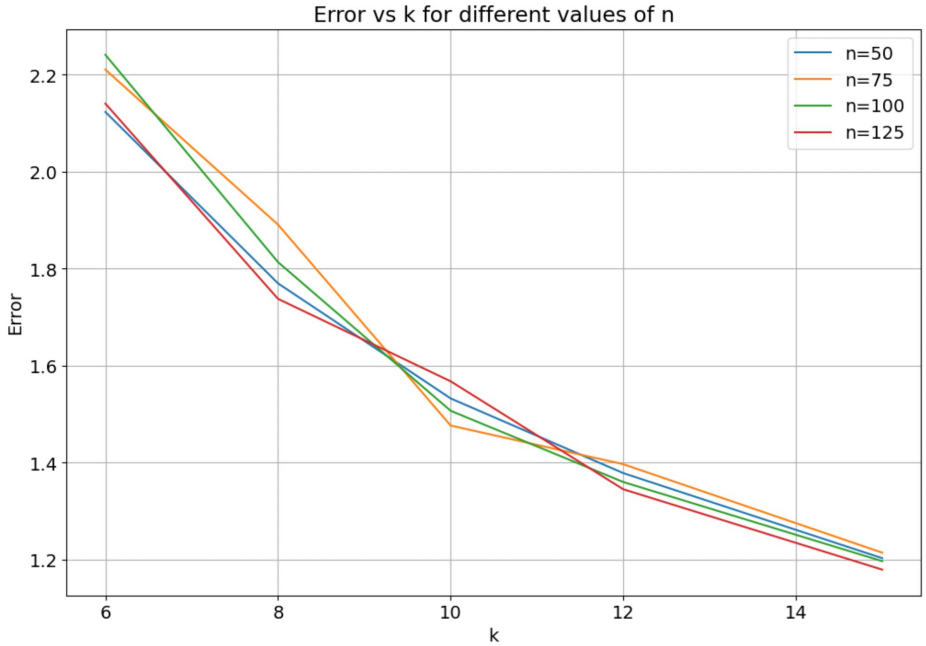
Results – Grid Search

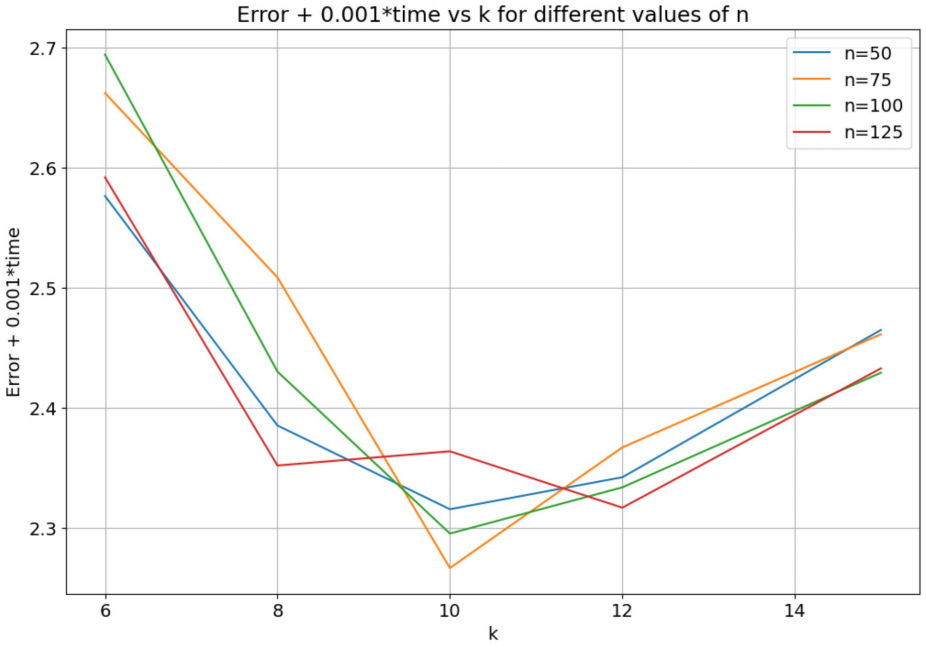
```
sigma, n_estimators, k, errors[i], times[i]
0.1 50 6 2.123250983860585 453.232919216156
0.1 50 8 1.7697958887252667 615.4992399215698
0.1 50 10 1.532520956577137 783.0041449069977
0.1 50 12 1.3784826844318918 963.6725449562073
0.1 50 15 1.2030712465081277 1261.8457012176514
0.1 75 6 2.210551039687014 451.79561018943787
0.1 75 8 1.8909582909170042 617.7272660732269
0.1 75 10 1.4766223296356207 789.9161767959595
0.1 75 12 1.396888805410257 970.119815826416
0.1 75 15 1.2144714680147513 1246.8568789958954
0.1 100 6 2.241098546302769 453.23779582977295
0.1 100 8 1.8135697719582127 616.643844127655
0.1 100 10 1.5070237202828594 788.2283709049225
0.1 100 12 1.3601275038884917 973.6084778308868
0.1 100 15 1.1969515264626422 1232.3524017333984
0.1 125 6 2.1404832101615003 451.5641610622406
0.1 125 8 1.7378186280110761 614.1331520080566
0.1 125 10 1.5678111284230938 795.9770369529724
0.1 125 12 1.3451873026163441 971.6059560775757
0.1 125 15 1.1794782507379347 1253.42227101326
0.3 50 6 4.286835848051948 454.43320393562317
0.3 50 8 3.8541404998810074 616.7751598358154
0.3 50 10 3.580935648253944 812.8118703365326
0.3 50 12 3.1768423955267004 981.7006332874298
0.3 50 15 2.9137530604750497 1262.8222060203552
0.3 75 6 4.259607815631118 453.2317440509796
0.3 75 8 3.779869507558781 634.3313491344452
0.3 75 10 3.4747383961343963 806.5165979862213
0.3 75 12 3.3252849123727146 1001.2140069007874
0.3 75 15 3.080503112997707 1284.6416409015656
0.3 100 6 4.124319321596761 455.2351200580597
0.3 100 8 3.737608490077014 613.6963279247284
0.3 100 10 3.4104726320928527 857.8154928684235
0.3 100 12 3.175827647757968 1097.3567090034485
0.3 100 15 2.8353585898221185 1292.2801749706268
0.3 125 6 4.183703446850557 452.571830034256
0.3 125 8 3.815476035675996 621.4726150035858
0.3 125 10 3.409905602243544 817.1797692775726
0.3 125 12 3.3362848861197043 987.097660779953
0.3 125 15 2.8495104267460425 1307.4048869609833
0.5 50 6 5.907816381196676 467.38573479652405
0.5 50 8 5.395212918970673 625.9393992424011
0.5 50 10 4.974361985050907 810.4163219928741
0.5 50 12 4.7324721877859695 981.8742320537567
0.5 50 15 4.4012864790267425 1258.0406680107117
0.5 75 6 5.776574460707119 448.6002109050751
0.5 75 8 5.357465981420109 624.325287103653
0.5 75 10 5.151092447623965 812.4187479019165
0.5 75 12 4.759355232207676 986.5483028888702
0.5 75 15 4.446821155148221 1301.2817389965057
0.5 100 6 5.738528535505697 456.9824409484863
0.5 100 8 5.349774513736412 622.6476709842682
0.5 100 10 5.075954105864357 814.2959086894989
0.5 100 12 4.708078826670521 992.0677070617676
0.5 100 15 4.460049010119934 1254.2637069225311
0.5 125 6 5.830854980126976 454.9686813354492
0.5 125 8 5.429897676524517 628.3578081130981
0.5 125 10 5.071865366518777 804.2803790569305
0.5 125 12 4.842387057604407 994.2819221019745
0.5 125 15 4.4639471190389886 1278.6514360904694
1 50 6 7.966131247610685 463.03659534454346
1 50 8 7.814180568153854 626.9705309867859
1 50 10 7.45132134819808 800.3845870494843
1 50 12 7.376658928821689 1042.9059393405914
1 50 15 7.041675484756364 1272.958072900772
1 75 6 7.952605763965242 459.40076875686646
1 75 8 7.80976002563384 616.2504198551178
1 75 10 7.52449701150576 795.4010350704193
```

Methods – Grid Search (Plotting)

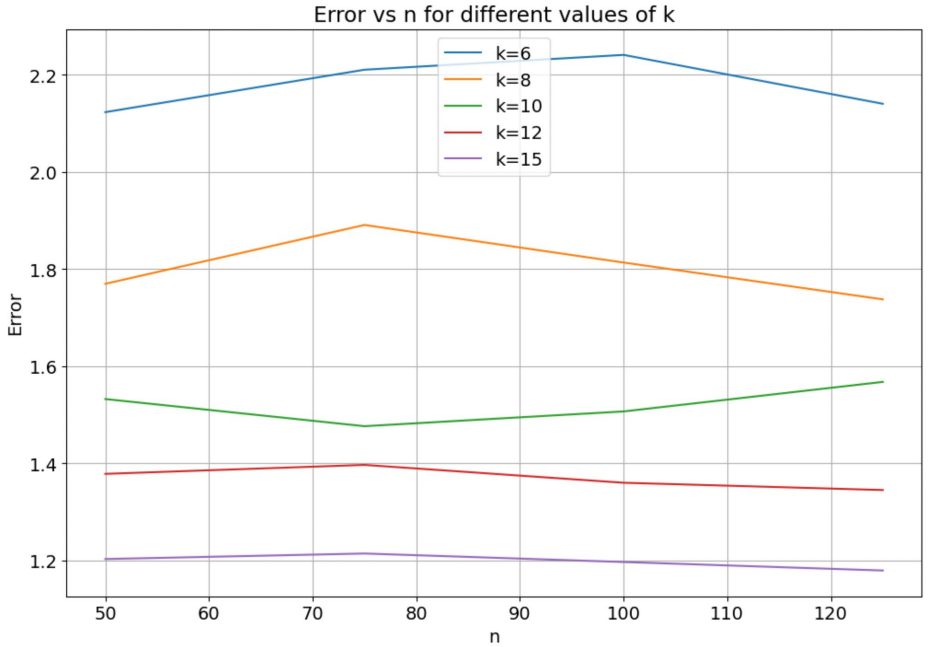
- The error seemed amazing at sigma = 0.1, and the results were plotted only for that sigma.
- I fixed k and varied n to observe the errors, as well as vice versa (all at sigma = 0.1).
- I also plotted the function (error + 0.001*time), which felt like I could see the minima that would have the best of both worlds (least time complexity and least error as well).

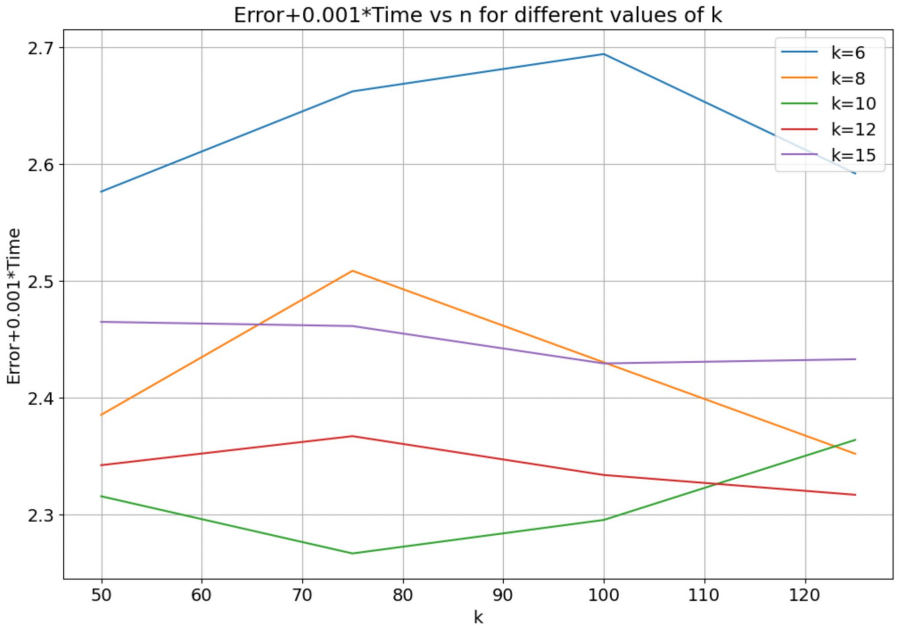
Results – Grid Search (Plotting)





Results – Grid Search (Plotting)





k = 10 and n = 75 seems to be ideal

Methods – Cross-Validation at Optimum Parameters

- Cross-validation was performed at the optimum parameters of n = 75 and k = 10.

Results – Cross-Validation at Optimum Parameters

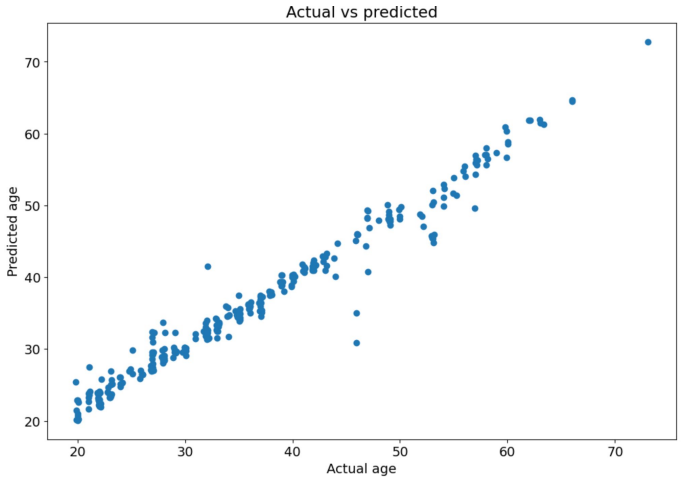
```
Iteration 1: Validation Error = 1.477621839683421
Iteration 2: Validation Error = 1.428831422922082
Iteration 3: Validation Error = 1.4934006967598052
Iteration 4: Validation Error = 1.3380493655651486
Iteration 5: Validation Error = 1.7180714842769105
Iteration 6: Validation Error = 1.4174241968114634
Iteration 7: Validation Error = 1.6799725522255298
Iteration 8: Validation Error = 1.681309359303384
Iteration 9: Validation Error = 1.5790709011461814
Iteration 10: Validation Error = 1.7832809149996844
Average validation error:  1.5597032733693612
Time:  842.0542769432068
```

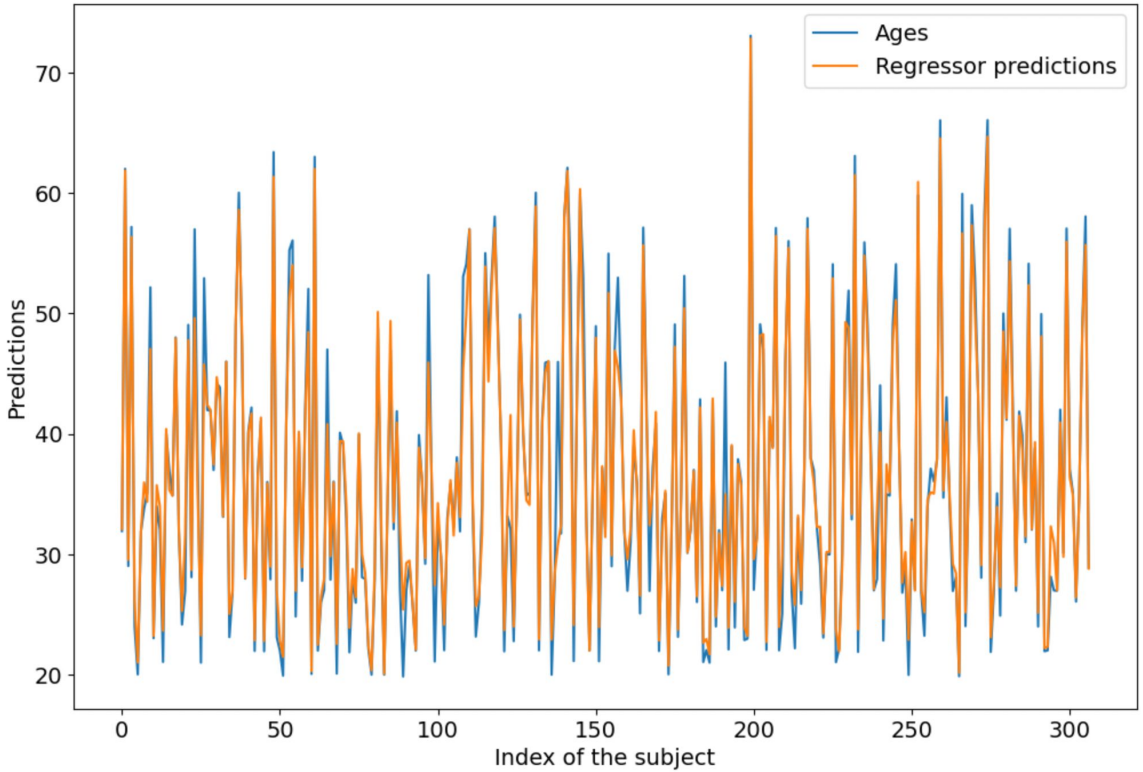
Methods – Testing

- We performed cross-validation considering only 80% of the augmented data. The remaining 20% was taken into the test set.
- This data was now used to test the model’s predictions and results were plotted.

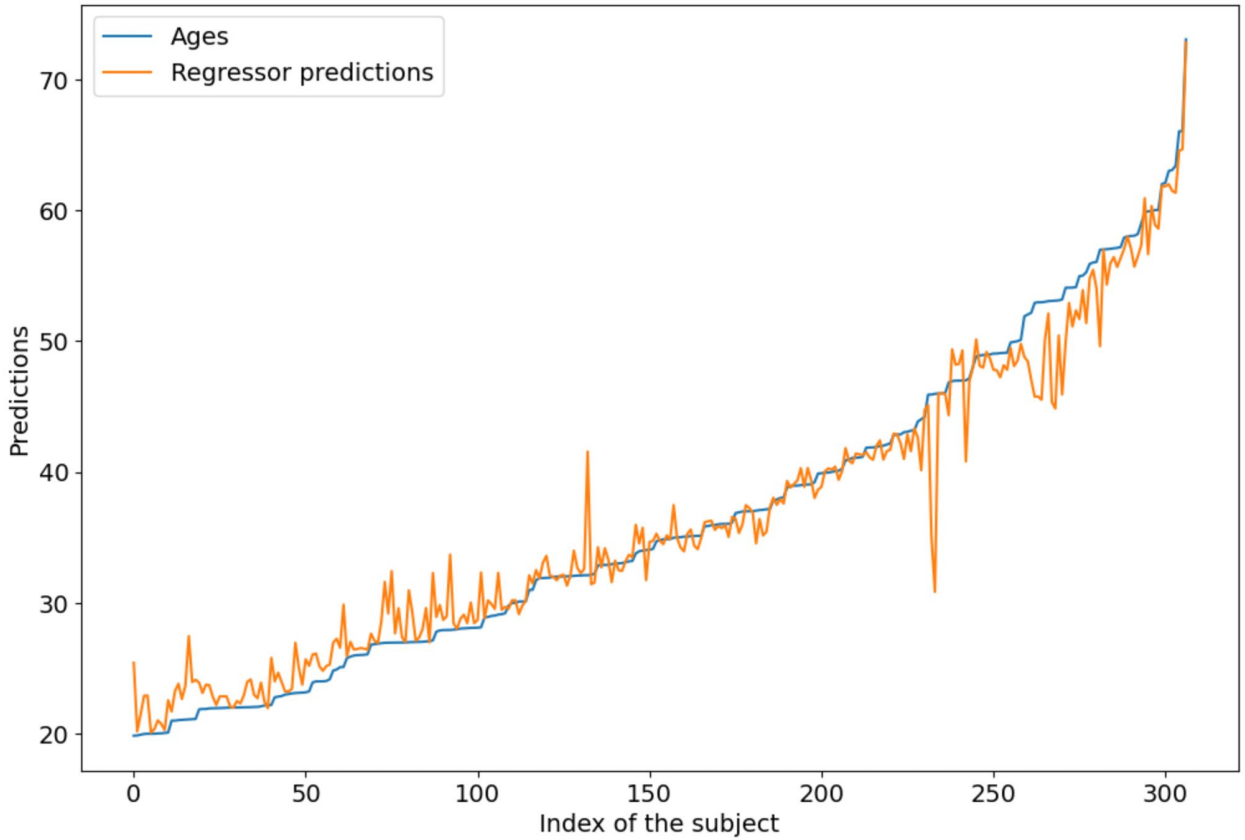
Results – Testing

```
Test measures:
Correlation: 0.9842323518884828
MAE:  1.4078725185729046
R2:  0.9624571936775189
```





Results – Testing



The issue above is that lowering sigma allows most elements to sit around discrete values, and this makes the validation error tend to zero as sigma decreases further and further. This is because of leakage of data from the train into the validation and test sets, so the model is able to memorise train data, and just sees the same data repeated in the validation and test sets.

Thus, first we should split the original data into train, validation and test datasets. Then, we should augment the train set alone to allow for attaining a good fit to the train dataset. We can optimise the parameters so that the fit is ideal on the validation set. However this optimisation process may lead to overfitting on the validation set, so we proceed to test the error of the model on the test dataset using these optimum parameters.

Methods – Changing the Augmentation Pipeline

- First the original 118 data points were divided into an 80-20 train-test split.
- Then, for k = 10, n = 75, and sigma = [0.2, 0.5, 1, 1.5, 2, …], K-fold cross validation was performed on the train data for each value of sigma.
- Once every train-validation split was made, the train data was augmented as per k, and this was fitted onto the Random Forest Regressor.
- Fold errors were computed on the validation set and were averaged across all runs

Results – Changing the Augmentation Pipeline

```
sigma = 0.2
Iteration 1: Validation Error = 11.290681484202157
Iteration 2: Validation Error = 8.879767805142372
Iteration 3: Validation Error = 10.38521890808616
Iteration 4: Validation Error = 10.059248307431195
Iteration 5: Validation Error = 6.691149999095297
Iteration 6: Validation Error = 10.297888981080515
Iteration 7: Validation Error = 10.613217058430612
Iteration 8: Validation Error = 14.631788737321834
Iteration 9: Validation Error = 9.930961933562875
Iteration 10: Validation Error = 9.12817917625674
Mean = 10.190810239060976

sigma = 0.5
Iteration 11: Validation Error = 12.815760971511398
Iteration 12: Validation Error = 17.21927293139512
Iteration 13: Validation Error = 11.106372712857649
Iteration 14: Validation Error = 7.584425037108593
Iteration 15: Validation Error = 9.679358875009775
Iteration 16: Validation Error = 10.887462112656575
Iteration 17: Validation Error = 7.8042702996585
Iteration 18: Validation Error = 8.800613984922204
Iteration 19: Validation Error = 8.33870603126519
Iteration 20: Validation Error = 13.66310379282829
Mean = 10.78993467492133

sigma = 1
Iteration 21: Validation Error = 10.610193061882004
Iteration 22: Validation Error = 11.985152135068644
Iteration 23: Validation Error = 14.043939586249573
Iteration 24: Validation Error = 7.328535587074032
Iteration 25: Validation Error = 11.703239138213847
Iteration 26: Validation Error = 10.280576551424765
Iteration 27: Validation Error = 8.560835525483482
Iteration 28: Validation Error = 12.739871711407984
Iteration 29: Validation Error = 11.937292319389844
Iteration 30: Validation Error = 10.310174553074763
Mean = 10.949981016926893
```

```
sigma = 1.5
Iteration 31: Validation Error = 10.163018013406576
Iteration 32: Validation Error = 7.757780034927189
Iteration 33: Validation Error = 11.915929784431732
Iteration 34: Validation Error = 8.522734111013646
Iteration 35: Validation Error = 15.13870640825816
Iteration 36: Validation Error = 8.097350785912713
Iteration 37: Validation Error = 10.62244625397755
Iteration 38: Validation Error = 13.426300407630796
Iteration 39: Validation Error = 8.176127090437303
Iteration 40: Validation Error = 12.328863946875767
Mean = 10.614925683687144

sigma = 2
Iteration 1: Validation Error = 10.503622572899971
Iteration 2: Validation Error = 11.731184134433068
Iteration 3: Validation Error = 12.850122501859195
Iteration 4: Validation Error = 10.514127452450307
Iteration 5: Validation Error = 7.930232380136129
Iteration 6: Validation Error = 9.81627694369923
Iteration 7: Validation Error = 12.232980808814482
Iteration 8: Validation Error = 8.32818314578333
Iteration 9: Validation Error = 9.23226969278037
Iteration 10: Validation Error = 8.093843320228277
Average validation error:  10.123284295308434

sigma = 2.5
Iteration 1: Validation Error = 6.969951729439877
Iteration 2: Validation Error = 10.884676190278993
Iteration 3: Validation Error = 9.845694639144144
Iteration 4: Validation Error = 8.437862218221904
Iteration 5: Validation Error = 11.013370404172658
Iteration 6: Validation Error = 12.464632893359473
Iteration 7: Validation Error = 10.904874106663875
Iteration 8: Validation Error = 9.148870586493961
Iteration 9: Validation Error = 6.582755522754493
Iteration 10: Validation Error = 14.333366790681493
Average validation error:  10.058605508121088
```

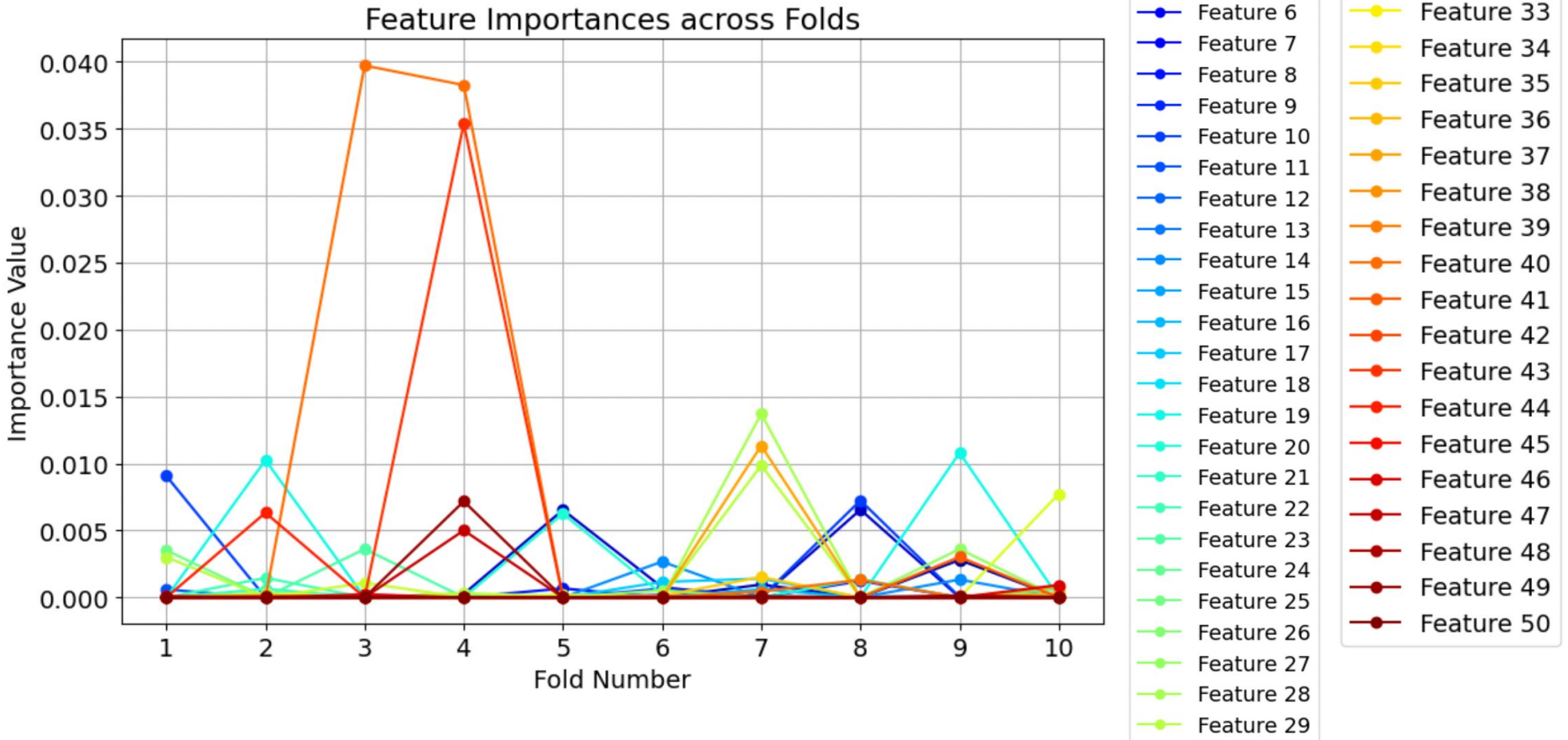
```
sigma = 3
Iteration 1: Validation Error = 11.010711854027663
Iteration 2: Validation Error = 6.971447119425984
Iteration 3: Validation Error = 9.970685422394931
Iteration 4: Validation Error = 11.121533287303347
Iteration 5: Validation Error = 8.894545812118443
Iteration 6: Validation Error = 13.156382839476818
Iteration 7: Validation Error = 10.053141437375197
Iteration 8: Validation Error = 9.027821160094671
Iteration 9: Validation Error = 11.367215366189729
Iteration 10: Validation Error = 7.548656369767473
Average validation error:  9.912214066817425

sigma = 4
Iteration 1: Validation Error = 8.904164247651305
Iteration 2: Validation Error = 8.99876898219417
Iteration 3: Validation Error = 8.844483992596384
Iteration 4: Validation Error = 11.231597414699342
Iteration 5: Validation Error = 9.441825537637776
Iteration 6: Validation Error = 9.092564588574348
Iteration 7: Validation Error = 16.4421570517758
Iteration 8: Validation Error = 6.734457046489257
Iteration 9: Validation Error = 10.427343203974134
Iteration 10: Validation Error = 10.629976229754902
Average validation error:  10.07473382953474

sigma = 5
Iteration 1: Validation Error = 9.626467586275874
Iteration 2: Validation Error = 10.992674824098629
Iteration 3: Validation Error = 6.019010086010706
Iteration 4: Validation Error = 8.333742582173029
Iteration 5: Validation Error = 12.731906985366672
Iteration 6: Validation Error = 10.270128368071662
Iteration 7: Validation Error = 12.23259470050709
Iteration 8: Validation Error = 7.9101196661705
Iteration 9: Validation Error = 14.369945242283004
Iteration 10: Validation Error = 7.657829326340261
Average validation error:  10.014441936729742
```

The original overfitting problem wasn't fixed.

Methods – Feature Importance

- Then experimented with feature importance of Random Forest, in hopes that reducing the feature set according to this might help tackle overfitting.
- However, it was noted that the feature importance was not even consistent across folds, either due to the small size of the dataset or because the model was overfitting to the specific data points so a change in the dataset would change the feature importances.
- Thus, it was clear that overfitting would need to be tackled before tackling feature importance.

Results – Feature Importance



Methods – Hyperparameters of the Random Forest Model

- Further experimented with hyperparameters of the Random Forest Model.
- The n_estimators, max_depth, and min_samples_split parameters must be set. For example, if max_depth is not set, the tree keeps splitting till it reaches purity. We obviously don't want complete purity for the training set since that leads to overfitting.
- Thus, a grid search was performed on these hyperparameters to observe the reduction of overfitting in the training error and to see the effect on validation error.

Results – Hyperparameters of the Random Forest Model

Train errors for (min_samples_split,max_depth) pairs

```
[(10, 3, 8.618556139626971),
 (10, 5, 6.731477858635867),
 (10, 8, 5.081527191698981),
 (10, 12, 4.806380726260778),
 (20, 3, 8.574351192110317),
 (20, 5, 6.94590919156784),
 (20, 8, 5.887001605353013),
 (20, 12, 5.873105116436852),
 (40, 3, 8.94417212048974),
 (40, 5, 7.741287063937614),
 (40, 8, 7.31581478645397),
 (40, 12, 7.317848240847026),
 (50, 3, 8.943055506714794),
 (50, 5, 7.98191398244803),
 (50, 8, 7.71979907950382),
 (50, 12, 7.621048812826537)]
```

Overfitting reduced (since train errors are larger)

Results – Hyperparameters of the Random Forest Model

Validation errors for (min_samples_split,max_depth) pairs

```
[(10, 3, 10.058211480205411),
 (10, 5, 10.29630041950953),
 (10, 8, 10.060921847867213),
 (10, 12, 10.281937612921135),
 (20, 3, 10.207241280850312),
 (20, 5, 10.085054531707033),
 (20, 8, 10.226046712458606),
 (20, 12, 10.500655661122945),
 (40, 3, 10.11328214235266),
 (40, 5, 10.234693643210791),
 (40, 8, 10.301503816299062),
 (40, 12, 10.037984303403036),
 (50, 3, 9.90035965193405),
 (50, 5, 10.070546745640879),
 (50, 8, 10.616120091867796),
 (50, 12, 10.30033597359564)]
```

Unfortunately, validation errors won't budge.

Methods – Feature Extraction Code (Server)

- Ran the “divide_epochs.m” MATLAB script on the server and prepared all the preprocessed data (152 subjects) for amplitude and spectral feature extraction.

Next Steps? (Some Ideas)

- Complete feature extraction (amplitude and spectral features) of the 152 subjects so that these features can be included in the model.
- Run MATLAB preprocessing script to finish rough preprocessing of the remaining subjects so they can be used as well.
- Try a basic bin-based **single** decision tree classifier model (perhaps 20 bins of 4 years width each). This would be the simplest version of the RF and I can try to build the RF from the ground to control overfitting.
- More graph-based features? GCN? (Might be useful to try out)

Week 10

25th July, 2023

Nikhil Anand

Progress Summary

- Completed all documentation on Github.
- Extracted amplitude and spectral domain features from the first 152 data points.
- Tried different combinations of features on the model and checked MAE.
- Preprocessed remaining 152 subjects using the script on the server (stored at ‘spacelin2/nikhil/preprocessed’).

GitHub

Move to the GitHub wiki.
I’ve updated the code and some wiki articles to go over the code.