BE521 Final Project Report

PREDICTION OF FINGER FLEXION FROM ELECTROCORTICOGRAPHY DATA

Ziyi Yang, Miranda Wang
University of Pennsylvania, Philadelphia, PA, USA
E-mail: ziyiyang@seas.upenn.edu, jingxinw@seas.upenn.edu

***Abstract* – This report presents the algorithms we developed to predict finger movements from electrocorticography (ECoG) data based on datasets from BCI Competition IV, which took place in Berlin at 2008. The goal of this project is to perform analysis of data, discover relevant relations within data, build prediction model based on optimal linear decoder method as described in Warland et al., 1997. to achieve a good result.**

***Keywords* – BE521, BCI Competition IV, ECoG data, Linear regression.**

I. INTRODUCTION

The data set we were given for this project was obtained from three epilepsy patients who had electrode grid placed on the surface of the brain to localize seizures (Miller Schalk 2008). Subjects were wearing data glove from Fifth Dimension Technologies, which makes it possible to record movements of all fingers. Every 2 seconds random cue was presented on computer monitor indicating which finger to move. Subject then moved required finger several times. Each cue was followed by 2 seconds rest interval. We got 300s long data set recorded with sampling rate 1000Hz. We split it into training(200s) and testing(100s) set to build our prediction model. We mainly focused on data post processing, feature extraction to optimize our filter weights matrix.

II. Methodology

This section presents the main method to create our prediction model.

A. ECoG Data processing

Electrocorticography (ECoG) data is obtained by placing grid of platinum electrodes directly on surface of the brain and recording readings from these electrodes.

Subject	Number of channels
1	62
2	48
3	64

Fig. 1. Number of channels in each subject data

Training data contains 200'000 samples at sampling rate 1000 Hz for each channel. Test data contains 100'000 samples at same sampling rate.

The very first step for post processing the data is to visualize it. By plotting out the standard deviation of each channel, we could find there are some channels contain outlier(as shown in figure 2 data points which would affect our prediction model. As we can tell from the plots, channel 55 in object1 and channel 21/38 in object2 does have noise data and we removed those channels.

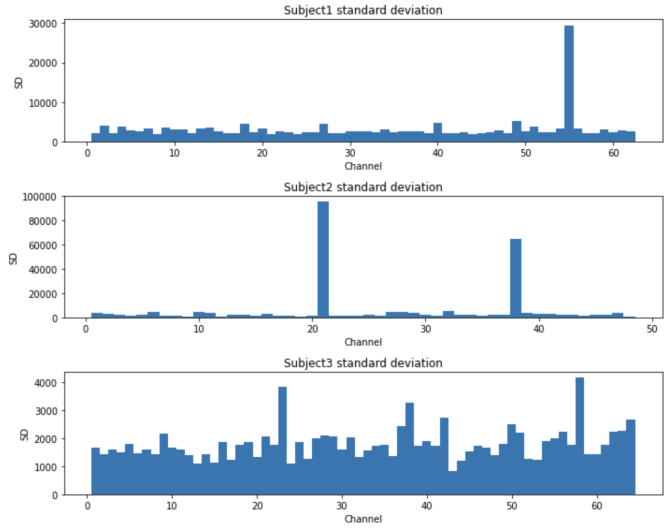

Fig. 2. Standard Deviation of each channel for each subject

Then we checked the correlation between each channel to optimize our channel selection. As shown in figure 3, the average correlation between each channel is 0.377 for subject1, 0.146 for subject2, 0.193 for subject3.

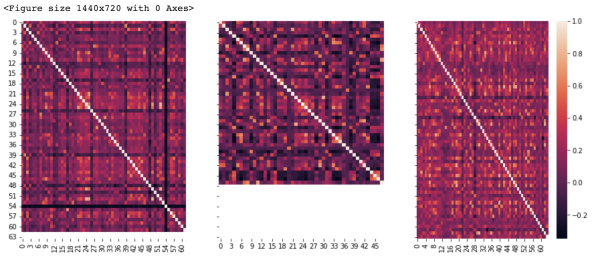

Fig. 3. Correlation between channels subject1 to 3 from left to right

B. Data Glove data

Data glove data is the record of finger movement of each subject. Our goal is to find the relation between ECoG data and Data glove data and try to predict the finger movement with the given ECoG data. The $dg_data is shown in figure 4$.

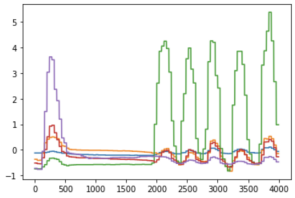

Fig. 4. First second of data glove data for subject 1

C. Prediction model algorithm

We followed the method that was given in final project part1, which is optimal linear decoder method(Warland et al., 1997).

1) Filtering and channel selection: After doing some research for the neural oscillation which is also known as brainwaves, there are frequency bands that better describes brain activities. Delta (1–4 Hz), theta (4–8 Hz), beta (13–30 Hz), low gamma (30–70 Hz), and high gamma (70–150 Hz) frequency bands. We first tried a butterworth bandpass filter with a focus on 1-200Hz signals.And the correlation heatmaps also shows that the correlation of the finger movement is higher within that frequency band(as shown in figure 5. We only plot the frequency correlation heatmap for object 2 since it crashes the colab when running the other two. In this process we found that some specific channel has a relative high correlation with finger flexion. Subject2 has more channels for each finger. Subject3 has less pronounced correlation differences, but still for most fingers there are specific significant channels and frequency ranges. Generally it means that flexion of specific finger is accompanied by increase in power for some frequency range in specific channel. The largest correlation obtained with this approach is for subject2 from channel 24 and frequency 110 Hz has correlation 0.37 with finger1 flexion data. We finally selected channel 1:5,10,15:28,34:47 for subject1, channel 7:13,15:30,31,34,35,36,38,43,44,45 for subject2, channel 7:29,34,35,41,44,46,48,49,51,54,55,57,61 for subject3. We also added the 37 ms delay to the ECoG signal as it was stated in the original BCI competetion documents that it is the hardware delay.

For normalizing the ECoG data we first tried Z-score method, but we found that common average reference method provides a better result.

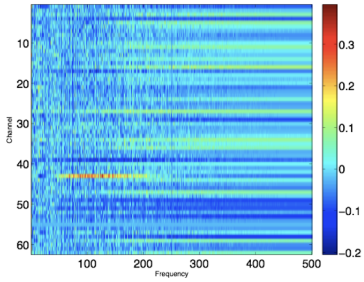

Fig. 5. Correlation with different frequency band for object2

2) Feature extraction: We used a window size of 100ms to extract our selected features. We have implemented the following features: Powerband(6 different frequency bands), linelength, zero-crossing, average, power, area. During the testing process, we noticed while choosing multiple features the final correlation decreases. So we decided to focus on choosing the best performing feature for each subject. So the final model we are using Power and linelength feature for subjust1 and only power feature for 2 and 3.

3) R-matrix: We found that changing the window size inside R matrix also impact our test result, By adjusting the R-matrix's to train the optimal linear decoder model and improve predictions correlations. We tried the number of windows used in each row of the output R matrix from 3 to 10 and found that 10 would give the best result. Since we were provided a limited amount of ECoG data with which to train the model, creating an R matrix that contains an increasing number of preceding feature windows improved the training and testing accuracy of our model, but which may lead to an over fitting. But after checking the leaderboard result, seems like the choice is decent.

4) Re-sampling data: Because we are using a window of 100ms and 50ms overlap to extract our features, the final data would be down sampled. In order to match the dimension of the R matrices for each subject, we had to down sample to the provided data glove finger flexion data. We tried a few different methods to do so, and found that the best results came from using a sliding time window-average down sampling.

After computing the optimal linear decoder predictions. We also need to up sample the result so that our prediction can match the original sampling rate. We first used cubic spline interpolation method to up sampling and it worked well. But we checked the original data glove data, looks like it has been up sampled with a more straight forward way by duplicate each data a certain times to match the sampling rate.So we duplicated each data point in our prediction 50 times and the leader board correlation slightly increased by 0.5

5) Discarded methods: More feature combination—We first followed the given method that used 4 or more feature combinations to create R-matrix. Even the selected feature

has a high correlation with the data glove data(about 20-30 percent).When out them together, the result is poor. This might be caused by wrongly normalized our feature windows. But since one or two feature worked well, we didn't go deep into this.

Machine learning model—First we tried logical regression and SVM by labelling each window with 1-5 represents the 5 fingers and assign dummy data glove amplitude to it, the result is not as good as we expected as the subject are moving there finger up and down, we couldn't simulate that process. And then we tried to use random forest method but the prediction result couldn't compete with filter matirx method so we decided to focus on the linear decoder.

III. Result

This section presents the final algorithm and observations

A. Flow chart

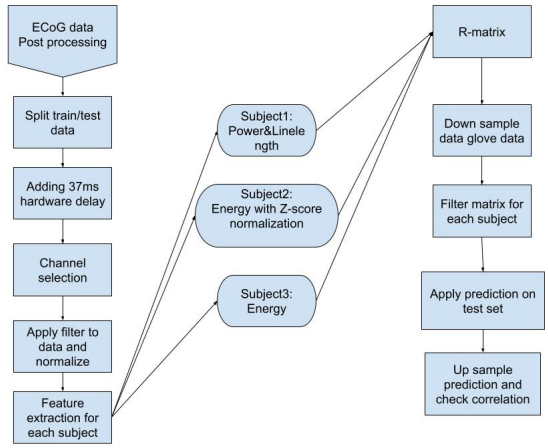

Fig. 6. Work Flow chart

B. Competition result

1) Leader Board: The final correlation for leader board competition is 0.6306

2) Hidden data: The correlation for hidden test set is 0.49

IV. Conclusion

A. 4th Finger discussion

Both of us were curious about the fact behind it. First we tried to find the correlation between each fingers.

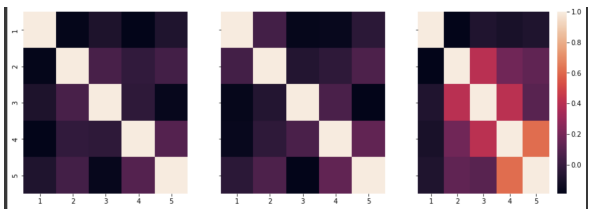

Fig. 7. Correlation between each finger

	subject1	subject2	subject3
1	0.133	0.211	0.108
2	0.211	0.273	0.305
3	0.191	0.205	0.367
4	0.207	0.272	0.419
5	0.213	0.268	0.351

Fig. 8. Correlation between each finger

Although in competition documentation it is said that finger high correlation with fingers 3 and 5, this correlation analysis shows that there is nothing special about finger 4. Instead thumb and index fingers seem to be most correlated with other fingers.

Since the potential muscle and tendon structures of finger 4 associate it with fingers 3 and 5 when moving the finger, it is assumed that there is a correlation between the flexion of finger 4 and fingers 3 and 5. If a person moves the ring finger, it is almost impossible not to move the adjacent finger 3 or 5. Vice versa, if you move finger 3 or finger 5, it is almost impossible not to move finger 4. In addition, when moving finger 3 or 5 alone, trying to keep finger 4 stationary requires muscle strength. Therefore, in the experiment of this competition, subjects may not be able to move fingers 3 or 5 independently without moving 4 slightly, especially considering that the length of 10 minutes may lead to finger muscle fatigue.

B. Sum-up

Our team enjoyed the process of the competition and thought it was a comprehensive review of be 521 course. We spent a lot of time drafting our method in google colabs. The most valuable of our methods is the iterative ability of our model: we can test many times, view the results, and modify them as needed. When we are aware of overfitting, we continue to optimize the algorithm and improve performance. Our team applied many concepts of course setting, such as post processing / filtering data, or simple functions, such as area and line length. In addition, we found some new technologies in this project, which gave us more insight. For example, we found some interesting prediction models in the research, which can be implemented in the future. From the time we spent the whole night testing various iterations to seeing our relevance score improve on the leaderboard, our team felt that we did well in this game and was eager to apply our knowledge to other projects and fields. Some of the projects we hope to explore include other ECoG data sets, virus tracking projects and foreign exchange trading algorithms. We thank professors and teaching assistants for providing us with feedback and course materials to guide us in designing machine learning models.

V. References

1. Working with ECoG data. [online] Available at: <https://mne.tools/stable/auto$_t$utorials/clinical/30$_e$cog.html> .

2. Warland, D. K., Reinagel, P., Meister, M. (1997). Decoding visual information from a population of retinal ganglion cells. Journal of neurophysiology.

3. Wang, Z., Miller, K. J., Schalk, G. (2011).

Prior knowledge improves decoding of finger flexion from electrocorticographic signals. Frontiers in neuroscience, 5, 127.

4. Neural oscillation. [online] Available at: <https://en.wikipedia.org/wiki/Neural_oscillation > .

5. learning discriminative patterns for self-paced EEG-based motor imagery detection. [online] Available at: <https://www.frontiersin.org/articles/10.3389/fnins.2012.00007/full>.

VI. Appendix

A. Code

```
#Import all pkgs here
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pickle
from scipy.stats import pearsonr
from scipy import signal as sig
from scipy.signal import ellip,butter, filtfilt,sosfiltfilt,iirnotch
from scipy.io import loadmat
from scipy.signal import butter, sosfilt, sosfreqz
from sklearn import preprocessing
from numpy.linalg import inv
#Drive file
from google.colab import drive
drive.mount('/content/drive')
```

```
#split training ecog data
ecog_1 = np.array(raw_training_data['train_ecog'][0][0])
ecog_2 = np.array(raw_training_data['train_ecog'][1][0])
ecog_3 = np.array(raw_training_data['train_ecog'][2][0])

#split training data glove data
dg_1 = np.array(raw_training_data['train_dg'][0][0])
dg_2 = np.array(raw_training_data['train_dg'][1][0])
dg_3 = np.array(raw_training_data['train_dg'][2][0])

#split testing data
ecog_test_1 = ecog_1[37:100037]
ecog_train_1 = ecog_1[100000:300000]
ecog_test_2 = ecog_2[37:100037]
ecog_train_2 = ecog_2[100000:300000]
ecog_test_3 = ecog_3[37:100037]
ecog_train_3 = ecog_3[100000:300000]

#split data glove data
dg_test_1 = dg_1[0:100000]
dg_train_1 = dg_1[100000:300000]
dg_test_2 = dg_2[0:100000]
dg_train_2 = dg_2[100000:300000]
dg_test_3 = dg_3[0:100000]
dg_train_3 = dg_3[100000:300000]

#split leaderboard data
ecog_lb_1 = np.array(leaderboard_data['leaderboard_ecog'][0][0])
ecog_lb_2 = np.array(leaderboard_data['leaderboard_ecog'][1][0])
ecog_lb_3 = np.array(leaderboard_data['leaderboard_ecog'][2][0])
```

```
from IPython.core.pylabtools import figsize
import pandas as pd
import seaborn as sn
f=plt.figure(figsize=(20,10))
f,(ax1,ax2,ax3) = plt.subplots(1,3,sharey=True,figsize=(
data = dg_1
df = pd.DataFrame(data,columns=np.arange(5)+1)
corrMatrix1 = df.corr()
g1 = sn.heatmap(corrMatrix1,cbar=False,ax=ax1)
g1.set_ylabel('')
g1.set_xlabel('')
result = np.zeros((5,3))

data = dg_2
df = pd.DataFrame(data,columns=np.arange(5)+1)
corrMatrix2 = df.corr()
g2 = sn.heatmap(corrMatrix2,cbar=False,ax=ax2)
g2.set_ylabel('')
g2.set_xlabel('')
print(corrMatrix2)
data = dg_3
df = pd.DataFrame(data,columns=np.arange(5)+1)
corrMatrix3 = df.corr()
g3 = sn.heatmap(corrMatrix3,ax=ax3)
g3.set_ylabel('')
g3.set_xlabel('')
print(corrMatrix3)


plt.show()
```

```
def filter_data(data, fs=1000,bandpass=[1,490],order=2):
  data_t = data.transpose()
  result = []
  nyq = fs/2
  sos = butter(order,bandpass,btype='bandpass',output='sos',fs=fs)
  for i in data_t:
    temp= sosfiltfilt(sos,i)
    result.append(temp)

  result = np.array(result)
  return result.transpose()
def car(data):
  for i in range(np.shape(data)[0]):
    mean=np.mean(data[i,:])
    for j in range(np.shape(data)[1]):
      data[i,j]-=mean
```

```
from scipy.signal import welch
def bandpower(x, fs=1000,band=[5,15],time=0.1):
    # f, Pxx = scipy.signal.periodogram(x, fs=fs)
    #f, Pxx = welch(x, fs, nperseg=fs*time)
    fmin=band[0]
    fmax=band[1]
    freq_data = np.fft.fft(x)
    frequency = np.linspace (0.0, 500, int(100/2))
    y = 2/100 * np.abs (freq_data [0:int(100/2)])
    ind_min = np.argmax(frequency > fmin) - 1
    ind_max = np.argmax(frequency > fmax) - 1
    return np.mean(y[ind_min:ind_max])

#average voltage feature
def ave_voltage(data):
  sum=np.sum(data)
  ave=sum/len(data)
  return ave

#line length feature
def LL(data):
  result = np.sum(np.absolute(np.ediff1d(data)))
  return result

def Area(data):
  result = np.sum(np.absolute(data))
  return result
```

```
def E(data):
  temp=[]
  for i in data:
    temp.append(i*i)
  return np.sum(temp)


def get_features_1(filtered_window, fs=1000):
  cn=filtered_window.shape[1]
  result=[]
  for i in range(cn):
    temp=filtered_window[:,i]
    ll=LL(temp)
    result.append(ll)
  a=np.divide((result-np.mean(result)),np.std(result))
  return a

def get_features_2(filtered_window, fs=1000):
  cn=filtered_window.shape[1]
  result=[]
  for i in range(cn):
    temp=filtered_window[:,i]
    e=E(temp)
    result.append([e])
  return result
```

```
#funtion to create R matrix
def create_R_matrix(features, N_wind):
  #append the copy of first N-1 raw
  features = np.append(features,features[0:N_wind-1,:],axis=0)
  M = len(features)
  nfeats = np.shape(features)[1]
  samples = int(NumWins(features[:,0],1,N_wind,1))
  R = np.ones([samples,N_wind*nfeats+1])
  R[:,0] = np.ones(np.shape(R[:,0]))
  for i in range(samples):
    for j in range(nfeats):
      R[i,1+j*N_wind:1+j*N_wind+N_wind] = features[i:i+N_wind,j]
  return R

#function to create f weight matrix
def f_matrix(R_matrix,dg_data):
  Y_matrix = downsample_data(dg_data)
  result=np.matmul(inv(np.matmul(R_matrix.transpose(),R_matrix)),np.matmul(R_matrix
  return result
```

```
#getting feats for each window
def NumWins(x, fs, winLen, winDisp):
  return (len(x) - winLen*fs + winDisp*fs) // (winDisp * fs)

def get_windowed_feats_1(raw_ecog, fs=1000, window_length=100, window_overlap=50):
  numWins = NumWins(raw_ecog,1000,0.1,0.05)
  i = 0
  pos = 0
  curr_winFeat = []
  result = []
  while(i<numWins):
    curr_winFeat = get_features_1(raw_ecog[pos:(pos+window_length)])
    result.append(curr_winFeat)
    pos +=window_overlap
    i += 1
  result = np.array(result)
  return result

def get_windowed_feats_2(raw_ecog, fs=1000, window_length=100, window_overlap=50):
  numWins = NumWins(raw_ecog,1000,0.1,0.05)
  i = 0
  pos = 0
  curr_winFeat = []
  result = []
  while(i<numWins):
    curr_winFeat = get_features_2(raw_ecog[pos:(pos+window_length)])
    result.append(curr_winFeat)
    pos +=window_overlap
    i += 1
  result = np.array(result)
  return result
```

```
def downsample_data(raw_ecog, fs=1000, window_length=100, window_overlap=50):
  numWins = NumWins(raw_ecog,1000,0.1,0.05)
  i = 0
  pos = 0
  curr_winFeat = []
  result = []
  while(i<numWins):
    curr_winFeat = get_ave(raw_ecog[pos:(pos+window_length)])
    result.append(curr_winFeat)
    pos +=window_overlap
    i += 1
  result = np.array(result)
  return result

def upsample_data(data,sample,up_sample):
  from scipy.interpolate import CubicSpline
  x=np.linspace(0,up_sample-100,sample)
  cs = CubicSpline(x, data)
  xx=np.arange(up_sample)
  arr_upsampled=np.array(cs(xx))
  return arr_upsampled
```