BE 521: Final Project Part 1

Spring 2022

Adapted by Kevin Xie

32 Points

Objective: Predict finger movements from ECoG Recordings

Due: March 31st

Project Overview

This final project involves predicting finger flexion using intracranial EEG (ECoG) in three human subjects. The data and problem framing come from the 4th BCI Competition (Miller et al. 2008). For the details of the problem, experimental protocol, data, and evaluation, please see the original 4th BCI Competition documentation (included as separate document). The remainder of the current document details your deliverables for part 1 of the project.

Important Deadlines

- Final Project Part 1
  - Due: March 31st
  - 32 Points
- Team Registration
  - Due: March 31st
  - 5 Points
- Checkpoint 1
  - Due: April 14th
  - 20 Points
- Checkpoint 2
  - Due: April 21st
  - 15 Points
- End of competition, submit algorithm (Canvas):
  - Due: April 25th
  - 15 Points
- Final Report
  - Due: April 28th
  - 60 Points
- Competition results (Final class session)
  - On: April 27th

The grading is structured so that going the extra mile is definitely rewarded. We want you to show what you've learned this semester, and to have some fun!

Writing Your Code

To get started with the final project we have provided a a series of method stubs for you to fill out. Your job for part 1 of the final project is to build a prediction pipeline that takes in the ECoG and dataglove finger angle recordings (serving as the data and labels respectively), then uses machine learning methods to generate predicted finger angles from the ECoG signals. The functions you will develop in this assignment are as follows:

- `get_windowed_feats` This function will take in raw ECoG data, and use the 2 following helper functions to filter the data, calculate sliding-window features.
  - `filter_data` This function will apply a filter to the raw data and return cleaned data
  - `get_features` This function will take in a window of cleaned data and return a vector of features for that window
- `create_R_matrix` This function will take in a feature matrix and return a response matrix as an adaptation of the optimal linear decoder method.

Optimal Linear Decoder

You will use the **optimal linear decoder** method as described in Warland et al., 1997. We will recapitulate the method in this section, but consult the paper for more details. Our ultimate goal is to predict the angle of each finger as it moves over time using data recorded from the ECoG channels.

The position data is captured for 300 seconds, which you will split up into *M* total time bins, and the number of ECoG channels, *v*, is 61, 46, and 64 for subject 1, 2, and 3 respectively.

The paradigm we adapt here tries to predict finger angle at a given time window using ECoG features calculated over the preceding *N* time windows, using the following steps:

First, features will be calculated across all *v* ECoG channels $\times$ *M* total time windows.

Then, following the approach that Warland et al., 1997 takes, we will construct a row vector corresponding to each time bin, that contains features for all the ECoG channels over the preceding *N* time bins (in the paper, spike counts are their features and they index neurons instead of ECoG channels). Thus, there will be a good amount of redundancy between row vectors of adjacent time bins, but that is okay.

Let r_i^j be the value of the feature channel j in time bin i. Let the response matrix **R** be defined as (copied from Warland et al. 1997):

$$\mathbf{R} = \begin{bmatrix} 1 & r_0^1 & r_1^1 & \cdots & r_{N-1}^1 & \cdots & r_0^\nu & \cdots & r_{N-1}^\nu \\ 1 & r_1^1 & r_2^1 & \cdots & r_N^1 & \cdots & r_1^\nu & \cdots & r_N^\nu \\ 1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & r_{M-1}^1 & r_M^1 & \cdots & r_{N+M-2}^1 & \cdots & r_{M-1}^\nu & \cdots & r_{N+M-2}^\nu \end{bmatrix}$$

This is also referred to as the design or feature matrix, with each column being a predictor, or feature. The column of 1's accounts for the intercept term in linear regression/decoding. Make sure you understand what this matrix means before moving on.

We denote the target matrix (e.g. the $\mathbf{M} \times 5$ matrix of finger angles) as the $\mathbf{Y}$ and the reconstruction (e.g. the predicted finger angles) as $\hat{\mathbf{Y}}$. Note that in Warland et al., 1997, this quantity is referred to as the stimulus vector since they are talking about decoding the stimulus from neural data after it. We, on the other hand, are trying to decode finger positions using the ECoG data before it, but we can conveniently use the same method.

Solving the minimum least-squares difference between the stimulus and reconstruction, $(\mathbf{Y} - \hat{\mathbf{Y}})^{\mathrm{T}}(\mathbf{Y} - \hat{\mathbf{Y}})$, we get the analytic form for the optimal filter,

$$\mathrm{f} = (\mathbf{R}^{\mathrm{T}}\mathbf{R})^{-1}(\mathbf{R}^{\mathrm{T}}\mathbf{Y})$$

This equation should take a familiar form. Warland et al., 1997 don't refer to it as such, but this is exactly the same as linear regression, one of the most commonly used algorithms in practical machine learning. Not only is this algorithm remarkably powerful, but it has a beautiful analytic form for learning the "weights" (here, the f matrix), a rarity in a field where almost all optimizations involve some sort of iterative algorithm. After learning the filter weights f, we can calculate the optimal predictions as:

$$\hat{\mathbf{Y}} = \mathbf{R}\mathrm{f}$$

Dataset

The dataset for part 1 is stored within `final_proj_part1_data.pkl`. The `.pkl` file type is a pickle file, which stores python objects. You can open the `.pkl` file with this code.

```
with open('final_proj_part1_data.pkl', 'rb') as f:
  proj_data = pickle.load(f)
```

This stores the data inside the file as a variable named proj_data.

NOTE: Python versions don't pickle with each other very well. This pickle file was made in Google Colab, running Python 3.7.12. If you are running your own installation of Python and cannot load the file, we recommend you either use Colab, or make a new environment with Python 3.7.12.

There are 3 subjects, each with their own Data Glove data (the glove they used to capture hand movements), and ECoG data. The data is represented as a dictionary with keys `'data_glove'` and `'ecog'`, storing the data glove and ecog data, respectively. These keys map to python lists of 3 items. Each item is an np.ndarray corresponding to a subject's data. See the pseudcode below.

```
proj_data = {
  'data_glove':[np.ndarray for subject 1, np.ndarray for subject 2, np.ndarray for subject 3],
  'ecog':[np.ndarray for subject 1, np.ndarray for subject 2, np.ndarray for subject 3]
}
```

All np.ndarray shapes for `'data_glove'` should be (m x 5), where m is the number of samples in the signal, and 5 is the number of fingers.

The np.ndarray shapes for `'ecog'` are (m x 61), (m x 46), and (m x 64), where m is the number of samples in the signal, and each subject had 61, 46, and 64 ecog channels, respectively.

The sampling rate of the data glove and ecog was 1000 Hz

Your task is to develop an algorithm to use the ECoG to predict finger movements that are captured by the Data Glove.

1. Getting Started (4 pts)

The following sections will walk you through the development of the prediction pipeline.

```
#Set up the notebook environment
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 butter, lfilter, filtfilt, ellip, resample
```

1.

Extract the dataglove and ECoG data for each subject from the pickle file. Feel free to copy the code snippet above. Split the data into a training and testing set (at least 50% of the data should be in the training set).

How many samples are there in the full ECoG recording (before splitting)? (1 pt)

How many samples do you have in your training set? In your testing set? (1 pt)

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
with open('drive/MyDrive/BE521_BCI/FinalProj/final_proj_part1_data.pkl', 'rb') as f:
  proj_data = pickle.load(f)
print("Number of samples in full ECoG recording:", np.size(proj_data['ecog'][0][:,0]))
```

```
Number of samples in full ECoG recording: 300000
```

```
keylist = list(proj_data.keys())
train_data = {'data_glove':[], 'ecog':[]}
```

```
test_data = {'data_glove':[], 'ecog':[]}
for key in keylist:
  for i in range(3):
    m,n = np.shape(proj_data[key][i])
    train_data[key].append(proj_data[key][i][:int(m/3*2),:])
    test_data[key].append(proj_data[key][i][int(m/3*2):,:])

print("Number of samples in training set:", np.size(train_data['ecog'][0][:,0]))
print("Number of samples in testing set:", np.size(test_data['ecog'][0][:,0]))
```

```
Number of samples in training set: 200000
Number of samples in testing set: 100000
```

2.

Next, complete the `filter_data` function. Test it using the raw data extracted in the prior step. What filter types and cutoff frequencies did you use? (2 pts)

```
def freq_domain(X, doplot = False):
  N = len(X)
  n = np.arange(N)
  T = N/200
  freq = n/T

  # freq = np.fft.fftfreq(N, 1/200)

  sd = np.fft.fft(X)
  if doplot:
    plt.plot(freq, np.abs(sd))
    # plt.xlim([0, 500])
  return [freq, np.abs(sd)]
```

```
def filter_data(raw_eeg, fs=1000):
  """
  Write a filter function to clean underlying data.
  Filter type and parameters are up to you. Points will be awarded for reasonable filter type, parameters and application.
  Please note there are many acceptable answers, but make sure you aren't throwing out crucial data or adversly
  distorting the underlying data!

  Input:
    raw_eeg (samples x channels): the raw signal
    fs: the sampling rate (1000 for this dataset)
  Output:
    clean_data (samples x channels): the filtered signal
  """

  m,n = np.shape(raw_eeg)

  # Band-pass filter:
  lowcut = 0.5
  highcut = 200

  b, a = butter(N = 5, Wn = [lowcut, highcut], fs = fs, btype='bandpass')
  y1 = np.stack(filtfilt(b,a, raw_eeg[:,i]) for i in range(n))
  clean_data = y1.T

  # Bandstop filter:
  bsfreq = np.array([[59, 61], [119, 121], [179, 181]])
  for k in range(np.shape(bsfreq)[0]):
    b1, a1 = butter(N = 4, Wn = bsfreq[k,:], btype = 'bandstop', fs = fs)
    y2 = np.stack(filtfilt(b1,a1, clean_data[:,i]) for i in range(n))
    clean_data = y2.T

  return clean_data
```

```
# test function
y = filter_data(proj_data['ecog'][0]) # proj_data_filtered
plt.figure(figsize=(15,6))
plt.subplot(121)
freq_domain(proj_data['ecog'][0][:,0], True)
```

```
plt.title('Freq-domain raw data')
plt.xlabel('Freq')

plt.subplot(122)
freq_domain(y[:,0], True)
plt.title('Freq_domain filtered data')
plt.xlabel('Freq')
```

```
train_fil = {'data_glove':[], 'ecog':[]}
test_fil = {'data_glove':[], 'ecog':[]}
for key in keylist:
  for i in range(3):
    train_fil[key].append(filter_data(train_data[key][i]))
    test_fil[key].append(filter_data(test_data[key][i]))
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: FutureWarning: arrays to stack must be passed as a
  exec(code_obj, self.user_global_ns, self.user_ns)
```

```
proj_fil = {'data_glove':[], 'ecog':[]}
for key in keylist:
  for i in range(3):
    proj_fil[key].append(filter_data(proj_data[key][i]))
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: FutureWarning: arrays to stack must be passed as a
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Refs:

https://mne.tools/dev/auto_tutorials/clinical/30_ecog.html

https://mne.tools/stable/auto_tutorials/preprocessing/30_filtering_resampling.html#tut-filter-resample

https://www.fieldtriptoolbox.org/tutorial/human_ecog/#preprocessing-of-the-neural-recordings

https://sccn.ucsd.edu/wiki/Makoto's_preprocessing_pipeline

2. Calculating Features (12 points)

Here you will complete the `get_windowed_feats` and `get_features` functions.

1.

We will calculate features across sliding time windows. if we use a suggested window length of 100ms with a 50ms window overlap, how many feature windows, M, will we have if we computed features using all the data in a given subject? Feel free to re-use code from previous homeworks.

```
def NumWins(x, fs, winLen, winDisp):
  xLen,_ = np.shape(x)
  NumWindows = int((xLen - winLen*fs)/(winDisp*fs))+1 # updated from "round((xLen - winLen*fs)/(winDisp*fs))"
  return NumWindows
```

```
#Your code here
winlen = 100e-3 # ms
win_overlap = 50e-3
windisp = winlen-win_overlap # ms

M1 = NumWins(proj_data['ecog'][0], 1000, winlen, windisp)
print("Number of feature windows M is:", M1)
```

```
Number of feature windows M is: 5999
```

2.

Now complete the `get_features` function. Please create **4 or more** different features to calculate for each channel in each time window. Features may include the average time-domain voltage, or the average frequency-domain magnitude in consecutive 15Hz frequency bands, bandpower of relevant frequency bands, etc...

```
def bandpower(data, sf, band, window_sec=None, relative=False):
    """Compute the average power of the signal x in a specific frequency band.
    Ref: https://raphaelvallat.com/bandpower.html
    Parameters
    ----------
    data : 1d-array
        Input signal in the time-domain.
    sf : float
        Sampling frequency of the data.
    band : list
        Lower and upper frequencies of the band of interest.
    window_sec : float
        Length of each window in seconds.
        If None, window_sec = (1 / min(band)) * 2
    relative : boolean
        If True, return the relative power (= divided by the total power of the signal).
        If False (default), return the absolute power.

    Return
    ------
    bp : float
        Absolute or relative band power.
    """
    from scipy.signal import welch
    from scipy.integrate import simps
    band = np.asarray(band)
    low, high = band

    # Define window length
    if window_sec is not None:
        nperseg = window_sec * sf
    else:
        nperseg = (2 / low) * sf

    # Compute the modified periodogram (Welch)
    freqs, psd = welch(data, sf, nperseg=nperseg)

    # Frequency resolution
    freq_res = freqs[1] - freqs[0]

    # Find closest indices of band in frequency vector
    idx_band = np.logical_and(freqs >= low, freqs <= high)
```

```
43
44    # Integral approximation of the spectrum using Simpson's rule.
45    bp = simps(psd[idx_band], dx=freq_res)
46
47    if relative:
48        bp /= simps(psd, dx=freq_res)
49    return bp
```

```
 1 def get_features(filtered_window, fs=1000):
 2   """
 3     Write a function that calculates features for a given filtered window.
 4     Feel free to use features you have seen before in this class, features that
 5     have been used in the literature, or design your own!
 6
 7     Input:
 8       filtered_window (window_samples x channels): the window of the filtered ecog signal
 9       fs: sampling rate
10     Output:
11       features (channels x num_features): the features calculated on each channel for the window
12   """
13   m,n = np.shape(filtered_window)
14   # Average time-domain voltage
15   MeanVol = np.stack(np.mean(filtered_window[:,i]) for i in range(n))
16
17   # Average freq-domain magnitude in several freq bands (kubanek et al, 2009)
18   freqbands = np.array([[10,25], [75,115], [125, 159], [160, 175]]) *m/200;
19   MeanAmp0 = np.stack( np.mean(freq_domain(filtered_window[:,i])[1][int(freqbands[0,0]):int(freqbands[0,1])] )
20       for i in range(n))
21
22   MeanAmp1 = np.stack( np.mean(freq_domain(filtered_window[:,i])[1][int(freqbands[1,0]):int(freqbands[1,1])] )
23       for i in range(n))
24
25   MeanAmp2 = np.stack( np.mean(freq_domain(filtered_window[:,i])[1][int(freqbands[2,0]):int(freqbands[2,1])] )
26       for i in range(n))
27
28   MeanAmp3 = np.stack( np.mean(freq_domain(filtered_window[:,i])[1][int(freqbands[3,0]):int(freqbands[3,1])] )
29       for i in range(n))
30
31   # Bandpower of freq band [0-20 Hz] (warland et al, 1997)
32   BP = np.stack( bandpower(filtered_window[:,i], 1000, [0.5, 20]) for i in range(n))
33
34   return  np.array([MeanVol, MeanAmp0, MeanAmp1, MeanAmp2, MeanAmp3, BP]).T
```

```
1 # test function
2 winsample = y[:int(winlen*1000), :]
3 featsample = get_features(winsample)
4 feat_reshape = featsample.T.reshape(-1)
5
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: FutureWarning: arrays to stack must be passed as a
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.7/dist-packages/scipy/signal/spectral.py:1966: UserWarning: nperseg = 4000 is greater than input length
  .format(nperseg, input_length))
```

3.

Now finish the `get_windowed_feats` function by putting the `filter_data` and `get_features` functions together to return a feature vector for each time window.

```
1 def get_windowed_feats(raw_ecog, fs, window_length, window_overlap):
2   """
3     Write a function which processes data through the steps of filtering and
4     feature calculation and returns features. Points will be awarded for completing
5     each step appropriately (note that if one of the functions you call within this script
6     returns a bad output, you won't be double penalized). Note that you will need
7     to run the filter_data and get_features functions within this function.
8
9     Inputs:
```

```
    raw_eeg (samples x channels): the raw signal
    fs: the sampling rate (1000 for this dataset)
    window_length: the window's length
    window_overlap: the window's overlap
  Output:
    all_feats (num_windows x (channels * features)): the features for each channel for each time window
      note that this is a 2D array.
"""
window_disp = window_length-window_overlap
N = NumWins(raw_ecog, fs = fs, winLen = window_length, winDisp = window_disp)

fil_ecog = filter_data(raw_ecog, fs)

all_feats = np.array([])
# for i in range(N):
#   startwin = int(i*window_disp*fs)
#   endwin = int(startwin + window_length*fs)
#   windata = fil_ecog[startwin:endwin,:]

all_feats = np.stack(get_features(fil_ecog[int(i*window_disp*fs):int(i*window_disp*fs+ window_length*fs),:], fs).T.reshape(-1)
for i in range(N))

return all_feats
```

```
# test function
A = get_windowed_feats(proj_data['ecog'][0], 1000, 100e-3, 50e-3)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning: arrays to stack must be passed as a "sequence" ty

/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: FutureWarning: arrays to stack must be passed as a
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:420: FutureWarning: arrays to stack must be passed as a "sequenc
  arrays = [asanyarray(arr) for arr in arrays]
/usr/local/lib/python3.7/dist-packages/scipy/signal/spectral.py:1966: UserWarning: nperseg = 4000 is greater than input length
  .format(nperseg, input_length))
```

3. Creating the Response Matrix (6 points)

In this section, you will develop code for your *create_R_matrix* function

1.

For our set of 62 channels in subject 1, what would the dimensions of the R matrix be if we calculated 6 di erent feature types per channel, and N = 3 time bins where the number of total time bins M is the number you calculated in 2.1? (1pt)

The dimension of R matrix would be (M x (N * v * 6 + 1). In this case, the dimension of R matrix with N =3, M = 5999, and v = 62 is (5998 x 1117)

2.

We do not have feature data to fill out the first N-1 data rows in the R matrix that will be used to predict the first N-1 finger angles. One way to work around this is to append a copy of the first N-1 rows of your feature matrix to the beginning of your feature matrix before calculating R. Make this adjustment in `create_R_matrix`, then compute the response matrix R. You can test whether your function is running correctly by running `create_R_matrix` with data from `testRfunction.pkl` using 3 windows and verifying that the quantity `np.mean(R)` is 25.4668 (5 points).

Double-click (or enter) to edit

```
with open('drive/MyDrive/BE521_BCI/FinalProj/testRfunction.pkl', 'rb') as f1:
  testR = pickle.load(f1)
```

```
def create_R_matrix(features, N_wind):
  """
  Write a function to calculate the R matrix
```

```
4
5   Input:
6     features (samples (number of windows in the signal) x channels x features):
7       the features you calculated using get_windowed_feats
8     N_wind: number of windows to use in the R matrix
9
10  Output:
11    R (samples x (N_wind*channels*features))
12  """
13  M,vf = np.shape(features)
14  features_new = np.append(features[:N_wind-1, :].copy(), features, axis=0)
15
16  R = np.stack((features_new[i:i+N_wind, :]).reshape(-1) for i in range(M))
17  R = np.append(np.ones([M,1]), R, axis = 1)
18
19  return R
```

```
1 R_test = create_R_matrix(testR, 3)
2 np.mean(R_test)
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: FutureWarning: arrays to stack must be passed as a
  exec(code_obj, self.user_global_ns, self.user_ns)
25.46678519830894
```

4. ML Training and Testing (10 points)

Here we will use the optimal linear decoder framework to predict finger angles, and additionally you will use one or more classifiers of your own choosing to make the prediction.

1.

Calculate the linear filter $\mathbf{f}$ as defined above for all 5 finger angles using features calculated from your training data. You will have to first down-sample the finger flexion data so that your feature matrix, $\mathbf{R}$, and your flexion data have the same number of time windows.

You will likely find np.matmul, and np.linalg.inv to be useful.

```
1 cor=[]
2 fs = 1000
3 winlen =  100e-3
4 win_overlap = 50e-3
5 for k in range(3): #3
6   feats_train = get_windowed_feats(train_fil['ecog'][k], fs, winlen, win_overlap)
7   feats_test = get_windowed_feats(test_fil['ecog'][k], fs, winlen, win_overlap)
8   #downsampling
9   M_train,_ = np.shape(feats_train)
10  M_test,_ = np.shape(feats_test)
11  Y_train = resample(train_fil['data_glove'][k], M_train)
12  Y_test = resample(test_fil['data_glove'][k], M_test)
13  # R matrices
14  R_train = create_R_matrix(feats_train, 3)
15  R_test = create_R_matrix(feats_test, 3)
16
17  f = np.matmul(np.linalg.inv(np.matmul(R_train.T, R_train)), np.matmul(R_train.T, Y_train))
18  Y_hat = np.matmul(R_test, f)
19
20  cor.append([(pearsonr(Y_hat[:,i], Y_test[:,i]))[0] for i in range(5)])
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: FutureWarning: arrays to stack must be passed as a "sequence" ty
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: FutureWarning: arrays to stack must be passed as a
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:420: FutureWarning: arrays to stack must be passed as a "sequenc
  arrays = [asanyarray(arr) for arr in arrays]
/usr/local/lib/python3.7/dist-packages/scipy/signal/spectral.py:1966: UserWarning: nperseg = 4000 is greater than input length
  .format(nperseg, input_length))
```

2.

Try one other machine learning classifier using your features and finger angle labels. Look back through previous homeworks to get some ideas

```
# PCA
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

# pca = PCA(n_components = 2)

svm_cor = []

for k in range(3): #3
  feats_train = get_windowed_feats(train_fil['ecog'][k], fs, winlen, win_overlap)
  feats_test = get_windowed_feats(test_fil['ecog'][k], fs, winlen, win_overlap)
  # data_pca = pca.fit_transform(feats_train)
  #downsampling
  M_train,_ = np.shape(feats_train)
  M_test,_ = np.shape(feats_test)
  Y_train = resample(train_fil['data_glove'][k], M_train)
  Y_test = resample(test_fil['data_glove'][k], M_test)
  #lableling
  labels_train = np.stack(np.argmax(np.abs(Y_train[i,:])) for i in range(M_train))
  labels_test = np.stack(np.argmax(np.abs(Y_test[i,:])) for i in range(M_test))
  # ML
  svm_train = make_pipeline(StandardScaler(), SVC(kernel = 'rbf')).fit(feats_train, labels_train)

  predict = svm_train.predict(feats_test)
  svm_cor.append((pearsonr(predict, labels_test))[0])

```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: FutureWarning: arrays to stack must be passed as a "sequence" t
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: FutureWarning: arrays to stack must be passed as a
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:420: FutureWarning: arrays to stack must be passed as a "sequenc
  arrays = [asanyarray(arr) for arr in arrays]
/usr/local/lib/python3.7/dist-packages/scipy/signal/spectral.py:1966: UserWarning: nperseg = 4000 is greater than input length
  .format(nperseg, input_length))
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2822: FutureWarning: arrays to stack must be passed as a
  if self.run_code(code, result):
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: FutureWarning: arrays to stack must be passed as a "sequence" t
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: FutureWarning: arrays to stack must be passed as a
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:420: FutureWarning: arrays to stack must be passed as a "sequenc
  arrays = [asanyarray(arr) for arr in arrays]
/usr/local/lib/python3.7/dist-packages/scipy/signal/spectral.py:1966: UserWarning: nperseg = 4000 is greater than input length
  .format(nperseg, input_length))
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2822: FutureWarning: arrays to stack must be passed as a
  if self.run_code(code, result):
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: FutureWarning: arrays to stack must be passed as a "sequence" t
  if sys.path[0] == '':
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: FutureWarning: arrays to stack must be passed as a
  exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.7/dist-packages/numpy/core/shape_base.py:420: FutureWarning: arrays to stack must be passed as a "sequenc
  arrays = [asanyarray(arr) for arr in arrays]
/usr/local/lib/python3.7/dist-packages/scipy/signal/spectral.py:1966: UserWarning: nperseg = 4000 is greater than input length
  .format(nperseg, input_length))
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2822: FutureWarning: arrays to stack must be passed as a
  if self.run_code(code, result):
```

```
#downsampling
svm_cor = []
pca = PCA(n_components = 300)
data_pca = pca.fit_transform(feats_train)
test_pca = pca.fit_transform(feats_test)
M_train,_ = np.shape(feats_train)
M_test,_ = np.shape(feats_test)
```

```
 8 Y_train = resample(train_fil['data_glove'][k], M_train)
 9 Y_test = resample(test_fil['data_glove'][k], M_test)
10 #labeling
11 labels_train = np.stack(np.argmax(np.abs(Y_train[i,:])) for i in range(M_train))
12 labels_test = np.stack(np.argmax(np.abs(Y_test[i,:])) for i in range(M_test))
13 # ML
14 svm_train = make_pipeline(StandardScaler(), SVC(kernel = 'rbf')).fit(feats_train, labels_train)
15 predict = svm_train.predict(feats_test)
16 svm_cor.append((pearsonr(predict, labels_test)))
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2822: FutureWarning: arrays to stack must be passed as a
  if self.run_code(code, result):
```

3.

Produce predictions on the testing set for each finger angle. Calculate the correlation coefficient between the predicted and test finger angles for each finger separately. Report you correlations here using the lineare filter, and when using the other classifier(s) that you tried.

You will find pearsonr to be helpful and already imported.

```
1 # Your code here
2 for k in range(3):
3   print('From 4.1, Correlation coefficient for 5 fingers of subject', k,':\n', cor[k] )
```

```
Correlation coefficient for 5 fingers of subject 0 :
 [0.1797790506034379, 0.106779431618751, 0.12247615365746448, 0.24395678461744172, 0.1743332084699997]
Correlation coefficient for 5 fingers of subject 1 :
 [0.1737090381222766, 0.2314693153540183, 0.16189015927254433, 0.2997046789911961, 0.2122137326461213]
Correlation coefficient for 5 fingers of subject 2 :
 [0.2893059796651726, 0.3476620493045314, 0.415482072887796, 0.48365000905135197, 0.3148291103286852]
```

```
1 for k in range(3):
2   print('From 4.2, Correlation coefficient for 5 fingers of subject', k,':\n', svm_cor[k] )
```

```
From 4.2, Correlation coefficient for 5 fingers of subject 0 :
 0.24503338398754224
From 4.2, Correlation coefficient for 5 fingers of subject 1 :
 0.14698012155746862
From 4.2, Correlation coefficient for 5 fingers of subject 2 :
 0.13542274811646135
```

Collaborators: Rohan Vemu, Tshepo Yane