Purdue University, West Lafayette

Fall 2021
BME 51100 – Biomedical Signal Processing

Mid-Term project Report
Submission Date- 10/28/2021

Submitted by
Nahian Ibn Hasan
PUID: 0032764564
Email: hasan34@purdue.edu

PURDUE
UNIVERSITY®

Introduction:

Brain-computer interfaces (BCIs) are an exciting possibility of modern-day Biomedical Engineering. They have a broad range of potential applications from augmenting human sensorimotor action in both medical (e.g., prostheses) and non-medical contexts (e.g., virtual-reality games), to neurofeedback for training and therapy (e.g., management of chronic pain, ADHD, rehabilitation after stroke). Electroencephalography (EEG) provides a non-invasive window to measure brain responses to various kinds of stimuli and how they interact with behaviors. In this project we explored a classic BCI paradigm that uses EEG -- a P300 based on-screen menu selection. Classically, the P300 is a positive response peak that occurs in the EEG when an environmental stimulus happens to match a target expectation. It has been suggested that the P300 could be exploited to device a mental prosthetic that can be used for spelling when the patient is otherwise paralyzed (e.g., see Kübler et al., 2009).

In this project, we will use the data collected by Ulrich Hoffmann's research group at EPFL accompanying their 2008 publication in the Journal of Neuroscience Methods (Hoffmann et al., 2008) [1]. Here, subjects were asked to concentrate on one of six different pictures on screen as they are periodically flashed one at a time and count the number of times the target flashed. The idea is that this simulates a scenario where a patient is trying to mentally select one of the six possible menu items on screen. When the target image (i.e., the one that the subject is concentrating on) flashes, the EEG data exhibits a P300 response (see Hoffmann et al., 2008 for details). The engineering problem is then one of decoding which of the six menu items the subject intended to select, by using just the EEG data. The faster the target is decoded from the EEG, the more quickly the action corresponding to the selected menu item can be executed. Further, the fewer the number of EEG channels required to decode the target, the interface becomes simpler and less intrusive. Also, if the features that differentiate the target menu item from the other items is consistent across multiple sessions for a given subjects, or is consistent across different individuals, then the BCI can be optimized for the "average" person without having to train the BCI for each individual or for each session.

Declaration:

This project is a part of the curriculum of BME51100 (Biomedical Signal Processing) at Purdue University, West lafayette. All of the codes presented here are not collected from any third party source. The analysis provided here are based on the instructions provided by the instructor. And this report serves as a project report. Some of the conclusions mentioned might be changed after further analysis. The corresponding codes can be found at [4].

Instructor:

Dr. Hari Bharadwaj
Assistant Professor of Speech, Language and Hearing Sciences and Biomedical Engineering
Purdue University
Department of Speech, Language and Hearing Sciences

Project Completed By:

Nahian Ibn Hasan
Graduate Student
Purdue University, West Lafayette
October 28, 2021

Part A

Section 1

Sub-section 1.1

Codes:

```
import os,glob
import scipy.io
import numpy as np
import datetime
def unpackStamp(x):
	y = np.int32(x[0])
	mo = np.int32(x[1])
	d = np.int32(x[2])
	h = np.int32(x[3])
	mi = np.int32(x[4])
	s = x[5]
	s_new = np.int32(np.floor(s))
	micros = np.int32((s - s_new) * 1e6)
	unpacked = datetime.datetime(y, mo, d, h, mi, s_new, micros)
	return unpacked
def events2samps(events, fs):
	firsteve_time = 0.4
	Nevents = events.shape[0]
	evesamps = np.zeros(Nevents)
	for k in range(Nevents):
		td = unpackStamp(events[k, :]) - unpackStamp(events[0, :])
		evesamps[k] = np.int32(np.round(td.total_seconds()*fs + firsteve_time*fs + 1))
	return evesamps
def data_read(data_directory,subject,sessions,fs):
	Events = {}
	for sess in sessions:
		EEG_data_files = os.listdir(data_directory+'/'+subject+'/'+sess)
		Events[sess] = {}
		for eeg in EEG_data_files:
			EEG_data_path = data_directory+'/'+subject+'/'+sess+'/'+eeg
			EEG_data = scipy.io.loadmat(EEG_data_path)
			#print(EEG_data.keys())
			target = int(EEG_data['target'].squeeze())
			stimuli = EEG_data['stimuli'].squeeze()
			targets_counted = int(EEG_data['targets_counted'].squeeze())
			data = EEG_data['data'].squeeze()
			events = EEG_data['events'].squeeze()
			events2samples = events2samps(events,fs)
			sub_name = eeg.split('.')[0]
			Events[sess][sub_name] =
{'target':target,'targets_counted':targets_counted,'stimuli':stimuli,'data':data,'events':events2samples}
	return Events
```

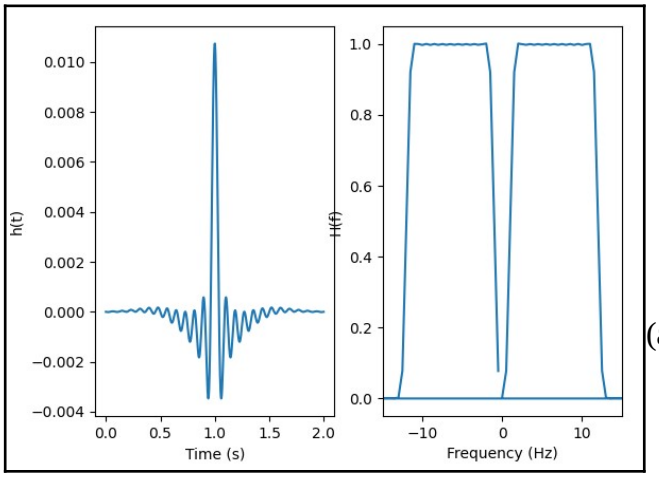
Sub-section 1.2

Codes:

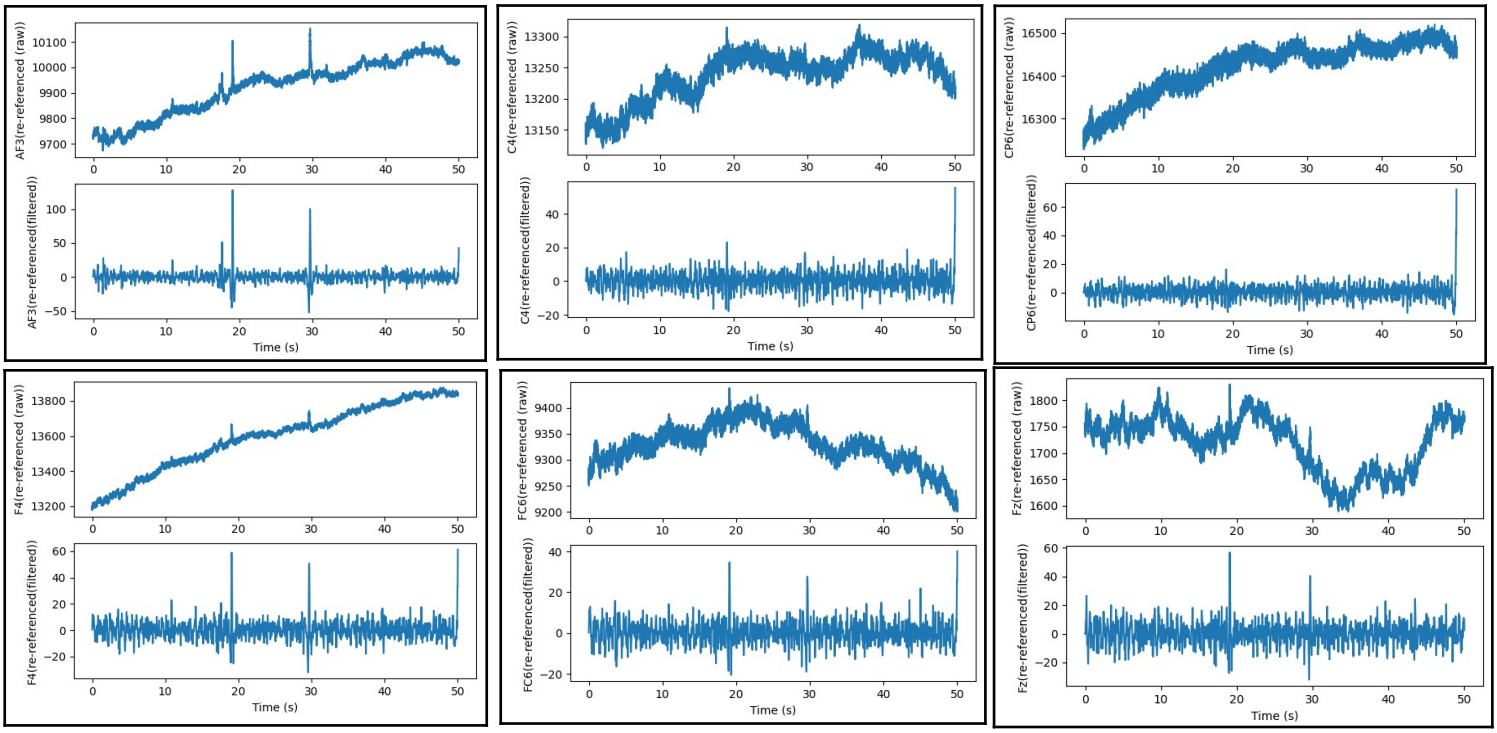
```
def filter_design(f_high, f_low,transition,fs):
        N = int(np.ceil(1./transition * fs)) #filter length
        h = signal.firwin(N, [f_high, f_low], pass_zero=False, fs=fs)
        t_h = np.arange(0, N) / fs
        plt.subplot(121)
        plt.plot(t_h, h)
        plt.xlabel('Time (s)')
        plt.ylabel('h(t)')
        H=fft(h)
        f = np.fft.fftfreq(len(h),d=1/fs)  # Frequency axis in Hz
        plt.subplot(122)
        plt.plot(f,np.abs(H))
        plt.xlabel('Frequency (Hz)')
        plt.ylabel('H(f)')
        plt.xlim([-15, 15])
        #plt.show()
        plt.savefig('bandpass_filter.png')
        return h,H
def filtered_events(event_data,fs,h,save_path,eeg_channels,verbose=0):
        filtered_data = np.zeros(event_data.shape)
        for i in range(event_data.shape[0]):
                eeg_filt_firwin = signal.filtfilt(h, 1, event_data[i,:],axis=0)
                if verbose:
                        if not os.path.exists(save_path):
                                os.makedirs(save_path)
                        figure = plt.figure()
                        t = np.arange(0,event_data.shape[1]/fs,1./fs)
                        ax1 = plt.subplot(211)
                        ax1.plot(t[0:102400], event_data[i,0:102400])#Plot the first 50s data
                        plt.ylabel(eeg_channels[i]+'(re-referenced (raw))')
                        ax2 = plt.subplot(212, sharex=ax1)
                        ax2.plot(t[0:102400], eeg_filt_firwin[0:102400])#Plot the first 50s data
                        plt.xlabel('Time (s)')
                        plt.ylabel(eeg_channels[i]+'(re-referenced(filtered))')
                        #plt.show()
                        plt.savefig(save_path+'/'+'filterred_signal_'+eeg_channels[i]+'.png')
                        plt.close()
                filtered_data[i,:] = eeg_filt_firwin
        #print(filtered_data.shape)
        return filtered_data
```

Outputs:

The filter_design() function designs the filter and returns the band-pass filter in time-domain and frequency-domain format. On the other hand, filtered_events() applies the filter on the re-referenced data and plots the results of the first 50s equivalent of data.



(a)



(b)

Figure 1: (a) FIR filter design (1-12Hz) with a sharp transition (0.5Hz), (b)Example EEG channel data after re-referencing with respect to the mastoid channels and band-pass-filtering in the range of [1-12Hz].

Comments:

Filtering an EEG signal removes the artifacts due to external sources, electrical machines, subject movement, eye movement, or head movement. The signal generated due to these stimuli might not be of interest for any researcher/ practitioner. These outliers also hinders the true nature of the response of the subject.

Sub-section 1.3, 1.4, 1.5

Codes:

```
def extract_epochs(events,data,epoch_length,eeg_channels,fs):
        epoch_results = np.zeros((len(events),data.shape[0],int(epoch_length)))
        events = events.astype('int')
        for ev in range(len(events)):
                for i in range(data.shape[0]):
                        epoch_results[ev,i,:] = data[i,events[ev]:events[ev]+epoch_length]
        return epoch_results
def extract_target_non_target(Epoch_data,Events,eeg_channels,epoch_length,baseline_DC_range,threshold):
        target_epochs = {}
        non_target_epochs = {}
        for key in Events.keys():
                target = Events[key]['target']
                stimuli = Events[key]['stimuli']
                events = Events[key]['events']
                stimu_count = Counter(stimuli.tolist())
                target_epochs[key] = {}
                non_target_epochs[key] = {}
                for ch in range(len(eeg_channels)):
                        target_epochs[key][eeg_channels[ch]] = np.zeros((stimu_count[target],epoch_length))
                        non_target_epochs[key][eeg_channels[ch]] = np.zeros((len(events)-
stimu_count[target],epoch_length))
                        targ = 0
                        ntarg = 0
                        for i in range(len(events)):
                                epoch = Epoch_data[key][i,ch,:]-np.average(Epoch_data[key]
[i,ch,0:baseline_DC_range]) # after removing baseline noise
                                if stimuli[i] == target:
                                        if not (max(np.absolute(epoch)) > threshold):
                                                target_epochs[key][eeg_channels[ch]][targ,:] = epoch
                                        targ += 1
                                else:
                                        if not (max(np.absolute(epoch)) > threshold):
                                                non_target_epochs[key][eeg_channels[ch]][ntarg,:] = epoch
                                        ntarg += 1
        return target_epochs,non_target_epochs
```

Outputs:

The extract_epochs() function extracts epochs based on the event information, and each of the extracted data are of 1000ms. On the other hand, extract_target_non_target() function subtracts the baseline DC average of each event based on the average value of first 100ms of each event. Next, noisy trials, which exceed the 40 microvolt threshold, are rejected. No down-sampling is applied at this stage. Next, the processed data are divided into two data structures – target_epochs and non_target_epochs. These data structures hold the respective epochs for each EEG channel and event, under a certain session. Hence, for each session, this function will provide the target epochs and non-target epochs for all events.

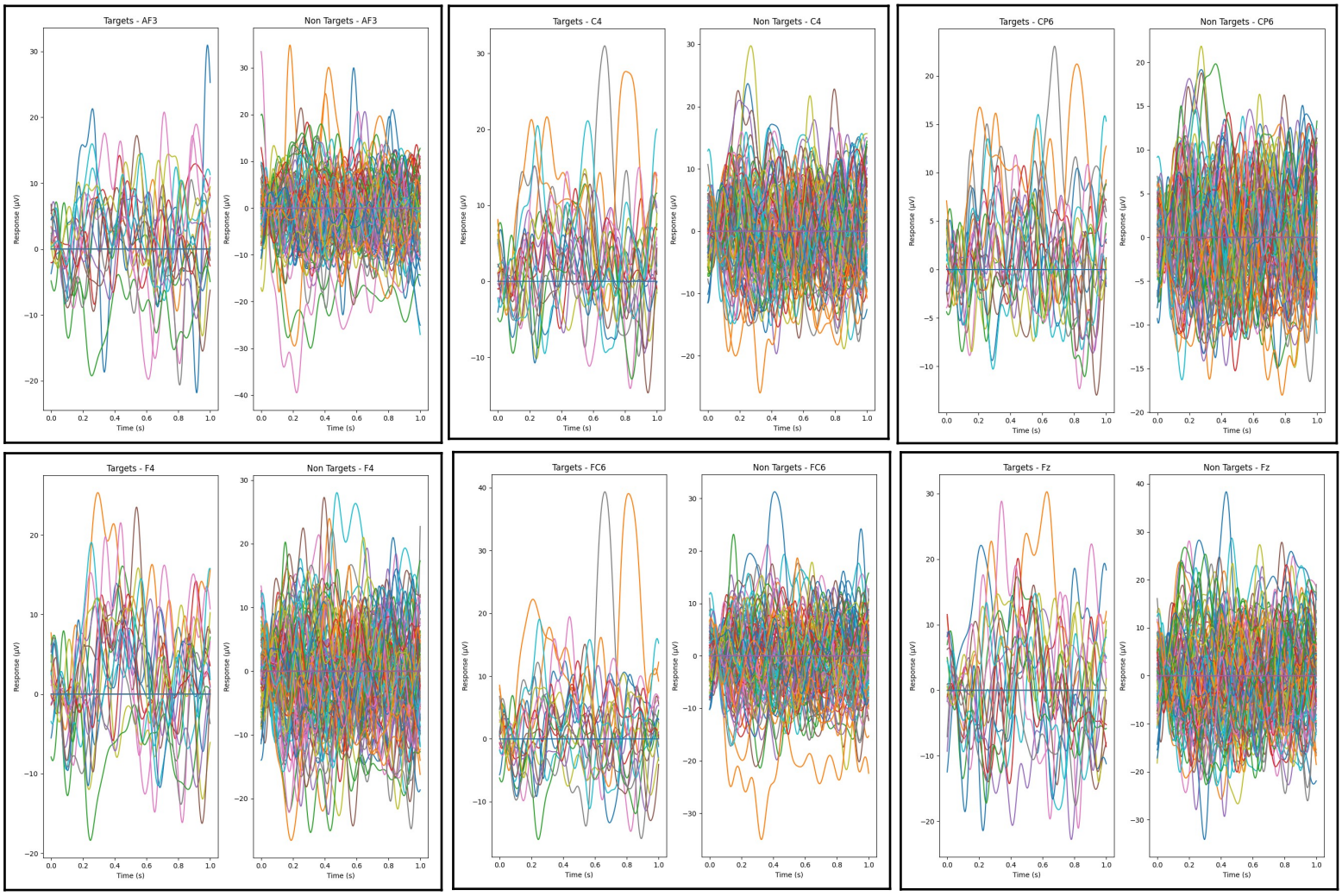


Figure 2: Example target and non-target data under a single session of a single subject after baseline removal and noisy trial rejection.

Comments:

While plotting targets and non-targets does not provide a good overview of the expected target response. But from the plots, it is obvious that the the baseline removal process is in effect. In the next section, the average over all of these responses are plotted, which provide an expected P300 response.

Part A

Section 2

Codes:

```
def plot_session_target_non_target(targets,non_targets,eeg_channels,save_path,subject,sess,epoch_length,fs,
verbose=0):
	time_axis = np.arange(0,epoch_length/fs,1/epoch_length)
	if verbose:
		for key in targets.keys():
			for ch in targets[key].keys():
				if not os.path.exists(save_path+key+'/'):
					os.makedirs(save_path+key+'/')
				#print(targets[key][ch].shape)
				#print(non_targets[key][ch].shape)
				plt.figure(figsize=(10, 10))
				plt.subplot(121)
				plt.plot(time_axis,targets[key][ch].T)
				plt.title('Targets - '+ch)
				plt.ylabel('Response (\u03bcV)')
				plt.xlabel('Time (s)')
				plt.subplot(122)
				plt.plot(time_axis,non_targets[key][ch].T)
				plt.ylabel('Response (\u03bcV)')
				plt.xlabel('Time (s)')
				plt.title('Non Targets - '+ch)
				#plt.show()
				plt.savefig(save_path+key+'/'+ch+'.png')
				plt.close()
	if not os.path.exists(save_path+'/Average_Results'):
		os.makedirs(save_path+'/Average_Results')
	keys = list(targets.keys())
	sum_trg = {}
	sum_ntrg = {}
	num_trg = {}
	num_ntrg = {}
	for ch in eeg_channels:
		avg_targets = np.zeros((targets[keys[0]][ch].shape[1],))
		trg_nums = 0
		for key in keys:
			for num_ev in range(targets[key][ch].shape[0]):
				avg_targets += targets[key][ch][num_ev,:]
				trg_nums += 1
		sum_trg[ch] = avg_targets
		num_trg[ch] = trg_nums
		avg_non_targets = np.zeros((non_targets[keys[0]][ch].shape[1],))
		ntrg_nums = 0
		for key in keys:
			for num_ev in range(non_targets[key][ch].shape[0]):
				avg_non_targets += non_targets[key][ch][num_ev,:]
				ntrg_nums += 1
		sum_ntrg[ch] = avg_non_targets
		num_ntrg[ch] = ntrg_nums
		avg_targets = avg_targets/trg_nums
		avg_non_targets = avg_non_targets/ntrg_nums
		difference = avg_targets - avg_non_targets
```

```
                plt.figure(figsize=(10, 10))
                plt.plot(time_axis,avg_targets,label='Average_Targets ('+str(trg_nums)+' trials)')
                plt.plot(time_axis,avg_non_targets,label='Average_Non_Targets ('+str(ntrg_nums)+' trials)')
                plt.plot(time_axis,difference,label='Differencce_Response')
                plt.title(subject+', '+sess+', channel '+ch)
                plt.ylabel('Average Response (\u03bcV)')
                plt.xlabel('Time (s)')
                plt.legend()
                #plt.show()
                plt.savefig(save_path+'/Average_Results/'+ch+'.png')
                plt.close()
        return sum_trg,sum_ntrg,num_trg,num_ntrg
def plot_subject_target_non_target(per_session_data,eeg_channels,save_path,subject,epoch_length,fs):
        time_axis = np.arange(0,epoch_length/fs,1/epoch_length)
        if not os.path.exists(save_path+'/Average_Results'):
                os.makedirs(save_path+'/Average_Results')
        sessions = list(per_session_data.keys())
        average_target = {}
        average_non_target = {}
        difference = {}
        total_trg = {}
        total_ntrg = {}
        for ch in eeg_channels:
                average_target[ch] = np.zeros(per_session_data[sessions[0]][0][ch].shape)
                average_non_target[ch] = np.zeros(per_session_data[sessions[0]][1][ch].shape)
                total_trg[ch] = 0
                total_ntrg[ch] = 0
                for sess in sessions:
                        average_target[ch] += per_session_data[sess][0][ch]
                        average_non_target[ch] += per_session_data[sess][1][ch]
                        total_trg[ch] += per_session_data[sess][2][ch]
                        total_ntrg[ch] += per_session_data[sess][3][ch]
                average_target[ch] = average_target[ch] / total_trg[ch]
                average_non_target[ch] = average_non_target[ch] / total_ntrg[ch]
                difference[ch] = average_target[ch] - average_non_target[ch]
                plt.figure(figsize=(10, 10))
                plt.plot(time_axis,average_target[ch],label='Average_Targets ('+str(total_trg[ch])+' trials)')
                plt.plot(time_axis,average_non_target[ch],label='Average_Non_Targets ('+str(total_ntrg[ch])+'
trials)')
                plt.plot(time_axis,difference[ch],label='Differencce_Response')
                plt.title(subject+', channel '+ch)
                plt.ylabel('Average Response (\u03bcV)')
                plt.xlabel('Time (s)')
                plt.legend()
                #plt.show()
                plt.savefig(save_path+'/Average_Results/'+ch+'.png')
                plt.close()
        return average_target,average_non_target,difference,total_trg,total_ntrg
```

Outputs:

The plot_session_target_non_target() function finds the average target and non-target response under a single session and plots the result for each channel. It also plots the difference between the average target and non-target responses. On the other hand, plot_subject_target_non_target() function finds the average target response, non-target response and the difference response for all of the sessions under a single subject and plots the result channel-wise.

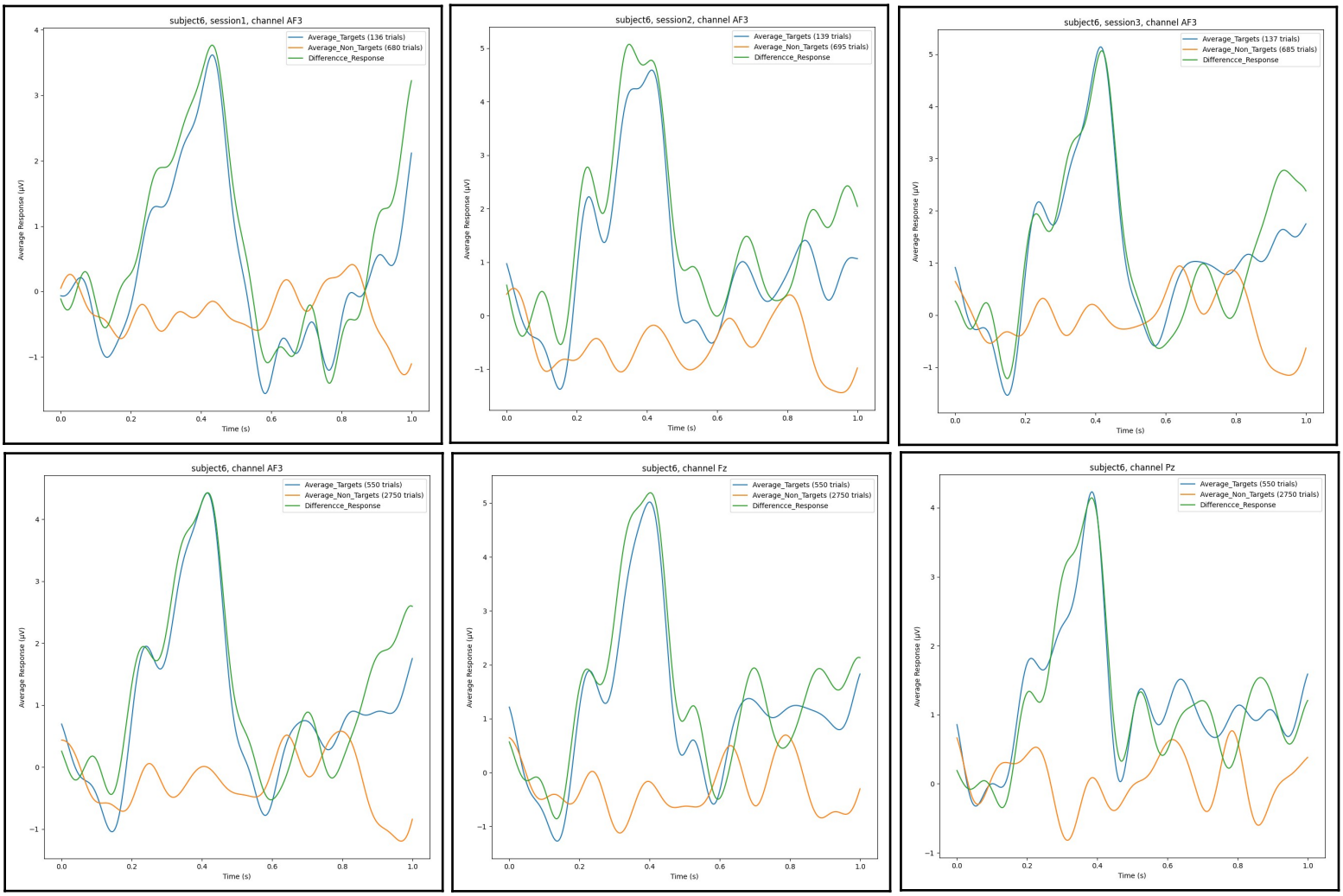


Figure 3: **(Top Row)** Example average target, non-target, and difference response for a single subject and single channel (AF3), but for different session. **(Bottom Row)** Example average target, non-target, and difference response for a single subject over all sessions; the average response is measured by averaging over almost 2750 epochs over all sessions.

Comments:

Each of the epoch is of 1000ms long. Since, inter-stimulus-interval was of 400ms, the last 600ms of each trial are overlapping with the first 600ms of the following trial. For the shown subject (subject 6), it seems the peak response for any target occurs at around 350ms-400ms. It seems almost all of the channels have unique and distinct peak response for this subject (subject 6 - healthy). Figure 3 shows several channels. But some channels (O1,O2,Oz,P8, PO4) provide even multiple peaks at around 300-400ms regime. In Figure 4, the multiple peak- channels are provided.

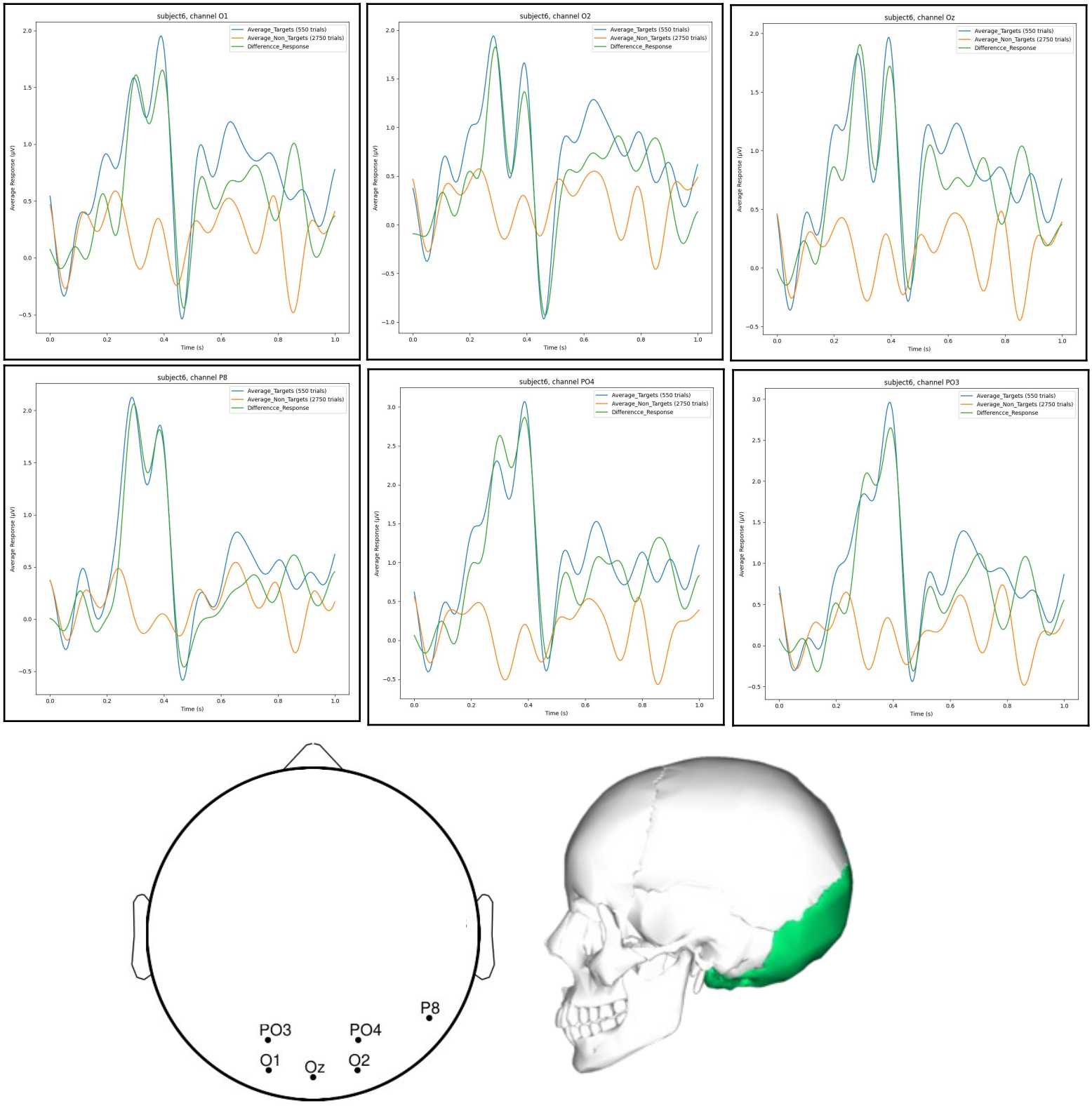


Figure 4: **(Top and Middle Row)** Example average target, non-target, and difference response for a single subject for the channels (O1,O2,Oz,P8,PO3,PO4) with multiple peaks. The bottom pictures shows the channel lead location over the human head.

Based on the data over other healthy subjects (subject 6,7,8,9), it seems, there is a clear peak response near the 300ms-400ms regime in almost all of the EEG channels. It seems the leads in the occipital region of human head, capture the sensitive event transition near the 400th ms. This region is actually consists of nerve cells which are directly related to the vision. This could be the reason, that there are multiple peaks near the event transition near the 400th ms. Figure 5 shows the responses for different channels for unhealthy subjects.

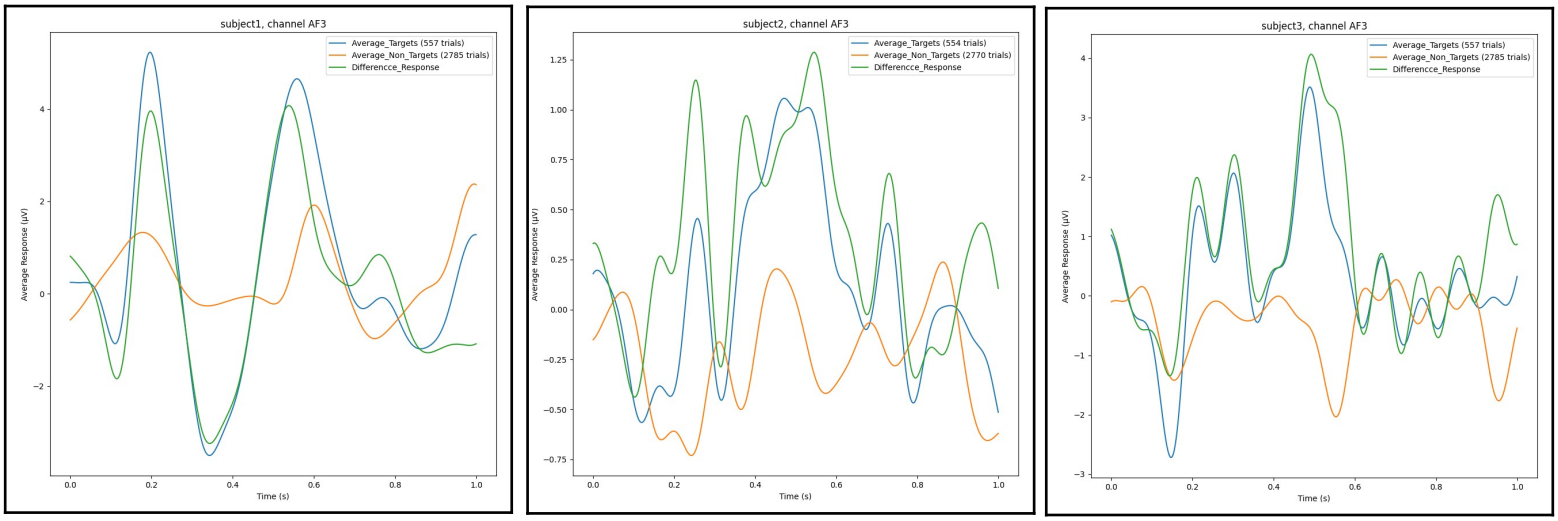


Figure 5: Example average target, non-target, and difference response for unhealthy subjects, where it is hard to find a solid peak response near the 300th ms. Subject 1 (left image) suffers from cerebral pulsy and mild disarthria, subject 2 (middle image) suffers from multiple sclerosis and mild disarthria, and subject 3 (right image) suffers from late-stage amyotrophic lateral sclerosis and severe disarthria. All of the subjects also have weak limb muscle control.

Part A

Section 3

Comparison of peak responses for different subjects

Fig. 6 shows the peak responses for all of the subjects for a single channel Fz. Fig. 7, 8, and 9 shows the similar responses for Cz, Pz, and Oz channels, respectively. It seems for healthy subjects, someone can clearly define a peak response near the 300th ms of the epoch. However, for unhealthy subjects, it is hard to find a peak near the 300th ms. However, someone can find a peak which a either before or past the 300th-400th ms regime. Hence, patients with certain neurological disorders, the P-300 response can be found outside the regime of interest. Also, the peak amplitude for healthy subjects (4uv - 6uv) is way higher than the unhealthy subjects (1uV-3uV). However, for subject 1, (although considered unhealthy), the peak amplitude might not be a distinct factor for classification. Rather the peak-position is important as well.

Based on this result, a mental prosthetic can also be defined for subjects with speech-paralysis and neurological disorders. The inter-subject responses can be analyzed from Fig. 6,7,8 and 9.

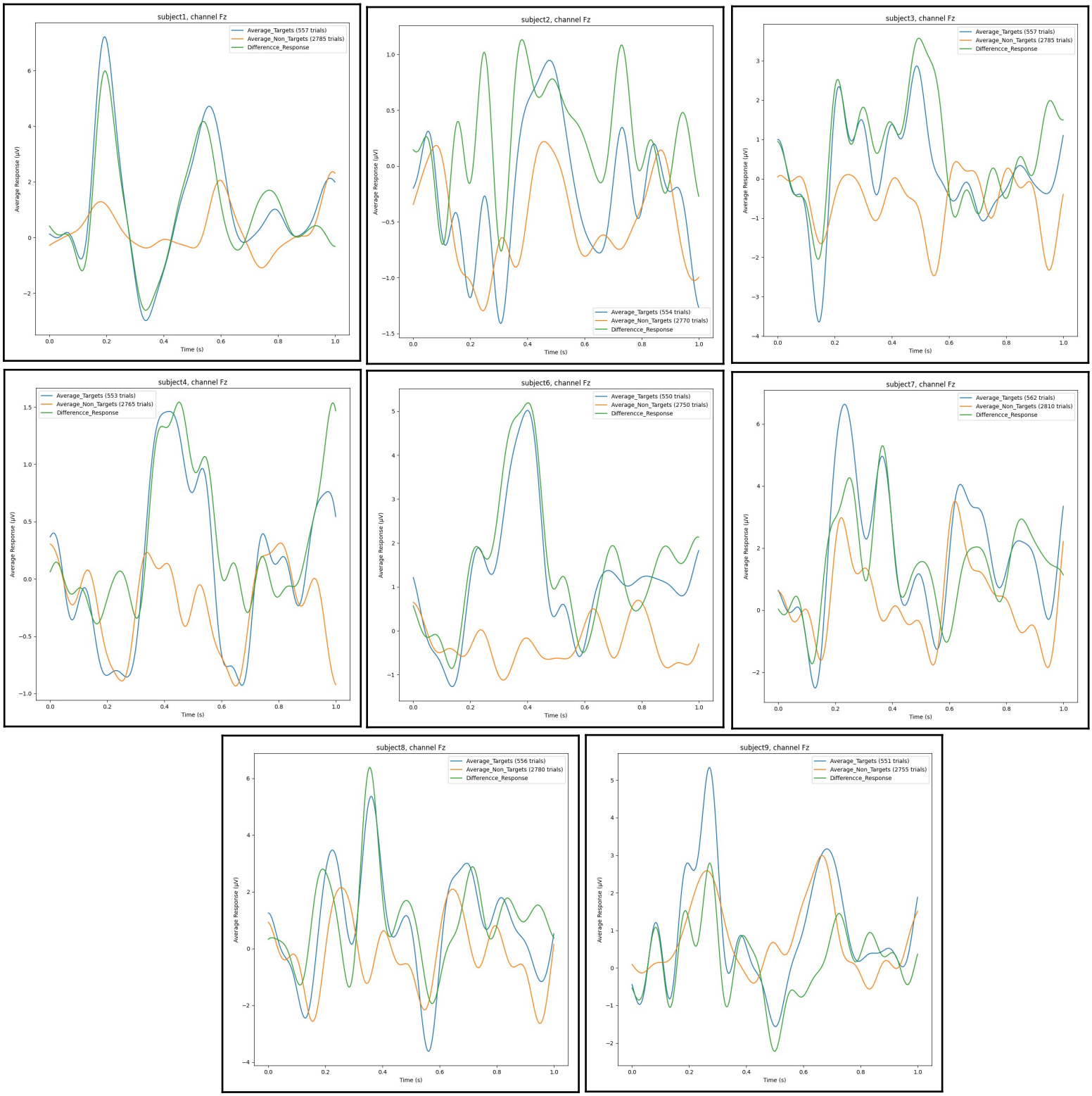


Figure 6: Example average target, non-target, and difference response for unhealthy subjects (subjects1,2,3,4) and healthy subjects (subjects 6,7,8,9) for channel Fz. The image legends provide the information about subject ID, total number of target trials and non-target trials.

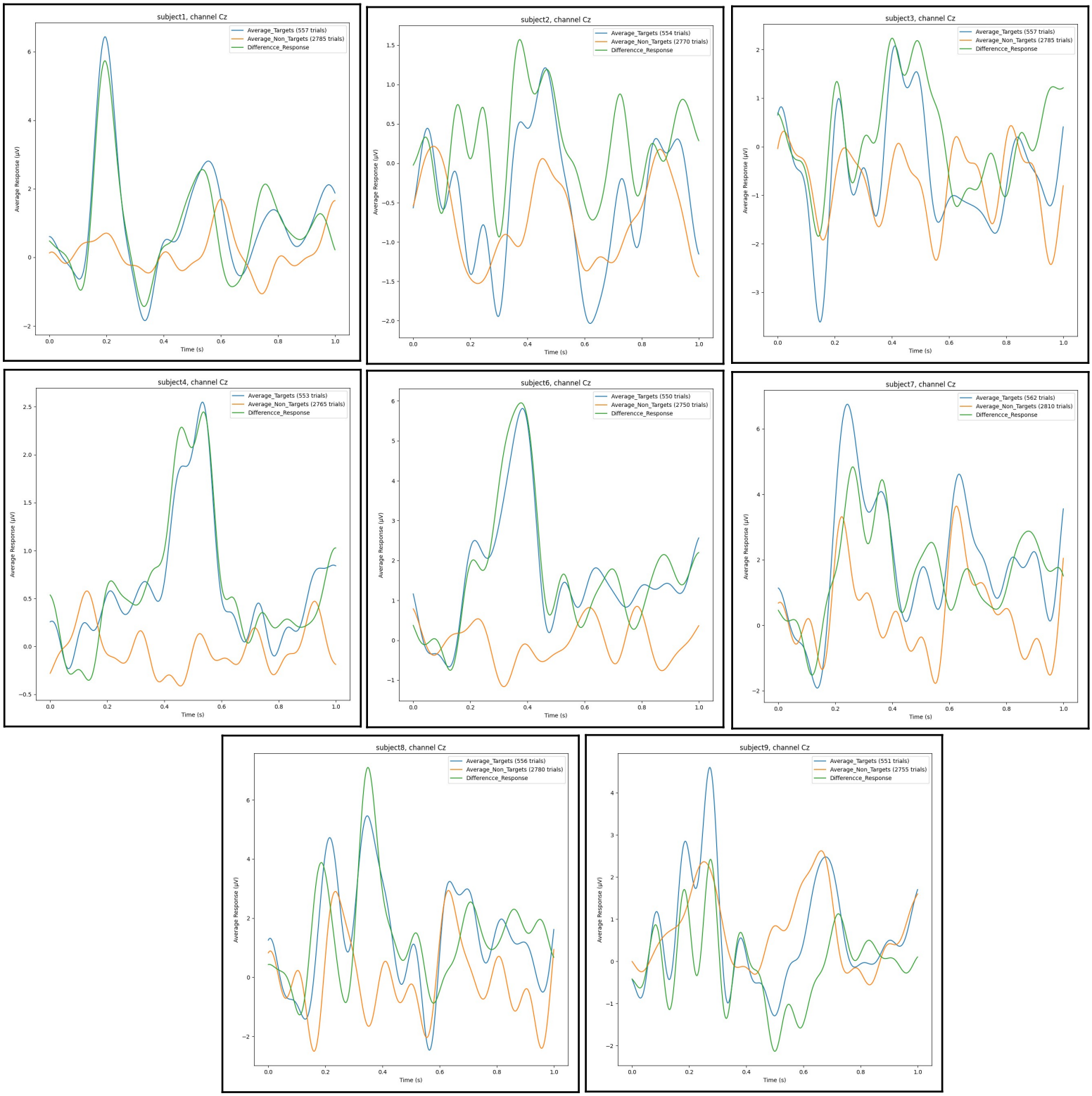


Figure 7: Example average target, non-target, and difference response for unhealthy subjects (subjects1,2,3,4) and healthy subjects (subjects 6,7,8,9) for channel Cz. The image legends provide the information about subject ID, total number of target trials and non-target trials.

Figure 8: Example average target, non-target, and difference response for unhealthy subjects (subjects1,2,3,4) and healthy subjects (subjects 6,7,8,9) for channel Pz. The image legends provide the information about subject ID, total number of target trials and non-target trials.

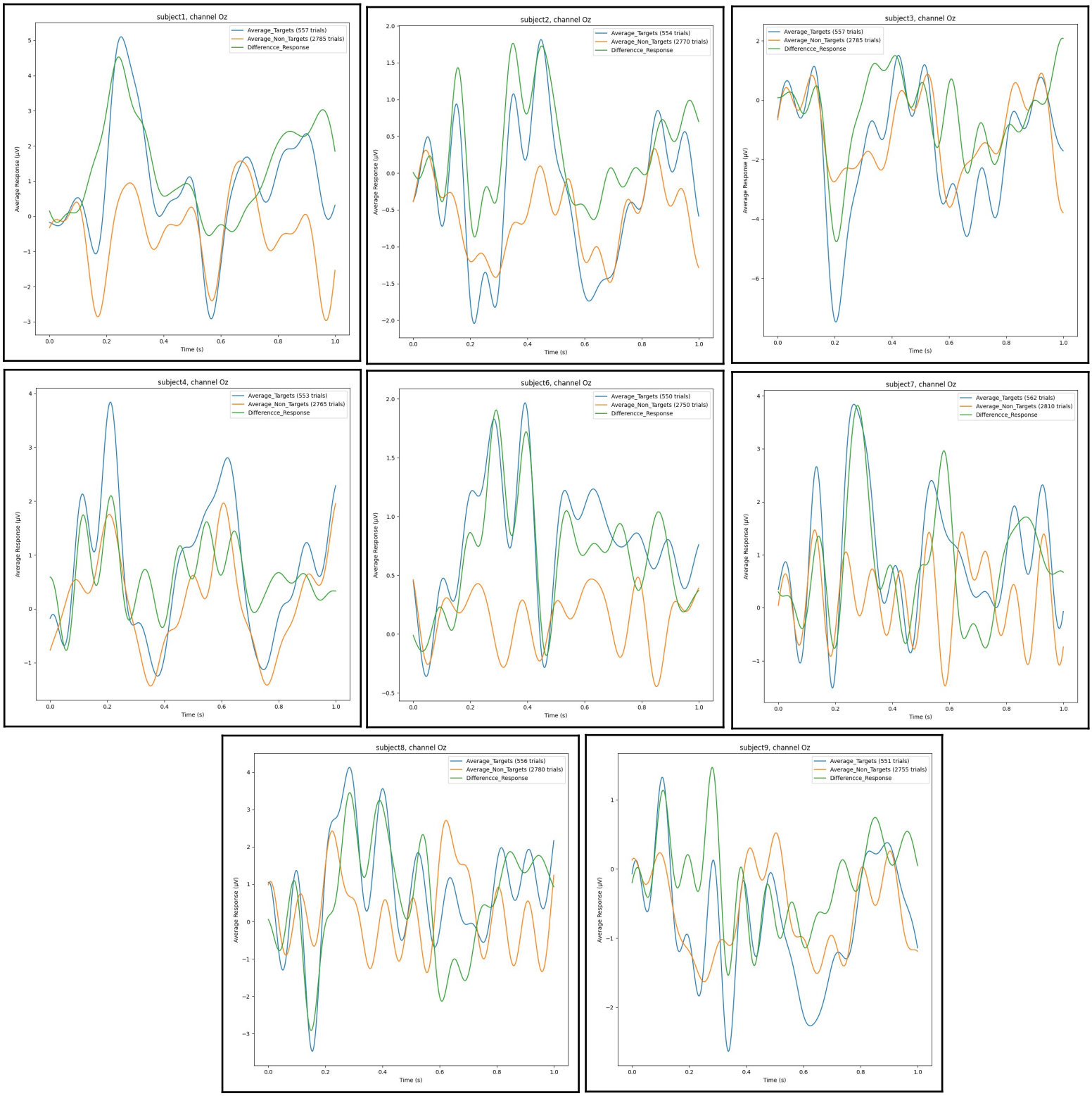


Figure 9: Example average target, non-target, and difference response for unhealthy subjects (subjects1,2,3,4) and healthy subjects (subjects 6,7,8,9) for channel Oz. The image legends provide the information about subject ID, total number of target trials and non-target trials.

Part B

Section 1

Code:

```
def find_all_targets_non_targets(target_epochs,non_target_epochs,eeg_channels):
	all_targets = {}
	all_non_targets = {}
	sessions = list(target_epochs.keys())
	for ch in eeg_channels:
		all_targets[ch] = list()
		all_non_targets[ch] = list()
		for sess in sessions:
			session_targets = target_epochs[sess]
			session_non_targets = non_target_epochs[sess]
			keys = list(session_targets.keys())
			for key in keys:
				for num_ev in range(session_targets[key][ch].shape[0]):
					all_targets[ch].append(session_targets[key][ch][num_ev,:])
			keys = list(session_non_targets.keys())
			for key in keys:
				for num_ev in range(session_non_targets[key][ch].shape[0]):
					all_non_targets[ch].append(session_non_targets[key][ch][num_ev,:])
		all_targets[ch] = np.array(all_targets[ch])
		all_non_targets[ch] = np.array(all_non_targets[ch])
	return all_targets,all_non_targets
def random_null_hypothesis_test(all_targets,all_non_targets,average_epoch_data,eeg_channels,subject,epoch_length,
fs,save_path,data_count_factor=1,offset=0,pool_size=1000):
	P_values = {}
	time_axis = np.arange(0,epoch_length/fs,1/epoch_length)
	if not os.path.exists(save_path+'/Hypothesis_Testing/data_count_factor_'+str(data_count_factor)+'/
offset_'+str(offset)):
		os.makedirs(save_path+'/Hypothesis_Testing/data_count_factor_'+str(data_count_factor)+'/
offset_'+str(offset))
	G = open(save_path+'/Hypothesis_Testing/data_count_factor_'+str(data_count_factor)+'/offset_'+str(offset)
+'/P_values.csv','w')
	G.write('Channel,P_value\n')
	for ch in eeg_channels:
		N = average_epoch_data[4][ch]#number of non-targets
		M = average_epoch_data[3][ch]#number of targets
		M = math.floor(M/data_count_factor)
		N = math.floor(N/data_count_factor)
		'''
		print('M = ',M)
		print('N = ',N)
		print(all_targets[ch].shape)
		print(all_non_targets[ch].shape)
		print('Number of targets = ',all_targets[ch].shape)
		print('Number of non targets = ',all_non_targets[ch].shape)
		'''
		random_differences = list()
		Peak_values = list()
		for p in range(pool_size):
			random_trg_average = np.zeros((1,all_targets[ch].shape[1]))
			random_ntrg_average = np.zeros((1,all_non_targets[ch].shape[1]))
			#Randomly choose M data indexes. Consider the data count factor
```

```
                                R = np.random.choice(M+N,M,replace=False)
                                for i in range(M+N):
                                        if i in R:
                                                if i < M:
                                                        random_trg_average += all_targets[ch][i+(M*offset),:]
                                                else:
                                                        random_trg_average += all_non_targets[ch][i-M+(N*offset),:]
                                        else:
                                                if i < M:
                                                        random_ntrg_average += all_targets[ch][i+(M*offset),:]
                                                else:
                                                        random_ntrg_average += all_non_targets[ch][i-M+(N*offset),:]
                                random_trg_average = random_trg_average / M
                                random_ntrg_average = random_ntrg_average / N
                                diff = (random_trg_average - random_ntrg_average).squeeze()
                                if p < 100:
                                        random_differences.append(diff)
                                Peak_values.append(np.max(diff))
                        actual_peak = max(average_epoch_data[2][ch])
                        P_values[ch] = len([val for val in Peak_values if val>actual_peak])/pool_size
                        G.write(ch+','+str(P_values[ch])+'\n')
                        random_differences = np.array(random_differences)
                        random_differences = np.reshape(random_differences,
[random_differences.shape[0],random_differences.shape[-1]])
                        #Plot the random average response and actual average response
                        plt.figure()
                        plt.plot(time_axis,random_differences[0:100,:].T,color='black')#Just plot the first 100 null examples
                        plt.plot(time_axis,average_epoch_data[2][ch],color='red',linewidth=5)
                        plt.title(subject+', channel '+ch)
                        plt.ylabel('Average Response (\u03bcV)')
                        plt.xlabel('Time (s)')
                        plt.legend(['Random differences','Actual Difference'])
                        #plt.show()
                        plt.savefig(save_path+'/Hypothesis_Testing/data_count_factor_'+str(data_count_factor)+'/
offset_'+str(offset)+'/'+ch+'.png')
                        plt.close()
                #plot the p-values
                plt.figure()
                p_vals = list()
                for ch in eeg_channels:
                        p_vals.append(P_values[ch])
                plt.bar(eeg_channels,p_vals)
                plt.title(subject)
                plt.ylabel('P_values')
                plt.xlabel('Channel')
                plt.xticks(rotation='vertical')
                plt.savefig(save_path+'/Hypothesis_Testing/data_count_factor_'+str(data_count_factor)+'/offset_'+str(offset)
+'/P_values.png')
                plt.close()
                return P_values
```

Outputs:
The find_all_targets_non_targets() functions gathers all targets and non-targets data from different sessions into single data structure for future analysis. On the other hand, random_null_hypothesis_test() function performs a statistical null-hypothesis test on the targets/non-targets data. This function also plots the 100 random null-examples (difference response) along with the actual difference response for all channels. Moreover, the function also plots the same if the data were subdivided into K segments. Also, this function plots and returns P-values for each channel based on the 1000 null examples.

Fig. 10 shows the example null-hypothesis test examples for several channels of a single subject (subject 6).

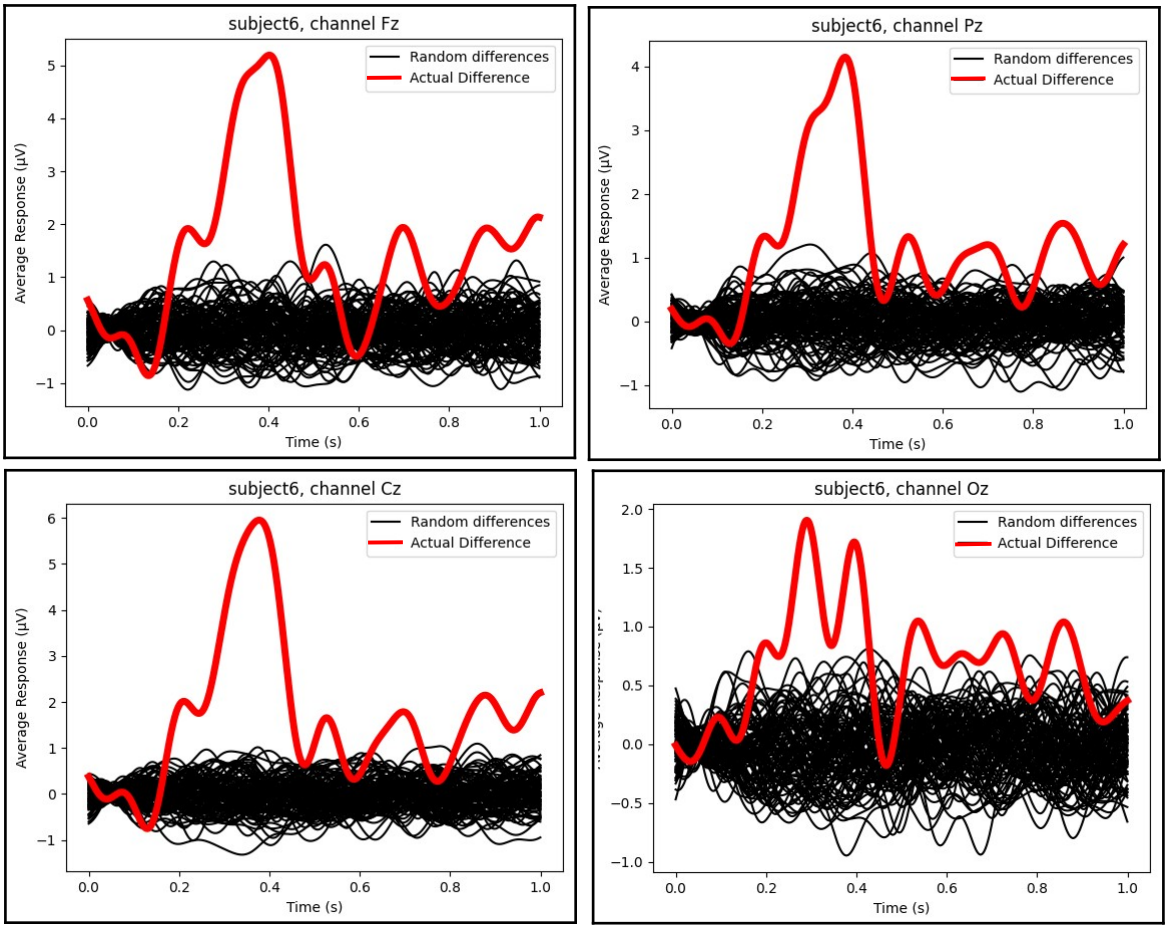


Figure 10: Average Difference response over 100 null-examples and the actual difference response (red plot) for a single subject (subject 6) and multiple channels (Fz, Cz, Pz, Oz).

Comments:
From figure 10, it seems, for this subject, the actual difference response is completely standing out as being different from the null-examples within the time frame of 0.25s -0.35s (roughly). However, to confirm this, the same channel information are provided for two other subjects (subject 2 and 9) in Fig. 11 and 12, respectively. It is evident, that such claim is not quite possible for unhealthy subjects (Fig. 11). But for healthy subjects (subject 6,7,8,9), it is quite valid (Fig. 10 and 12).

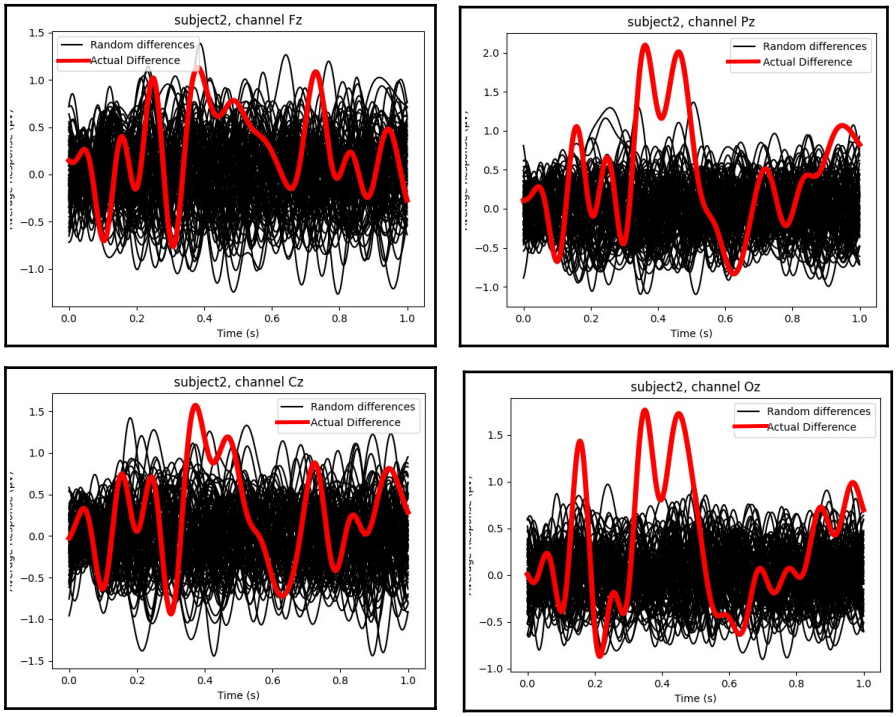


Figure 11: Average Difference response over 100 null-examples and the actual difference response (red plot) for a single subject (subject 2) and multiple channels (Fz, Cz, Pz, Oz).

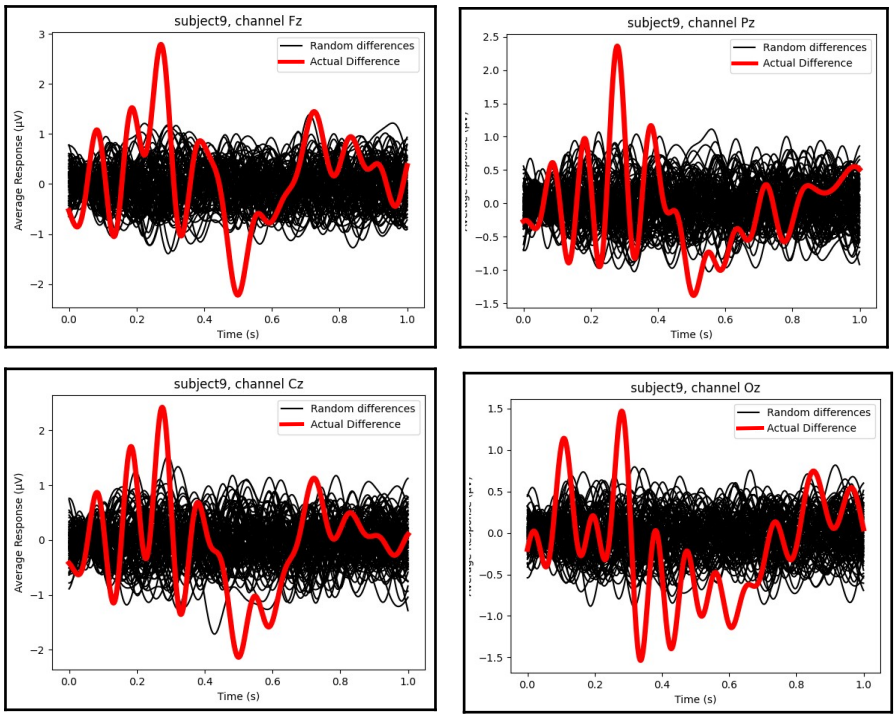


Figure 12: Average Difference response over 100 null-examples and the actual difference response (red plot) for a single subject (subject 9) and multiple channels (Fz, Cz, Pz, Oz).

Part B

Section 2

Comments:

The p-values for each of the channels for all subjects are provided in Fig 13. The P-values are gained over 1000 null examples, when the positive-peak-height is considered as the respective feature look into. Apart from subject 2, and some other few channels, almost all of the channels for other subjects, the statistical test results in extremely low p-values.

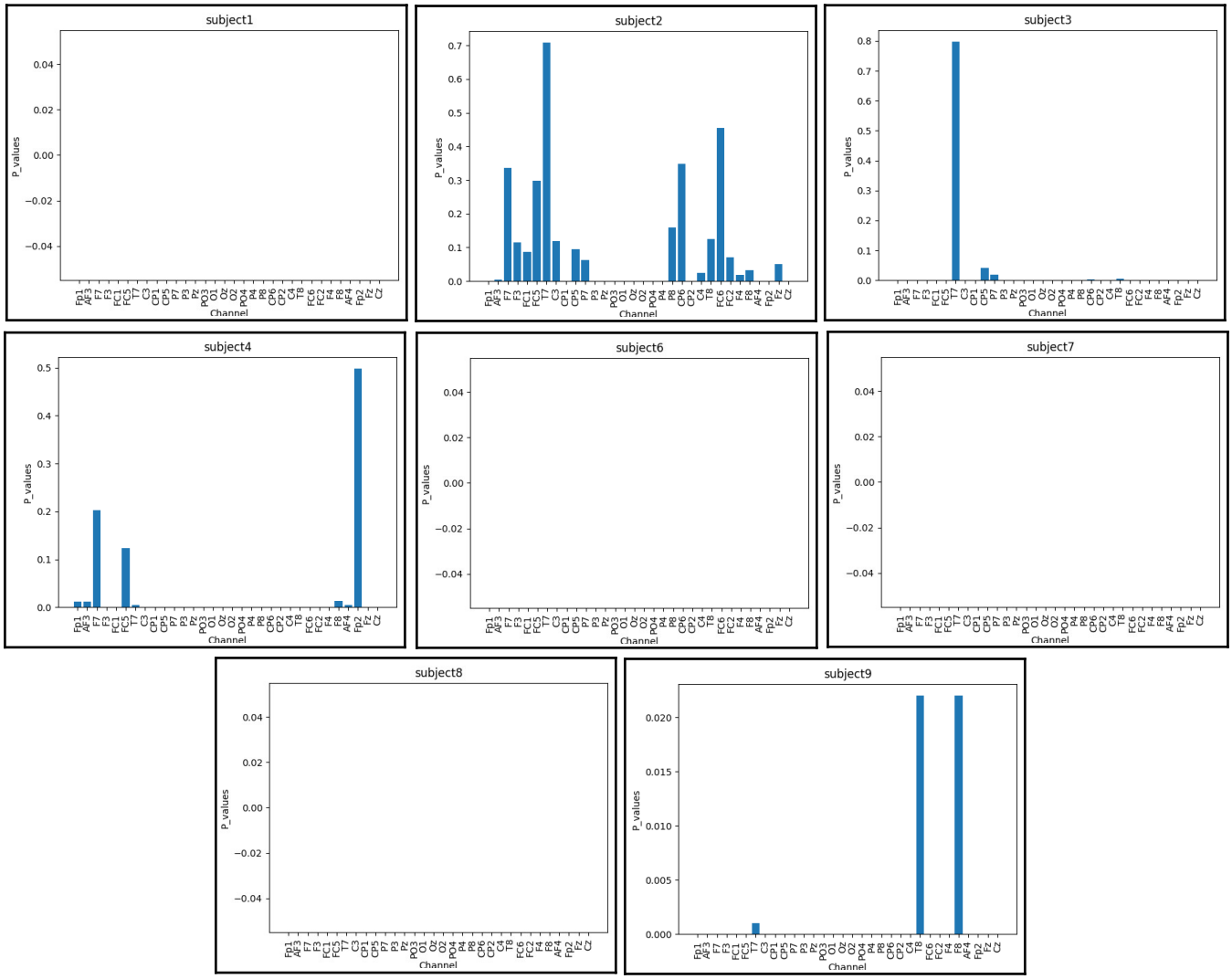


Figure 13: P_values across subjects for different channels over 1000 null-examples.

Part B

Section 3

Comments:

The statistical test is performed with Cz,Pz, and Oz channels as well and results are shown in Fig. 10, 11 and 12. For healthy subjects, the time interval of 250ms-350ms shows a distinct response between actual difference response and null-difference-examples across Cz,Pz, and Oz channel. But for unhealthy subjects, this does not hold. The corresponding p-values are also shown in Fig. 13.

Part C

Section 1

Comments:

Fig. 14 shows the p-values across subjects when there are 4 subdivisions. The null-hypothesis test is done for data worth of one subdivision. Fig. 15 shows the same result with 24 subdivisions, and the null-test is done over the data worth of one subdivision.

From the plots of Fig. 13,14 and 15, it is evident that when the number of subdivisions is increased, the p-values also increase (on an average) across subjects as well. So, when we have less amount of epoch information (target and non-target), the higher p-values means the target and non-target responses cannot be differentiated. Hence, for the null hypothesis to be rejected, we need higher number of target/non-target data for a single subject. That is actually important for target localization and P-300 response localization.

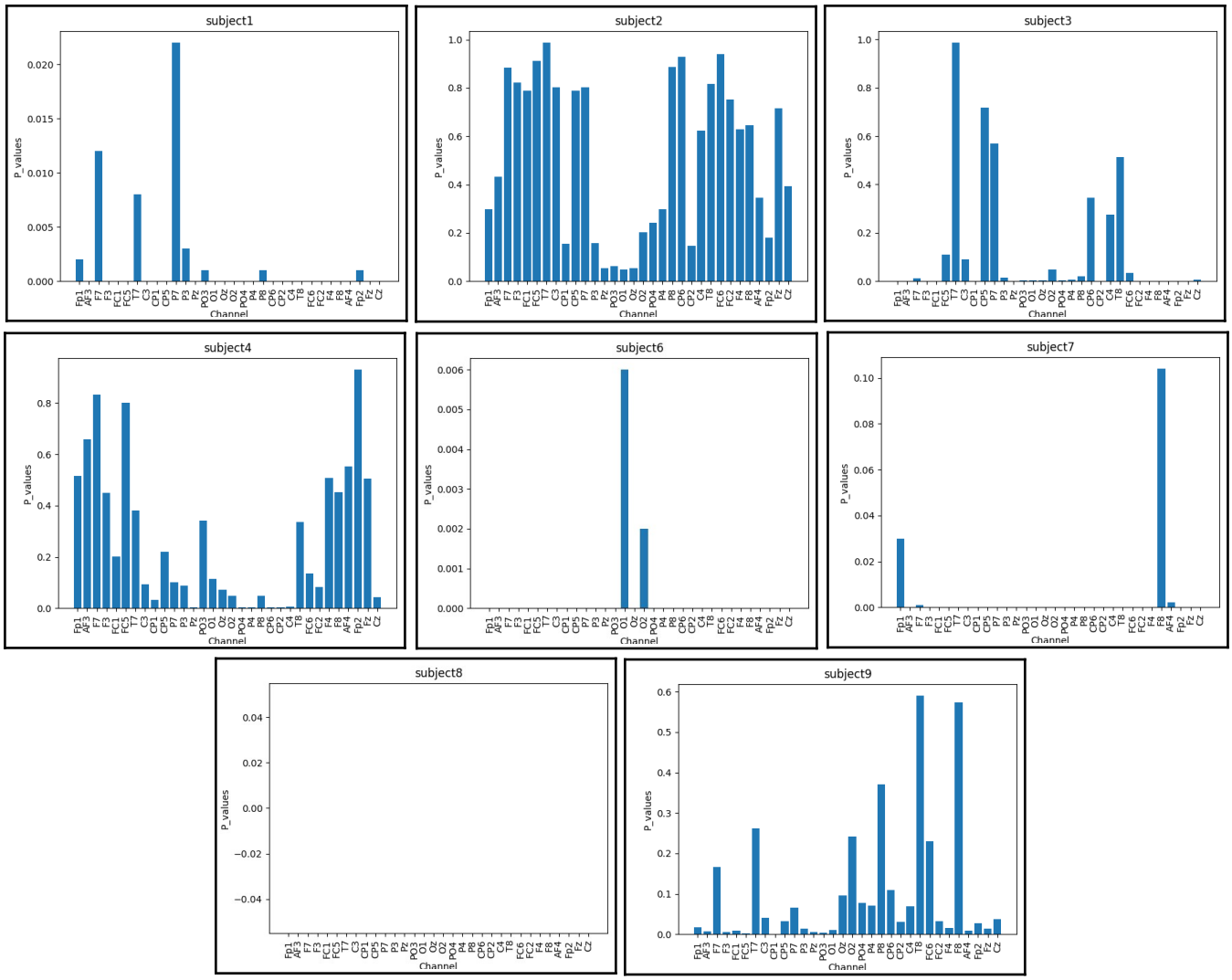


Figure 14: P_values across subjects for different channels over 1000 null-examples when the test is done over ¼ th of the whole data.

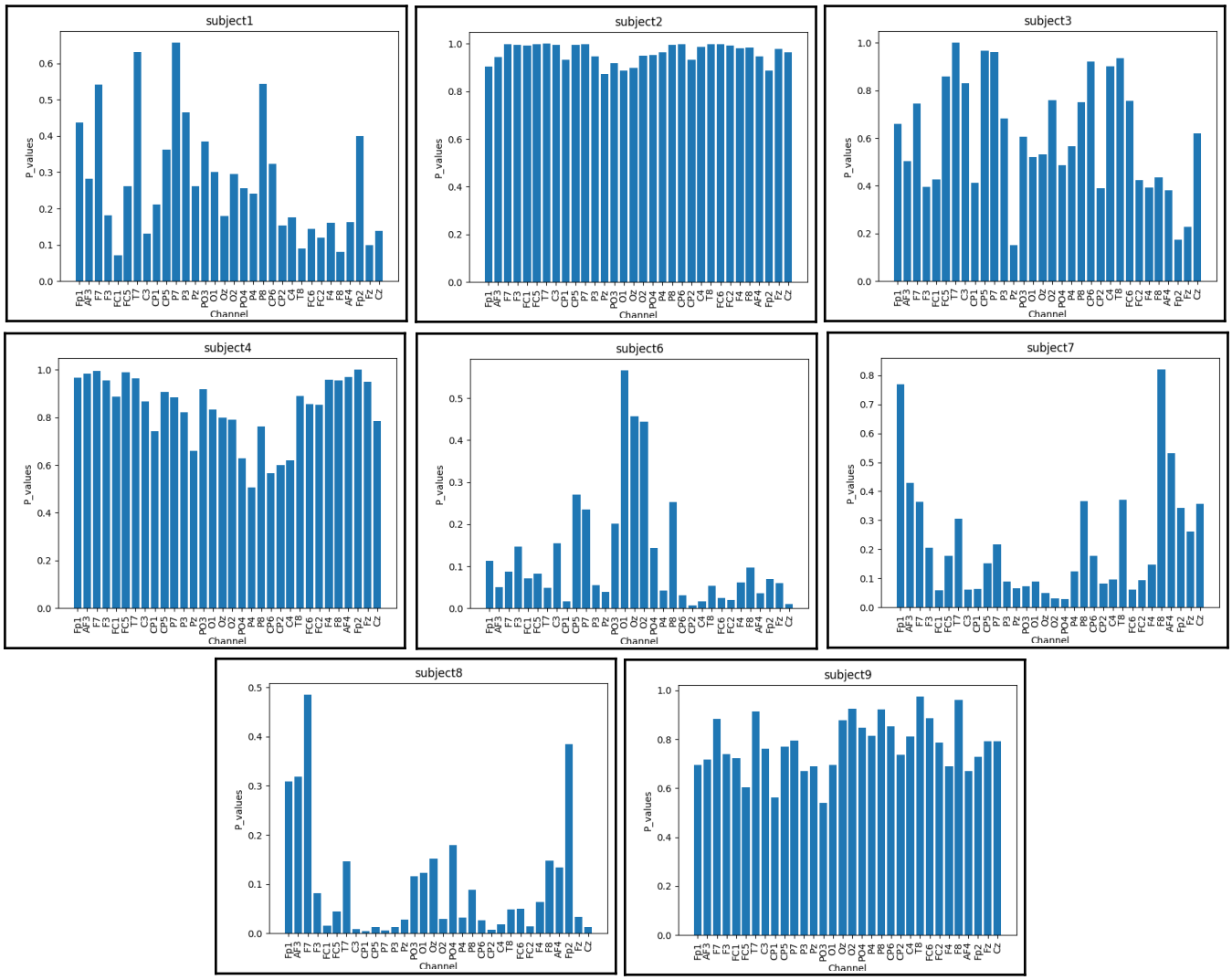


Figure 15: P_values across subjects for different channels over 1000 null-examples when the test is done over 1/24 th of the whole data.

Part C

Section 2

Consistency of P-values across individual subdivisions

Fig. 16 and 17 show the result of p-values across different subdivisions for different subjects and channel Fz.

It seems as we increase the number of subdivisions, there is an increasing of p-values variation across subdivisions. Hence, we have lower number of events information, the statistical difference between target and non-target information vary across subdivisions, and it becomes harder to regenerate the result. Hence, for BCI applications, it becomes harder to generalize the result for multiple subjects (even with healthy people).

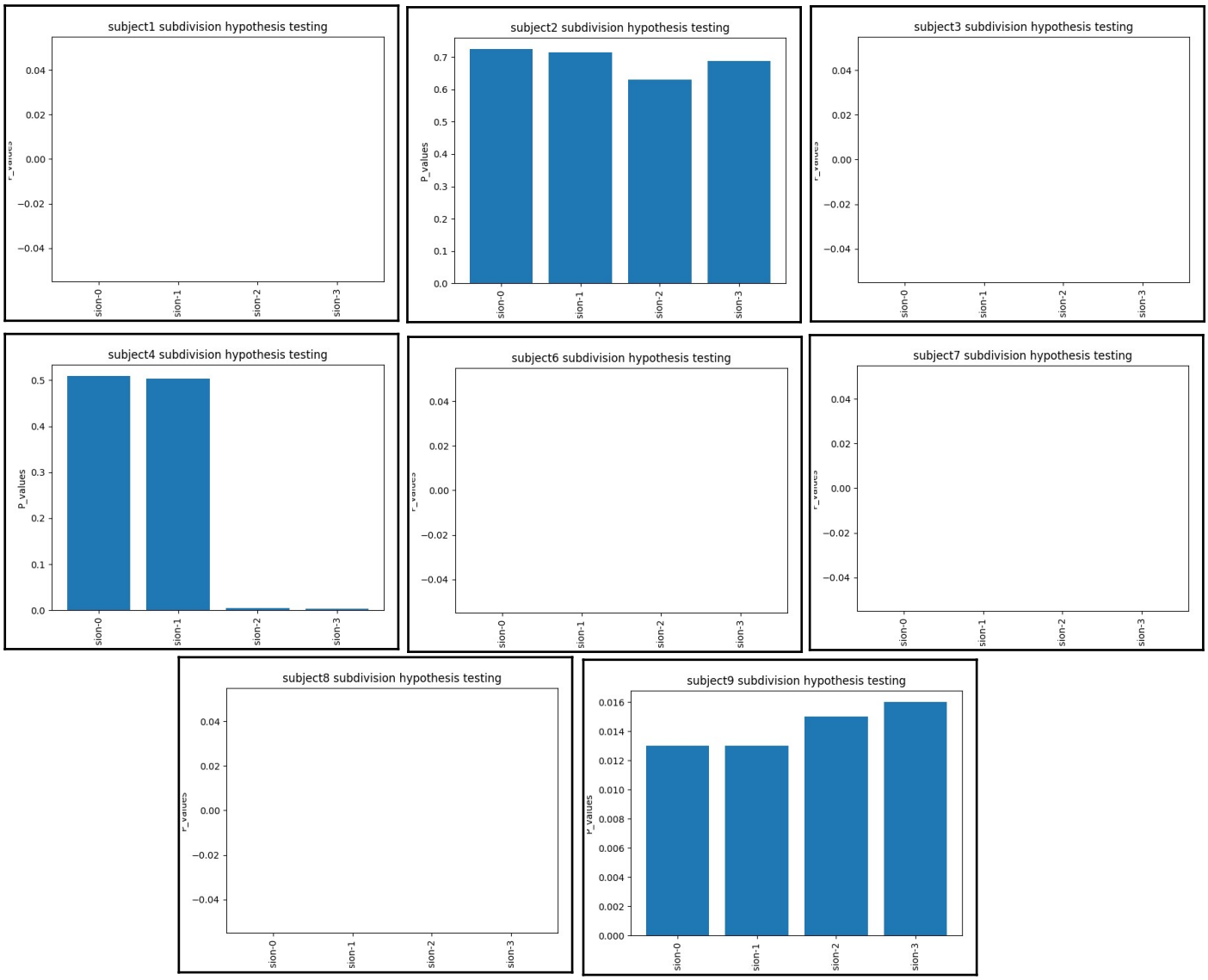


Figure 16: P_values across subdivisions for channel Fz of different subjects when the test is done over 1/4 th of the whole data. Along the x-axis, it is the ID of the subdivision and along the y-axis, it is the p-value.

Analyzing the plots of Fig. 16 and 17, it is evident that for 1/24th of the data, the variation is much larger than the variation of 1/4th of the data. The same is true for other channels as well.

Just to verify this, Fig. 18 shows the variation of P-values across subdivisions across different channels for a single subject (subject 6).

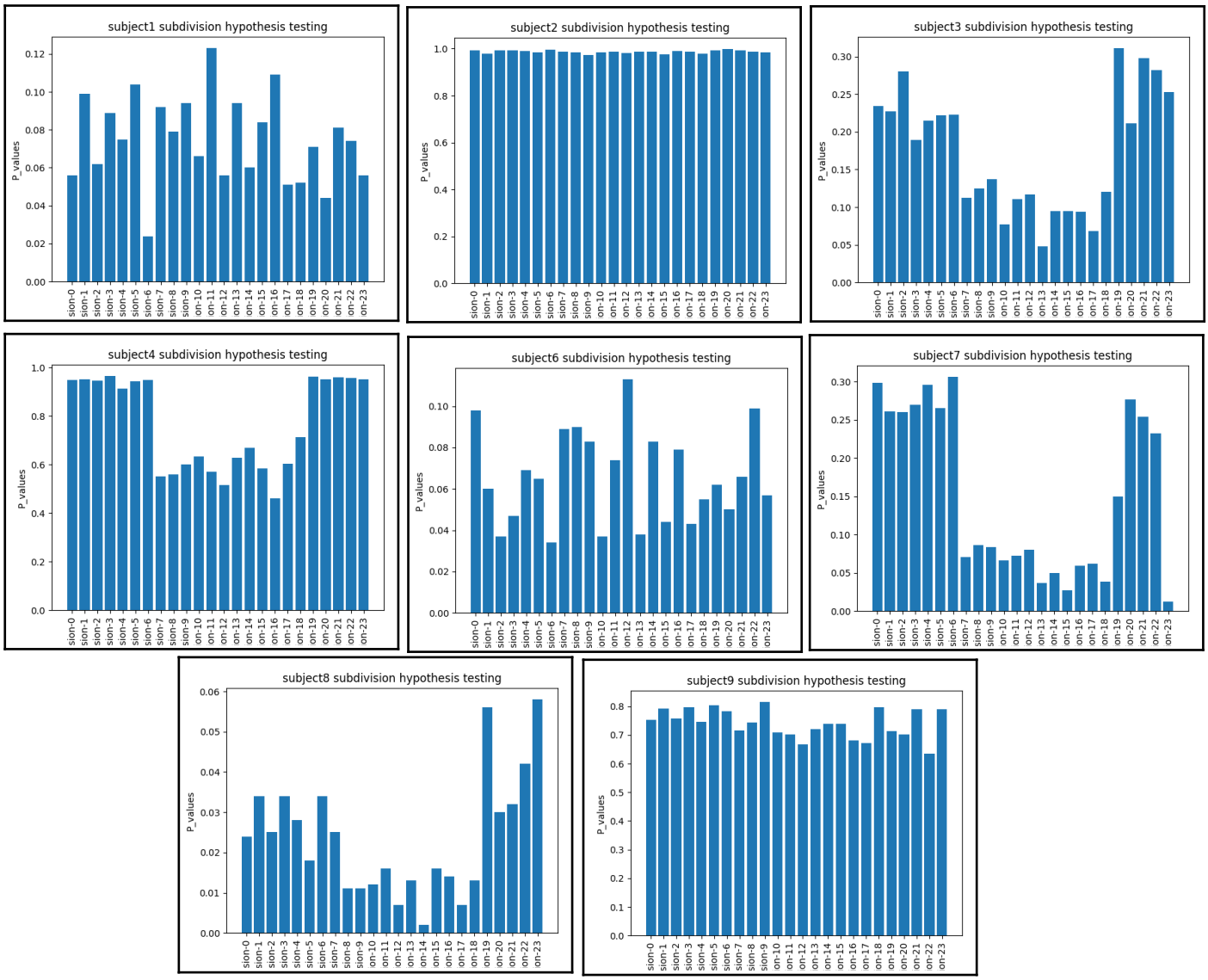


Figure 17: P_values across subdivisions for channel Fz of different subjects when the test is done over 1/24 th of the whole data. Along the x-axis, it is the ID of the subdivision and along the y-axis, it is the p-value.

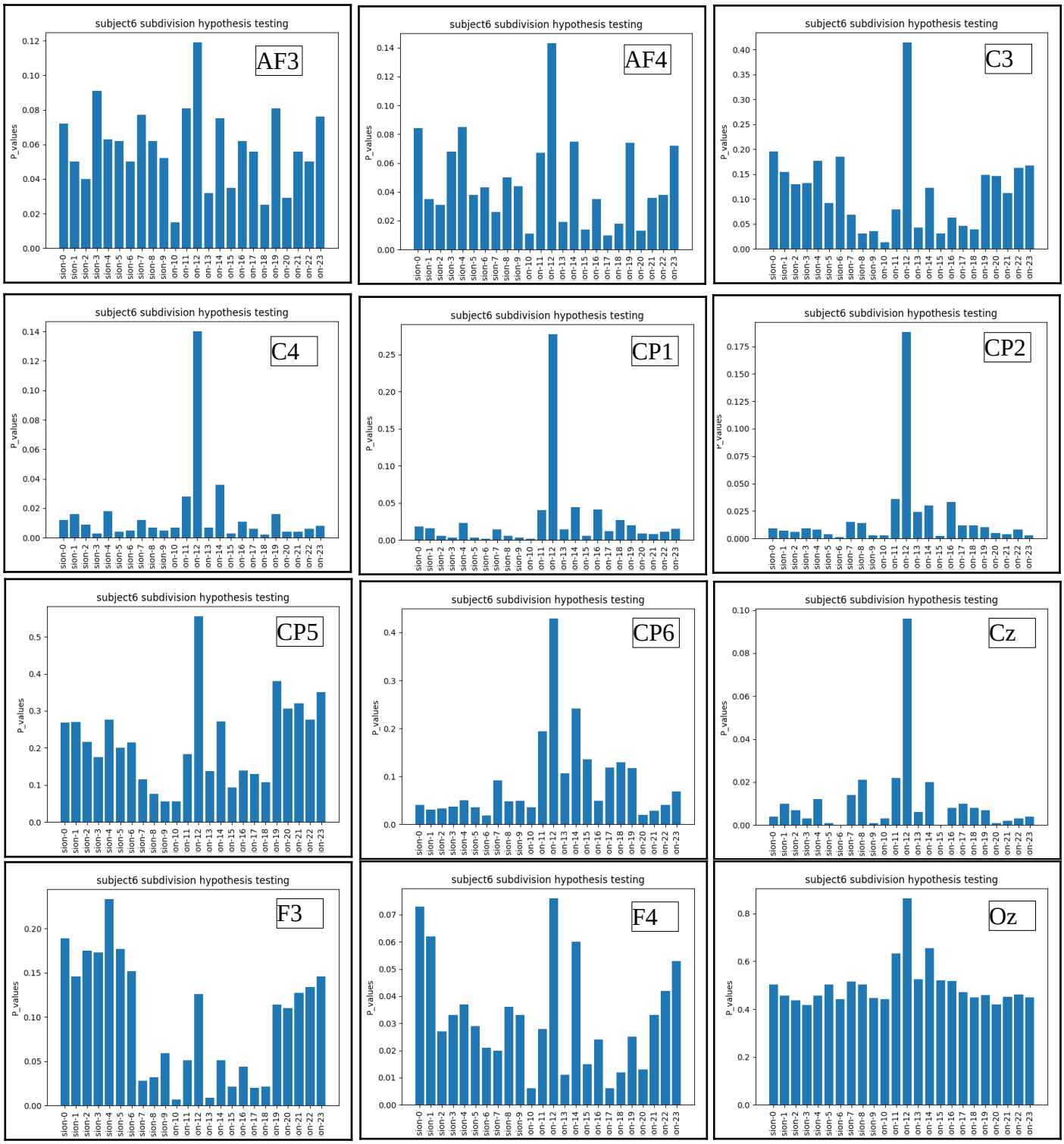


Figure 18: P_values across subdivisions of different channels of subject 6 when the test is done over 1/24 th of the whole data. Along the x-axis, it is the ID of the subdivision and along the y-axis, it is the p-value. The inset of each plot shows the channel name.

Part C

Section 3

Consistency of P-values across individual subdivisions

Fig. 16 and 17 show the resultant p-values across subdivisions for all of the subjects (channel Fz), when 1/4th and 1/24th of the data used for hypothesis testing. It is obvious from the analysis of section 2 in part C that as we increase the number of subdivisions, the variation of p-values also increases. Hence, it is almost impossible to reject the null-hypothesis and becomes harder to differentiate between target and non-target response. Also, training on one subject, and testing on other subjects is quite impossible with such low number of data. Since, the results are varying across subdivisions, it is also reasonable to assume that such model will not be generalized and cannot be regenerated.

Part C

Section 4

Principal Component Analysis (PCA)

Code:

```
def calculate_covariance_matrix(X):
	C = np.zeros((X.shape[1],X.shape[1]))
	for i in range(0,X.shape[1]):
		for j in range(0,X.shape[1]):
			C[i,j] = np.matmul(np.transpose(X[:,i]),X[:,j])
	C = C/X.shape[1]
	return C
def calculate_eigen_vectors(waveforms,num_eigs,K,save_path):
	waveforms = waveforms.astype('float')
	C = calculate_covariance_matrix(waveforms)
	#C = np.cov(np.transpose(waveforms))
	plt.figure()
	plt.matshow(C)
	plt.colorbar()
	#plt.show()
	plt.savefig(save_path+'/Covariance_Matrix_K_'+str(K)+'.png')
	[w,v] = np.linalg.eig(C)#w=eigenvalues, v=eigenvectors; the column v[:,i] is the eigenvector corresponding
to the eigenvalue w[i].
	ws = heapq.nlargest(num_eigs, w)
	ind = [list(w).index(i) for i in ws]
	Q = np.zeros((waveforms.shape[1],num_eigs))
	Q[:,:] = v[:,ind]
	plt.figure()
	plt.plot(Q)
	plt.legend(['Q'+str(i) for i in range(1,Q.shape[1]+1)])
	#plt.show()
	plt.savefig(save_path+'/Eigen_vectors_K_'+str(K)+'.png')
	return Q,v,w
def get_low_dimensional_projections(v,Q,waveforms,K,save_path):
	ak = list()
	bk = list()
	for i in range(0,waveforms.shape[0]):
		ak.append(np.dot(waveforms[i,:],Q[:,0]))
		bk.append(np.dot(waveforms[i,:],Q[:,1]))
```

```
        ak,bk = np.array(ak),np.array(bk)
        plt.figure()
        plt.scatter(ak,bk)
        plt.xlabel('ak')
        plt.ylabel('bk')
        plt.savefig(save_path+'/Projections_K_'+str(K)+'.png')
        return ak,bk
def clustering_data(ak,bk,num_trg,num_ntrg,save_path,K,boundary):
        cl1_indices = np.where(bk<=boundary)
        cl1_a = ak[cl1_indices]
        cl1_b = bk[cl1_indices]
        plt.figure()
        plt.scatter(cl1_a,cl1_b)
        cl2_indices = np.where(bk>boundary)
        cl2_a = ak[cl2_indices]
        cl2_b = bk[cl2_indices]
        plt.scatter(cl2_a,cl2_b)
        plt.xlabel('ak')
        plt.ylabel('bk')
        #plt.show()
        plt.savefig(save_path+'/Cluster_K_'+str(K)+'_b_'+str(boundary)+'.png')
        return cl1_indices[0],cl2_indices[0]
def perf_measure(y_actual, y_pred):
        TP = 0
        FP = 0
        TN = 0
        FN = 0
        for i in range(len(y_pred)):
                if y_actual[i]==y_pred[i]==1:
                        TP += 1
                if y_pred[i]==1 and y_actual[i]!=y_pred[i]:
                        FP += 1
                if y_actual[i]==y_pred[i]==0:
                        TN += 1
                if y_pred[i]==0 and y_actual[i]!=y_pred[i]:
                        FN += 1
        return TP, FP, TN, FN
def ROC_plot(pca_data,ak,bk,Q,num_trg,num_ntrg,ground_truth,K,save_path):
        boundaries = np.arange(np.min(bk),np.max(bk),(np.max(bk)-np.min(bk))/20)
        hit_rate = list()
        false_alarm = list()
        for i in range(len(boundaries)):
                predictions = np.zeros((ground_truth.shape[0],1))
                cl1_indices,cl2_indices = clustering_data(ak,bk,num_trg,num_ntrg,save_path,K,boundaries[i])
                predictions[cl1_indices] = 1
                predictions[cl2_indices] = 0
                [TP,FP,TN,FN] = perf_measure(ground_truth, predictions)
                hit_rate.append(TP/(TP+FN))#Sensitivity
                false_alarm.append(FP/(FP+TN))
                #plot_any_cluster_waveform(pca_data,cl1_indices,ak,bk,Q,K,'b_'+str(i)+'_cl1',save_path)
                #plot_any_cluster_waveform(pca_data,cl2_indices,ak,bk,Q,K,'b_'+str(i)+'_cl2',save_path)
        #Plot the ROC curve
        plt.figure()
        plt.plot(false_alarm,hit_rate)
        plt.title('ROC Curve (K = '+str(K)+')')
        plt.xlabel('False Alarm Rate (1-specificity)')
```

```
	plt.ylabel('Sensitivity (Hit Rate)')
	plt.savefig(save_path+'/ROC_K_'+str(K)+'.png')
	plt.close()
def plot_any_cluster_waveform(waveforms,ind,ak,bk,Q,K,name,save_path):
	plt.figure()
	for i in ind:
		plt.plot(waveforms[i,:])
	plt.xlabel('sample number')
	plt.ylabel('Signal amplitude (uv)')
	plt.savefig(save_path+'/All_Epochs_K_'+str(K)+'.png')
	cl_a = ak[ind]
	cl_b = bk[ind]
	mean_a, mean_b = np.mean(cl_a),np.mean(cl_b)
	print('mean a = ',mean_a)
	print('mean b = ',mean_b)
	plt.plot(mean_a*Q[:,0]+mean_b*Q[:,1],'k', linewidth=3)
	plt.savefig(save_path+'/Cluster_'+name+'_K_'+str(K)+'.png')
def data_prepare(pca_channels,all_targets,all_non_targets,eeg_channels,fs,K,windowing=False,
time_window=[0.2,0.5],downsampling=False,downsampling_rate=32):
	all_targets = {k: all_targets.item().get(k) for k in pca_channels}#keep the pca channels only
	all_non_targets = {k: all_non_targets.item().get(k) for k in pca_channels}
	pca_trg_data = all_targets[pca_channels[0]]
	for i in range(1,len(pca_channels)):
		pca_trg_data = np.concatenate((pca_trg_data,all_targets[pca_channels[i]]),axis=-1)
	pca_ntrg_data = all_non_targets[pca_channels[0]]
	for i in range(1,len(pca_channels)):
		pca_ntrg_data = np.concatenate((pca_ntrg_data,all_non_targets[pca_channels[i]]),axis=-1)
	ground_truth = list()#1=Target, 0=non-target
	if K==1:
		pca_data = (np.concatenate((pca_trg_data,pca_ntrg_data),axis=0)).squeeze()
		ground_truth =
(np.concatenate((np.ones((pca_trg_data.shape[0],1)),np.zeros((pca_ntrg_data.shape[0],1))),axis=0)).squeeze()
	else:
		pca_data = list()
		for row in range(0,pca_trg_data.shape[0]-K,K):
			pca_data.append(np.average(pca_trg_data[row:row+K,:],axis=0))
			ground_truth.append(1)
		for row in range(0,pca_ntrg_data.shape[0]-K,K):
			pca_data.append(np.average(pca_ntrg_data[row:row+K,:],axis=0))
			ground_truth.append(0)
		pca_data = (np.array(pca_data)).squeeze()
		ground_truth = np.array(ground_truth)
	num_trg = math.floor(pca_trg_data.shape[0]/K)
	num_ntrg = math.floor(pca_ntrg_data.shape[0]/K)
	del pca_trg_data
	del pca_ntrg_data
	print('PCA Data Shape (K = {}) = {}'.format(K,pca_data.shape))
	if downsampling:
		pca_data = pca_data[:,0:-1:math.floor(fs/downsampling_rate)]
		fs = downsampling_rate
	if windowing:
		pca_data = pca_data[:,math.floor(time_window[0]*fs):math.floor(time_window[1]*fs)]
	print('PCA Data Shape after post processing (K = {}) = {}'.format(K,pca_data.shape))
	return pca_data,ground_truth,num_trg,num_ntrg
```

Comments:
4 channels (Fz, Pz, Cz and Oz) have been selected for further feature selection. But the 4*2048=8192D signal is reduced to a two dimensional signal through 3 steps.

1. **Down-sampling**:
   The data is down-sampled from 2048Hz to a 32Hz signal by selecting every 64th sample of the signal.
2. **Time-Windowing**:
   Based on the new sampling rate (32Hz), the respective section of the signal (250ms-350ms) is extracted based on the results from section 3 of part B results. Although we already know from the analysis that, it is not always true for unhealthy subjects.
3. **PCA**:
   Next, two principal components are chosen to be final two dimensions based on the principal component analysis technique.

Fig. 19 shows the covariance matrix across all target/non-target events for all subjects.

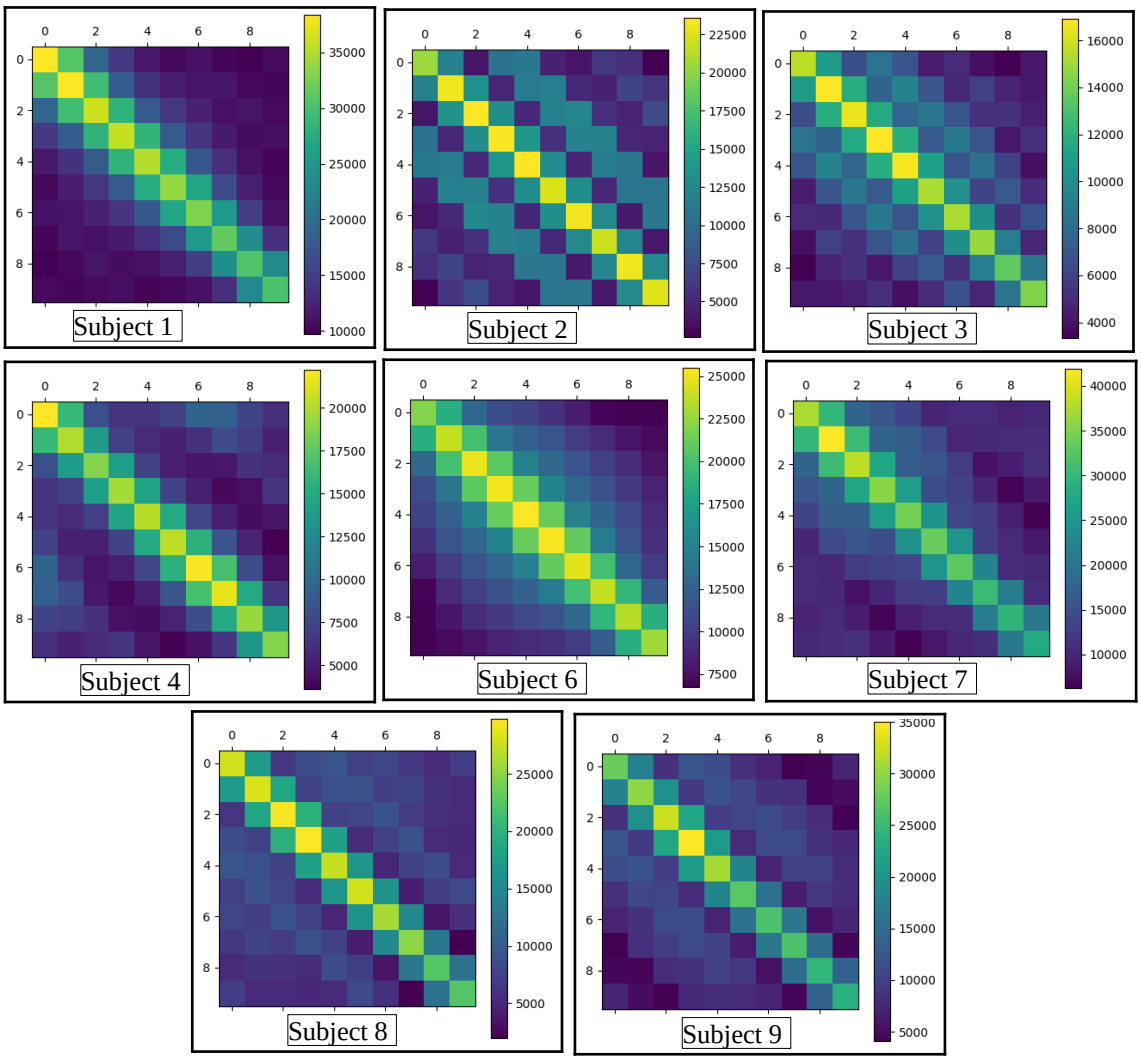


Figure 19: Covariance matrix across all targets/non-targets events for all subjects. Only 4 channels are considered (Fz,Cz,Pz,Oz). The data are first down-sampled, then time-windowed, and finally PCA is applied.

Fig. 20 shows the two principal eigenvectors across all target/non-target events for all subjects.

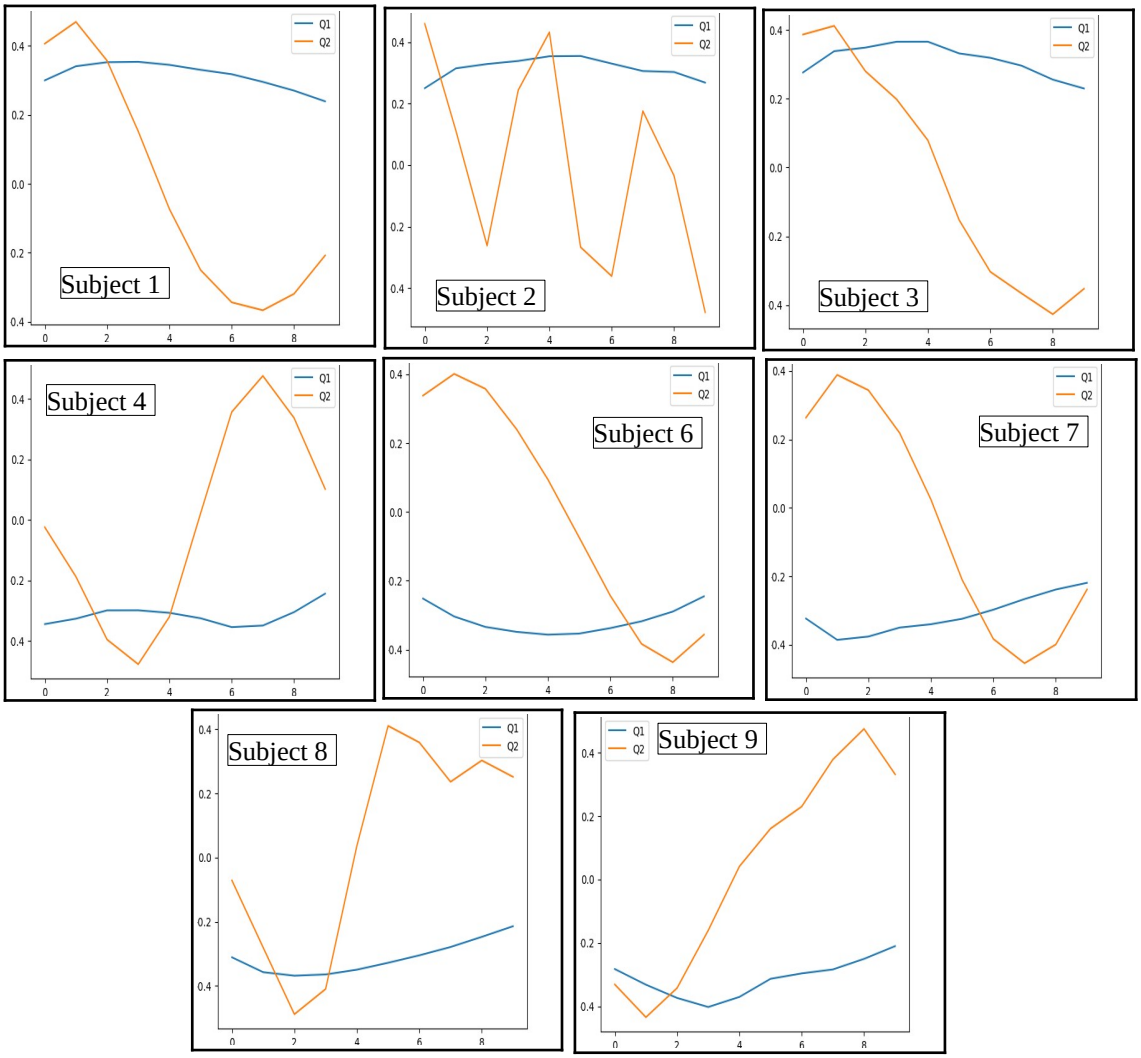


Figure 20: Two principal eigenvectors across all targets/non-targets events for all subjects. Only 4 channels are considered (Fz,Cz,Pz,Oz). The data are first down-sampled, then time-windowed, and finally PCA is applied.

Part C

Section 5

Low dimensional Projection of events based on PCA:

Fig. 21 shows the two dimensional projections of all target/non-target events for all subjects. The blue color indicates the target events and the orange color indicates the non-target events. It is seen that the clusters are not properly separated. Hence, it is not quite possible to draw a boundary between clusters.

In figure 22, 23 and 24, the same projections are shown when K numbers of data are averaged together within each class. K is 10,25 and 50 in Fig. 22, 23 and 24, respectively. It seems, when K in increased, a rough boundary can be drawn between two clusters more easily. However, that boundary is not uniform across subjects. It seems, for healthy subjects, it is easier to draw a boundary than unhealthy subjects. So, increasing K doesn't necessarily differentiate the cluster for unhealthy subjects. But, for healthy subjects, it is quite prominent.

Also, if we notice the projection scales on the x and y axis, it seems, higher values of K results into a clustering into a more condensed space. However, that is true for both clusters. Hence, the relative spacing between clusters among their own members remains the same.

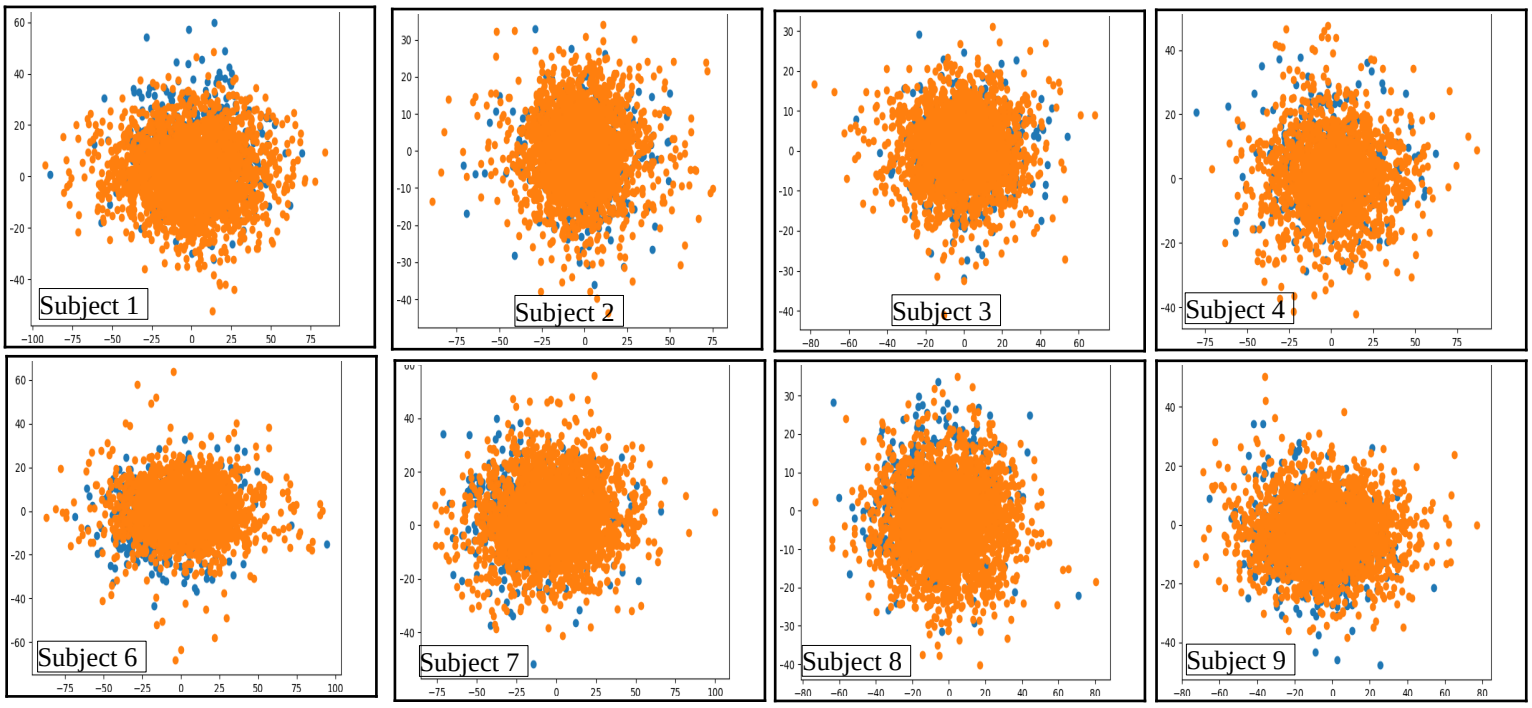


Figure 21: Two dimensional projections of all targets/non-targets events for all subjects. Only 4 channels are considered (Fz,Cz,Pz,Oz). The data are first down-sampled, then time-windowed, and finally PCA is applied.

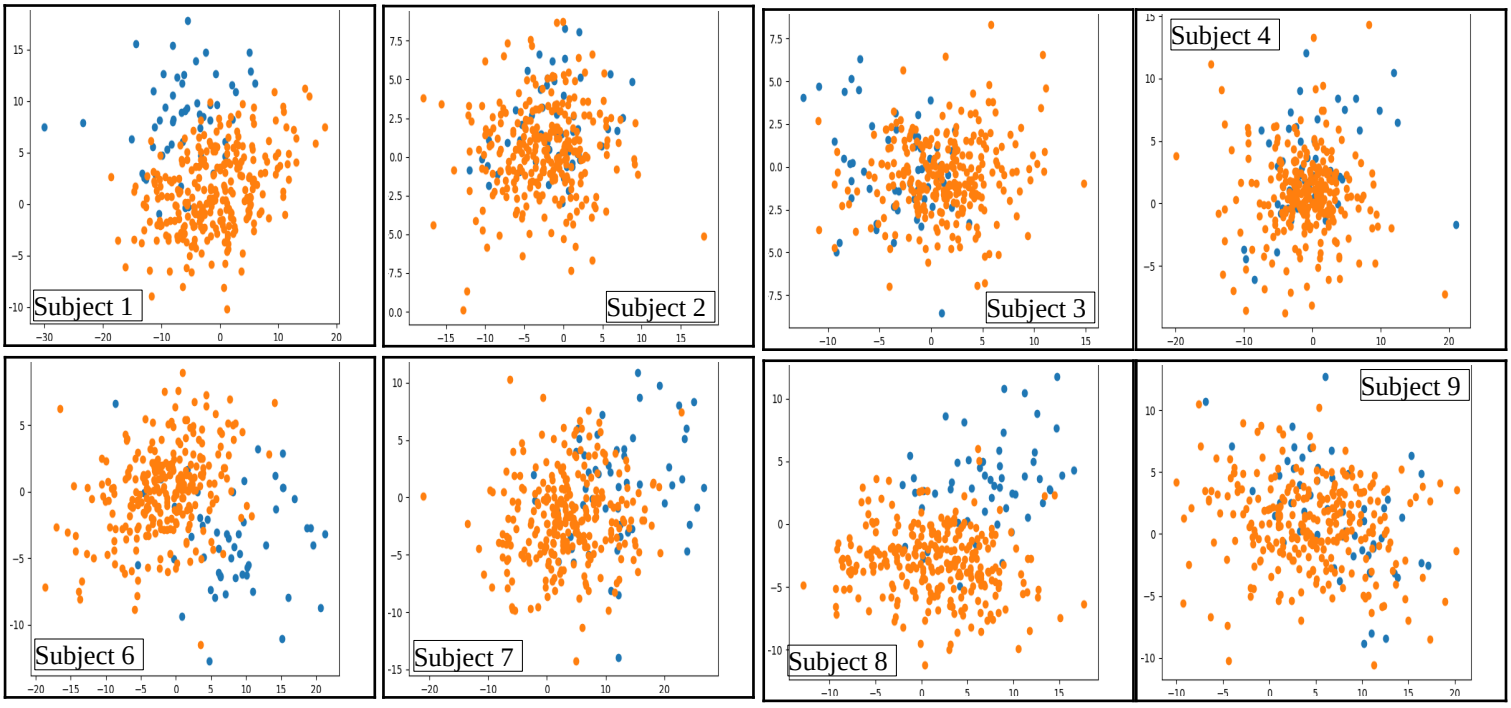


Figure 22: Two dimensional projections of all targets/non-targets events for all subjects when K=10. Only 4 channels are considered (Fz,Cz,Pz,Oz). The data are first down-sampled, then time-windowed, and finally PCA is applied.

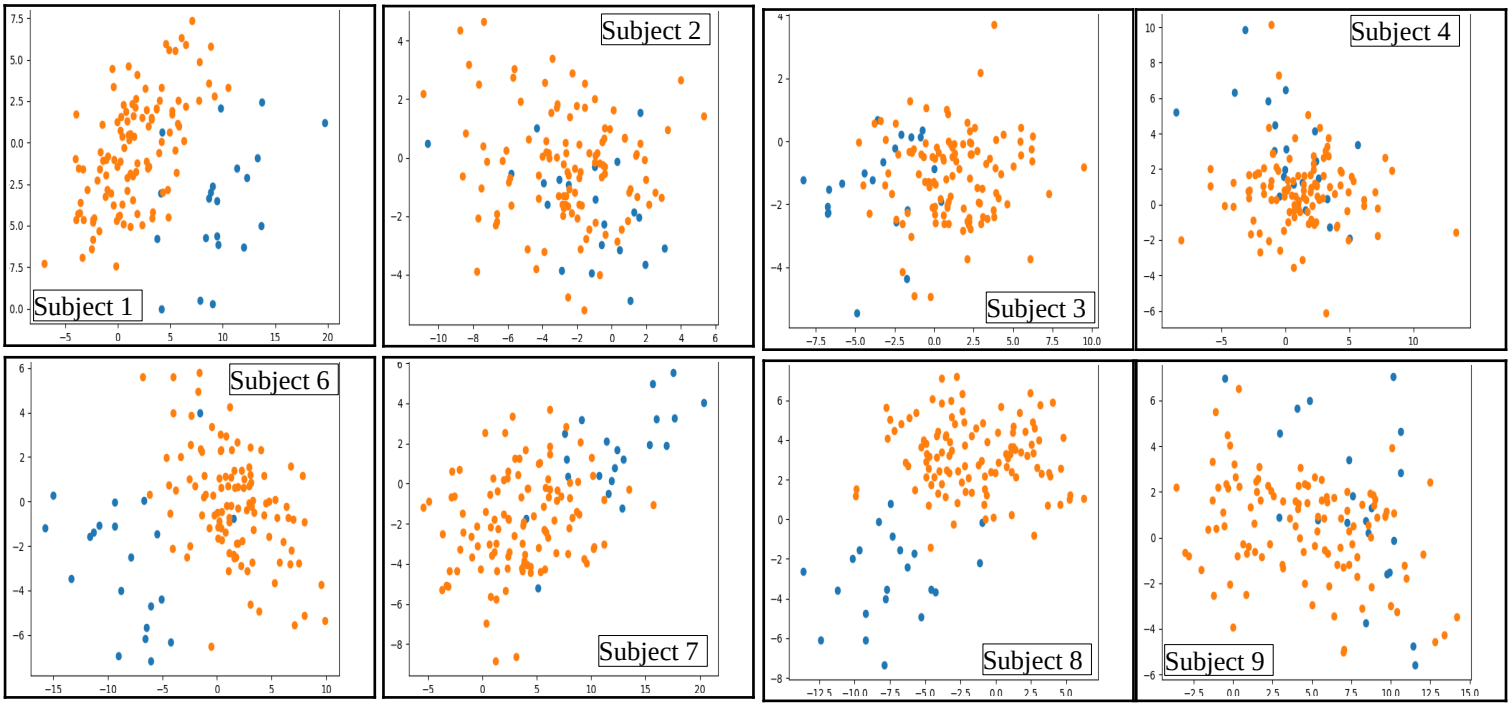


Figure 23: Two dimensional projections of all targets/non-targets events for all subjects when K=25. Only 4 channels are considered (Fz,Cz,Pz,Oz). The data are first down-sampled, then time-windowed, and finally PCA is applied.

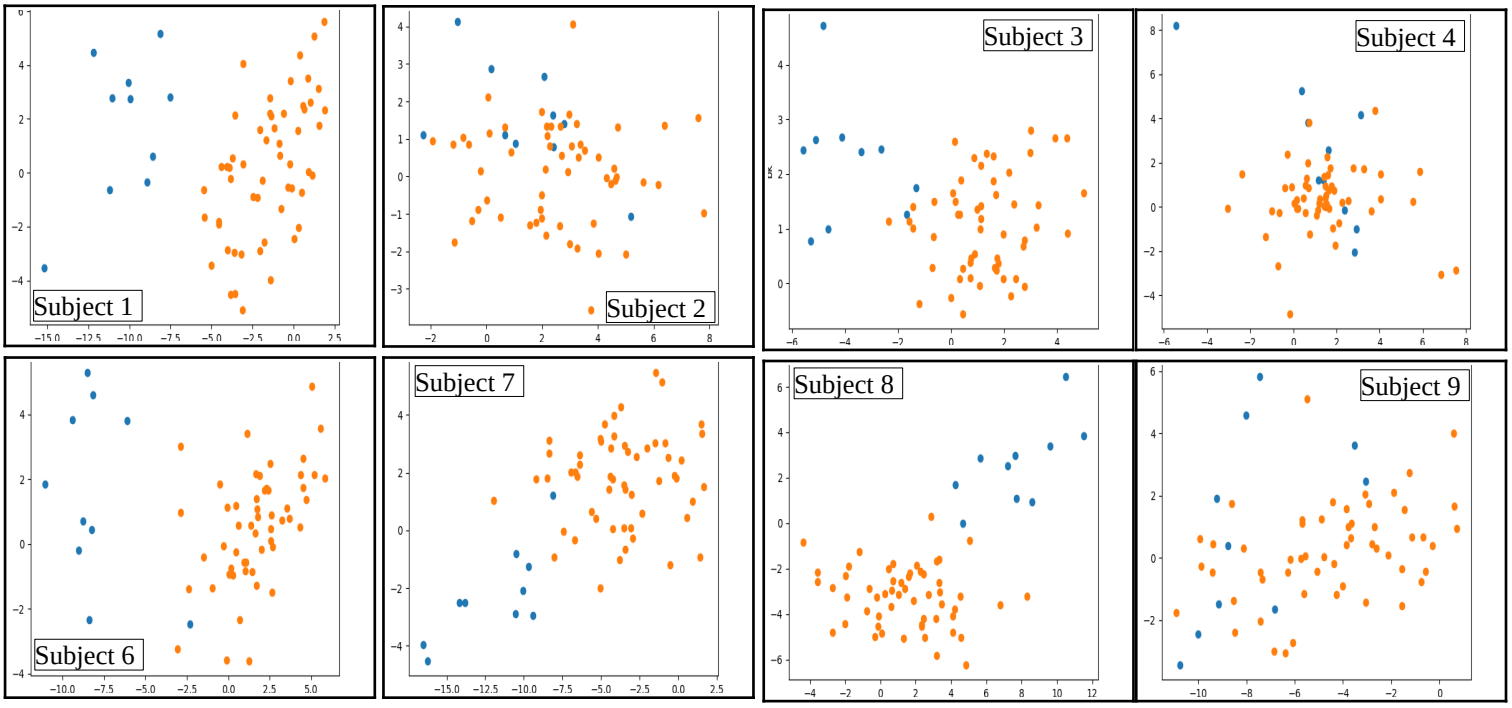


Figure 24: Two dimensional projections of all targets/non-targets events for all subjects when K=50. Only 4 channels are considered (Fz,Cz,Pz,Oz). The data are first down-sampled, then time-windowed, and finally PCA is applied.

Part C

Section 6

Classification Accuracy and ROC Curves:

Fig. 25, 26, 27, 28 shows the ROC curve for K=1, 10, 25, and 50 respectively across all subjects. The ROC curves were prepared with different boundary selection in the two-dimensional projections of the events. For smoother curve, 20 different boundaries were applied.

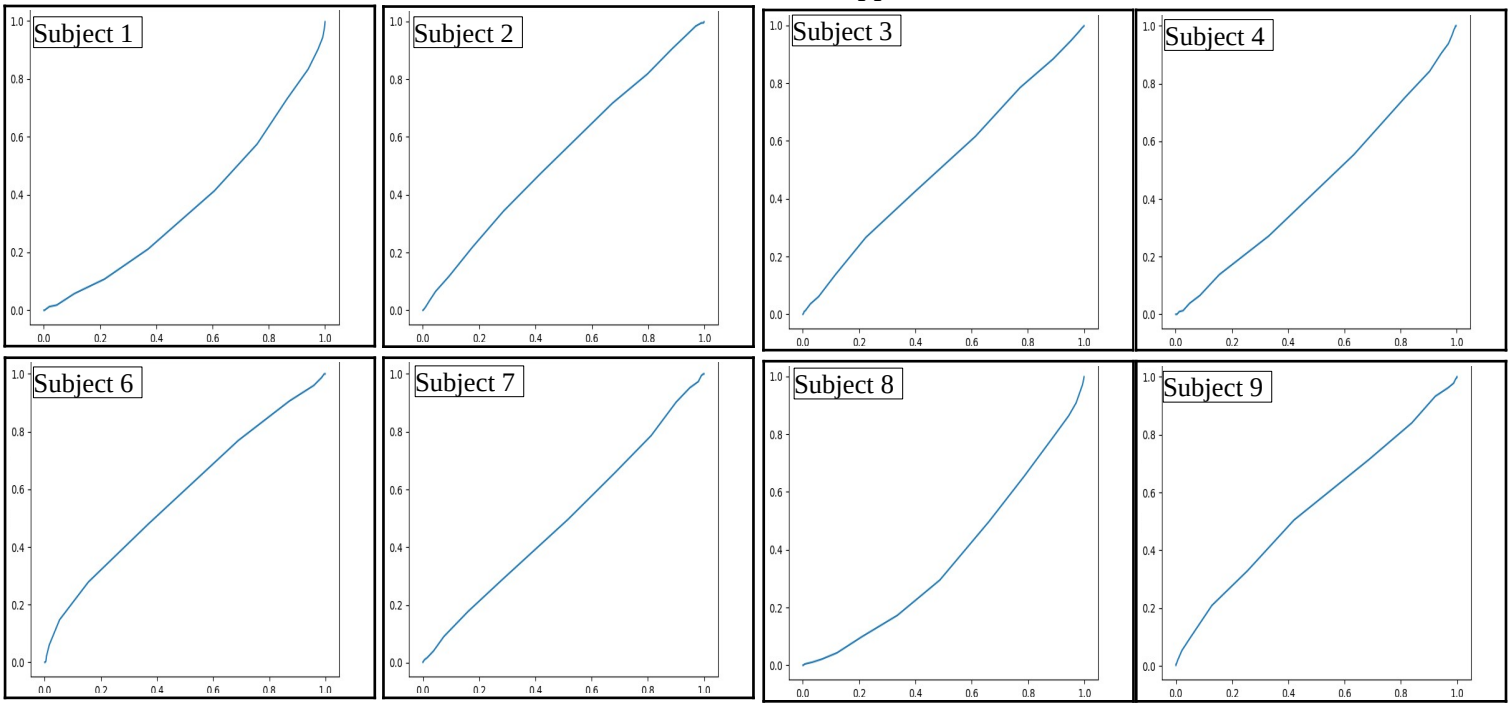


Figure 25: ROC curve when K=1. Only 4 channels are considered (Fz,Cz,Pz,Oz). The data are first down-sampled, then time-windowed, and finally PCA is applied.

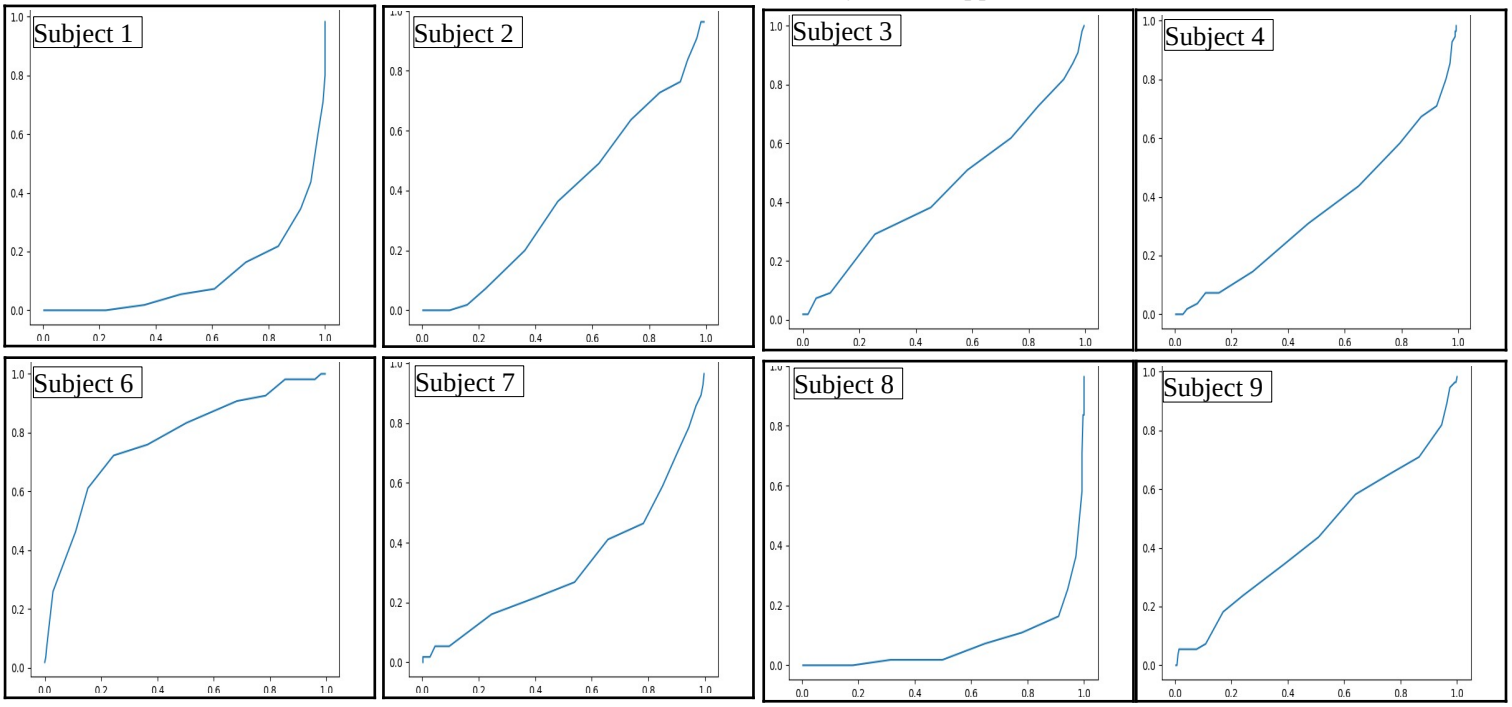


Figure 26: ROC curve when K=10. Only 4 channels are considered (Fz,Cz,Pz,Oz). The data are first down-sampled, then time-windowed, and finally PCA is applied.

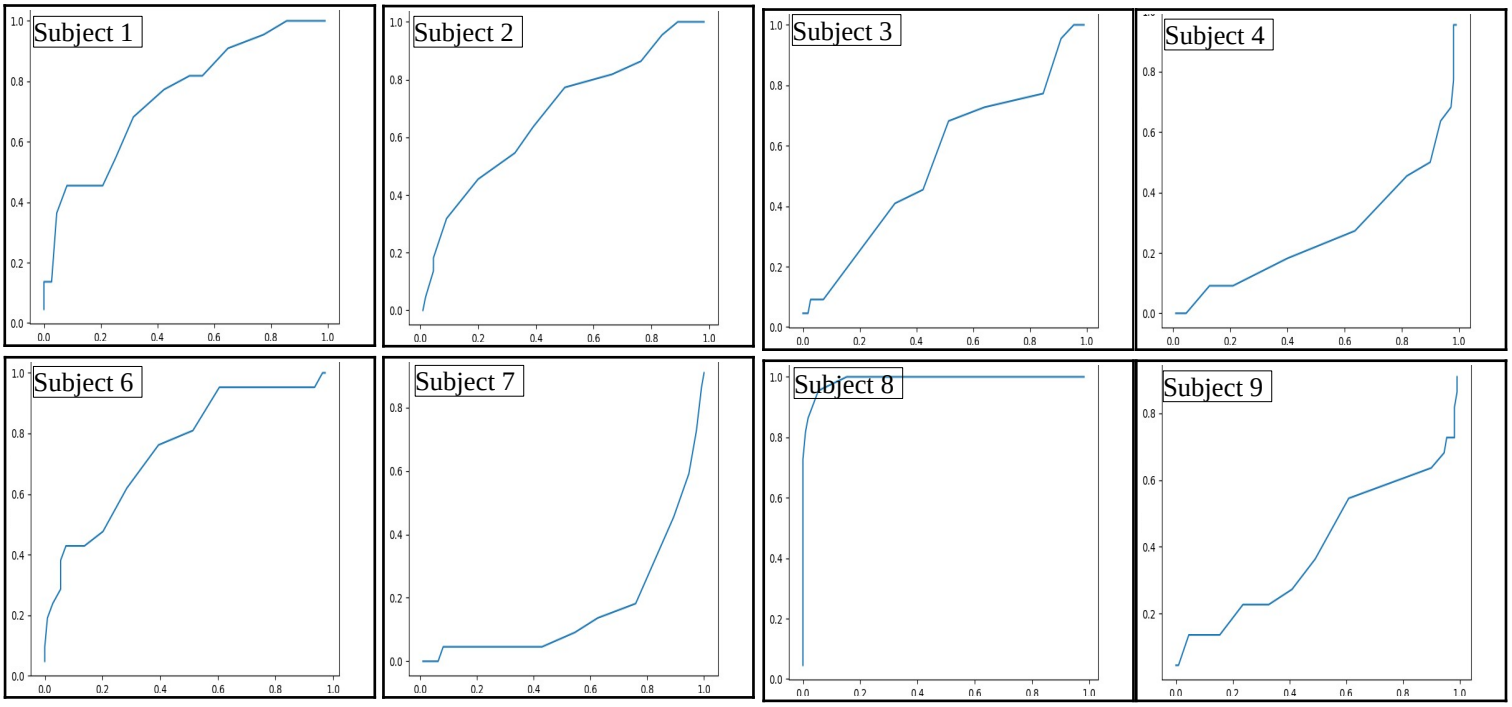


Figure 27: ROC curve when K=25. Only 4 channels are considered (Fz,Cz,Pz,Oz). The data are first down-sampled, then time-windowed, and finally PCA is applied.

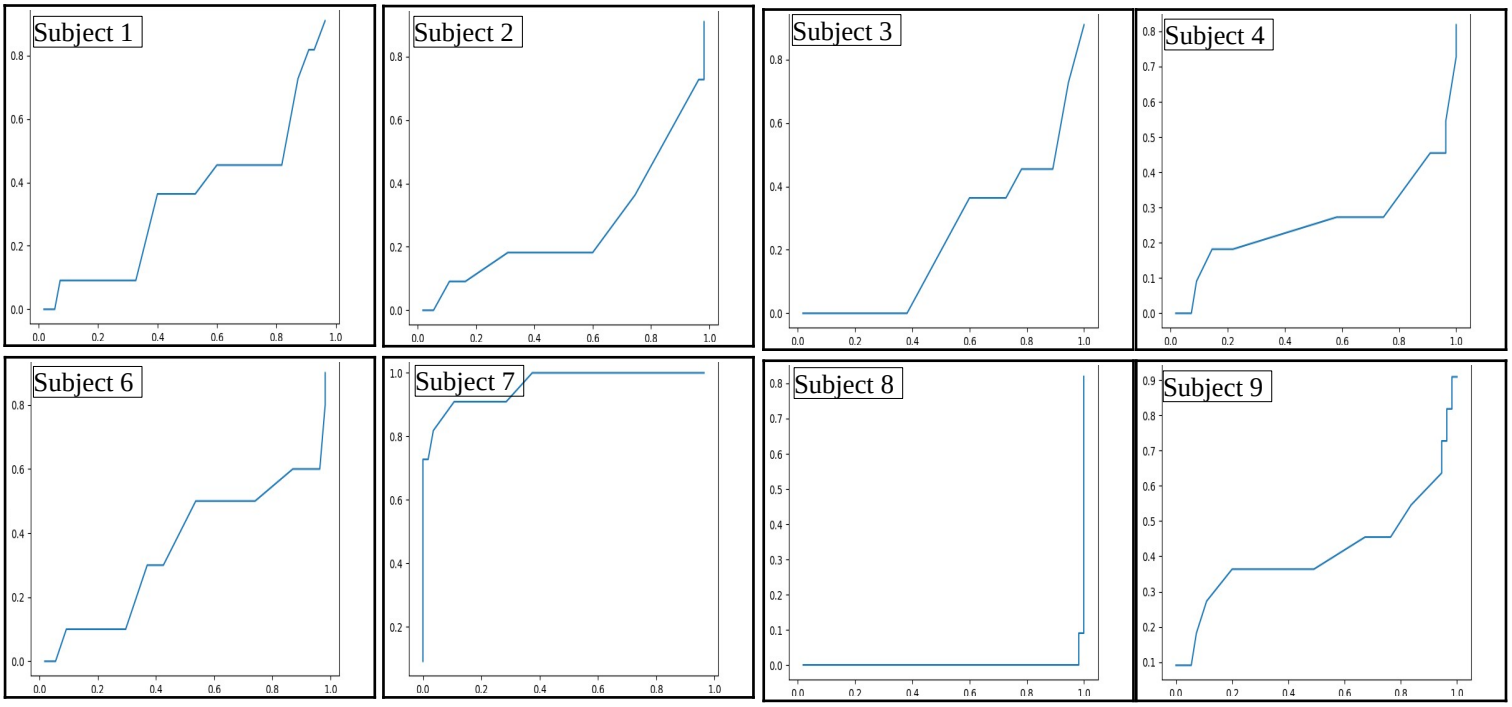


Figure 28: ROC curve when K=50. Only 4 channels are considered (Fz,Cz,Pz,Oz). The data are first down-sampled, then time-windowed, and finally PCA is applied.

Comments:
Analyzing the ROC curves of Fig. 25, 26, 27, and 28, it seems, the way the data were selected (4 channels), down-sampled, time-windowed, application of PCA, it might not be possible to generalize the performance across subjects or classes. For this whole analysis, the following hyper-parameters were selected-

1. Epoch length = 1000ms
2. Baseline DC value calculation = based on first 100ms of epochs
3. Movement noise threshold = 40ms
4. Hypothesis test pool size = 1000
5. PCA channels = ['Fz', 'Cz', 'Pz', 'Oz']
6. Time window selection before PCA = 250ms – 350ms
7. Down-sampling rate before PCA = 32Hz
8. PCA - number of principal components selected = 2

For analyzing the performance of the BCI, other evaluation metrics that could be used -

1. Classification accuracy between target and non-target events
2. Average Classification Accuracy across healthy and unhealthy subjects
3. Precision of performance
4. Sensitivity of the performance w.r.t. specific hyperparameters
5. F1 score/ F2 score
6. Efficiency [2]
7. Utility - average benefit achievable with a BCI [2]

Discussion:
The performance of the BCI can be increased by different combinations of the above hyper-parameters. Also, it is possible to apply deep learning based algorithms on the raw EEG channel information. Moreover, a combination of the above procedures and deep learning based algorithms can be applied. For example, the processed data can be used as input data to the deep learning network for automatic feature extraction and classification. This is actually useful for high number of subjects. Either Multi-layered perceptron networks can be used or a more recent 1D convolutional neural network can be applied. [3]

Reference:
[1]Hoffmann, U., Vesin, J. M., Ebrahimi, T., & Diserens, K. (2008). An efficient P300-based brain–computer interface for disabled subjects. Journal of Neuroscience methods, 167(1), 115-125.

[2] Thompson, D. E., Quitadamo, L. R., Mainardi, L., Gao, S., Kindermans, P. J., Simeral, J. D., ... & Huggins, J. E. (2014). Performance measurement for brain–computer or brain–machine interfaces: a tutorial. *Journal of neural engineering*, *11*(3), 035001.

[3] Hasan, N. I., & Bhattacharjee, A. (2019). Deep learning approach to cardiovascular disease classification employing modified ECG signal from empirical mode decomposition. *Biomedical Signal Processing and Control*, *52*, 128-140.

[4] GitHub Repository for this project: **https://github.com/NahianHasan/Braiin_Computer_Interface_P300**