پردازش سیگنال الکترومایوگرام به منظور شناسایی حرکات دست

استاد درس : آقای دکتر جاهد

اعضای گروه :

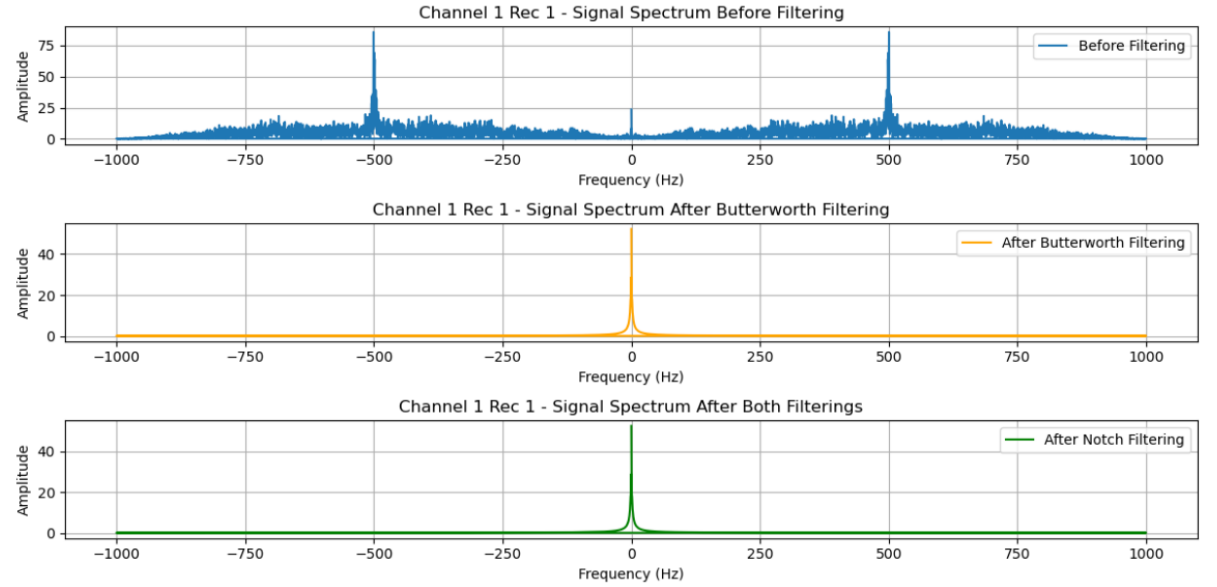
پانته آ عمویی	400101656	پرسش های الف و ب
نیلوفر حسین زاده	400101001	پرسش های ج و د
مهدی طباطبایی	400101515	پرسش های ه و ی

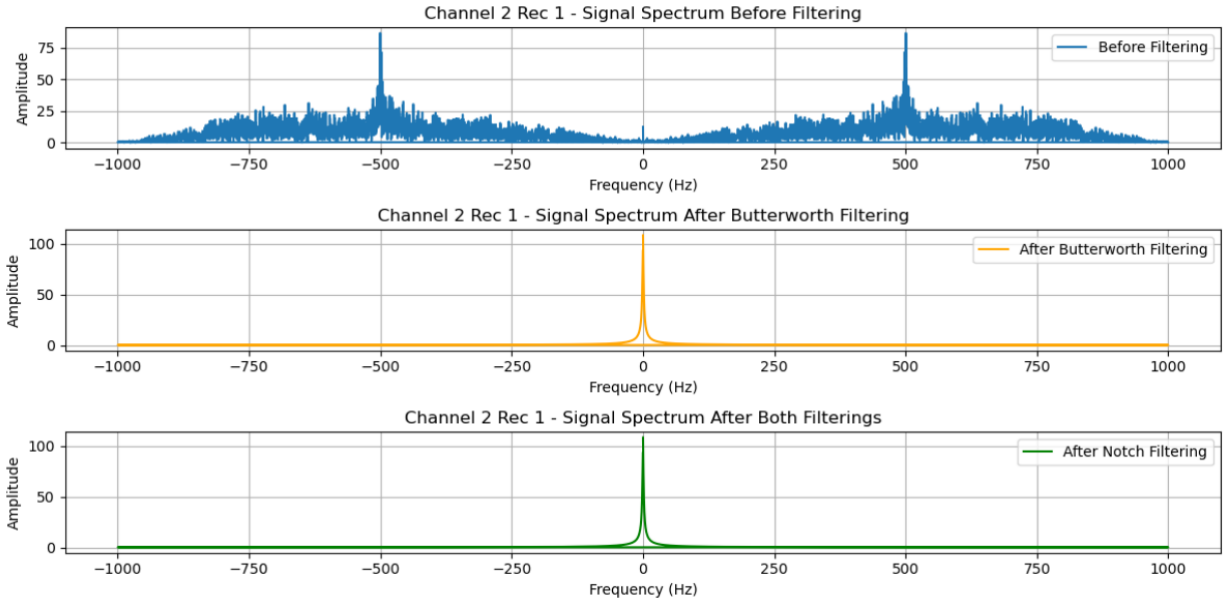
(تمام کدها در پایتون می باشد)

الف) در ابتدا سیگنالهای EMG را به صورت هرکانال جداگانه فیلتر پایینگذر و حذف هارمونیک های برق شهر اعمال کنید.
(پیشنهاد: فیلتر باتروورث مرتبه اول با فرکانس قطع پایین یک هرتز)

ابتدا یک فیلتر باتروورث با فرکانس قطع 1 هرتز به سیگنال اعمال می کنیم. سپس یک فیلتر میان نگذر برای حذف فرکانس برق شهر 50 هرتز(با Quality Factotr برابر 30)

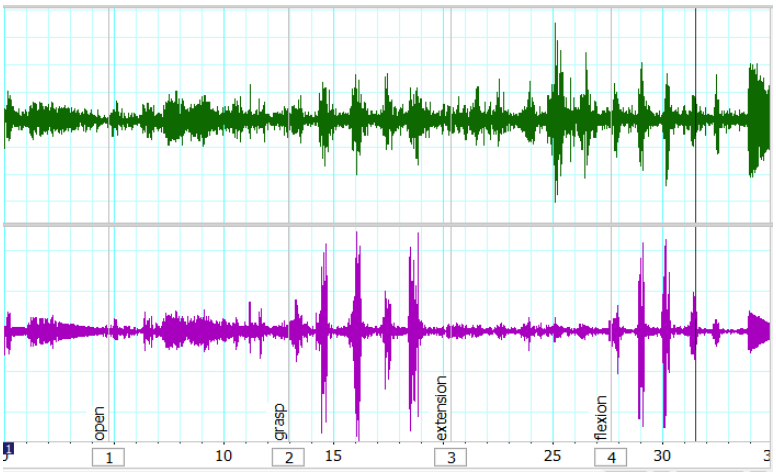
نمایش طیف فرکانسی سیگنال مربوط به هر کانال قبل و بعد از اعمال فیلتر ها :





ب) سیگنال را بر اساس دادگان آموزش و تست جدا کنید و سپس دادگان را براساس کانال نرمالیزه کنید. به دو صورت نرمالیزه کردن سیگنال امکانپذیر میباشد: نرمالیزه دامنه بین صفر و یک – نرمالیزه به میانگین صفر و واریانس یک یکی از دو روش را انتخاب و روی دادگان آموزش محاسبه و اعمال و سپس روی دادگان تست اعمال نمایید. همچنین دلیل و چرایی جداسازی دادگان در این مرحله را ذکر کنید.

با توجه به سیگنال های بدست آمده در آزمایشگاه(در ذیل هم آورده شده)، می توانیم بازه ی زمانی مربوط به هر عمل شامل open, grasp, extension و flexion را بدست بیاوریم. از آنجایی که فرکانس نمونه برداری این برنامه 200 هرتز می باشد، می توان با ضرب کردن این بازه های زمانی در عدد 200 ، بازه های زمانی را به اندیس های متناظرشان در وکتور بدست آمده (دیتای مورد نظر در فرمت mat.) تبدیل کرد.



برای جدا کردن داده ها به test و train، 0.2 از هر بازه زمانی را به test و 0.8 را به train اختصاص می دهیم.

کد استفاده شده در مورد توضیحات بالا :

```
# time spans for each label (in seconds)
labels = [(4.73, 12.93), (12.93, 20.33), (20.33, 27.63), (27.63, 34.85)]

# time interval of each span
x1_interval = 12.93*200 - 4.73*200
x2_interval = 20.33*200 - 12.93*200
x3_interval = 27.63*200 - 20.33*200
x4_interval = 34.85*200 - 27.63*200


# data of each time span for channel 1
x1_train_channel1 = filtered_notch_channel1_rec1_data[int(4.73*200) : int(4.73*200 + 0.8*x1_interval)]
x1_test_channel1 = filtered_notch_channel1_rec1_data[int(4.73*200 + 0.8*x1_interval) : int(12.93*200)]
x2_train_channel1 = filtered_notch_channel1_rec1_data[int(12.93*200) : int(12.93*200 + 0.8*x2_interval)]
x2_test_channel1 = filtered_notch_channel1_rec1_data[int(12.93*200 + 0.8*x2_interval) : int(20.33*200)]
x3_train_channel1 = filtered_notch_channel1_rec1_data[int(20.33*200) : int(20.33*200 + 0.8*x3_interval)]
x3_test_channel1 = filtered_notch_channel1_rec1_data[int(20.33*200 + 0.8*x3_interval) : int(27.63*200)]
x4_train_channel1 = filtered_notch_channel1_rec1_data[int(27.63*200) : int(27.63*200 + 0.8*x4_interval)]
x4_test_channel1 = filtered_notch_channel1_rec1_data[int(27.63*200 + 0.8*x4_interval) : int(34.85*200)]

# data of each time span for channel 2
x1_train_channel2 = filtered_notch_channel2_rec1_data[int(4.73*200) : int(4.73*200 + 0.8*x1_interval)]
x1_test_channel2 = filtered_notch_channel2_rec1_data[int(4.73*200 + 0.8*x1_interval) : int(12.93*200)]
x2_train_channel2 = filtered_notch_channel2_rec1_data[int(12.93*200) : int(12.93*200 + 0.8*x2_interval)]
x2_test_channel2 = filtered_notch_channel2_rec1_data[int(12.93*200 + 0.8*x2_interval) : int(20.33*200)]
x3_train_channel2 = filtered_notch_channel2_rec1_data[int(20.33*200) : int(20.33*200 + 0.8*x3_interval)]
x3_test_channel2 = filtered_notch_channel2_rec1_data[int(20.33*200 + 0.8*x3_interval) : int(27.63*200)]
x4_train_channel2 = filtered_notch_channel2_rec1_data[int(27.63*200) : int(27.63*200 + 0.8*x4_interval)]
x4_test_channel2 = filtered_notch_channel2_rec1_data[int(27.63*200 + 0.8*x4_interval) : int(34.85*200)]
```

سپس داده های test و train را به میانگین صفر و واریانس یک نرمالایز می کنیم.

```
# normalize data to mean = 0 and variance = 1
def normalize_data(data):
    normalized_data = []
    mean = []
    std = []
    for i in range(4):
        mean.append(np.mean(data[i]))
        std.append(np.std(data[i]))
        normalized_data.append((data[i] - mean[i]) / std[i])
    normalized_data = [normalized_data[0], normalized_data[1], normalized_data[2], normalized_data[3]]
    return normalized_data
```

پس داده های train و test به صورت زیر بدست می آیند :

```
# X(features)
x_train_channel1_normalized = normalize_data(train_channel1)
x_test_channel1_normalized = normalize_data(test_channel1)
x_train_channel2_normalized = normalize_data(train_channel2)
x_test_channel2_normalized = normalize_data(test_channel2)


# y(Labels)
y = [1, 2, 3, 4]
```

دلیل تفکیک داده‌ها به مجموعه‌های test و train در این مرحله، آماده‌سازی داده‌ها برای آموزش و ارزیابی یک مدل یادگیری ماشینی است. با تقسیم داده‌ها به مجموعه‌های test و train جداگانه، مطمئن می‌شویم که مدل بر روی بخشی از داده‌ها آموزش داده شده و در قسمت دیگری که در طول آموزش ندیده است، آزمایش می‌شود. این به ارزیابی عملکرد تعمیم مدل و تشخیص هر گونه مشکل بیش از حد برازش کمک می کند.

این جداسازی تضمین می‌کند که معیارهای عملکرد مدل، مانند دقت یا میزان خطا، شاخص‌های قابل اعتمادی از اثربخشی آن در دنیای واقعی هستند. این به تشخیص اینکه آیا مدل می‌تواند به خوبی به داده‌های جدید و دیده نشده تعمیم دهد یا اینکه بیش از حد با داده‌های آموزشی تطبیق دارد کمک می‌کند.

ج) سیگنال را به پنجره های 200 میلی ثانیه با هم پوشانی 190 میلی ثانیه (گام 10 میلی ثانیه) تقسیم کنید.(برا اساس فرکانس نمونه برداری تعداد نمونه های هر پنجره متفاوت خواهد بود)

بدین ترتیب برای پنجره بندی سیگنال از این تابع استفاده نمودم و تابع را روی دادگان آموزش و تست هر کانال اعمال کردم.

```python
def divide_signal_into_windows(signal, window_length_ms, overlap_ms, fs):
    window_length_samples = int(window_length_ms / 1000 * fs)
    overlap_samples = int(overlap_ms / 1000 * fs)

    num_samples = len(signal)
    current_position = 0

    windows = []

    while current_position + window_length_samples < num_samples:
        window = signal[current_position:current_position + window_length_samples]
        windows.append(window)
        current_position += overlap_samples

    return windows
```

```python
# first channel
list_of_windows_of_X_train_channel1 = []
for i in range(len(x_train_channel1_normalized)):
  res = divide_signal_into_windows(x_train_channel1_normalized[i],200,190,fs)
  for j in range(len(res)):
    list_of_windows_of_X_train_channel1.append(res[j])
list_of_windows_of_X_test_channel1 = []
for i in range(len(x_test_channel1_normalized)):
  res = divide_signal_into_windows(x_test_channel1_normalized[i],200,190,fs)
  for j in range(len(res)):
    list_of_windows_of_X_test_channel1.append(res[j])
# second channel
list_of_windows_of_X_train_channel2 = []
for i in range(len(x_train_channel2_normalized)):
  res = divide_signal_into_windows(x_train_channel2_normalized[i],200,190,fs)
  for j in range(len(res)):
    list_of_windows_of_X_train_channel2.append(res[j])
list_of_windows_of_X_test_channel2 = []
for i in range(len(x_test_channel2_normalized)):
  res = divide_signal_into_windows(x_test_channel2_normalized[i],200,190,fs)
  for j in range(len(res)):
    list_of_windows_of_X_test_channel2.append(res[j])
```

د)حداقل سه مورد از ویژگی های ذکر شده در قسمت قبل را از هر پنجره استخراج نموده و در قالب یک دیتافریم به همراه برچسب آن ذخیره کنید.

در این قسمت برای ساختن دیتافریم از سه تابع استفاده کردیم.تابع set_label که برای ست کردن برچسب متناظر پنجره است که به این شرح است:هر لیست که درقسمت قبل ساختیم شامل تمامی پنجره های مربوط به آن دادگان و کانال مورد نظر است و به ترتیب شامل هر چهار حرکت است.به همین خاطر یک چهارم پنجره های اول مربوط به حرکت اول،یک چهار دوم مربوط به حرکت دوم و به همین ترتیب الی آخر.

```python
def set_label(n,num):
  if num >= 0 and num<= int(n/4):
    return 1
  elif num > int(n/4) and num<= int(n/2):
    return 2
  elif num > int(n/2) and num<= int(3*n/4):
    return 3
  else:
    return 4
```

تابع بعدی مربوط به استخراج ویژگی های آماری برای هر پنجره است که من ویژگی های MAV,VAR,STD,RMS را انتخاب نمودم.(به دلیل اینکه کاربردی تر هستند)

```
def extract_features(array):
  results = []
  # mean absolute value
  abs_array = np.abs(array)
  MAV = statistics.mean(abs_array)
  results.append(MAV)

  # variance
  VAR = np.var(array)
  results.append(VAR)

  # standard deviation
  STD = np.std(array)
  results.append(STD)

  # root mean square
  RMS = np.sqrt(np.mean(np.square(array)))
  results.append(RMS)

  return results
```

تابع آخر مربوط به ساختن دیتافریم با در نظر گرفتن برچسب متناظر آن پنجره است که به این صورت است:

```
def creating_dataframe(lst):
  # creating dataframe
  data = []
  num = len(lst)
  for i in range(num):
    result = extract_features(lst[i])
    result.append(set_label(num,i))
    data.append(result)
  df = pd.DataFrame(data, columns=['MAV', 'VAR', 'STD', 'RMS','label'])
  return df
```

Head مربوط به دادگان آموزش و تست هر دو کانال نیز در اینجا قابل مشاهده است (پس از ساخت دیتافریم):

```
        MAV       VAR       STD       RMS  label
0  1.547199  0.002001  0.044736  1.547845      1
1  1.400154  0.001771  0.042085  1.400787      1
2  1.295584  0.000506  0.022495  1.295779      1
3  1.238879  0.000184  0.013563  1.238953      1
4  1.180228  0.000567  0.023811  1.180468      1
---------
        MAV       VAR       STD       RMS  label
0  1.678339  0.028122  0.167697  1.686696      1
1  1.124321  0.027900  0.167033  1.136661      1
2  0.625484  0.017702  0.133050  0.639478      1
3  0.201304  0.015478  0.124409  0.236646      1
4  0.157631  0.010840  0.104115  0.188308      1
        MAV       VAR       STD       RMS  label
0  2.774115  0.018547  0.136186  2.777456      1
1  2.349412  0.014759  0.121486  2.352551      1
2  1.975009  0.011330  0.106444  1.977875      1
3  1.647478  0.008644  0.092972  1.650099      1
4  1.355278  0.007181  0.084741  1.357924      1
---------
        MAV       VAR       STD       RMS  label
0  1.677450  0.045234  0.212683  1.690879      1
1  1.062595  0.019173  0.138466  1.071579      1
2  0.623428  0.017670  0.132927  0.637442      1
3  0.232601  0.011896  0.109067  0.256903      1
4  0.114805  0.009934  0.099671  0.143853      1
```

ه) دو مورد از الگوریتمهای SVM و LDA و KNN و Forest Random را روی دادگان آموزش train کنید و روی دادگان تست test کنید. نتیجه را تحت قالب matrix confusion و دقت نهایی برای هر فرد گزارش دهید.

الگوریتم های SVM و RandomForest را اعمال میکنیم و نتایج را می یابیم.

ابتدا تابعی می نویسم تا xtrain و ytrain و xtest و ytest را بیابیم. برای اینکار با توجه به ویژگی های استخراج شده و دو چنل موجود یک ماتریس با 8 ستون برای x داریم و با توجه به لیبل ها هم یک وکتور برای y.

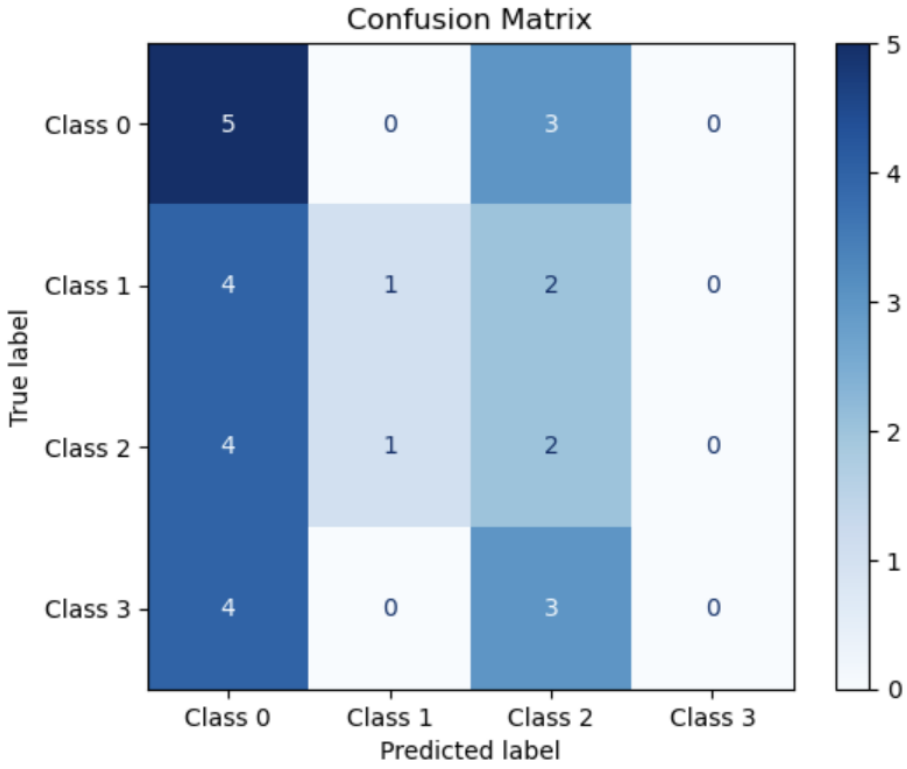
```python
def find_param(df1, df2):

    # Select the first 4 columns from each DataFrame
    selected_columns_df1 = df1.iloc[:, :4]
    selected_columns_df2 = df2.iloc[:, :4]

    # Concatenate the selected columns horizontally (axis=1) into a 2D matrix
    combined_matrix = pd.concat([selected_columns_df1, selected_columns_df2], ax

    # Convert the combined matrix to a NumPy array (if needed)
    x = combined_matrix.to_numpy()

    # Select the 5th column from each DataFrame
    selected_column_df1 = df1.iloc[:, 4]

    # Convert the combined matrix to a NumPy array (if needed)
    y = selected_column_df1.to_numpy()

    return x, y

X_train, y_train = find_param(ch1_train, ch2_train)
X_test, y_test = find_param(ch1_test, ch2_test)
```

حال الگوریتم svm را اعمال میکنیم.

```
# Create an SVM classifier with an RBF kernel
classifier_rbf = SVC(kernel='rbf')
classifier_rbf.fit(X_train, y_train)

# Predict classes for the testing data
y_pred = classifier_rbf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Define class labels (if needed)
class_names = ["Class 0", "Class 1", "Class 2", "Class 3"]  # Replace with your c

# Create a ConfusionMatrixDisplay object and plot it
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=class_
disp.plot(cmap='Blues', values_format='.0f')
plt.title("Confusion Matrix")
plt.show()
```

خروجی را مشاهده میکنیم:

```
Accuracy: 0.28
```



حال الگوریتم RF را اعمال میکنیم:

```
# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model on the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Define class labels
class_names = ['Class 0', 'Class 1', 'Class 2', "Class 3"]  # Replace with your c

# Plot the confusion matrix using Seaborn
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticl
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix (Random Forest)")
plt.show()
```

خروجی را میبینیم:

```
Accuracy: 0.31
```

در اینجا class 0 = label1, class 1 = label 2, class 2 = label 3, class 3 = label 4

ی) چالش این تمرین و ایده برای حل آن یا بهبود دقت در صورت امکان با شفافیت بیان کنید.

در قسمت ب، تبدیل زمان به اندیس های وکتور از جمله چالش های ما بود.

همچنین در قسمت د برای ست کردن لیبل برای هر پنجره مشکل بود، که در ابتدا فرکانس سمپلینگ را اشتباه داده بودیم و طول وکتور کم شده بود، و نمی توانست پنجره بندی کند.

در قسمت ه نیز برای یادگیری الگوریتم های لرنینگ کمی چالش بود که با پرسش از تی ای حل شد. همچنین در آن بخش برای یافتن xTrain, xTest, yTrain, yTest نیز مشکلاتی بود که در اخر با استفاده از فیچر ها و لیبل ها این بخش کامل شد.