GEL4072-7072: Workshop 7 - Artificial intelligence in biomedical systems

Content & Objectives

1. Biomedical data logging (done in previous workshops)
2. Data loader (and EDA - Exploratory Data Analysis)
3. Segmentation
4. Labeling (classification)
5. Feature extraction
6. Train/test split
7. Training
8. Validation / testing

TODO: Download/upload your dataset in the right directory (./data/ from your code/notebook)

Packages import & function definitions (run this section first)

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import RepeatedKFold
from sklearn import preprocessing
```

In [ ]:
```
def getMAV(x):
    '''
    Computes the Mean Absolute Value (MAV)
    :param x: EMG signal vector as [1-D numpy array]
    :return: Mean Absolute Value as [float]
    '''
    MAV = np.mean(np.abs(x))
    return MAV

def getRMS(x):
    '''
    Computes the Root Mean Square value (RMS)
    :param x: EMG signal vector as [1-D numpy array]
    :return: Root Mean Square value as [float]
    '''
    RMS = np.sqrt(np.mean(x**2))
    return RMS

def getVar(x):
    '''
    Computes the Variance of EMG (Var)
    :param x: EMG signal vector as [1-D numpy array]
    :return: Variance of EMG as [float]
    '''
    N = np.size(x)
    Var = (1/(N-1))*np.sum(x**2)
    return Var

def getSD(x):
    '''
    Computes the Standard Deviation (SD)
    :param x: EMG signal vector as [1-D numpy array]
    :return: Standard Deviation as [float]
    '''
    N = np.size(x)
    xx = np.mean(x)
    SD = np.sqrt(1/(N-1)*np.sum((x-xx)**2))
    return SD

def getZC(x, threshold=0):
    '''
    Computes the Zero Crossing value (ZC)
    :param x: EMG signal vector as [1-D numpy array]
    :return: Zero Crossing value as [float]
    '''
    N = np.size(x)
    ZC=0
    for i in range(N-1):
        if (x[i]*x[i+1] < 0) and (np.abs(x[i]-x[i+1]) >= threshold):
            ZC += 1
    return ZC

def getSSC(x, threshold=0):
    '''
    Computes the Slope Sign Change value (SSC)
    :param x: EMG signal vector as [1-D numpy array]
    :return: Slope Sign Change value as [float]
    '''
    N = np.size(x)
    SSC = 0
    for i in range(1, N-1):
        if (((x[i] > x[i-1]) and (x[i] > x[i+1])) or ((x[i] < x[i-1]) and (x[i
] < x[i+1]))) and ((np.abs(x[i]-x[i+1]) >= threshold) or (np.abs(x[i]-x[i-1])
>= threshold)):
            SSC += 1
    return SSC
```

In [ ]:
```
def segment_dataset(filepath, window_length=200, classes=None):
    files = os.listdir(filepath)
    fileNames = []
    data = []
    labels = []
    for f in files:
        if f.endswith('.csv'):
            fileNames.append(f)

    for i in fileNames:
        fileName, fileType = i.split('.')
        metaData = fileName.split('-')      # [0]: Gesture/Label, [1]: Trial

        if np.in1d(int(metaData[0]), classes):
          data_read_ch0 = np.loadtxt(filepath+i, delimiter=',')[6]  # Choosing
channel 6 as first channel for this exercise
          data_read_ch1 = np.loadtxt(filepath+i, delimiter=',')[17] # Choosing
channel 17 as second channel for this exercise

          len_data = len(data_read_ch0)
          n_window = int(len_data/window_length)

          data_windows_ch0 = [data_read_ch0[w*window_length:w*window_length+wi
ndow_length] for w in range(n_window)]
          data_windows_ch1 = [data_read_ch1[w*window_length:w*window_length+wi
ndow_length] for w in range(n_window)]

          data += [(a, b) for a, b in zip(data_windows_ch0, data_windows_ch1)]

          labels += [int(metaData[0])]*n_window
        else:
          pass

    return data, labels

def features_dataset(data, MAV=True, RMS=True, Var=True, SD=True, ZC=True, SSC
=True):
    dataset = []
    for d in data:
        feature_vector = []
        if MAV==True:
            feature_vector += [getMAV(d[0])]
            feature_vector += [getMAV(d[1])]
        if RMS==True:
            feature_vector += [getRMS(d[0])]
            feature_vector += [getRMS(d[1])]
        if Var==True:
            feature_vector += [getVar(d[0])]
            feature_vector += [getVar(d[1])]
        if SD==True:
            feature_vector += [getSD(d[0])]
            feature_vector += [getSD(d[1])]
        if ZC==True:
            feature_vector += [getZC(d[0])]
            feature_vector += [getZC(d[1])]
        if SSC==True:
            feature_vector += [getSSC(d[0])]
            feature_vector += [getSSC(d[1])]
        dataset += [feature_vector]
    return dataset
```

Main exercise

In this exercise, we will build a dataset from 2 channels of EMG signals recorded on the forearm during the contractions of up to 6 different hand gestures.

First, we will complete the exercise with the signals provided in the directory ./data/. The dataset follows a nomenclature corresponding to AAA-BBB.csv, where AAA corresponds to the gesture label and BBB to the identification of the repetition.

The gestures are:

- 0: closed fist
- 1: thumb up
- 2: tri-pod pinch
- 3: neutral hand position
- 4: fine pinch (index+thumb)
- 5: pointed index

Note that the provided dataset is too small to properly train a machine learning algorithm. It is provided as such for educational purpose.

1. Exploratory Data Analysis (EDA)

Explore the data files, plot signals, etc.

In [ ]:
```
# Loading recorded EMG signals into numpy arrays
gesture_0_0 = np.loadtxt('./data/000-000.csv', delimiter=',')
gesture_0_1 = np.loadtxt('./data/000-001.csv', delimiter=',')
gesture_0_2 = np.loadtxt('./data/000-002.csv', delimiter=',')

gesture_1_0 = np.loadtxt('./data/001-000.csv', delimiter=',')
gesture_1_1 = np.loadtxt('./data/001-001.csv', delimiter=',')
gesture_1_2 = np.loadtxt('./data/001-002.csv', delimiter=',')

gesture_2_0 = np.loadtxt('./data/002-000.csv', delimiter=',')
gesture_2_1 = np.loadtxt('./data/002-001.csv', delimiter=',')
gesture_2_2 = np.loadtxt('./data/002-002.csv', delimiter=',')

gesture_3_0 = np.loadtxt('./data/003-000.csv', delimiter=',')
gesture_3_1 = np.loadtxt('./data/003-001.csv', delimiter=',')
gesture_3_2 = np.loadtxt('./data/003-002.csv', delimiter=',')
```

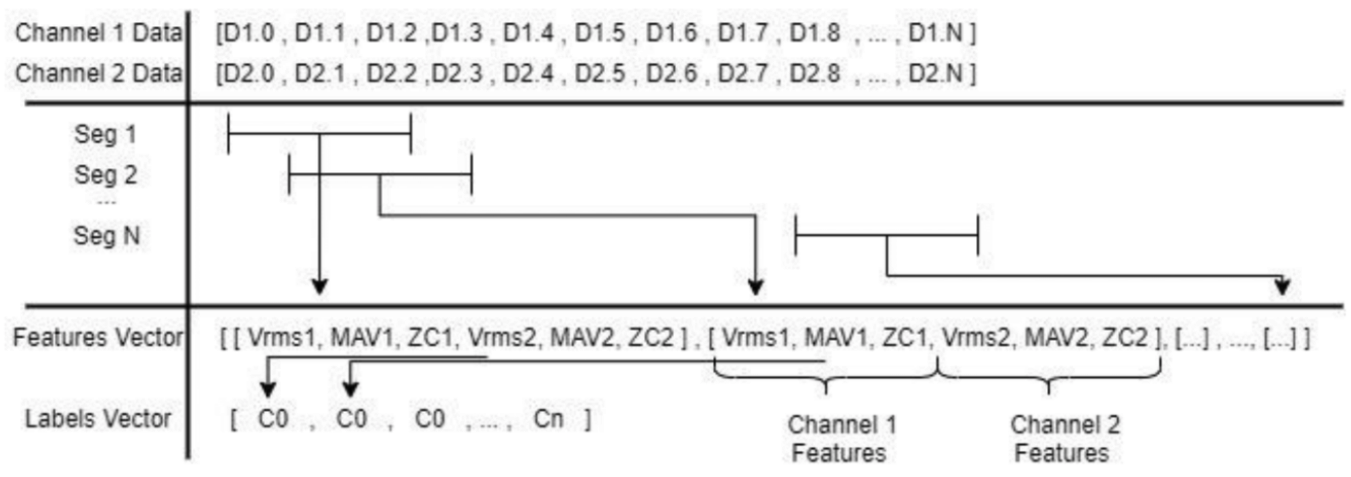
In [ ]:
```
# Define a function to plot the FFT of a signal
def plotfft(signal, fs, axis=[0, 500, 0, 3e10]):
  ps = np.abs(np.fft.fft(signal))**2
  time_step = 1/fs
  freqs = np.fft.fftfreq(signal.size, time_step)
  idx = np.argsort(freqs)
  plt.plot(freqs[idx], ps[idx])
  plt.axis(axis)
  return
```

In [ ]:
```
# Exploratory data analysis (EDA)
data = gesture_0_0[6]   # Use index 6 and 17 for this exercise
                        # (channel 6 and 17 of original data)

plt.figure(figsize=(15, 3))
plt.subplot(1, 2, 1)
plt.title('EMG Signal')
plt.plot(data)
plt.axis([0, None, -6000, 6000])

plt.subplot(1, 2, 2)
plt.title('FFT')
plotfft(data, fs=1000)

plt.show()
```

2. Segmentation, feature extraction & labeling

To generate a data point, segment the EMG signals in windows (window length depends on application). For each window:

- Calculate the chosen features
- Record the results in a feature vector (including features from both channels in one vector as shown below)
- Save the class label in a variable (the label is a numerical value associated with a gesture


Channel 1 Data
[D1.0 , D1.1 , D1.2 ,D1.3 , D1.4 , D1.5 , D1.6 , D1.7 , D1.8 , ... , D1.N]
Channel 2 Data
[D2.0 , D2.1 , D2.2 ,D2.3 , D2.4 , D2.5 , D2.6 , D2.7 , D2.8 , ... , D2.N]
Seg 1
Seg 2
...
Seg N
Features Vector
[[Vrms1, MAV1, ZC1, Vrms2, MAV2, ZC2] , [Vrms1, MAV1, ZC1, Vrms2, MAV2, ZC2], [...] , ..., [...]]
Labels Vector
[C0 , C0 , C0 , ... , Cn]
Channel 1 Features
Channel 2 Features

3. Data splitting and training

The feature vectors set will be used as training data set and test data set. To shuffle the dataset, we will be using the Kfold from the Scikit-learn package. This function will select arbitrary indices in the dataset. These indices will be used to separate the feature vectors list and the labels list in a train data set and a test data set. This process will be done 3 times to get a mean score/accuracy. This technique is used when the available dataset is relatively small to obtain a metric that is less biased towards a limited dataset.

We will define a LDA classifier with the object LinearDiscriminantAnalysis () from the

Scikit

learn python machine learning library . Below, the object is instantiated in the variable clf. The training process is done with the function fit where the algorithm learns the discriminating functions from the features and labels of the training data set. The performance of the training is evaluated with the function score where it predicts the labels of the input d ata and compares it to its true labels. A function predict exists to return the predicted labels directly.

In [ ]:
```
# Loading recorded EMG signals into numpy arrays
path = './data/'

classes = [0, 1, 2, 3, 4, 5]

data, labels = segment_dataset(path, window_length=150, classes=classes)

features_set = features_dataset(data, MAV=True, RMS=True, Var=True, SD=True, Z
C=True, SSC=True)

features_set = preprocessing.scale(features_set) # preprocessing module import
ed from sklearn
```

In [ ]:
```
feat_x = 10  # 0-1: MAV, 2-3: RMS, 4-5: Var, 6-7: SD, 8-9: ZC, 10-11: SSC
feat_y = 11

for c in classes:
  ind = np.where(np.array(labels)==c)
  plt.scatter([f[feat_x] for f in features_set[ind]], [f[feat_y] for f in feat
ures_set[ind]], label='Class '+str(c))
plt.legend()
plt.show()
```

In [ ]:
```
avgScoreTemp = []

kFold_rep = 3
kFold_splits = 3
kFold = RepeatedKFold(n_splits=kFold_splits, n_repeats=kFold_rep)

for i in range(kFold_rep):
  for i_Train, i_Test in kFold.split(features_set):
    clf = LinearDiscriminantAnalysis()
    X_train, X_test = [features_set[j] for j in i_Train], [features_set[k] for
k in i_Test]
    y_train, y_test = [labels[l] for l in i_Train], [labels[m] for m in i_Test
]

    clf.fit(X_train, y_train)
    currentScore = clf.score(X_test, y_test)

    avgScoreTemp += [currentScore]

avgScore = sum(avgScoreTemp)/len(avgScoreTemp)
print('Mean score with k-fold validation: {}'.format(avgScore))
```