MRC Brain Network Dynamics Unit

UNIVERSITY OF OXFORD

Stimulating States of Parkinsonian Tremor

Internship Report

Cagnan Group

Submitted by: Muhammad Roshan Mughees

Project Supervisors: Associate Prof. Hayriye Cagnan

Dr. Tim West

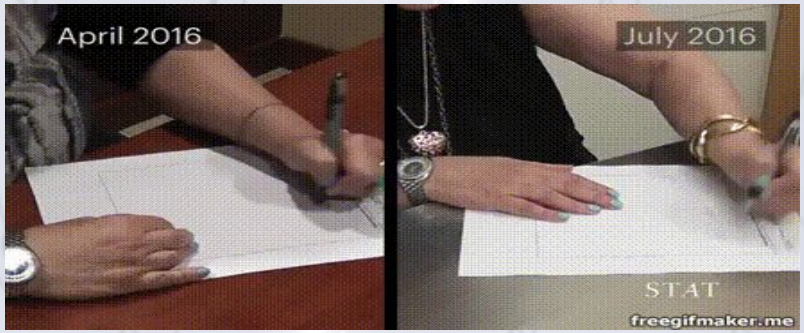
MAIA

Problem Statement / Motivation

- Tremor is defined as a rhythmic, involuntary oscillatory movement of a body part
- AFFECTING AS MANY AS 41 MILLION PATIENTS WORLDWIDE
- Caused by neurological degeneration

Patients cannot even do **basic tasks**
No permanent cure possible until today
Medications work but lose their effectiveness

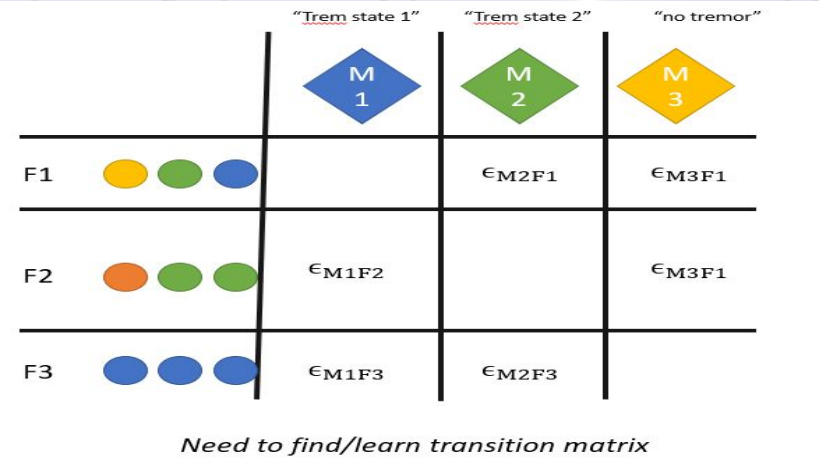
PROBLEM:
Tremor is <u>variable</u> -> This makes it hard to treat (**optimal stimulation parameters** are changing) -> so we need to define states -> understand how to treat

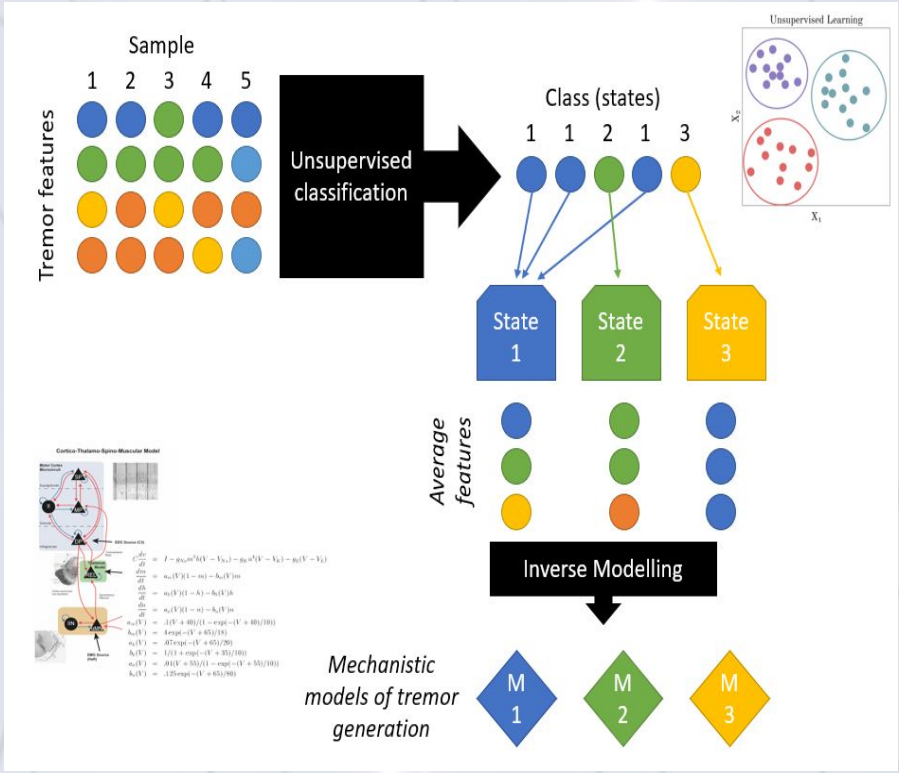


Solution Overview

Tremor changes throughout the day and depending on activity/medication.

- Detect tremor states and their characteristics
- Find parameters to change from one state to another.





Presentation Overview

- Preprocessing of datasets
- Details of feature space calculated
- Clustering algorithm
- Optimization of clustering algorithm
- Validation on simulated data(from oscillators)
- Results on unsupervised data under:
  - Single subject with unique data
  - Single subject with overlapping data (80%)
  - Nine subjects with unique data
- Future work / Other points

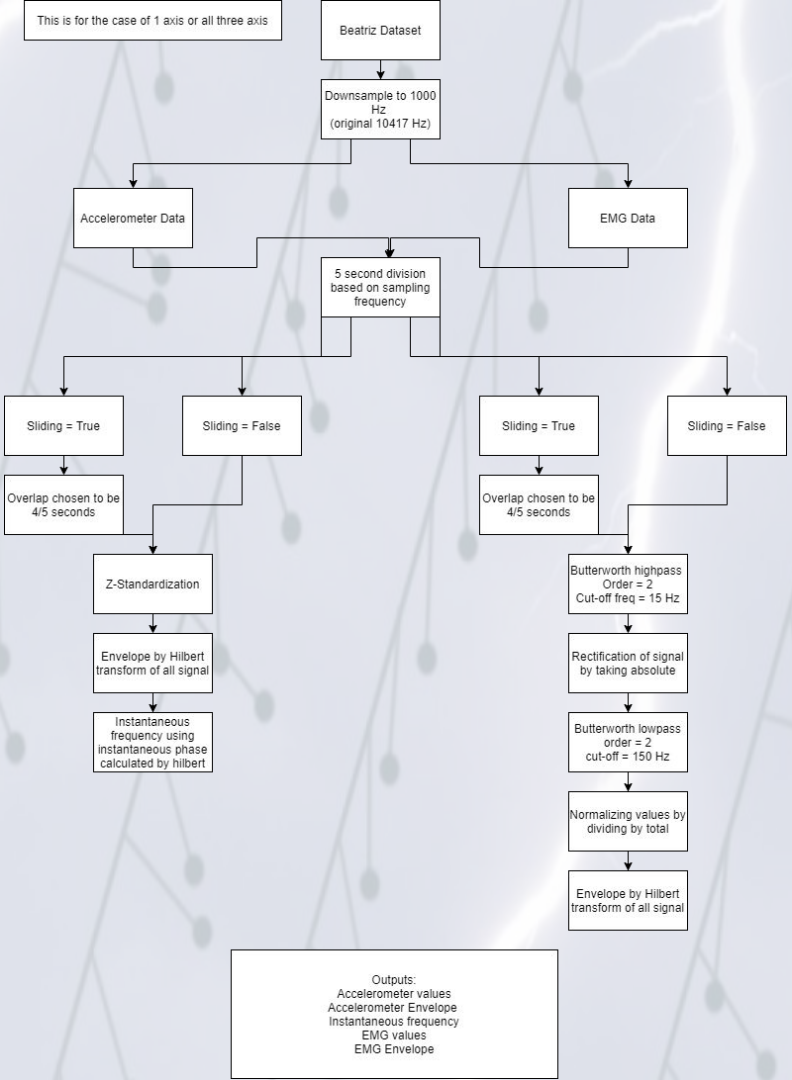
Methodology

Preprocessing pipeline

Dataset (5-6 minutes of recordings) extraction:

- Total patients/samples **10**
- Accelerometer
- EMG
- Accelerometer data **downsampled -> 1000 Hz**, so each sample has **1/1000 second duration** (**original 10417 Hz**)
- High pass filtered the EMG at 15 Hz -> remove **DC-offset** - output signal centered around zero
- Low pass filtered the EMG at 150 Hz -> EMG signals are mostly dominant until up to 150 Hz

Window size + overlap is parameter to be optimized

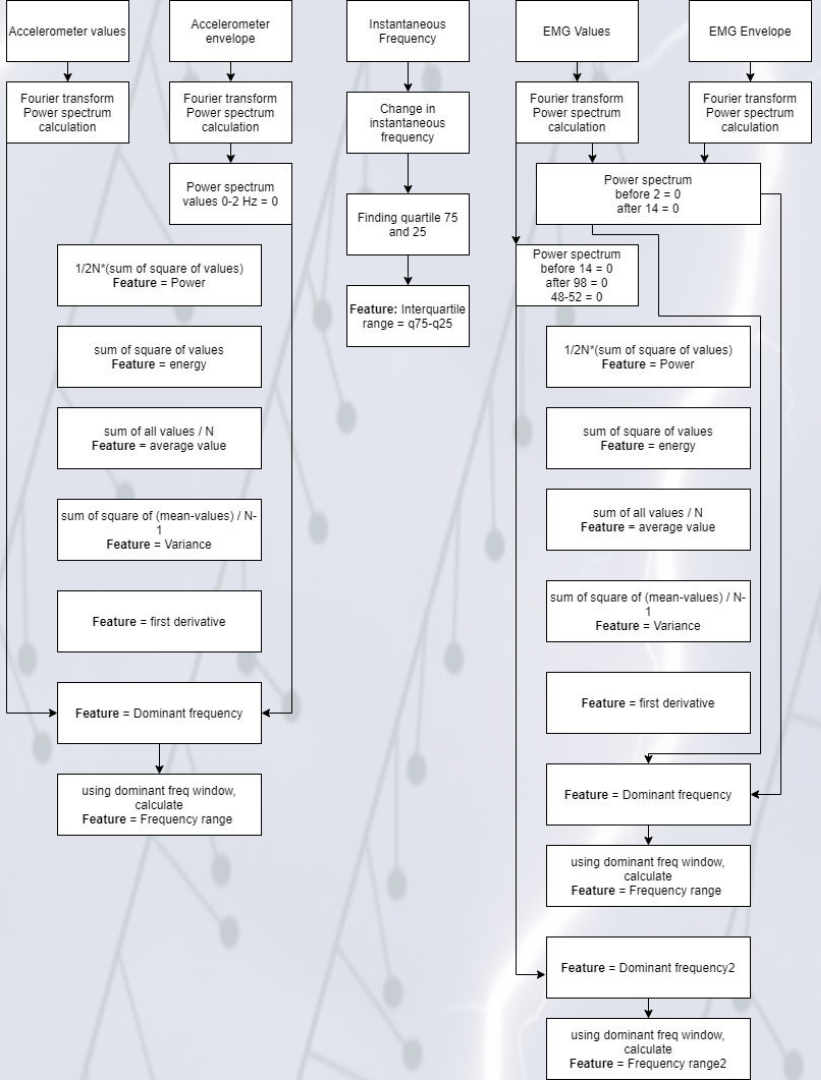


Feature Extraction

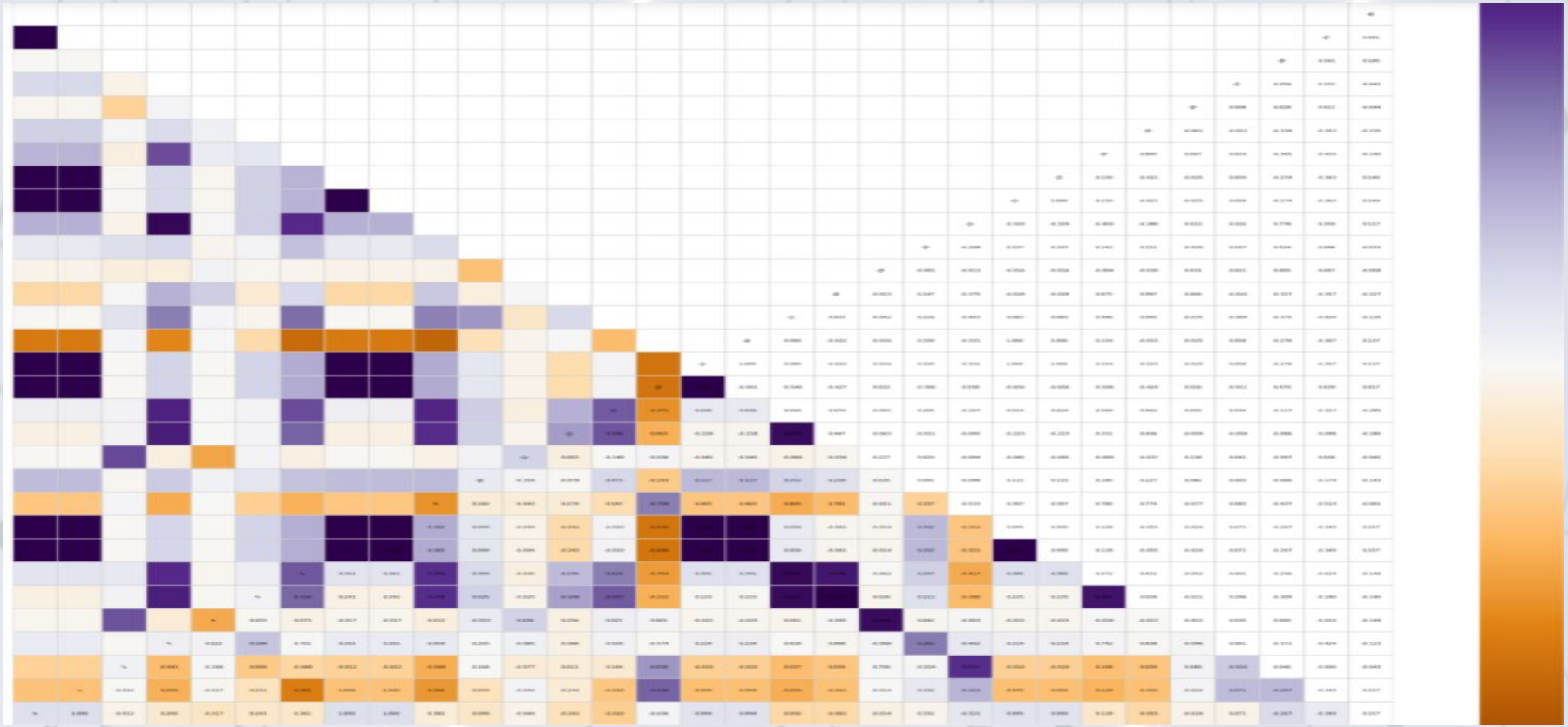
- Features are:
  - Power
  - Energy
  - Average value
  - Variance
  - First derivative
  - Dominant frequency
  - Frequency range
- 31 features in total
  - 7 of Accelerometer values
  - 7 of Accelerometer Envelope
  - 1 of Tremor Stability Index
  - 9 of EMG values
  - 7 of EMG Envelope
- Computed for each of the 5 second instance

The features have correlations with each other

Here N is the number of values in the 5 second interval



Correlations among features



Clustering Algorithm

As it is unsupervised data, we do not know the actual clusters possible.

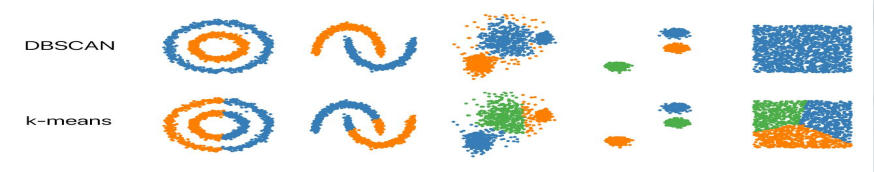
DBScan:

- Based on core point, border point and noise point
- No need to tell the number of clusters (like KMeans) + detects outliers too
- But challenge is to find right hyperparameters **eps** and **min_samples values**
    - **Eps (epsilon distance):** minimum distance between samples to be labeled as same cluster
    - **Min_samples:** minimum number of points to make a cluster to be considered one not outliers

Finding the optimal values:

- Now using K-means algorithm along with Silhouette distance to find the optimal number of clusters.
- Using that optimal cluster number to find minimum distance(**eps**) b\w points
- then using that distance and optimal cluster number to find **min_samples** parameter of DBScan

Source:
https://medium.com/@mohantysandip/a-step-by-step-approach-to-solve-dbscan-algorithms-by-tuning-its-hyper-parameters-93e693a91289



Results and Validation

Simulated data 1 (ACC only) + without overlap

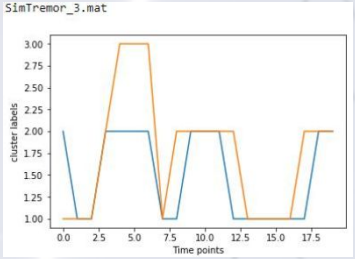
Simulated data obtained from model of tremor that contains different transitions and have different dynamics.

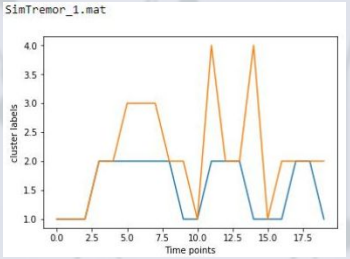
We use clustering to decode what the transitions are

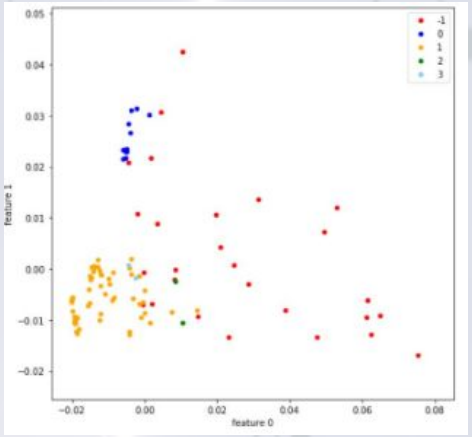
- Yellow = Actual
- Blue = Predicted

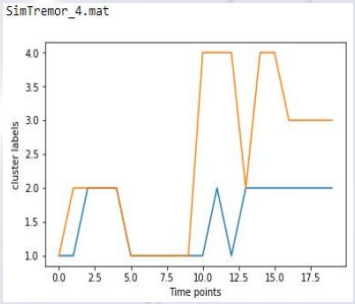
Source: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1009281

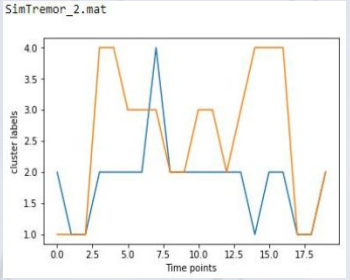
- States being close together and absorbing one another
- The difference between readings is not significant

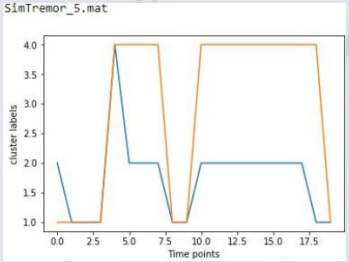












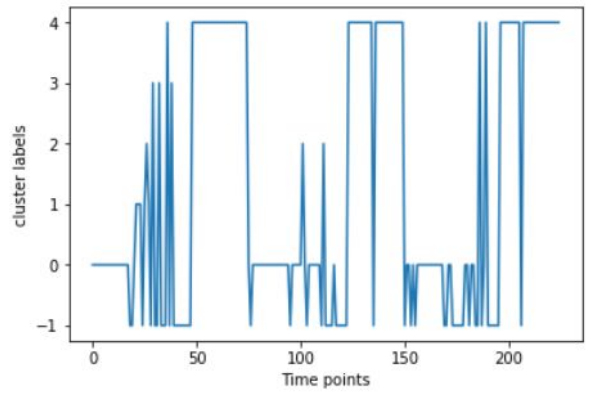

```
from sklearn.cluster import DBSCAN
clustering = DBSCAN(eps=0.011, min_samples=2).fit(norm_matrix)
cluster=clustering.labels_
print(len(set(cluster)))
unique, counts = np.unique(cluster, return_counts=True)
dict(zip(unique, counts))
```

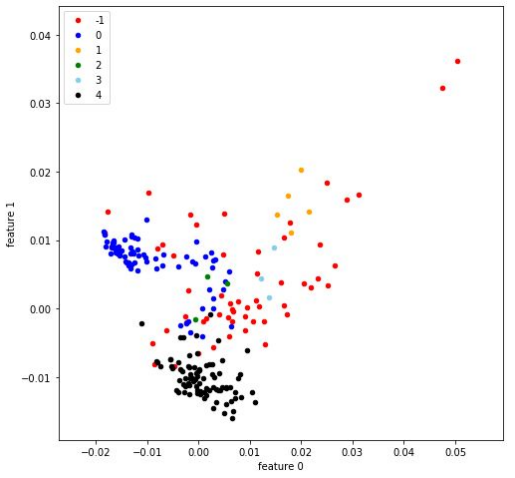
```
5
{-1: 27, 0: 11, 1: 58, 2: 2, 3: 2}
```

One Person All Axis (ACC+EMG) (without overlap)

Using one person, all axis.

- Optimal clusters = 4
- Min_points = 4
- Eps = 0.01031
- 18 = EMG variance
- 10 = ACC_env variance
- CPU time to compute - ~1-2 mins






```
from sklearn.cluster import DBSCAN
clustering = DBSCAN(eps=0.01031, min_samples=4).fit(norm_matrix)
cluster=clustering.labels_
print(len(set(cluster)))
unique, counts = np.unique(cluster, return_counts=True)
dict(zip(unique, counts))
```

```
6

{-1: 56, 0: 74, 1: 5, 2: 3, 3: 3, 4: 84}
```

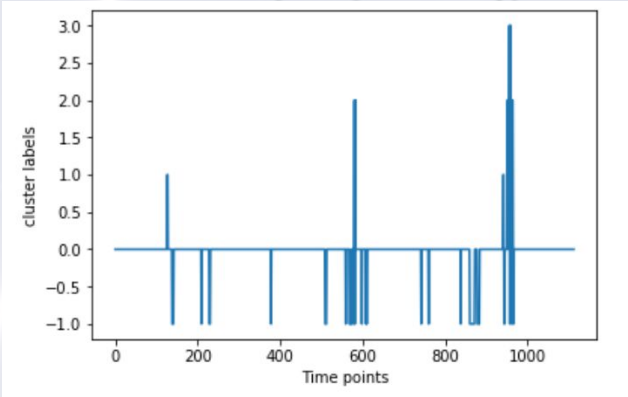
```
print(pca.explained_variance_ratio_)
# number of components
n_pcs= pca.components_.shape[0]
most_important = [np.abs(pca.components_[:][i]).argmax() for i in range(n_pcs)]
print(most_important)
```

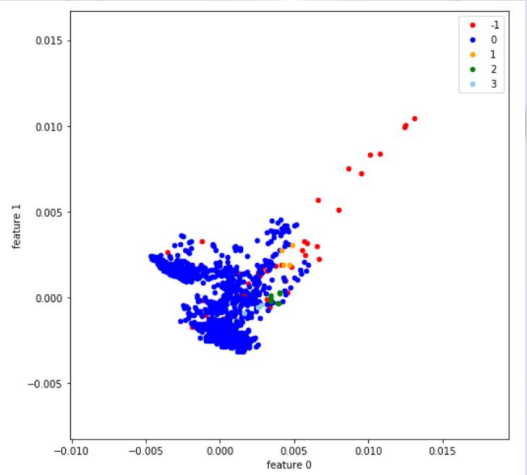
```
[0.32240092 0.22684091 0.19854599 0.06516051 0.03687473 0.03654934
 0.02119217 0.01952025 0.01768636 0.01105906]
[18, 18, 10, 3, 13, 13, 23, 14, 23, 6]
```

One Person All Axis (ACC+EMG) (with overlap)

Using one person, all axis.

- Optimal clusters = 4
- Min_points = 4
- Eps = 0.0023000000000000004
- 18 = EMG variance
- 10 = ACC_env variance



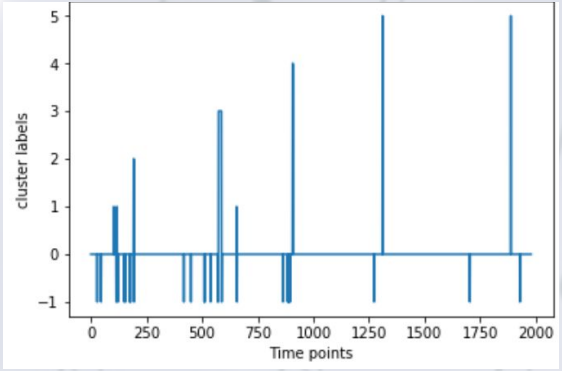


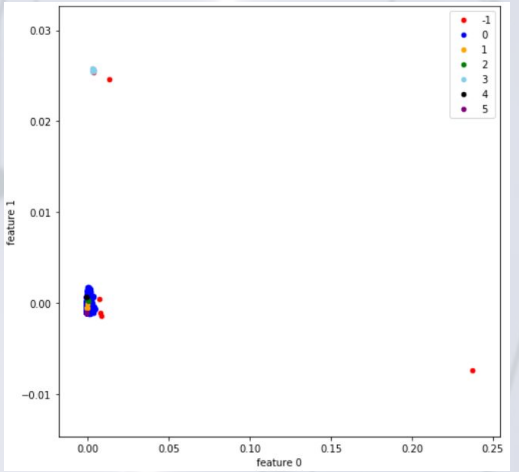

```
print(pca.explained_variance_ratio_)
# number of components
n_pcs= pca.components_.shape[0]
most_important = [np.abs(pca.components_[:][i]).argmax() for i in range(n_pcs)]
print(most_important)
```

```
[0.32259502 0.22853658 0.16328989 0.07447037 0.04350702 0.03677964
 0.03058722 0.02575668 0.01964407 0.01244224]
[18, 18, 10, 30, 3, 13, 23, 6, 25, 14]
```

All Person All Axis (ACC+EMG) Beatriz Dataset without overlap

- Optimal clusters = 2
- But here I put 5 to show that if we change number of clusters then results change
- 18 = EMG variance
- 24 = EMG_envelope power






```
print(pca.explained_variance_ratio_)
# number of components
n_pcs= pca.components_.shape[0]
most_important = [np.abs(pca.components_[:][i]).argmax() for i in range(n_pcs)]
print(most_important)
```

```
[0.76555548 0.15087007 0.01736329 0.01710454]

[18, 24, 10, 21]
```

Modular code

- Modification of code to make it more user friendly
- For using all these functions:
  - python main.py
- For loading data:
  - import preprocessing_helper
  - Data = preprocessing_helper.ThreeAxisACC(directory)
- Similarly can run for 1-axis, 3-axis Acc or EMG with or without sliding window of any length

feature_extraction.py

```
import matplotlib.pyplot as plt
# ^^^ pyforest auto-imports - don't write above this line
import numpy as np
import scipy.io
from scipy.fftpack import fft
from scipy import signal
from scipy.signal import hilbert
from matplotlib import rcParams

def TSI_feature(acc_instFreq):...

def dom_freq_with_range(power_spectrum1, frequency):...

def featureACC(data, cover, sampling_rate):...

def featureEMG(data, cover,sampling_rate):...
```

preprocessing_helper.py

```
import statistics
import numpy as np
import os
import scipy.io
import pandas as pd
import math
from scipy import signal
from scipy.signal import hilbert
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import KMeans, DBSCAN

def testing_preprocessing(name,fs):...

def z_score(df):...

#JUST FOR PUTTING THE RANGE OF NUMBERS
def seq(start, stop, step=1):...

def AllPersonAllaxis(acc_withoutEnv):...

def FirstPersonFirstaxis(acc_withoutEnv):...

def ACC_preprocessing_helper(df_cars, criterian,fs):...

def sliding_ACC_preprocessing_helper(df_cars, criterian, fs, seconds):...

def ThreeAxisACC(directory_acc,sampling_freq,sliding=False,seconds=1):...

def OneAxisACC(directory_acc,sampling_freq):...

def butter_highpass(cutoff, fs, order=2):...

def butter_highpass_filter(data, cutoff, fs, order=2):...

def butter_lowpass(cutoff, fs, order=2):...

def butter_lowpass_filter(data, cutoff, fs, order=2):...

def sliding_EMG_preprocessing_helper(sine, criterian, fs, seconds):...

def EMG_preprocessing_helper(sine, criterian, fs):...

def ThreeAxisEMG(directory_emg, sampling_freq, sliding=False, seconds=1):...

def optimal_cluster_value(norm_matrix):...

def optimal_eps_value(norm_matrix,n_clusters):...

def DB_cluster(norm_matrix,eps_finalVal,cluster_final):...

def nnormalize(chek):...
```

Windows Defender configuration updated

Future work / Other points

Todo tasks:

- Clustering on Beatriz dataset with overlap
- Clustering on Carolina's tremor data - posture and spiral drawing - known "transitions"
- Validation on simulated data 1(with overlap) and 2(with and without overlap)
- See the feature importance using Univariate ANOVA
- Refine the code to be a pipeline usable for everyone (GITHUB)

Feature extraction using Deep learning:

- Regarding transfer learning:
  - Pretrained CNNs (that have been trained on real images) will be much confused by your 2D reshaped data
  - Pretrained timenet (RNN) wouldn't have worked either because of the difference in domain
- Solution is to learn the representation from scratch, given "enough" data
- For starting we can use 1D convolutional autoencoders

Thank you for the opportunity !

Any Questions?