Experiments1_Ivan

October 17, 2023

[ ]:
```
import numpy as np
import pandas as pd
import sklearn.svm as svm
import sklearn.cluster as cl
import matplotlib.pyplot as plt
import itertools
from typing import Dict
import random
```

[ ]:
```
# read in files
def load_dataset(file: str) -> Dict:

    items = ["RawData",
             "IAV-M_NEG_RawData",
             "IAV-M_POS_RawData",
             "IBV-M_NEG_RawData",
             "IBV-M_POS_RawData",
             "MHV_NEG_RawData",
             "MHV_POS_RawData",
             "RSV-N_NEG_RawData",
             "RSV-N_POS_RawData",
             "SARS-N1_NEG_RawData",
             "SARS-N1_POS_RawData",
             "SARS-N2_NEG_RawData",
             "SARS-N2_POS_RawData",
            ]
    path_map = {}
    df_all_map = {}
    for name in items:
        path_map[name] = file + f'{name}.csv'
        df_all_map[name] = pd.read_csv(path_map[name])

    return df_all_map
```

[ ]:
```
#Using Gernots plot to get the overview of the data and the clusters
def pairwise_plots_dbscan(df, eps, min_samples = 5):
    np_features = df.drop(["x-coordinate_in_pixel"," y-coordinate_in_pixel","␣
 ↪index"], axis=1)
```

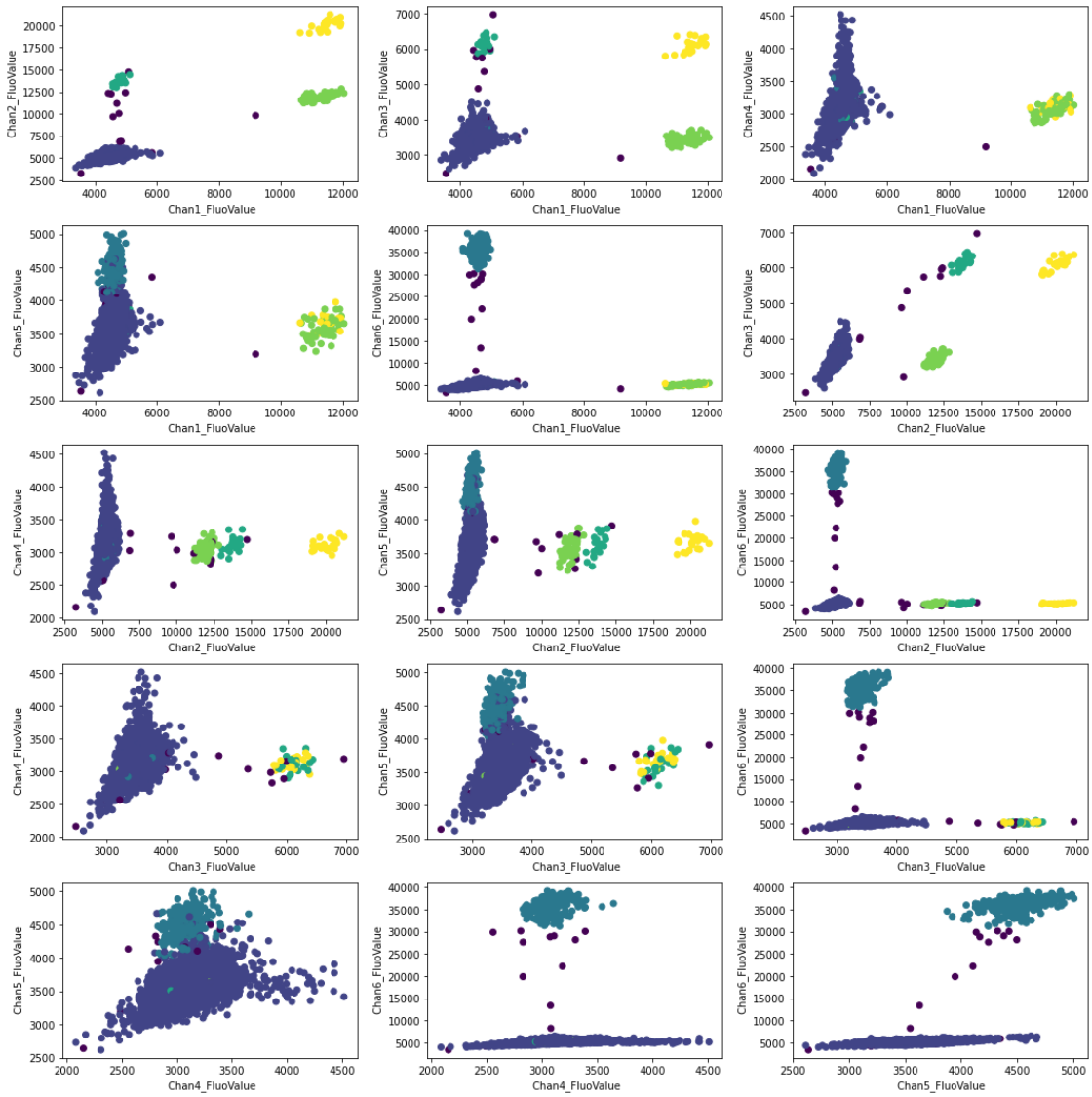
```
    np_features = np_features.to_numpy()
    classifier = cl.DBSCAN(eps = eps, min_samples = min_samples)
    preds = classifier.fit_predict(np_features)


    print(f"Number of outliers: {len([x for x in preds if x == -1])}") # print␣
↪number of outliers
    print(f"Number of clusters: {max(preds)+1}") # print number of clusters


    combinations = itertools.combinations(df.columns[2:-1], 2)
    fig, ax = plt.subplots(5, 3, sharex=False, sharey=False)
    fig.set_figheight(15)
    fig.set_figwidth(15)
    for i, combination in enumerate(combinations):
        np_features = df.loc[:, combination]
        np_features = np_features.to_numpy()

        ax[i //3, i %3].set_xlabel(combination[0])
        ax[i //3, i %3].set_ylabel(combination[1])
        ax[i //3, i %3].scatter(np_features[:, 0], np_features[:, 1], c = preds)
    fig.tight_layout()
```

0.1 A3 data set

Identifying clusters on the **A3 data set**

[ ]:
```
file = "../Data/6P-wastewater-samples-labelled/droplet-level-data/RawData/
↪6P-wastewater-samples-labelled_S-0697825-9_A3_"
df = load_dataset(file)["RawData"]
pairwise_plots_dbscan(df, eps = 700, min_samples = 5)
```

```
Number of outliers: 23
Number of clusters: 5
```



0.2 Finding the control cluster

The **find_zero_cluster** function identifies the cluster (given the set of a labelled data) with the smallest L2 norm. This should correspond to the control sample from the experiment.

```
[ ]: def find_zero_cluster(data : np.ndarray, labels : np.ndarray, n_clusters : int)␣
     ↪-> int:
         N = data.shape[0]

         #computing the size and norm of each cluster
         cluster_norm = np.zeros(n_clusters, float)
         cluster_size = np.zeros(n_clusters, int)
         for i in range(0, N):
```

```
        cluster_size[labels[i]] += 1
        cluster_norm[labels[i]] += np.linalg.norm(data[i, :])

    #finding the cluster with the smallest average norm
    avg_norms = np.divide(cluster_norm, cluster_size)

    return np.argmin(avg_norms)
```

0.3 Labelling clusters with respect to the control cluster

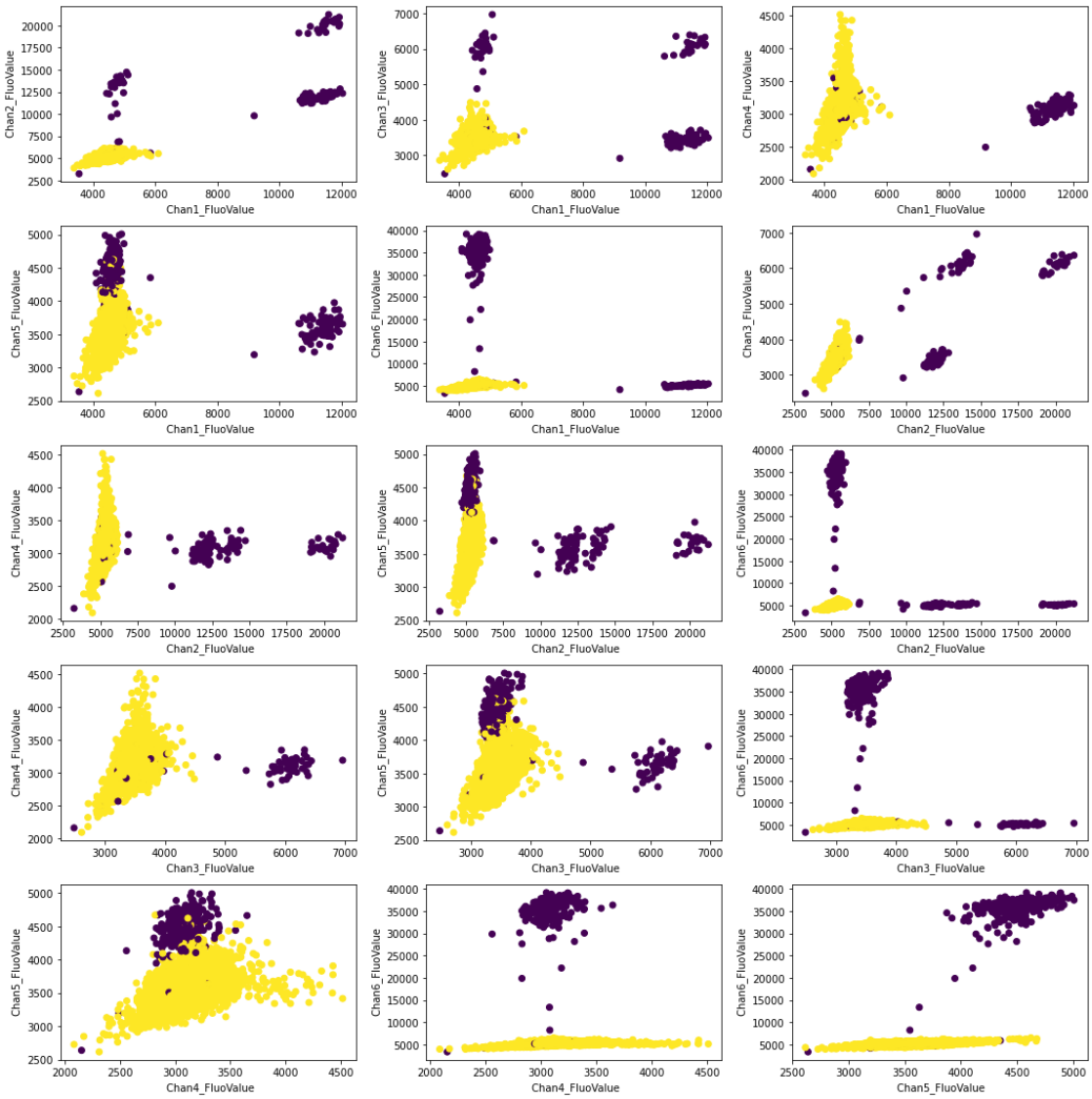
The **get_labelling** function compares a cluster with a zero cluster along each axis and selects the coordinates in which it is active

[ ]:
```
def get_labelling(zero_cluster : np.ndarray, my_cluster : np.ndarray) ->␣
↪list[bool]:
    dim = my_cluster.shape[1]
    zero_size = zero_cluster.shape[0]
    other_size = my_cluster.shape[0]
    active = []
    for i in range(0, dim):
        #set up the classification framework
        X = np.concatenate((zero_cluster, my_cluster))

        #true labels are both of our clusters concatenated
        true_labels = np.array([0]*zero_size + [1]*other_size)

        #run 1D svm
        clf = svm.SVC(kernel='linear', C=1.0)
        clf.fit(X, true_labels)

        #test how good the fit was
        score = clf.score(X, true_labels)
        #if SVM was good at separating our clusters then one of them is active␣
↪in the coordinate
        if score < 1:
            active.append(1)
        else:
            active.append(0)
    return active
```

0.4 Testing on the A3 data set

Testing the performance of the **find_zero_cluster** function. Plotting the zero cluster along all coordinates together with all the other clusters for comparasing

[ ]:
```
data = df.drop(["x-coordinate_in_pixel"," y-coordinate_in_pixel"," index"],␣
 ↪axis=1)
data = data.to_numpy()
classifier = cl.DBSCAN(eps = 700, min_samples = 5)
labels = classifier.fit_predict(data)

zero_cluster_id = find_zero_cluster(data, labels, max(labels) + 1)
is_zero = lambda x: x == zero_cluster_id
zero_cluster_mask =  is_zero(labels)

#using Gernots plotting code
combinations = itertools.combinations(df.columns[2:-1], 2)
fig, ax = plt.subplots(5, 3, sharex=False, sharey=False)
fig.set_figheight(15)
fig.set_figwidth(15)
for i, combination in enumerate(combinations):
    np_features = df.loc[:, combination]
    np_features = np_features.to_numpy()

    ax[i //3, i %3].set_xlabel(combination[0])
    ax[i //3, i %3].set_ylabel(combination[1])
    ax[i //3, i %3].scatter(np_features[:, 0], np_features[:, 1], c =␣
 ↪zero_cluster_mask)
fig.tight_layout()
```



Testing the performance of the **get_labelling** function on the A3 data set

[ ]:
```
#looking at the random cluster
my_cluster_id = random.randint(0, max(labels))

#isolating the clusters
zero_cluster = data
my_cluster = data

for i in range(0, data.shape[0]):
    if labels[i] != zero_cluster_id:
        np.delete(zero_cluster, i, 0)
if labels[i] != my_cluster_id:
```

```
        np.delete(my_cluster, i, 0)


# get the labels through the labelling routine
active = get_labelling(zero_cluster, my_cluster)

#only keep these two clusters
two_clusters = np.concatenate((zero_cluster, my_cluster))
two_labels = np.array([0] * zero_cluster.size() + [1] * zero_cluster.size())

#using Gernots plotting code
combinations = itertools.combinations(df.columns[2:-1], 2)
fig, ax = plt.subplots(5, 3, sharex=False, sharey=False)
fig.set_figheight(15)
fig.set_figwidth(15)
for i, combination in enumerate(combinations):
    np_features = df.loc[:, combination]
    np_features = np_features.to_numpy()

    ax[i //3, i %3].set_xlabel(combination[0])
    ax[i //3, i %3].set_ylabel(combination[1])
    ax[i //3, i %3].scatter(np_features[:, 0], np_features[:, 1], c =␣
 ↪zero_cluster_mask)
fig.tight_layout()
```

[ ]: