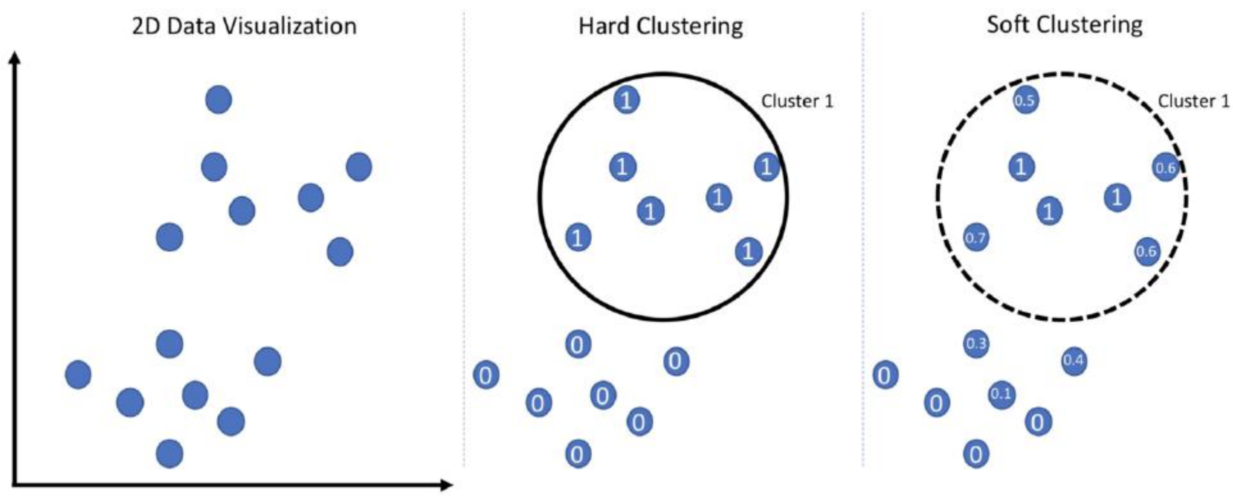
Project 3 on Fuzzy system

Subject: Fuzzy C-mean clustering

Name: Hesam Mousavi

Student number: 9931155

Master student



In [1]:

```
import numpy as np
from fcmeans import FCM
from my_io import read_dataset_to_X_and_y
from copy import deepcopy
import matplotlib.pyplot as plt
```

Build Class to easily have all the variables

I like to have my variable all together so I build a class and named it UniSet(short form of universal set)

Read dataset with my function on my_io module that can shuffle sample and correct missing values also normalized the feature.

In here I shuffle data and use class-mean for the missing values then normalized it with the z-score method(zero-mean unit-variance)

I use all the features(12) and change sex from m, f to 0, 1 (actually I map each string to a specific number in my_io module)

In [2]:

```
class UniSet():
    def __init__(self, file, range_feature, range_label,
                 normalization='scaling', min_value=0.1, max_value=1,
```

```python
                  shuffle=False, about_nan='class_mean'):
        np.random.seed(1)
        sample, label = read_dataset_to_X_and_y(
            file, range_feature, range_label, normalization, min_value, max_valu
            shuffle=shuffle, about_nan=about_nan)
        self.universal = sample.astype(float)
        self.label = label
        self.number_of_feature = sample.shape[1]
        self.size_of_universal = sample.shape[0]
        self.diffrent_label = np.unique(label)
        self.number_of_diffrent_label = self.diffrent_label.shape[0]


uni_total = UniSet(
    'dataset/hcvdat0.csv', (2, 14), (1, 2),
    normalization='scaling', shuffle=True, about_nan='class_mean')


print(f'The whole dataset is {uni_total.universal.shape} matrix')
```

```
The whole dataset is (615, 12) matrix
```

> ### Details
>
> In my_io module I have a function named read_dataset_to_X_and_y that get dataset file, range of attributes that are our features, range of attributes that are our labels, normalization which is our normalization method, shuffle which if be True our samples be shuffled, and about_nan that can be "delete" which delete samples with NA values or "class_mean" which replace NA values with mean of that feature in the sample class
>
> Also as I mentioned above this function can get string attributes too by mapping each string to a specific value so now our labels $\in [0, 4]$
>
> I change NA value with class-mean because It doesn't change the similarity(or distance) of two samples in one class
>
> In my class, I have all things that I'll need such as universal (sample data), their label, number of features, size of universal (dataset), different labels (unique labels), and number of different labels.
>
> Our labels in this dataset is attributed [1, 2) and features are attributed [2, 14) (12 features)

Split the whole dataset to Train and Test

As I shuffle the dataset before, now I just consider the first 80% of the data for the train and the rest for the test case

In [3]:

```python
def split_train_test(universe: UniSet, train_size: float) -> list[UniSet]:
    train = deepcopy(universe)
    test = deepcopy(universe)
```

```
    train.size_of_universal = \
        int(universe.size_of_universal*train_size)
    train.universal = \
        universe.universal[0:train.size_of_universal]
    train.label = \
        universe.label[0:train.size_of_universal]
    test.size_of_universal = (
        universe.size_of_universal - train.size_of_universal)
    test.universal = \
        universe.universal[train.size_of_universal:]
    test.label = \
        universe.label[train.size_of_universal:]

    return train, test


uni_train, uni_test = split_train_test(uni_total, 0.8)
print(f'The train dataset is {uni_train.universal.shape} matrix')
print(f'The test dataset is {uni_test.universal.shape} matrix')
```

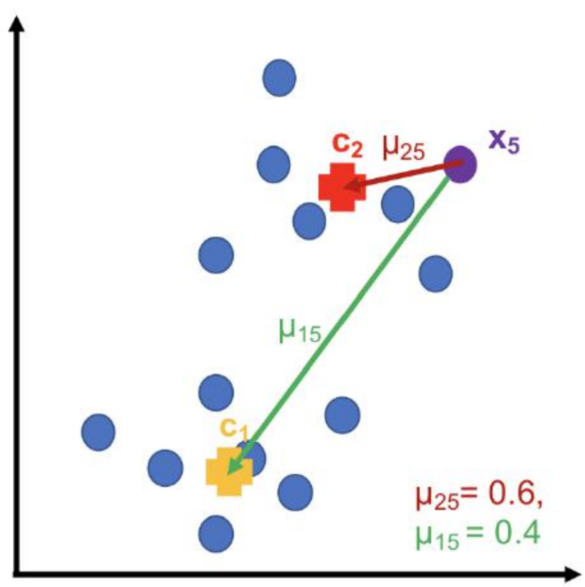
```
The train dataset is (492, 12) matrix
The test dataset is (123, 12) matrix
```

> Details
>
> I create two classes for train and test by copying the total set and just changing universal, level, and size of universal for both train and test

Our parameters

$\mu_{i,j}$: the probability that the jth data point belongs to the ith cluster which the sum of $\mu_{i,j}$ over C cluster centers is 1 for every data point j

c_i: the center of the ith cluster



Objective function

$$J = \sum_{i=1}^{C} \sum_{j=1}^{N} \mu_{ij}^{m} \|x_j - c_i\|^2$$

Accuracy

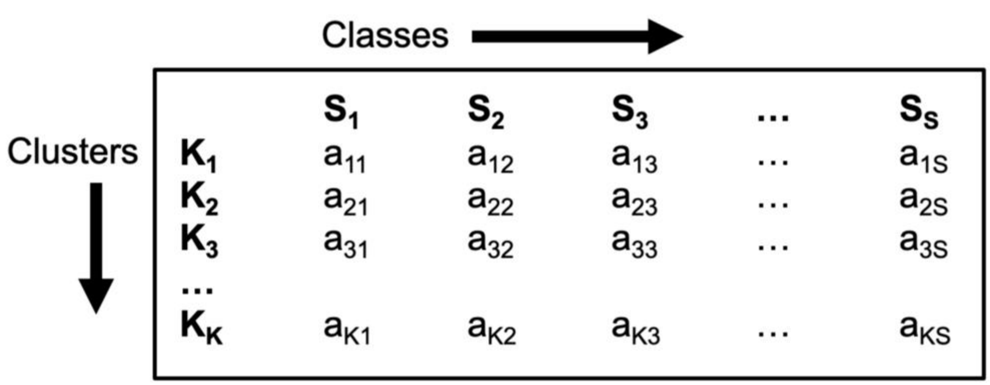
I use **Confusion matrix** to find the label of clusters (argmax in each row) and then choose f1-score as accuracy metric because as α increase precision increase and recall decrease and I want to find α that satisfy both

In [4]:

```
def evaluate(gold_label: np.ndarray, predict_label: np.ndarray,
             method: str = 'f1-score') -> float:
    diffrent_label_in_gold_label = np.unique(gold_label)
    diffrent_label_in_predict_label = np.unique(predict_label)
    confusion_matrix = np.array(
        list(map(lambda k: list(map(
            lambda s: sum((predict_label == k)*(gold_label == s))[0],
            diffrent_label_in_gold_label)),
            diffrent_label_in_predict_label)))
    precision = np.sum(
        np.max(confusion_matrix, axis=1)) / np.sum(confusion_matrix)
    recall = np.sum(
        np.max(confusion_matrix, axis=0)) / np.sum(confusion_matrix)
    if(method == 'precision'):
        return precision
    if(method == 'recall'):
        return recall
    if(method == 'f1-score'):
        return 2 * ((precision*recall)/(precision+recall))
```

Details

Confusion matrix



Its $(K * S)$ matrix that $a_{k,s}$ = total number of samples clustered to the k^{th} cluster and belongs to the s^{th} class.

$$\text{Precision} = \frac{\sum_k \max_s \{a_{ks}\}}{\sum_k \sum_s a_{ks}}$$

$$\text{Recall} = \frac{\sum_s \max_k \{a_{ks}\}}{(\sum_k \sum_s a_{ks} + U)}$$

$$F1 - score = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Fit module

Using train samples (without labels) as data points and number of clusters equal to number of diffrent label we have

In [5]:

```
fcm = FCM(n_clusters=uni_train.number_of_diffrent_label)
fcm.fit(uni_train.universal)
```

Predict labels

Now we use trained module on test data to find their labels by first, find the probability for each sample belonging to each cluster then finding the most probable cluster as sample cluster, and at the end using a confusion matrix to relabel our cluster to their actual labels

In [6]:

```
fcm_centers = fcm.centers
print(f'The centers is {fcm.centers.shape} matrix')

fcm_mu = fcm.soft_predict(uni_test.universal)
print(f'The μ is {fcm_mu.shape} matrix')

fcm_predict_labels = np.argmax(fcm_mu, axis=1).reshape((-1,1))
print(f'The predicted label is {fcm_predict_labels.shape} matrix')

accuracy = evaluate(uni_test.label, fcm_predict_labels, 'precision')
print(f'Our accuracy on test data is {np.round(accuracy*100, 2)}%')
```

```
The centers is (5, 12) matrix
The μ is (123, 5) matrix
The predicted label is (123, 1) matrix
Our accuracy on test data is 88.62%
```

> ### Details
>
> As I mentioned in the parameter section our μ here is a matrix (sample, cluster) in which each row is the probability that our sample belongs to each cluster so the sum of each row is equal to 1
>
> Our predicted label here is a cluster that our module created in the training phase and their label names are different from our actual labels name so we use a confusion matrix to relabel them to be able to find our accuracy

Thanks for your time