Cluster analysis or clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). Clustering is not one specific algorithm, but the task of grouping data. This can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Out of 100 published clustering algorithms there are common clustering algorithm,

Hard Clustering:

Connectivity-based clustering (Hierarchical Clustering) Centroid-based Clustering (k-means clustering) Distribution-based Clustering Density-based Clustering

Soft Clustering:

Fuzzy Clustering

Fuzzy Clustering

In **Fuzzy clustering** each element has a set of membership coefficients corresponding to the degree of being in a given cluster. Points close to the center of a cluster, may be in the cluster to a higher degree than points in the edge of a cluster. The degree, to which an element belongs to a given cluster, is a numerical value varying from 0 to 1.I

Fuzzy clustering algorithms seeks to **minimize cluster memberships** and distances, but we will focus on Fuzzy C-Means Clustering algorithm.

Fuzzy c-means developed in 1973 and improved in 1981. It's very similar to k-means algorithm in a structure way:

1.Choose number of clusters.

2.Assign coefficients randomly to each data point for being in the clusters.

3.Repeat until algorithm converged (Objective Function C minimizes cluster memberships and distances):

4.Compute the centroid for each cluster

5.Compute each data points' coefficients of being in the clusters.

The main difference with k-means cluster is that **objective function for fuzzy c-means algorithm** allows different cluster membership with probability values, where k-means cluster has strict objective function allows only one cluster membership.

Importing Libraries

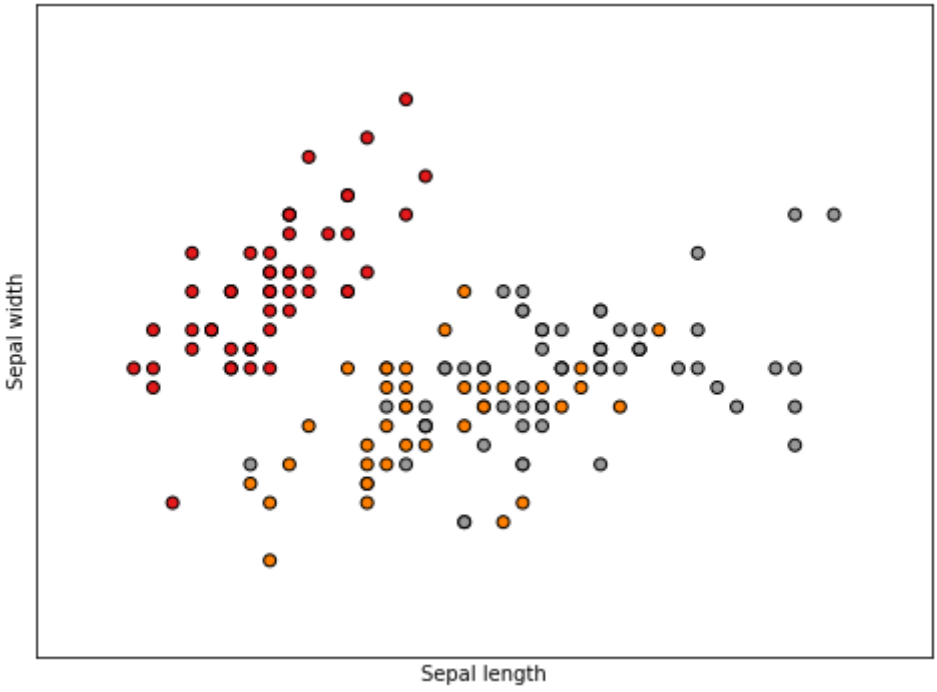
```
import pandas as pd # reading all required header files
import numpy as np
import random
import operator
import math
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from sklearn import datasets
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
```

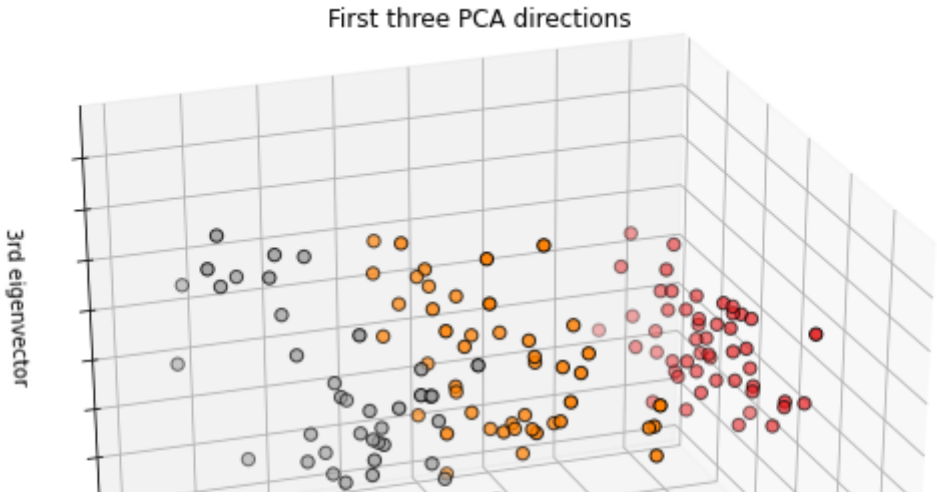
```
# import some data to play with
iris = datasets.load_iris()
```

```
iris
```

```
       [7.6, 3. , 6.6, 2.1],
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
       [6.5, 3.2, 5.1, 2. ],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]]),
'feature_names': ['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)'],
'filename': '/usr/local/lib/python3.6/dist-packages/sklearn/datasets/data/iris.csv',
'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10')}
```

```
X = iris.data[:, :2]  # we only take the first two features.
y = iris.target

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

plt.figure(2, figsize=(8, 6))
plt.clf()

# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1,
            edgecolor='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())

# To getter a better understanding of interaction of the dimensions
# plot the first three PCA dimensions
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
X_reduced = PCA(n_components=3).fit_transform(iris.data)
ax.scatter(X_reduced[:, 0], X_reduced[:, 1], X_reduced[:, 2], c=y,
           cmap=plt.cm.Set1, edgecolor='k', s=40)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])

plt.show()
```





Installing package fuzzy-c-means

```
1 !pip install fuzzy-c-means
```

```
Collecting fuzzy-c-means
  Downloading https://files.pythonhosted.org/packages/99/fa/55219e166bb52dd7ed8e35d74
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.6/dist-package
Installing collected packages: fuzzy-c-means
Successfully installed fuzzy-c-means-0.0.6
```

Importing Libraries

```
1 #https://pypi.org/project/fuzzy-c-means/
2 from fcmeans import FCM
3 from sklearn.datasets import make_blobs
```

```
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt
from seaborn import scatterplot as scatter
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnir
  import pandas.util.testing as tm
```

Create artificial dataset

```
#create artifitial dataset
n_samples = 50000
n_bins = 3  # use 3 bins for calibration_curve as we have 3 clusters here
centers = [(-5, -5), (0, 0), (5, 5)]
```

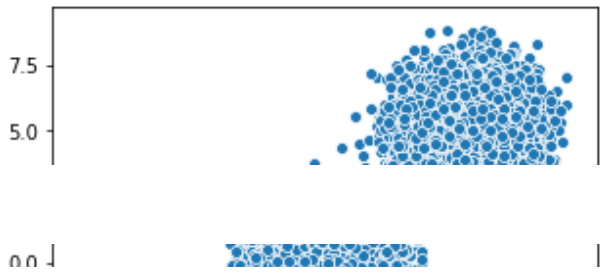
Applying Fuzzy C-means

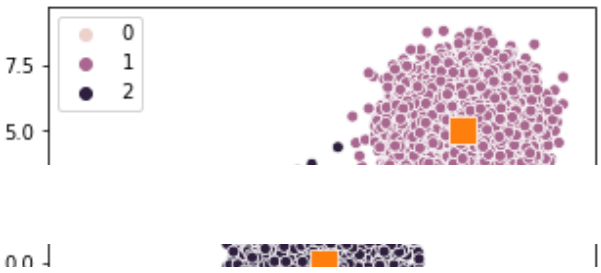
```
X,_ = make_blobs(n_samples=n_samples, n_features=2, cluster_std=1.0,
                  centers=centers, shuffle=False, random_state=42)

# fit the fuzzy-c-means
fcm = FCM(n_clusters=3)
fcm.fit(X)
```

```
<fcmeans.fcm.FCM at 0x7f9f79fe79e8>
```

```
# outputs
fcm_centers = fcm.centers
fcm_labels  = fcm.u.argmax(axis=1)
```

```
# plot result
%matplotlib inline
f, axes = plt.subplots(1, 2, figsize=(11,5))
scatter(X[:,0], X[:,1], ax=axes[0])
scatter(X[:,0], X[:,1], ax=axes[1], hue=fcm_labels)
scatter(fcm_centers[:,0], fcm_centers[:,1], ax=axes[1],marker="s",s=200)
plt.show()
```






```
1
```

```
1 from google.colab import drive
2 drive.mount("/content/drive")
```


```
1 import pandas as pd
2 import numpy as np
3 import random
4 import operator
5 import math
6
7
8 df_full = pd.read_csv("/content/drive/My Drive/Colab Notebooks/clustering/SPECTF_N
9 columns = list(df_full.columns)
10 features = columns[:len(columns)-1]
11 class_labels = list(df_full[columns[-1]])
12 df = df_full[features]
13
14 # Number of Attributes
15 num_attr = len(df.columns) - 1
16
17 # Number of Clusters
18 k = 2
19
20 # Maximum number of iterations
21 MAX_ITER = 1000
22
23 # Number of data points
24 n = len(df)
25
26 # Fuzzy parameter
27 m = 2.00
28
29 def accuracy(cluster_labels, class_labels):
30     county = [0,0]
31     countn = [0,0]
32     tp = [0, 0]
33     tn = [0, 0]
34     fp = [0, 0]
35     fn = [0, 0]
36
37     for i in range(len(df)):
38         # Yes = 1, No = 0
39         if cluster_labels[i] == 1 and class_labels[i] == 'Yes':
40             tp[0] = tp[0] + 1
41         if cluster_labels[i] == 0 and class_labels[i] == 'No':
```

```
            tn[0] = tn[0] + 1
        if cluster_labels[i] == 1 and class_labels[i] == 'No':
            fp[0] = fp[0] + 1
        if cluster_labels[i] == 0 and class_labels[i] == 'Yes':
            fn[0] = fn[0] + 1

    for i in range(len(df)):
        # Yes = 0, No = 1
        if cluster_labels[i] == 0 and class_labels[i] == 'Yes':
            tp[1] = tp[1] + 1
        if cluster_labels[i] == 1 and class_labels[i] == 'No':
            tn[1] = tn[1] + 1
        if cluster_labels[i] == 0 and class_labels[i] == 'No':
            fp[1] = fp[1] + 1
        if cluster_labels[i] == 1 and class_labels[i] == 'Yes':
            fn[1] = fn[1] + 1

    a0 = float((tp[0] + tn[0]))/(tp[0] + tn[0] + fn[0] + fp[0])
    a1 = float((tp[1] + tn[1]))/(tp[1] + tn[1] + fn[1] + fp[1])
    p0 = float(tp[0])/(tp[0] + fp[0])
    try:
        p1= float(tp[1])/(tp[1] + fp[1])
    except ZeroDivisionError:
        print ('0')
        p1=0
    r0 = float(tp[0])/(tp[0] + fn[0])
    r1 = float(tp[1])/(tp[1] + fn[1])

    accuracy = [a0*100,a1*100]
    precision = [p0*100,p1*100]
    recall = [r0*100,r1*100]

    return accuracy, precision, recall


def initializeMembershipMatrix():
    membership_mat = list()
    for i in range(n):
        random_num_list = [random.random() for i in range(k)]
        summation = sum(random_num_list)
        temp_list = [x/summation for x in random_num_list]
        membership_mat.append(temp_list)
    return membership_mat


def calculateClusterCenter(membership_mat):
    cluster_centers = list(zip(*membership_mat))
    #cluster_centers = list()
    for j in range(k):
        x = list(cluster_centers[j])
        xraised = [e ** m for e in x]
        denominator = sum(xraised)
        temp_num = list()
        for i in range(n):
            data_point = list(df.iloc[i])
            prod = [xraised[i] * val for val in data_point]
```

```
            prod = [xraised[i] * val for val in data_point]
            temp_num.append(prod)
        numerator = map(sum, zip(*temp_num))
        center = [z/denominator for z in numerator]
        cluster_centers.append(center)
    return cluster_centers


def updateMembershipValue(membership_mat, cluster_centers):
    p = float(2/(m-1))
    for i in range(n):
        x = list(df.iloc[i])
        distances = [np.linalg.norm(list(map(operator.sub, x, cluster_centers[j])))
        for j in range(k):
            den = sum([math.pow(float(distances[j]/distances[c]), p) for c in rang
            membership_mat[i][j] = float(1/den)
    return membership_mat


def getClusters(membership_mat):
    cluster_labels = list()
    for i in range(n):
        max_val, idx = max((val, idx) for (idx, val) in enumerate(membership_mat[i
        cluster_labels.append(idx)
    return cluster_labels


def fuzzyCMeansClustering():
    # Membership Matrix
    membership_mat = initializeMembershipMatrix()
    curr = 0
    while curr <= MAX_ITER:
        cluster_centers = calculateClusterCenter(membership_mat)
        membership_mat = updateMembershipValue(membership_mat, cluster_centers)
        cluster_labels = getClusters(membership_mat)
        curr += 1
    print(membership_mat)
    return cluster_labels, cluster_centers


labels, centers = fuzzyCMeansClustering()
a,p,r = accuracy(labels, class_labels)

print("Accuracy = " + str(a))
print("Precision = " + str(p))
print("Recall = " + str(r))
```



```
1
```

```
import pandas as pd
import numpy as np
import random
import math

def eucledian_dist(a,b):
    ans = float(np.sqrt(np.sum((a-b)**2)))
    return ans

if __name__== '__main__':
    df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/clustering/SPECTF_Ne
    df = df.sample(frac=1)
    #print df.head()
    df = df.values
    print (df.shape)
    no_of_clusters = 2
    m = 2
    rows = random.sample(range(0,df.shape[0]),no_of_clusters)
    centers = []
    for i in range(no_of_clusters):
        centers.append(df[rows[i],:df.shape[1]-1])
    centers = np.array(centers,dtype=np.float)
    #print( centers
    membership_matrix = np.zeros((df.shape[0],no_of_clusters),dtype=np.float)

    delta = 0.01
    itr=0
    flag=1

    while itr<500 and flag==1:
        print( "-----------------------------------------------------------------
        for i in range(membership_matrix.shape[0]):
            for j in range(membership_matrix.shape[1]):
                numerator = eucledian_dist(df[i,:44],centers[j,:])
                if numerator==0:
                    membership_matrix[i,j] = 1
                    for k in range(j):
                        membership_matrix[i,k] = 0
                    break
                else:
                    sumratios = 0
                    for k in range(centers.shape[0]):
                        dist = eucledian_dist(df[i,:44],centers[k,:])
                        if dist==0:
                            ratio = 0
                        else:
                            ratio = float(numerator)/dist
                        ratio = math.pow(ratio,2.0/(m-1))
                        sumratios += ratio
                    membership_matrix[i,j] = 1/float(sumratios)

```

```
        '''
        for i in range(membership_matrix.shape[0]):
            print( np.sum(membership_matrix[i])
        '''
        #print( centers
        flag=0
        for i in range(centers.shape[0]):
            numerator=np.zeros(44)
            denominator=0
            for j in range(df.shape[0]):
                #print( df[j,:44].shape
                numerator = numerator + (df[j,:44] * math.pow(membership_matrix[j,
                denominator += math.pow(membership_matrix[j,i],m)
            if all(i<delta for i in centers[i]-(numerator/denominator)) is False:
                flag=1
            centers[i] = numerator/denominator
        itr+=1
        print( membership_matrix)
        print( itr)
        print( "----------------------------------------------------------------------
```