Clustering

I. Introduction

Clustering analysis is a unsupervised learning technique that is applied on an unlabelled database to identify sets of objects that belong to groups of similar properties or interests. It is an application used in many fields including image analysis, bioinformatics, social networks, social science, and so on...
The objective in this project is to apply two types of clustering analysis:

- a hard-clustering analysis, the K-means model.
- a type of soft-clustering analysis, the Gaussian mixture model.

The difference between hard and soft clustering is that, in one case, every single data point is assigned to only one cluster and boundaries are clearly visible, while in the other case, a weight is applied to each data point telling how confident we are that it belongs to a given cluster, and as such boundaries are more fuzzy.

The most popular database for education purposes, the Iris dataset [1], will be used for the analysis. It contains 150 datapoints each having 4 different attributes and belonging to one of three different classes. The goal will be to identify the class of each datapoint using a clustering model.

[1] https://archive.ics.uci.edu/ml/index.php

II. Methodology

1. K-means model

As mentionned above, K-means will seek to assign each datapoint to one of the K clusters. In the case of unlabelled datasets, the value of K is arbitrary, and several runs of the model with various values of K might be needed before reaching an appropriate partitionning of the data.

For each datapoint x_i, we associate a cluster assignment $c_i \in \{1, \cdots, K\}$.
And each cluster in $\{1, \cdots, K\}$ is represented by a mean vector μ_k, called a centroid, that defines the center of a cluster.
We are going to find the centroids such that all the datapoints assigned to a cluster are the closest to that same cluster centroid (we will use the squared Euclidian distance of x_i to μ_i). We want to minimize the following objective function:

$$\mu^*, c^* = \underset{\mu,c}{argmin} \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|^2$$

To solve this, we are going to split the model variables into two different sets. One set is going to contain all the vectors μ_k, and the other set will contain all the cluster assignments c_i.

We are first going to randomly initialize our centroids. Then we are going to iterate, updating each c_i and μ_k back and forth between the two following steps until the objective function stops changing:

- *Step 1*: Update each c_i by setting c_i to be the index of the cluster that the point x_i is closest to, in the Euclidian sense:

$$c_i = \underset{k}{argmin} \|x_i - \mu_k\|^2$$

- *Step 2*: Update each μ_k by taking the average of all of the data assigned to a particular centroid. So for the k^{th} centroid we simply average all of the data that was assigned to the k centroid according to the most recent update of each c_i:

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{n} x_i \mathbb{1}\{c_i = k\} \quad \text{with} \quad N_k = \sum_{i=1}^{n} \mathbb{1}\{c_i = k\}$$

2. Gaussian mixture model

The Gaussian mixture model is a probabilistic soft-clustering model where the weight vector ϕ_i is a probability distribution of data point x_i being assigned to cluster c_i.

GMM works by first defining:

- A prior distribution π_k on the k^{th} cluster.
- A likelihood distribution on the i^{th}observation x_i given it comes from the k^{th} cluster, that we write as a Gaussian distribution of mean μ_k and covariance Σ_k: $N(x_i \,|\, \mu_k, \Sigma_k)$.

The goal of the GMM is to maximize the log of the marginal likelihood of each datapoint x_i over π, μ and Σ in order to find those 3 parameters:

$$\mathcal{L} = \sum_{i=1}^{n} ln\ p\,(x_i \,|\, \pi, \mu, \Sigma) = \sum_{i=1}^{n} ln\ \sum_{k=1}^{K} \pi_k\, N(x_i \,|\, \mu_k, \Sigma_k)$$

We first initialize the parameters such as:

- π is initialized as a uniform distribution in the interval $[0, 1]$.
- Each μ_k is randomly initialized on one of the x_i datapoint location.
- Each Σ_k is initialized as an identity matrix.

Then the iteration is done in two steps until convergence of the objective function $\mathcal{L}$:

- Step 1: For each data point, the posterior probability $\phi_i(k)$ of coming from each of the k different clusters is calculated:

$$\phi_i(k) = \frac{\pi_k\, N(x_i \,|\, \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j\, N(x_i \,|\, \mu_j, \Sigma_j)}, \quad \text{for } k = 1, \cdots, K$$

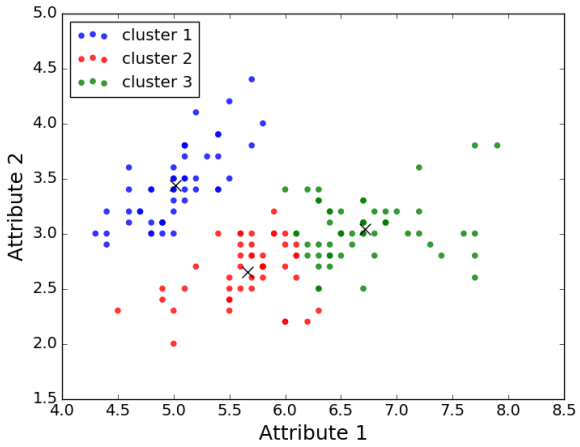
- Step 2: By defining $n_k = \sum_{i=1}^{n} \phi_i(k)$, we update the model parameters as follow:

$$\pi_k = \frac{n_k}{n}, \qquad \mu_k = \frac{1}{n_k} \sum_{i=1}^{n} \phi_i(k) x_i, \qquad \Sigma_k = \frac{1}{n_k} \sum_{i=1}^{n} (x_i - \mu_i)(xi - \mu_i)^T$$

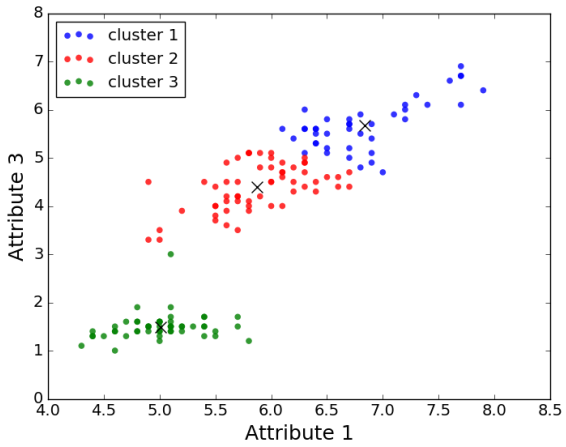
III. **Results**

1. K-means model

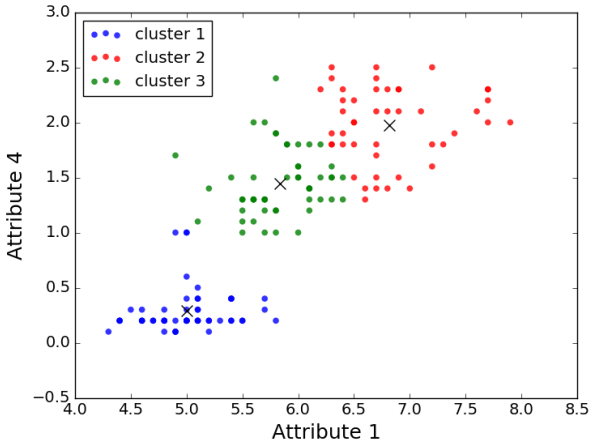
In order to visually see the partitionning and centroids location, only two of the four original attributes were used (see Figure 1). For each couple of attributes, comparisons were made between predictions and real class types.



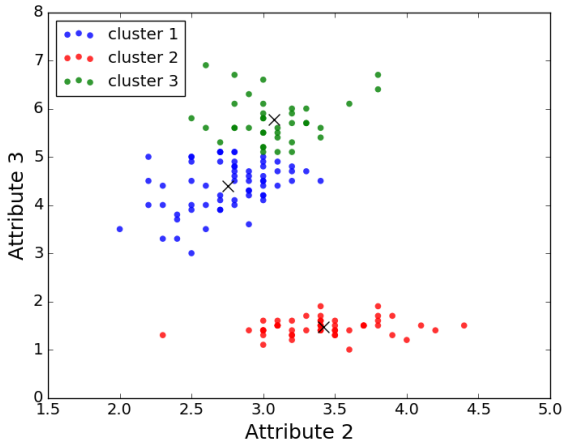
(a) Attributes 1 and 2



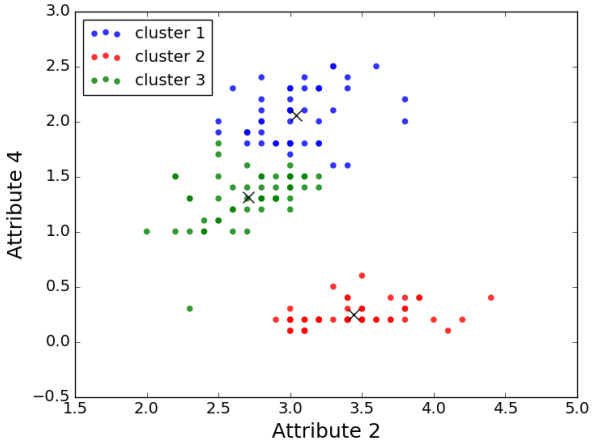
(b) Attributes 1 and 3



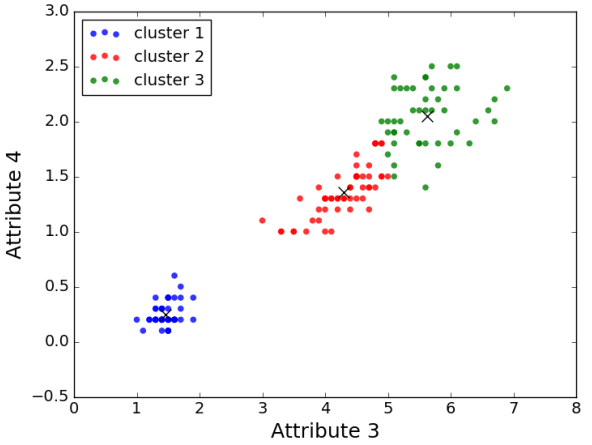
(c) Attributes 1 and 4



(d) Attributes 2 and 3



(e) Attributes 2 and 4



(f) Attributes 3 and 4

Figure 1: Various clusters plots with centroids for 2 of the 4 attributes

Attributes	Cluster 1	Cluster 2	Cluster 3
1 & 2	100%	76%	68%
1 & 3	100%	94%	74%
1 & 4	100%	82%	74%
2 & 3	100%	100%	74%
2 & 4	98%	92%	88%
3 & 4	100%	96%	88%
1 to 4	100%	96%	72%

Table 1: Rate of accurate predictions obtained with K-means

2. Gaussian mixture model

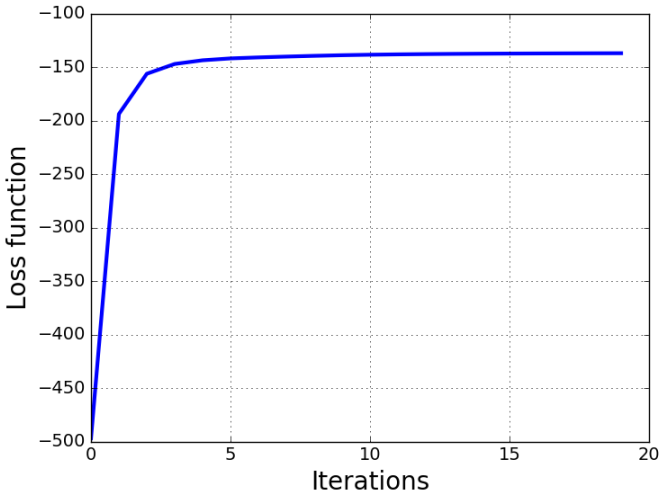


Figure 2: Objective function $\mathcal{L}$ of the GMM over iteration time

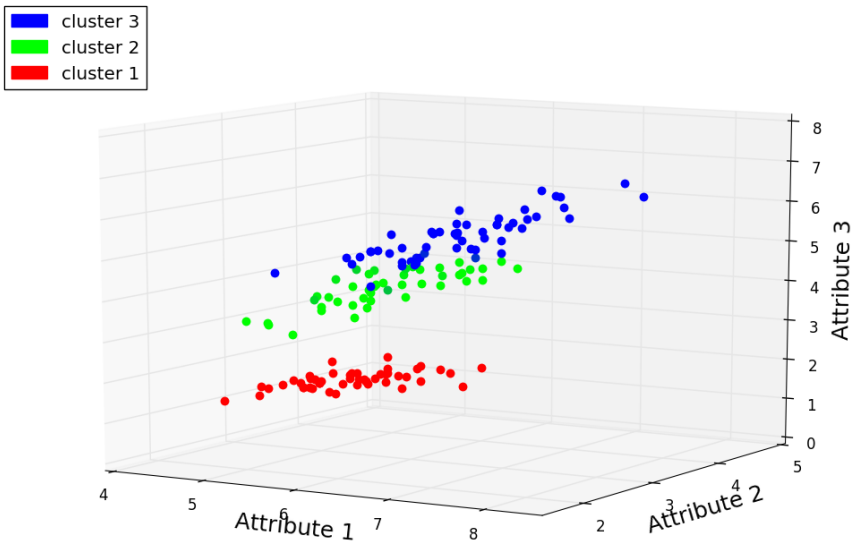


Figure 3: 3D plot of the clusters using all attributes with the GMM

IV. Python Code

```
from __future__ import division
import numpy as np
import sys
import random as rd
from matplotlib import pyplot as plt
import matplotlib.patches as mpatches
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.ticker as ticker
from scipy.stats import multivariate_normal as MvN

"""
This script should be launched from the console using the following command line:
python script_file X_data Label_data nb_cluster Clustering_type nb_iter
ex:    > python Clustering.py X.csv Y.csv 3 Kmeans 50
or:    > python Clustering.py X.csv Y.csv 3 GMM 50
Where:
.script_file is the name of this python file
.X_data is a .csv file of the data nodes coordinates
there should be one column per dimension, so for instance 2 columns if we are in
.Label_data is a .csv file that should contain the known labels corresponding
to the nodes in the X_data file
.nb_cluster is the number of clusters we want to partition the data into (5 is th
.Clustering_type is the type of clustering we want to apply:
Kmeans if we want to apply hard-clustering
GMM if we want to apply soft-clustering
.nb_iter is the number of iterations we are going to use
"""


class Kmeans:
    def __init__(self, nb_clust, nb_iter):
        self.X = np.genfromtxt(sys.argv[1], delimiter=",")
        self.Y = np.genfromtxt(sys.argv[2], delimiter=",")
        self.nb_clust = nb_clust
        self.nb_iter = nb_iter

        # Random initialisation of the mean mu = (mu_1,...,m_K)
        self.mu = np.zeros((nb_clust, len(self.X[0])))
        # print(self.mu.shape)
        for k in range(self.nb_clust):
            self.mu[k] = rd.choice(self.X)

        self.ci = np.zeros((len(self.X)))
        self.nk = np.zeros((nb_clust))

    def KMeans(self):
        LX = len(self.X)  # nb of datapoints

```

```
        ### do the following steps for self.nb_iter iterationsb ###

        for iter in range(self.nb_iter):
            centerslist = []
            print('iteration %s' % iter)

            # calculate for each cluster the measured Euclidian distance between
            # and select for c_i the cluster with the minimal distance
            count = 0
            for xi in self.X:
                val1 = 1e10
                clust = -1
                for k in range(self.nb_clust):
                    mu = self.mu[k]
                    val2 = np.dot(xi - mu, xi - mu)
                    # if count == 133:
                    # print('xi: %s, mu: %s, xi-mu: %s' % (xi, mu, xi - mu))

                    if val2 < val1:
                        val1 = val2
                        clust = k
                        # if count == 133:
                        # print('count: %s, k:%s, np.dot:%s' % (count, k, val2))
                self.ci[count] = clust
                count += 1

            # calculate numbers of indicators c_i for each cluster k
            for k in range(self.nb_clust):
                nbk = 0
                for i in range(LX):
                    if self.ci[i] == k:
                        nbk += 1
                self.nk[k] = nbk
                if self.nk[k] == 0:
                    self.nk[k] = 1
            # print('nk: %s'%self.nk)

            # Update the mean mu = (mu_1,...,m_K)
            for k in range(self.nb_clust):
                val = np.zeros((len(self.X[0])))
                count = 0
                for i in range(LX):
                    if self.ci[i] == k:
                        val += self.X[i]
                self.mu[k] = val / self.nk[k]
                centerslist.append(self.mu[k])

            # save centroids into file for each iteration
            filename = "centroids-" + str(iter + 1) + ".csv"  # "i" would be each
            np.savetxt(filename, centerslist, delimiter=",")
            # print(Kmeans.check_prediction(self, self.ci))

        return self.mu, self.ci
```

```
102
103     def check_prediction(self, ci):
104         LX = len(self.X)  # nb of datapoints
105         cTT = 50
106         ct1 = 0
107         ct2 = 0
108         ct3 = 0
109         print(ci)
110         for i in range(LX):
111             if 0 <= i < cTT:
112                 if i == 0:
113                     val = ci[i]
114                 if ci[i] == val:
115                     ct1 += 1
116             elif cTT <= i < 2 * cTT:
117                 if i == cTT:
118                     val = ci[i]
119                 if ci[i] == val:
120                     ct2 += 1
121             elif 2 * cTT <= i < 3 * cTT:
122                 if i == 2 * cTT:
123                     val = ci[i]
124                 if ci[i] == val:
125                     ct3 += 1
126
127         return ct1, ct2, ct3
128
129     def plot_data(self, ci, mu, nb):
130         x1 = []
131         y1 = []
132         x2 = []
133         y2 = []
134         x3 = []
135         y3 = []
136         x4 = []
137         y4 = []
138         x5 = []
139         y5 = []
140         mux = []
141         muy = []
142
143         # get mean
144         for k in range(len(mu)):
145             mux.append(mu[k][0])
146             muy.append(mu[k][1])
147
148         ## plot datapoints per cluster
149         colors = ('b', 'r', 'g', 'c', 'y')
150         groups = ("cluster 1", "cluster 2", "cluster 3", "cluster 4", "cluster 5"
151         for i in range(len(self.X)):
152             if ci[i] == 0:
153                 x1.append(self.X[i][0])
154                 y1.append(self.X[i][1])
```

```
            elif ci[i] == 1:
                x2.append(self.X[i][0])
                y2.append(self.X[i][1])
            elif ci[i] == 2:
                x3.append(self.X[i][0])
                y3.append(self.X[i][1])
            elif ci[i] == 3:
                x4.append(self.X[i][0])
                y4.append(self.X[i][1])
            elif ci[i] == 4:
                x5.append(self.X[i][0])
                y5.append(self.X[i][1])

        g1 = (x1, y1)
        g2 = (x2, y2)
        g3 = (x3, y3)
        g4 = (x4, y4)
        g5 = (x5, y5)
        data = (g1, g2, g3, g4, g5)

        # Create plot
        fig = plt.figure()
        #ax = fig.add_subplot(1, 1, 1, axisbg="1.0") #python 2.7
        ax = fig.add_subplot(1, 1, 1)
        for data, color, group in zip(data, colors[0:nb], groups[0:nb]):
            x, y = data
            ax.scatter(x, y, alpha=0.8, c=color, edgecolors='none', s=30, label=g
        plt.scatter(mux, muy, c='k', marker='x', s=100)
        plt.legend(loc=2)
        plt.show()


class GMM:
    def __init__(self, nb_clust, nb_iter):
        self.X = np.genfromtxt(sys.argv[1], delimiter=",")
        self.nb_clust = nb_clust
        self.nb_iter = nb_iter
        self.nk = np.zeros((nb_clust))

        # Random initialisation of the mean mu = (mu_1,...,m_K)
        self.mu = np.zeros((nb_clust, len(self.X[0])))
        for k in range(self.nb_clust):
            self.mu[k] = rd.choice(self.X)

        # Initialisation of the sigma_k matrix as identity matrix
        self.sigma = np.zeros((nb_clust, len(self.X[0]), len(self.X[0])))
        for k in range(self.nb_clust):
            self.sigma[k] = np.matrix(np.identity(len(self.X[0])), copy=False)

        # Initialisation of the conditional posterior probability distribution
        self.phi = np.zeros((len(self.X), nb_clust))

        # Initialisation of pi as a uniform distribution
```

```
208         self.pi = np.random.uniform(low=0.0, high=1.0, size=nb_clust)
209 
210     def EMGMM(self):
211         LX = len(self.X)  # nb of datapoints
212         L_save = []  # list saving loss function at each iteration
213 
214         ### do the following steps for self.nb_iter iterationsb ###
215 
216         for iter in range(self.nb_iter):
217             print("iteration: %s" % (iter + 1))
218             # Initialize objective function at the beginning of each iteration
219             L = 0
220 
221             ## E-step: generate phi posterior probability
222             # print("E-step")
223             # calculate for each datapoint x_i the sum of pi_k*N_k(mu_k,Sigma_k)
224             sum_pi_j_N = np.zeros((len(self.X)))
225             for i in range(LX):
226                 val = 0
227                 for k in range(self.nb_clust):
228                     val += self.pi[k] * MvN.pdf(self.X[i], mean=self.mu[k], cov=s
229                     # print(MvN.pdf(self.X[i], mean=self.mu[k], cov=self.sigma[k]
230                 sum_pi_j_N[i] = val
231 
232             # calculate phi & objective function
233             L = 0  # Initialize objective function at the beginning of each itera
234             for i in range(LX):
235                 val2 = 0
236                 for k in range(self.nb_clust):
237                     val = self.pi[k] * MvN.pdf(self.X[i], mean=self.mu[k], cov=se
238                     val2 += val
239                     self.phi[i][k] = val / sum_pi_j_N[i]
240                 L += np.log(val2)
241             L_save.append(L)
242 
243             ## M-step
244             # print("M-step")
245             # update empirical distribution pi_k using expected nb of pts n_k com
246             for k in range(self.nb_clust):
247                 val = 0
248                 for i in range(LX):
249                     val += self.phi[i][k]
250                 self.nk[k] = val
251                 if self.nk[k] == 0:
252                     self.nk[k] = 1
253 
254             for k in range(self.nb_clust):
255                 self.pi[k] = self.nk[k] / LX
256 
257             # update mean mu_k
258             for k in range(self.nb_clust):
259                 val = np.zeros((len(self.X[0])))
260                 for i in range(LX):
```

```
                    # print(self.phi[i][k],self.X[i])
                    val += self.phi[i][k] * self.X[i]
                self.mu[k] = val / self.nk[k]

            # update covariance sigma_k
            for k in range(self.nb_clust):
                val = np.zeros((len(self.X[0]), len(self.X[0])))
                for i in range(LX):
                    diff_iT = self.X[i] - self.mu[k]
                    diff_i = diff_iT[np.newaxis, :].T
                    aa = np.zeros((len(self.X[0]), len(self.X[0])))
                    bb = np.zeros((len(self.X[0]), len(self.X[0])))
                    bb[0] = diff_iT
                    for j in range(len(self.X[0])):
                        aa[j][0] = diff_i[j]

                    val += self.phi[i][k] * np.dot(aa, bb)
                self.sigma[k] = val / self.nk[k]

            # save pi, phi, mu & sigma into files for each iteration
            filename = "pi-" + str(iter + 1) + ".csv"
            np.savetxt(filename, self.pi, delimiter=",")
            filename = "phi-" + str(iter + 1) + ".csv"
            np.savetxt(filename, self.phi, delimiter=",")
            filename = "mu-" + str(iter + 1) + ".csv"
            np.savetxt(filename, self.mu, delimiter=",")  # this must be done at

            for k in range(self.nb_clust):  # k is the number of clusters
                filename = "Sigma-" + str(k + 1) + "-" + str(
                    iter + 1) + ".csv"  # this must be done 5 times (or the numbe
                np.savetxt(filename, self.sigma[k], delimiter=",")

        # save objective function
        np.savetxt("Loss_function.csv", L_save, delimiter=",")

        # Plot objective function
        plt.plot(L_save, linewidth=3)
        axes = plt.gca()
        # axes.set_ylim([-0.5e6, 0])
        plt.ticklabel_format(axis='y', style='sci')  # , scilimits=(-2, 2))
        plt.grid()
        plt.xlabel('Iterations', fontsize=20)
        plt.ylabel('Loss function', fontsize=20)
        axes.tick_params(labelsize=14)
        plt.show()

        return self.pi, self.mu, self.sigma, self.phi

    def plot_GMM(self, phi, nb):

        # create two vectors with data point coordinates (we are considering to b
        x = []
        y = []
```

```
314         for i in range(len(self.X)):
315             x.append(self.X[i][0])
316             y.append(self.X[i][3])
317 
318         # adjust phi for colors
319         col = phi[:]
320         for i in range(len(self.X)):
321             for k in range(nb):
322                 if col[i][k] < 0.01:
323                     col[i][k] = 0
324                 elif col[i][k] > 1:
325                     col[i][k] = 1
326                     # else:
327                     #     col[i][k] *= 1
328                     # col[i][k] = int(round(col[i][k]))
329 
330         np.savetxt("colors.csv", col, delimiter=",")
331 
332         groups_colors = {'cluster 1': 'r', 'cluster 2': [0, 1, 0], 'cluster 3': '
333         # Create plot
334         fig = plt.figure()
335         # ax = fig.add_subplot(1, 1, 1, axisbg="1.0") # python 2.7
336         ax = fig.add_subplot(1, 1, 1)
337         # handles, labels = ax.get_legend_handles_labels()
338         # ax.legend(colors, groups)
339         patchList = []
340         for key in groups_colors:
341             data_key = mpatches.Patch(color=groups_colors[key], label=key)
342             patchList.append(data_key)
343 
344         ax.legend(handles=patchList, loc=2)
345 
346         for i in range(len(self.X)):
347             ax.scatter(x[i], y[i], alpha=0.8, c=[col[i][0], col[i][1], col[i][2],
348                        s=40)  # , label=group)
349 
350         plt.xlabel('Attribute 1', fontsize=18)
351         plt.ylabel('Attribute 4', fontsize=18)
352         ax.tick_params(labelsize=14)
353 
354         plt.show()
355 
356 
357     def plot_GMM_3D(self, phi, nb):
358         # create two vectors with data point coordinates (we are considering to b
359         x = []
360         y = []
361         z = []
362 
363         for i in range(len(self.X)):
364             x.append(self.X[i][0])
365             y.append(self.X[i][1])
366             z.append(self.X[i][2])
```

```
367
368         # adjust phi for colors
369         col = phi[:]
370         for i in range(len(self.X)):
371             for k in range(nb):
372                 if col[i][k] < 0.01:
373                     col[i][k] = 0
374                 elif col[i][k] > 1:
375                     col[i][k] = 1
376
377
378         np.savetxt("colors.csv", col, delimiter=",")
379
380         groups_colors = {'cluster 1': 'r', 'cluster 2': [0, 1, 0], 'cluster 3': '
381         # Create plot
382         fig = plt.figure()
383         ax = fig.add_subplot(111, projection='3d')
384
385         patchList = []
386         for key in groups_colors:
387             data_key = mpatches.Patch(color=groups_colors[key], label=key)
388             patchList.append(data_key)
389
390         ax.legend(handles=patchList, loc=2)
391
392         for i in range(len(self.X)):
393             ax.scatter(x[i], y[i], z[i], zdir='z', s=50, c=[col[i][0], col[i][1],
394
395         ax.set_xlabel('Attribute 1', fontsize=18)
396         ax.set_ylabel('Attribute 2', fontsize=18)
397         ax.set_zlabel('Attribute 3', fontsize=18)
398         tick_spacing = 1.0
399         ax.xaxis.set_major_locator(ticker.MultipleLocator(tick_spacing))
400         ax.yaxis.set_major_locator(ticker.MultipleLocator(tick_spacing))
401         ax.zaxis.set_major_locator(ticker.MultipleLocator(tick_spacing))
402         ax.tick_params(labelsize=12)
403
404         plt.show()
405
406
407 if __name__ == "__main__":
408     try:
409         assert len(sys.argv) > 5, "Missing input arguments"
410         iter = int(sys.argv[5])  # nb of iterations
411         nb = int(sys.argv[3])    # nb of clusters
412         Cl_type = sys.argv[4]    # type of clustering
413         if Cl_type == "Kmeans":
414             mu, ci = Kmeans(nb, iter).KMeans()
415             ct1, ct2, ct3 = Kmeans(nb, iter).check_prediction(ci)
416             Kmeans(nb, iter).plot_data(ci, mu, nb)
417             print("ct1 = %s; ct2 = %s; ct3 = %s" % (ct1, ct2, ct3))
418         elif Cl_type == "GMM":
419             pi, mu, sigma, phi = GMM(nb, iter).EMGMM()
```

```
420             GMM(nb, iter).plot_GMM(phi, nb)
421             GMM(nb, iter).plot_GMM_3D(phi, nb)
422         else:
423             print("Clustering type not known")
424     except Exception as e:
425         print(e)
```