HOMEWORK 1/2 - ISYE/CSE 6740

Table of content

Exercise 3 - Homework1: Political blogs dataset

1) If we have k clusters in the graph, it means that for each cluster (from 1 to k) the political blogs that are in this cluster have a stronger relationship in terms of political views. Then the number k here means the number of possible political orientations that exist in the political blogs in the data set.

2)

Creation_Adjency (Adjency=Adjacency*)

Input: Edges, Nodes that are lists of lists

Output: Adjency_matrix with DataFrame type

```
def Creation_Adjency(Edges,Nodes):
    Matrix_size =len(Nodes)
    Adjacency_Matrix=np.zeros(shape=(Matrix_size,Matrix_size))

    for edge in Edges:
        Adjacency_Matrix[edge[0]-1][ edge[1]-1]=1
        Adjacency_Matrix[edge[1]-1][ edge[0]-1]=1
    return pd.DataFrame(Adjacency_Matrix)
```

Delete

Input: Adjency Matrix

Output: Index of rows columns that contains only 0

```
def delete(Adjacency_Matrix) :
    Index_remove=[]
    for i in range (0,len(Adjacency_Matrix)):
        if sum(Adjacency_Matrix.iloc[i])==0:
            Index_remove.append(i)
    return  Index_remove
```

Creation_Degree_Matrix

Input: Adjency_matrix

Output: D (the degree matrix associated to the Adjency Matrix)

```
def Creation_Degree_Matrix(Adjency_matrix):
    Size=len(Adjency_matrix)
    D=pd.DataFrame(np.zeros(shape=(Size,Size)))
    Degree=[]
    for row in range(0,len(Adjency_matrix)):
        degree=sum(Adjency_matrix.iloc[row])
        Degree.append(degree)
    for i in range(0,len(Degree)):
        D.iloc[i,i]=Degree[i]
    return D
```

Graph_Laplacian

Inputs: Adjency Matrix and D

Outputs: L=D-A the Graph Laplacian

```
def Graph_Laplacian(D,A):
    return D-A
```

Eigen_Value_Vect_selected

Inputs: Graph Laplacian L and the number of cluster k

Output: array of eigenvectors that are associated with the lowest eigenvalues

```
def Eigen_Value_Vect_selected(L,k):
    Value, Vector=np.linalg.eig(L)
    Vector=np.real(Vector)
    Value=np.real(Value)
    value_sorted=sorted(Value.tolist())
    Eigen_Vectors_Selected=[]
    for eigVal in value_sorted[0:k]:
        Eigen_Vectors_Selected.append(Vector[:,Value.tolist().index(eigVal)])
    return np.asarray(Eigen_Vectors_Selected)
```

Cluster:

Inputs: Number of clusters, Edges and Nodes

Outputs: Clusters of blogs with their associated political side : 0 or 1

```python
def Cluster(n_clusters,Edges,Nodes):
    A=Creation_Adjency(Edges,Nodes)
    Delete=delete(A)
    D=Creation_Degree_Matrix(A)
    L=Graph_Laplacian(D,A)
    L=L.drop(index=Delete)
    L=L.drop(columns=Delete)
    Columns_index_L=list(L.columns) #Rows
    Eigen_Vectors_Selected =Eigen_Value_Vect_selected(L,n_clusters)
    data_set=np.transpose(Eigen_Vectors_Selected)
    kmeans = KMeans(n_clusters).fit(data_set)
    label=list(kmeans.labels_)
    clusters=[]
    for i in range(0,n_clusters):
        clusters.append([])

    for j in range (0,len(label)):
        b=Columns_index_L[j]
        a=Nodes[b][2]
        clusters[label[j]].append(a)

    return clusters
```

List_cluster_Majority_labels:

Inputs: Edges, Nodes, k the number of clusters+1

Outputs: List of lists of clusters and their political side (0 or 1), Dictionnary with the majority of labels for each cluster.

```python
def List_cluster_Majority_labels(Edges,Nodes,k):

    List_clusters=[]
    Majority_label={}
    for j in range(2,k):
        List_clusters.append(Cluster(j,Edges,Nodes))
        Majority_label['Majority_label_cluster k = {}'.format(j)]=[]

    index_cluster=1
    for i in List_clusters:
        index_cluster+=1
        for j in i:
            if j.count(1)>j.count(0):
                Majority_label['Majority_label_cluster k = {}'.format(index_cluster)].append(1)
            else:
                Majority_label['Majority_label_cluster k = {}'.format(index_cluster)].append(0)

    print(List_clusters)
    print(Majority_label)

    return List_clusters, Majority_label
```

For k=2:

{'Majority_label_cluster k = 2': [1, 0]}

We see that the blogs are splited in two parts: one part with most of the political view equal 1 and the other equal to 0.

For k=3

'Majority_label_cluster k = 3': [1, 0, 1]}

We see another cluster with majority of 1 appearing when we are looking for 3 clusters

For k=4:

'Majority_label_cluster k = 4': [1, 0, 1, 1]}

We see another cluster with majority of 1 appearing when we are looking for 4 clusters

3) To calculate the mismatch rate, I first implemented a function that calculates the mismatch rate of for a group of cluster taking the mean of the rate of each cluster.

Mismatch_mean:

Input: group of cluster wich is defined by k=2,3..

Outputs: mean of the mismatch rate from each cluster in the group k=2,3…

```
def mismatch_mean(cluster):
    mismatch=[]

    count=0
    for Cluster in cluster:
        if Cluster!=[]:
            count+=1
            if Cluster.count(1)>Cluster.count(0):   #1 is in majority
                nbr_0=Cluster.count(0)
                mismatch.append(nbr_0/len(Cluster))
            else:
                nbr_1=Cluster.count(1)
                mismatch.append(nbr_1/len(Cluster))
    return sum(mismatch)/len(cluster)
```

Mismatch_rate

Inputs: Mismatch_mean(method used to calculate the mismatch), k(Number of cluster)+1, Edges, Nodes

Outputs : Dictionary of the mismatch rate for each k=2,3….

```
def mismatch_rate(mismatch_mean,k,Edges,Nodes):
    List_clusters, Majority_label=List_cluster_Majority_labels(Edges,Nodes,k)
    Mismatch_rate={}
    for j in range(2,k):
        Mismatch_rate['Mismatch_rate_cluster k = {}'.format(j)]=[]

    index_cluster=1
    for i in List_clusters:
        index_cluster+=1
        Mismatch_rate['Mismatch_rate_cluster k = {}'.format(index_cluster)].append(mismatch_mean(i))


    print(Mismatch_rate)
```

For instance, for:

mismatch_rate (mismatch_mean, 5, Edges, Nodes)

We get:

{'Mismatch_rate_cluster k = 2': [0.2397708674304419],

'Mismatch_rate_cluster k = 3': [0.1603721948549535],

'Mismatch_rate_cluster k = 4': [0.12047697368421052]}

4) In order to find the number of clusters to achieve a reasonably small mismatch rate, I plot the mismatch rate for each k, with k from 2 to 200. Then I took the minimum of the mismatch rate calculated and fund its k.

Mismatchgraph

Inputs: k+1, Edges, Nodes, mismatch_mean

Outputs: plot of the mismatch rate in function of k, the smallest mismatch rate and its k associated

```
def mismatchgraph(k,Edges,Nodes,mismatch_mean):
    X=[]
    Y=[]
    count=0
    for j in range (2,k):
        X.append(j)
        cluster=Cluster(j,Edges,Nodes)
        Y.append(mismatch_mean(cluster))
        # Y2.append(mismatch_sum(cluster))
        # Y3.append(mismatch_max(cluster))
        count+=1
        print(count)

    plt.xlabel('K')
    plt.ylabel('Mismatch Rate')
    plt.plot(X,Y,label='Mean of mismatchs per cluster')
    print('the smallest Mismatch_rate is {} for a value of k = {}'.format(min(Y),X[Y.index(min(Y))]))
    plt.legend()
    plt.savefig('Evolution of the mismatch in function of k.png')
    plt.show()
```

For:

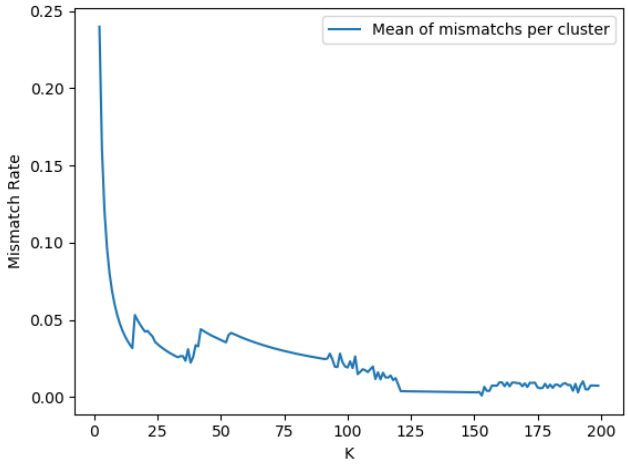
mismatchgraph (200, Edges, Nodes, mismatch_mean)

We have:

The smallest mismatch rate is:

Output: ‘the smallest Mismatch_rate **is 0.0009255741315774339** for a value of **k = 153’**

And we obtained this graph:

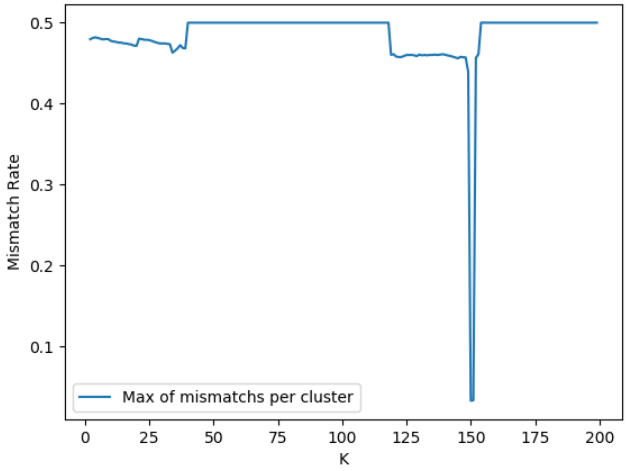


We observe that the mismatch rate is constantly decreasing while K is growing.

I also use another method to calculate the mismatch rate. This one is base on tacking the max of the mismatch for each k:

```
def mismatch_max(cluster):
    mismatch=[]

    count=0
    for Cluster in cluster:
        if Cluster!=[]:
            count+=1
            if Cluster.count(1)>Cluster.count(0):   #1 is in majority
                nbr_0=Cluster.count(0)
                mismatch.append(nbr_0/len(Cluster))
            else:
                nbr_1=Cluster.count(1)
                mismatch.append(nbr_1/len(Cluster))
    return max(mismatch)
```

Then we obtain:



With the output:

'the smallest Mismatch_rate is **0.03270223752151463** for a value of **k = 150'**

Both methods (mean and max) are yielding an optimal number of clusters of around 150. Both methods allow us to set up an optimal number of clusters around 150 as they validate each other.

5) For k = 153 we have the smallest mismatch within the clusters. This means that we can cluster the Nodes in k groups in which blogs have closest ties and thus share the same political view. Then, we can assume that there is k political orientation in the blogs listed.

Homework 2

PCA: Food consumption in European countries

1) In this first part of the exercise, we will consider m=16 data points that correspond to each country in the dataset (countries are the rows of the data set). And their features, in each column are the food consumption by products. Then, each data point as a size of (1,16).

Indice	Country	eal coffe	stant coff	Tea	weetene	Biscuits	owder sou	Tin soup	Potatoes	rozen fis	zen vegg	Apples	Oranges	inned frui	Jam	Garlic	Butter	Margarine	Olive oil	Yoghurt	risp brea
0	Germany	90	49	88	19	57	51	19	21	27	21	81	75	44	71	22	91	85	74	30	26
1	Italy	82	10	60	2	55	41	3	2	4	2	67	71	9	46	80	66	24	94	5	18
2	France	88	42	63	4	76	53	11	23	11	5	87	84	40	45	88	94	47	36	57	3
3	Holland	96	62	98	32	62	67	43	7	14	14	83	89	61	81	15	31	97	13	53	15
4	Belgium	94	38	48	11	74	37	23	9	13	12	76	76	42	57	29	84	80	83	20	5
5	Luxembourg	97	61	86	28	79	73	12	7	26	23	85	94	83	20	91	94	94	84	31	24
6	England	27	86	99	22	91	55	76	17	20	24	76	68	89	91	11	95	94	57	11	28
7	Portugal	72	26	77	2	22	34	1	5	20	3	22	51	8	16	89	65	78	92	6	9
8	Austria	55	31	61	15	29	33	1	5	15	11	49	42	14	41	51	51	72	28	13	11
9	Switzerland	73	72	85	25	31	69	10	17	19	15	79	70	46	61	64	82	48	61	48	30
10	Sweden	97	13	93	31	61	43	43	39	54	45	56	78	53	75	9	68	32	48	2	93
11	Denmark	96	17	92	35	66	32	17	11	51	42	81	72	50	64	11	92	91	30	11	34
12	Norway	92	17	83	13	62	51	4	17	30	15	61	72	34	51	11	63	94	28	2	62
13	Finland	98	12	84	20	64	27	10	8	18	12	50	57	22	37	15	96	94	17	21	64
14	Spain	70	40	40	18	62	43	2	14	23	7	59	77	30	38	86	44	51	91	16	13
15	Ireland	30	52	99	11	80	75	18	2	5	3	57	52	46	89	5	97	25	31	3	9

2) As we want to reduce the dimension of the data points, we are looking to do a projection of each data point on a reduced space. In this exercise we want to project a data point that as 16 dimensions (16 features) into a à 2-dimensional space. Then, we are looking for 2 directions (w : vector) that maximize the variance of the data set in order minimize the loose of information by the dimensional reduction.
We can formulate this such that:

With m data point x^i, with i ∈ {1,2…..,m} :

$$max_{w:\|w\|\leq 1} \frac{1}{m}\sum_{i=1}^{m}(w^T x^i - w^T \mu^i)^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(w^T(x^i - w^T\mu^i)\right)^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(w^T(x^i - w^T\mu^i)\right)\left(w^T(x^i - w^T\mu^i)\right)$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(w^T(x^i - w^T\mu^i)\right)\left((x^i - w^T\mu^i)w\right)$$

$$= w^T\frac{1}{m}\sum_{i=1}^{m}\left((x^i - \mu^i)(x^i - \mu^i)\right)w$$

$$= w^T Cov(x) w$$

Ps: as we want to miximize this expression considering w, we must bound w such that to converge. This is why we have $w: \|w\| \leq 1$

Therefore, we use the Lagrangian function to put in equation this optimization problem with this constraint:

$$L(w,\lambda) = w^T Cov(x) w + \lambda(1 - \|w\|^2)$$

By derivating by respect to w, :

$$\frac{\partial L}{\partial w} = 2Cov(x)w - 2\lambda w = 0$$

$$Cov(x)w = \lambda w$$

Then, the optimal solution is:

$$max_{w:\|w\|\leq 1}\ \lambda w w^T = max_{w:\|w\|\leq 1}\ \lambda \|w\|^2$$

As w is bounded, this maximization problem turns into finding the largest eigenvalue of Cov and its eigenvector associated.

Finally to find the rest of the principal component, we are looking to find the eigen vectors that have also their eigenvalue associated large (the second largest, third largest …). The number of chosen eigenvector depends on the number of dimensions that we want to project the data points. Each vector by definition of the eigen decomposition are ortho-nomal which satisfies the maximization problem.

3) Explanation of the code :

After importing the csv file, in Row, we are centering all the data points around the mean so as to calculate the covariance matrix .

```
Food_consumption = pd.read_csv ('food-consumption.csv')
mean_data=Food_consumption.mean(axis=0)
m_points=len(Food_consumption.index)

Row=[]
for row in range(0,len(Food_consumption.index)):
    Row.append(np.array(Food_consumption.iloc[row,1:].tolist())-np.asanyarray(mean_data).T)

X=np.asarray(Row)
Xt=X.T

Cov=(1/m_points)*Xt.dot(X)
```

Then, we are selecting k=2 largest eigenvalues with their eigen vector associated in the covariance matrix.

```
k=2
def Eigen_Value_Vect_selected(data_set,k):
    Value, Vector=np.linalg.eig(data_set)
    Vector=np.real(Vector)
    Value=np.real(Value)
    value_sorted=sorted(Value.tolist(),reverse=1)
    Eigen_Vectors_Selected=[]
    for eigVal in value_sorted[0:k]:
        Eigen_Vectors_Selected.append(Vector[:,Value.tolist().index(eigVal)])
    return np.asarray(Eigen_Vectors_Selected), value_sorted[0:k]

Eigen_vector, Eigen_values=Eigen_Value_Vect_selected(Cov,k)
```

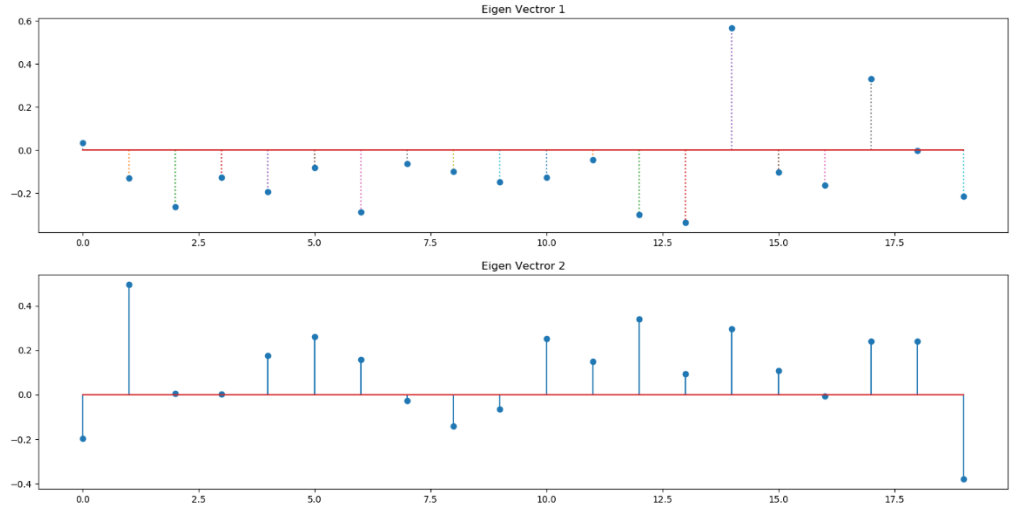
For each data point we project them in the k=2 Dimensionnal space:

```
reduced_representation=[]

for data_point in Row:
    reduced_representation.append([Eigen_vector[i].dot(data_point)/m.sqrt(Eigen_values[i]) for i in range(0,k)])
```

We can now plot the weight vectors (eigen vectors) using the stem method:

```
plt.subplot(2, 1, 1)
plt.title('Eigen Vectror 1')
plt.stem(list(Eigen_vector)[0],linefmt=':')
plt.subplot(2, 1, 2)
plt.title('Eigen Vectror 2')
plt.stem(list(Eigen_vector)[1])
plt.figure()
plt.savefig('Eigen Vectors for Countries=Features.png')
plt.show
```

We have then:



Axis are corresponding to :

1	2	3	4	5	6	7	8	9	10
Real Coffee	Instant Coffee	Tea	Sweetener	Biscuit	Powder soup	Tin soup	Potatoes	Frozen fish	Frozen veggies
11	12	13	14	15	16	17	18	19	20
Apple	Oranges	Tinned fruit	Jam	Garlic	Butter	margarine	Olive oil	Yoghurt	Crisp bread

First eigen vector:

- Gives a lot of importance to Garlic and Olive oil consumption
- Nullifies the impact of Real Coffee, Oranges and Yoghurt consumption

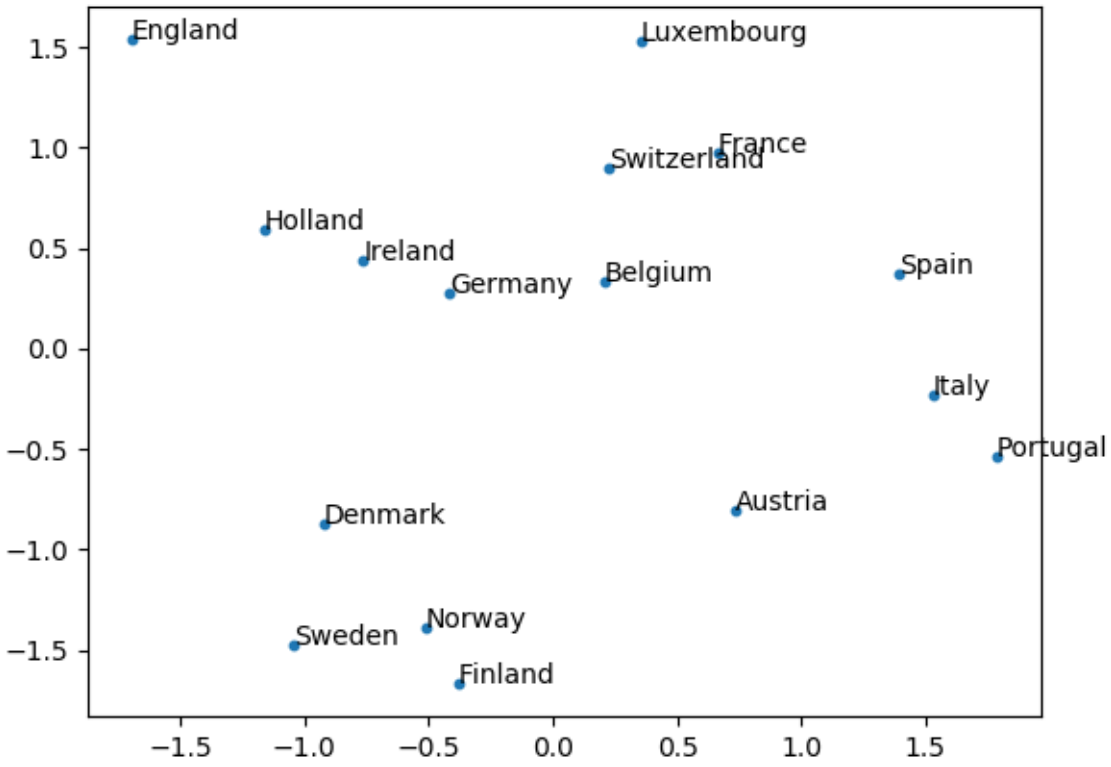
Second vector:

- Gives a lot of importance to instant coffee and Crisp bread consumption
- Nullifies the impact of Margarine, Tea, Sweetener, Potatoes and Oranges and Yoghurt consumption

Therefore, most countries can be represented by their Garlic, Olive Oil, instant Coffee and Crisp bread consumption features. However, other food consumption such as Oranges, or Yoghurt are no a differentiating element between countries.

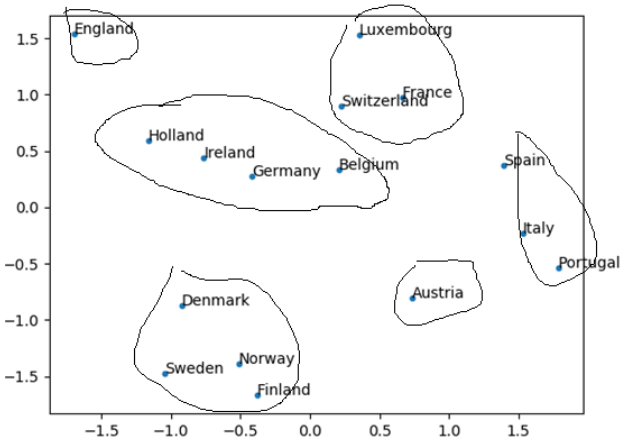
4) By plotting a two-dimensional representation of the data set, we get:

We observe that



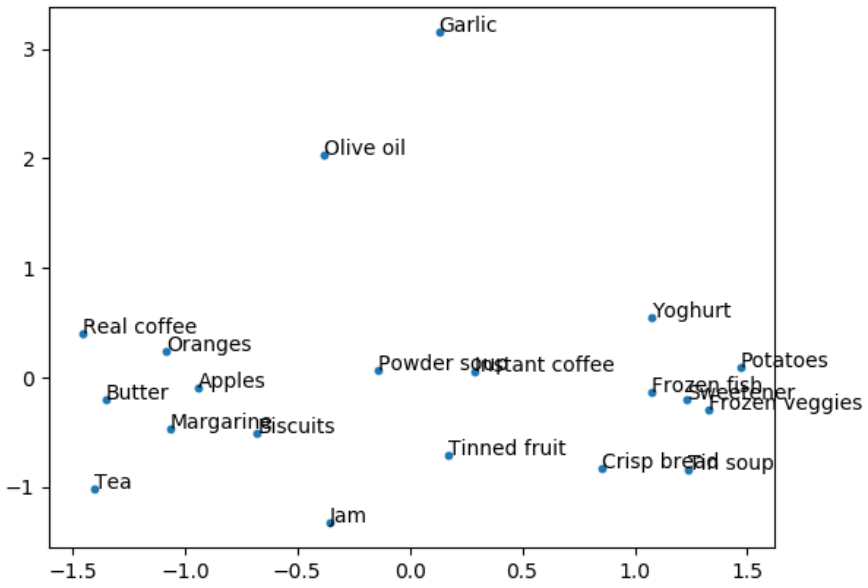
We observe that countries seem to be clustered by their location. For instance, Denmark, Norway, Sweden and Finland are closer than the South European countries. This result is logic as close countries are sharing similar food consumption. In fact, Austria is 'alone' as it is separated from the rest of the world geographically. However England is also far away from other points, which means that the English food consumption is unique in its region.

We can observe the same pattern of cluster by region as follows :

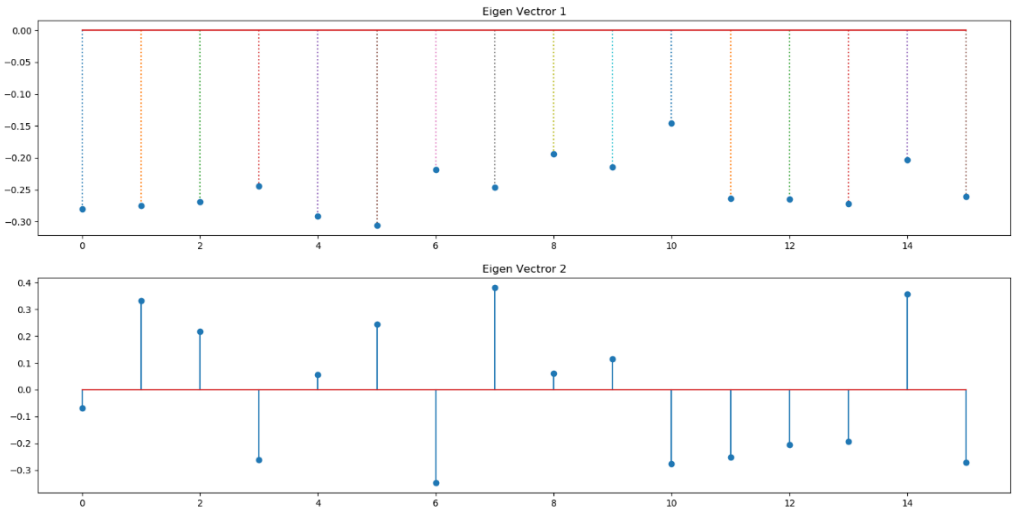


It could be interesting to use K-means algorithm to better cluster the countries by their food consumption.

5) Now using Countries as features and Food item as data points we have:



We observe that food items are clustered by their type of use. For instance, we have Butter, Margarine and Biscuits that are closer to each other as there are close food items. In the same way, Garlic and Olive oil are closer to each other as there are use in the same way while cooking. Then food items seem to be clustered by the food habit of each countries/region of the world.



Germany	Italy	France	Holland	Belgium	Luxembourg	England	Portugal
1	2	3	4	5	6	7	8
Austria	Switzerland	Sweden	Danmark	Norway	Finland	Spain	Ireland
9	10	11	12	13	14	15	16

First eigen vector:

- Gives a lot of importance to all countries and nullifies none of them in the projection

Second vector:

- Gives a lot of importance to Spain, Portugal and England for projecting the food items
- Reduce the impact of Belgium and Austria in the projection

Order of faces using ISOMAP

a)

First we have to generate a raw Adjency matrix with Euclidian distance as follow with the data_set imported :

Inputs: Dataset and the distance function needed to calculate the distance between two data points.

Outputs: Adjacency matrix with all the distance

```
mat = scipy.io.loadmat('isomap.mat')
data_set=pd.DataFrame(mat['images'])
```

```
def euclidian_distance(x,y): #Takes data frame
    x=np.asarray(x)
    y=np.asarray(y)
    return m.sqrt(np.vdot((x-y).T,x-y))

def Creation_Adjency(data_set,Dist):
    Matrix_size =len(data_set.iloc[0,:])
    Adjacency_Matrix=np.zeros(shape=(Matrix_size,Matrix_size))
    for i in range(0,Matrix_size):
        for j in range(0,Matrix_size):
            vector1=data_set.iloc[:,i]
            vector2=data_set.iloc[:,j]
            distance=Dist(vector1,vector2)
            Adjacency_Matrix[i][j]=distance
    return Adjacency_Matrix
```

Then the objective is to have at least for each nodes 100 neighbors. Thus, after sorting each columns and taking the 101th value that goes to a list, we return the value of the maximum of this list. This value becomes our threshold:

```
def threshold(Creation_Adjency):
    B=Creation_Adjency.copy()
    Max=[]
    for i in range(0,len(B[:][0])):
        a=sorted(B[:,i].tolist())[101]
        Max.append(a)
    return max(Max)
```

We use know this threshold as a filter for our row Adjency matrix:

```
def Adjency(Creation_Adjency,treshold):

    for i in range(0,len( Creation_Adjency[:][0])):
        for j in range(0,len( Creation_Adjency[:][0])):
            if Creation_Adjency[i][j]>treshold:
                Creation_Adjency[i][j]= 1000000

    return Creation_Adjency
```

The value 1000000 means that the two points are far away from each other.

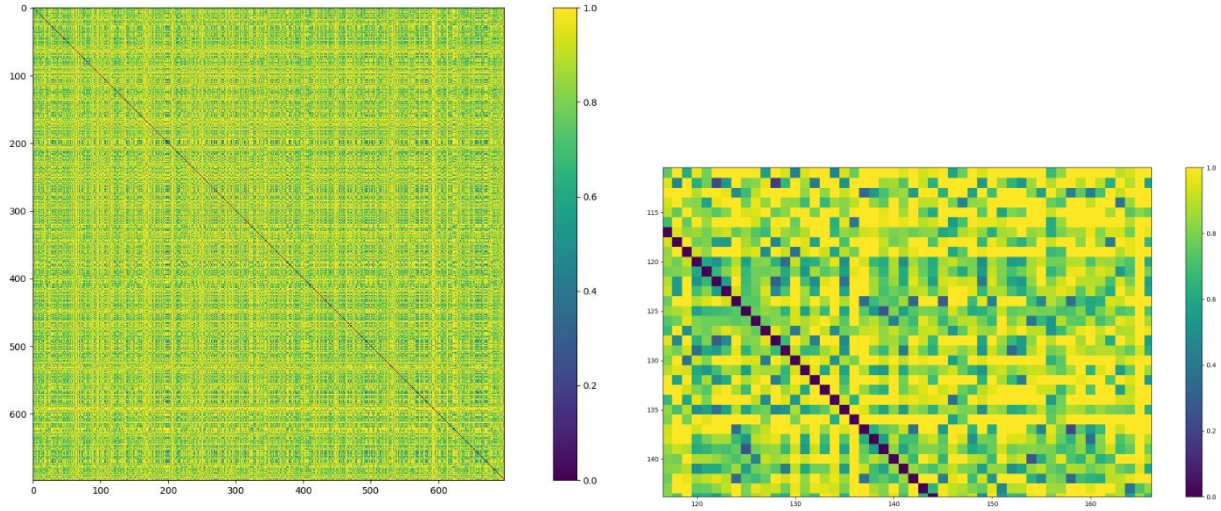
If now we show the Adjency matrix, the values are to much spread. To avoid a knotty plot of the matrix we "normalize" each value. This is who I proceed:

- For values under the threshold: we take this values divides by the threshold +1
- For values above the threshold (=1000000) we replace the value by 1

```
Adjency_dist=Creation_Adjency(data_set,euclidian_distance)
e=threshold(Adjency_dist)
AdjencyMarix=Adjency(Adjency_dist,e)

A_to_show=AdjencyMarix.copy()
for i in range(0,len(A_to_show)):
    for j in range(0,len(A_to_show)):
        if A_to_show[i][j]>e+1:
            A_to_show[i][j]=1
        else :
            A_to_show[i][j]=A_to_show[i][j]/(int(e)+1)
```

We get the following plot :

```
plt.figure()
plt.imshow(A_to_show)
plt.colorbar()
plt.show()
```



b)

After using the Matrix_D function given in the subject, we implement the C_matrix:

```
def C_matrix(D):
    H=np.eye(len(D)) - np.ones((len(D),len(D)))*1/len(D)
    D=D*D
    a=-0.5*H.dot(D)
    b=a.dot(H)
    return b
```

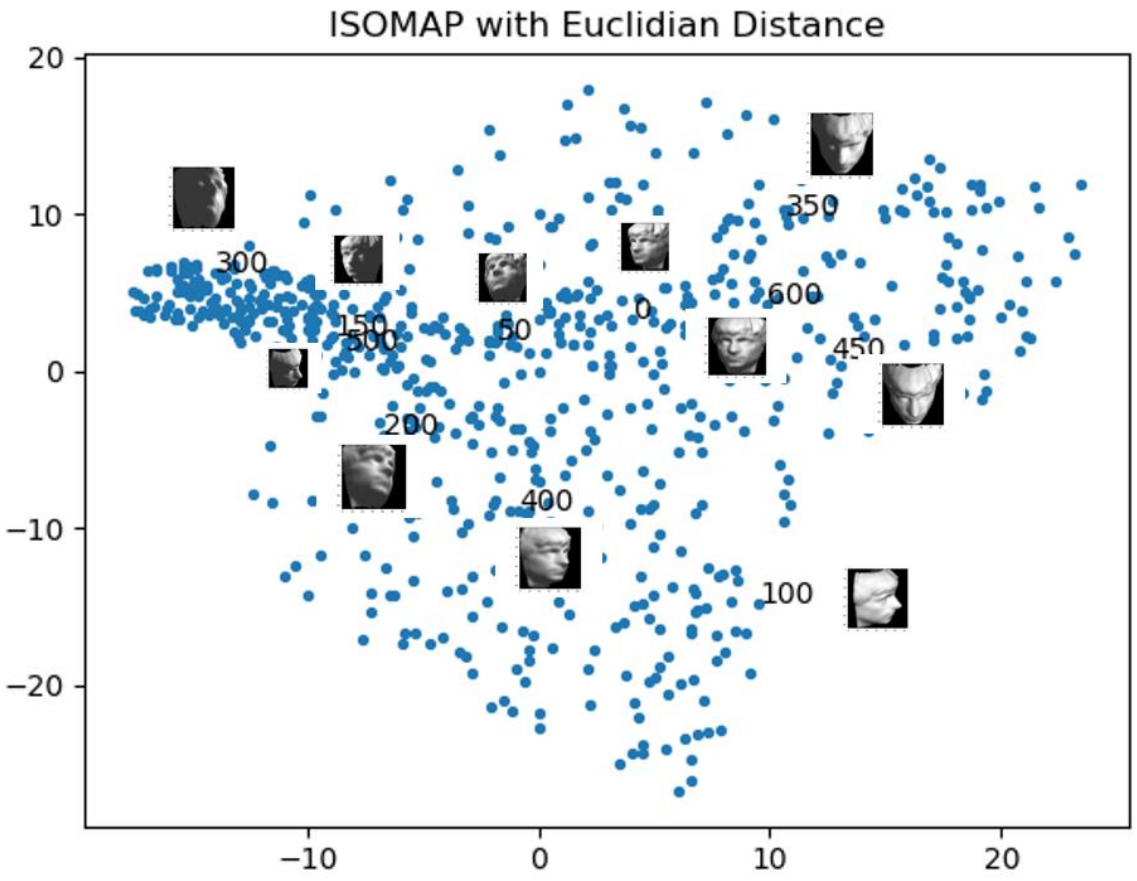
Then, we can easily get the top 2 eigenvalues and their associated eigenvectors:

```
def Eigen_Value_Vect_selected(data_set,k):
    Value, Vector=np.linalg.eig(data_set)
    Vector=np.real(Vector)
    Value=np.real(Value)
    value_sorted=sorted(Value.tolist(),reverse=1)
    Eigen_Vectors_Selected=[]
    for eigVal in value_sorted[0:k]:
        Eigen_Vectors_Selected.append(Vector[:,Value.tolist().index(eigVal)])
    return np.asarray(Eigen_Vectors_Selected), value_sorted[0:k]
```

Now, we implement the 698x2 Z matrix:

```
def Reduced_Z(Eigen_vector, Eigen_values):
    Z=np.zeros(shape=(len(Eigen_values),len(Eigen_values)))
    Z[0][0]=Eigen_values[0]**(0.5)
    Z[1][1]=Eigen_values[1]**(0.5)


    return (Eigen_vector.T).dot(Z)
```

We can now plot the k = 2-dimensional embedding space:



We see that from different region the head is turned on one side to another. For instance, on the top right-hand corner, the head is going down. On the top middle zone, the head is flipping to the left side and from the right-hand corner to the bottom, the head is flipping to the right side. Each image are grouped by region on the 2 dimension plot depending on the orientation of the face. The plot is ordering the faces.

b)

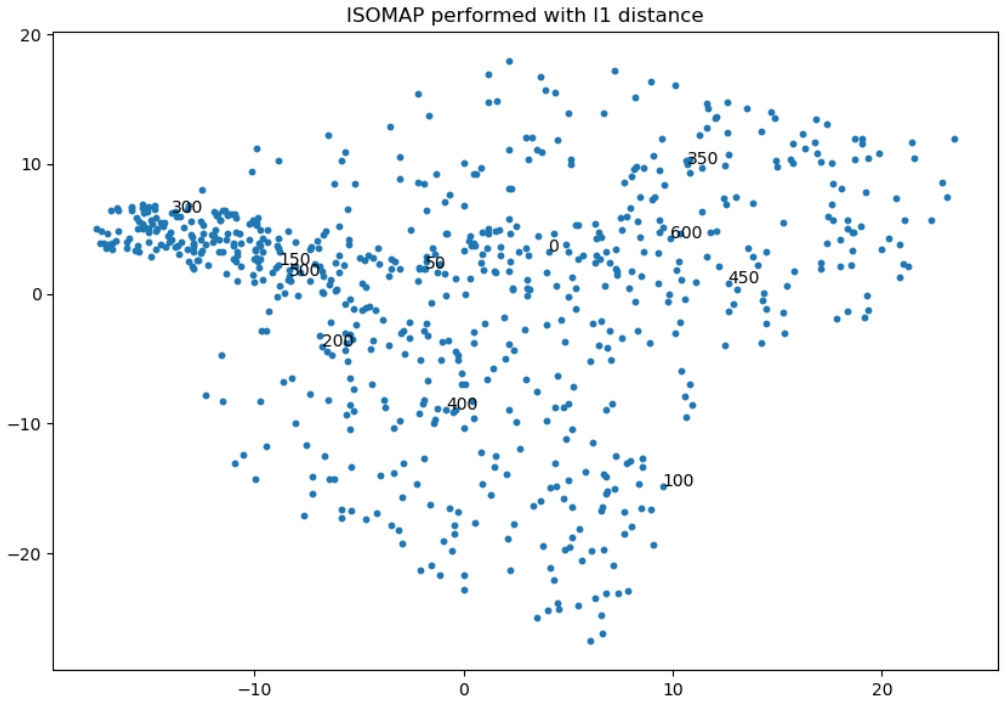
For calculating the Manhattan distance, we use this function:

```
def manhattan_dist(x,y):
    x=np.asarray(x)
    y=np.asarray(y)
    a=abs(x-y)
    a=a.tolist()
    return sum(a)
```

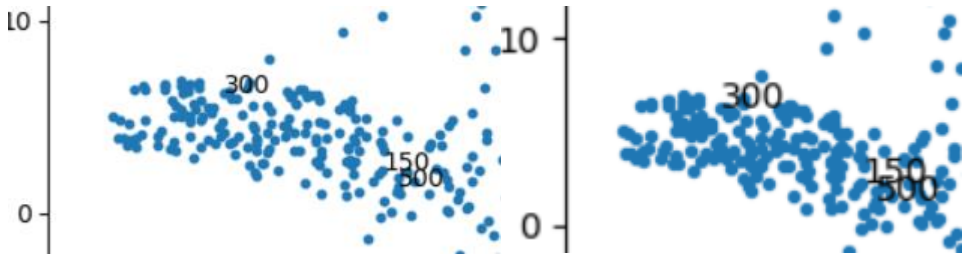
Then using the same procedure as the question b) we get:

```
Adjency_l1=Creation_Adjency(data_set,manhattan_dist)
e1=threshold(Adjency_l1)
AdjencyMarix_l1=Adjency(Adjency_l1,e1)

D2=Matrix_D(AdjencyMarix_l1)
C_matrix2=C_matrix(D2)
Eigen_vector2, Eigen_values2=Eigen_Value_Vect_selected(C_matrix2,2)

Z2=Reduced_Z(Eigen_vector2, Eigen_values2)
Z2=Z2.tolist()

X2=[]
Y2=[]
for points in Z:
    X2.append(points[0])
    Y2.append(points[1])
plt.figure()
for i in points_selected:
    plt.annotate(str(i),(X2[i],Y2[i]))
a=plt.scatter(X2,Y2,s=10)
plt.title('ISOMAP performed with l1 distance')
plt.show()
```

We obtain this plot:



Globally, we have the same shape as the scatter plot with the euclidian distance. However, the top left-hand corner is more spread than the Euclidian plot.



We can notice that the labeled points are still in the same areas as the euclidian plot. Therefore, in this example, choosing a different similarity seems to not impact the result.

d)

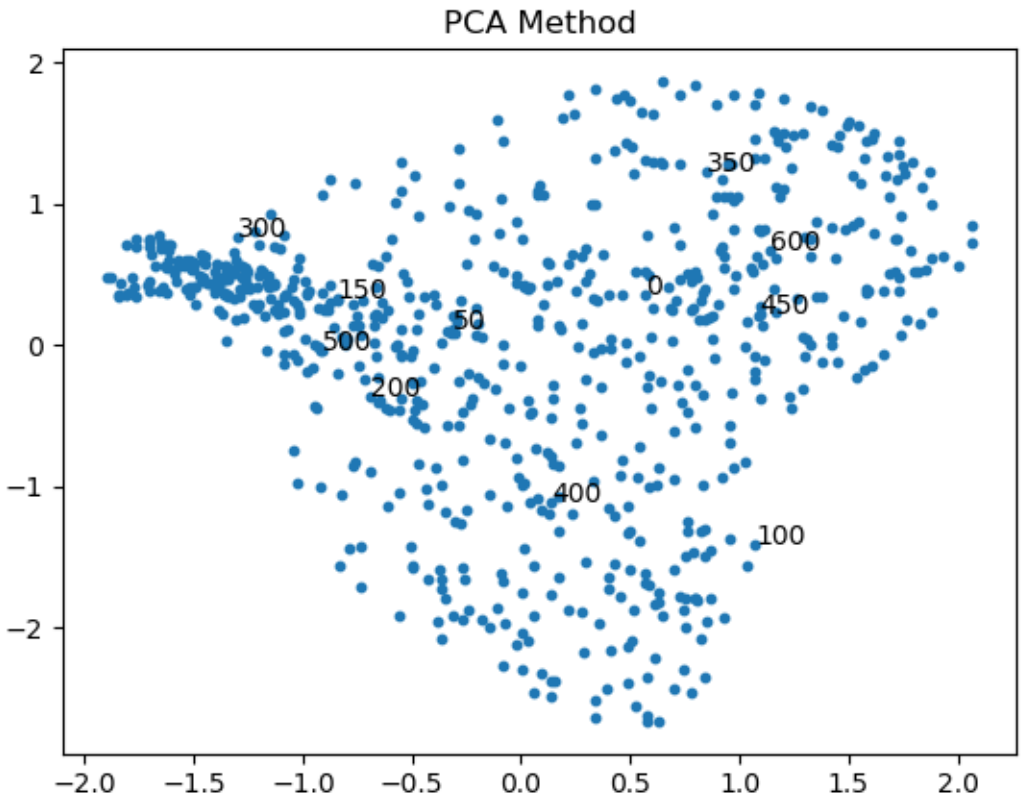
We implement the PCA methode by creating the Covariance matrix of the data points :

```
mean_data=data_set.mean(axis=1)
Row=[]
for row in range(0,len(data_set.iloc[0,:])):
    Row.append(np.array(data_set.iloc[:,row].tolist())-np.asanyarray(mean_data).T)

X=np.asarray(Row)
Xt=X.T

Cov=(1/len(data_set.iloc[0,:]))*Xt.dot(X)
```

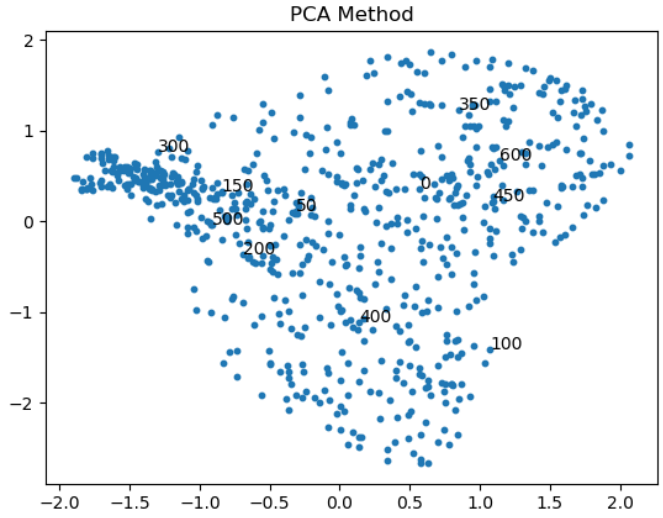
Then after getting the top 2 eigen values and their associated eigen vectors we can perform PCA in two dimension for this data set :

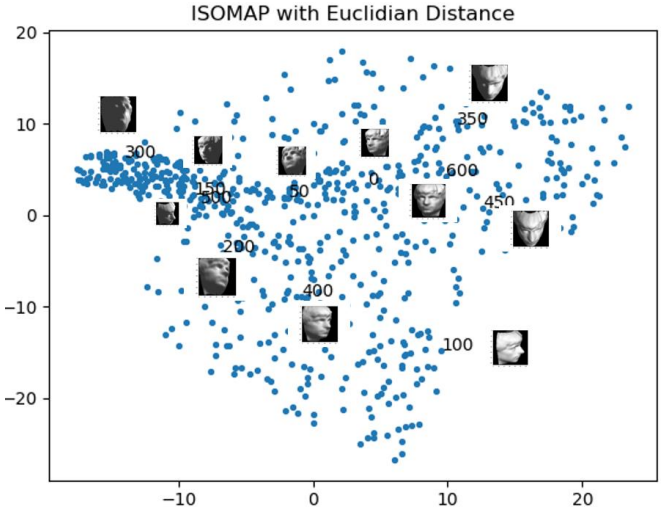
```
Eigen_vector3, Eigen_values3=Eigen_Value_Vect_selected(Cov,2)

reduced_representation=[]

for data_point in Row:
    reduced_representation.append([Eigen_vector3[i].dot(data_point)/m.sqrt(Eigen_values3[i]) for i in range(0,2)])

X3=[]
Y3=[]
for points in reduced_representation:
    X3.append(points[0])
    Y3.append(points[1])
plt.figure()
for i in points_selected:
    plt.annotate(str(i),(X3[i],Y3[i]))
plt.scatter(X3,Y3,s=10)
plt.title('PCA Method')
plt.show()
```

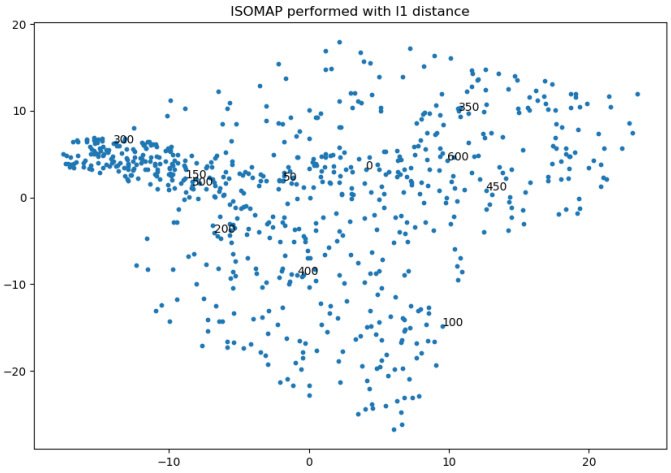
We obtain :


PCA Method
2
1
0
−1
−2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
350
300
600
150
0
450
50
500
200
400
100

Then in this exercice we had these three plots :







The projection with PCA has globally the same shape as the ISOMAP one. In addition, we note that the labeled points are still in the same areas as the ISOMAP plots. Therefore, projection using PCA instead of ISOMAP seems not in this example show a more meaningful projectionas all the projection are ordering the picture by the orientation of the face.

(Bonus) Eigenfaces and simple face recognition

1)

We first import all the images except the test images and we put them in a list by subject as follow:

```
Subject01_1 = Image.open("yalefaces\\subject01.glasses.gif")
Subject01_2 = Image.open("yalefaces\\subject01.happy.gif")
Subject01_3 = Image.open("yalefaces\\subject01.leftlight.gif")
Subject01_4 = Image.open("yalefaces\\subject01.noglasses.gif")
Subject01_5 = Image.open("yalefaces\\subject01.normal.gif")
Subject01_6 = Image.open("yalefaces\\subject01.rightlight.gif")
Subject01_7 = Image.open("yalefaces\\subject01.sad.gif")
Subject01_8 = Image.open("yalefaces\\subject01.sleepy.gif")
Subject01_9 = Image.open("yalefaces\\subject01.surprised.gif")
Subject01_10 = Image.open("yalefaces\\subject01.wink.gif")

Subject02_1 = Image.open("yalefaces\subject02.glasses.gif")
Subject02_2 = Image.open("yalefaces\\subject02.happy.gif")
Subject02_3 = Image.open("yalefaces\\subject02.leftlight.gif")
Subject02_4 = Image.open("yalefaces\\subject02.noglasses.gif")
Subject02_5 = Image.open("yalefaces\\subject02.normal.gif")
Subject02_6 = Image.open("yalefaces\\subject02.rightlight.gif")
Subject02_7 = Image.open("yalefaces\\subject02.sad.gif")
Subject02_8 = Image.open("yalefaces\\subject02.sleepy.gif")
Subject02_9 = Image.open("yalefaces\\subject02.wink.gif")

Subject1=[Subject01_1,Subject01_2,Subject01_3,Subject01_4,Subject01_5,Subject01_6,
          Subject01_7,Subject01_8,Subject01_9,Subject01_10]
Subject2=[Subject02_1,Subject02_2,Subject02_3,Subject02_4,Subject02_5,Subject02_6,
          Subject02_7,Subject02_8,Subject02_9]
```

Then, we implement the function ‘reducing’ to reduce by 4 the quality of images in a list:

```
def reducing(subject_list):
    Reduced_image_Subject=[]

    count=0

    for i in subject_list:
        Subject1[count]=np.asarray(i)
        count+=1

    for img in subject_list:
        red_im=signal.decimate(np.asarray(img).astype(float),4,n=None,ftype='iir',axis=-1,zero_phase=True)
        red_im=signal.decimate(red_im.T,4,n=None,ftype='iir',axis=-1,zero_phase=True)
        Reduced_image_Subject.append(list(red_im.T.reshape(80*61)))
    Reduced_image_Subject=np.asarray(Reduced_image_Subject)

    return np.asarray(Reduced_image_Subject)
```

```
Array_Image_Sub1=reducing(Subject1)
Array_Image_Sub2=reducing(Subject2)
```

The objective now is to compute the covariance matrix for both subjects:

```
Subject1_DF=pd.DataFrame(Array_Image_Sub1)
Subject2_DF=pd.DataFrame(Array_Image_Sub2)

Row_Subject1=[]
Row_Subject2=[]
mean_Subject1=Subject1_DF.mean(axis=0)
mean_Subject2=Subject2_DF.mean(axis=0)

for row in range(0,len(Subject1_DF.index)):
    Row_Subject1.append(np.array(Subject1_DF.iloc[row,:].tolist())-np.asanyarray(mean_Subject1).T)

for row in range(0,len(Subject2_DF.index)):
    Row_Subject2.append(np.array(Subject2_DF.iloc[row,:].tolist())-np.asanyarray(mean_Subject2).T)

X_Subject1=np.asarray(Row_Subject1)
Xt_Subject1=X_Subject1.T

X_Subject2=np.asarray(Row_Subject2)
Xt_Subject2=X_Subject2.T


Cov_Subject1=(1/len(Subject1_DF.index))*Xt_Subject1.dot(X_Subject1)
Cov_Subject2=(1/len(Subject2_DF.index))*Xt_Subject2.dot(X_Subject2)
```

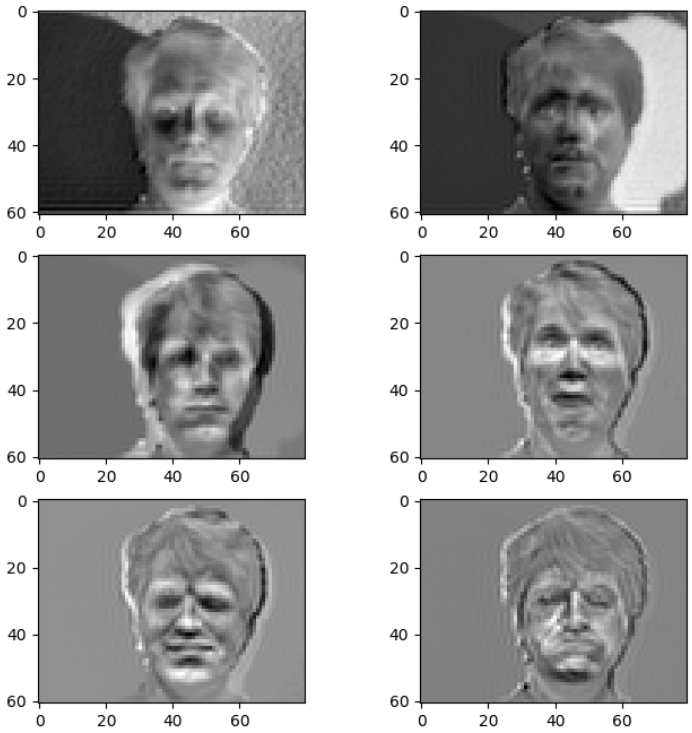
Finally by using the previous function ‘Eigen_Value_Vect_selected’ we obtain the top 6 eigenfaces of both of the subjects:

```
def eigenfaces(k):
    Eigen_vector_S1, Eigen_values_S1=Eigen_Value_Vect_selected(Cov_Subject1,k)
    Eigen_vector_S2, Eigen_values_S2=Eigen_Value_Vect_selected(Cov_Subject2,k)

    Eigenfaces1=[]
    Eigenfaces2=[]

    for i in range (0,k):
        Eigenfaces1.append(Eigen_vector_S1[i])
        Eigenfaces2.append(Eigen_vector_S2[i])

    return  Eigenfaces1,  Eigenfaces2

Eigenfaces1, Eigenfaces2=eigenfaces(6)
```

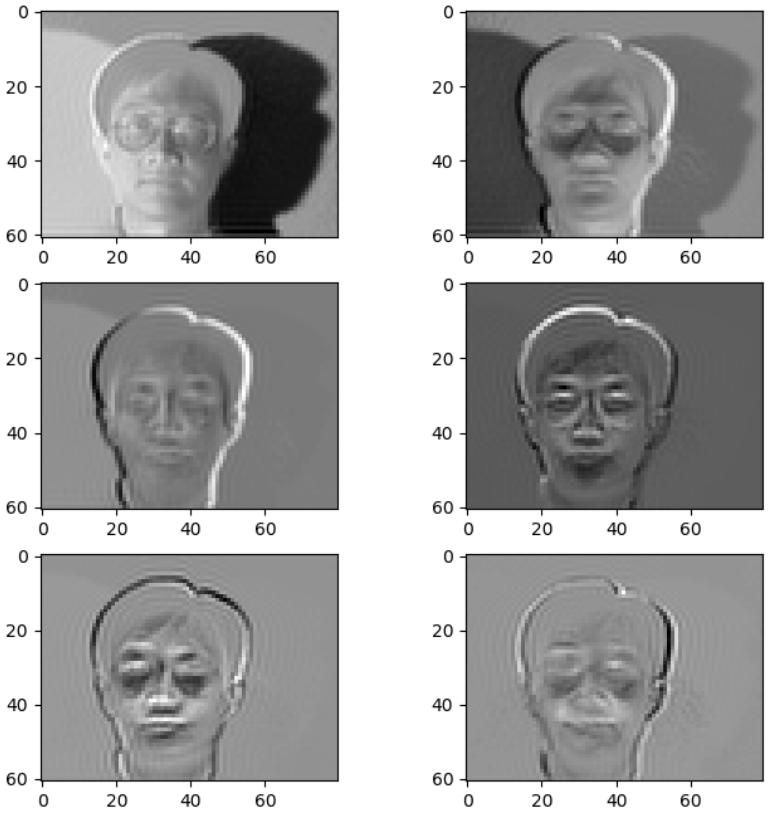
Here is the plot after resizing the vectors:

```
Eigenfaces1, Eigenfaces2=eigenfaces(6)
for i in Eigenfaces1:
    plt.imshow(np.reshape(np.asarray(i),(61,80)),cmap="gray")
    plt.show()
```

Subject 1:



Subject 2:



For the first 6 eigenfaces for each subject we see some face's features highlighted whether it is the glasses, the face shape, the geometry of the face or the light/shade orientation in the picture.

2)

After importing the test images we reduce their size by 4:

```
Subject01_test = [Image.open("yalefaces\\subject01-test.gif")]
Subject02_test = [Image.open("yalefaces\\subject02-test.gif")]

Reduced_image_Subject1=reducing(Subject01_test)
Reduced_image_Subject2=reducing(Subject02_test)
```

Then we implement the function 'score' to calculate the normalized inner product score of the 2 vectorized test images with the vectorized eigenfaces:

```
def scoresub(eigenface,test):
    score=np.vdot(test,np.multiply(eigenface,255))/(np.linalg.norm(np.multiply(eigenface,255))*np.linalg.norm(test))
    return abs(score)
```

si_j: denotes the score of the eigenface i (subject i) with the test image j (corresponding to the subject j)

We get:

```
s1_1=scoresub(Eigenfaces1[0], Reduced_image_Subject1)
s1_2=scoresub(Eigenfaces1[0], Reduced_image_Subject2)
s2_2=scoresub(Eigenfaces2[0], Reduced_image_Subject2)
s2_1=scoresub(Eigenfaces2[0], Reduced_image_Subject1)
```

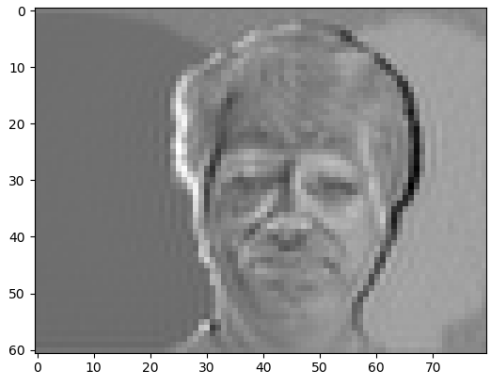
s1_1	s1_2	s2_1	s1_2
0.87230	0.69430	0.0820	0.40590

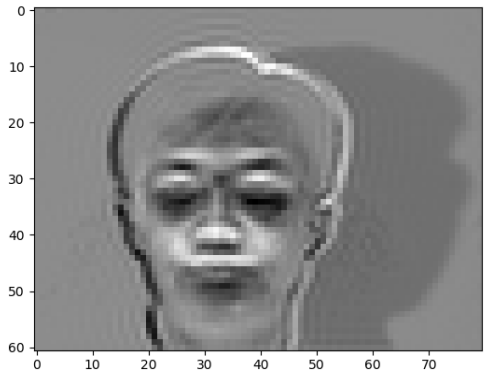
As expected, the score is higher when the subject of the test image is the same as the eigenface chosen. Therefore, it is possible to associate a test image to a subject depending on its score. Higher is the score for one subject, more likely the test image is to be the picture of this specific subject.

However, we see that the difference between the scores, no matter the test image chosen and the subject of the eigenface are close. Thus, associating a test image to a subject can be difficult : the model is not accurate.

To improve this face recognition, we can first take images with higher definition. In addition it is possible to take the mean of the top three eigenfaces of one subject (such that this image is the mean of the top eigenvectors of the subject). This would make a more robust score. However, by doing so we reduce the accuracy of the model as we are mixing images and thus reduce the quality of the eigenface picture.

Here are the mean eigenfaces of subject one and two:





The scores we obtain are:

S1_1	S1_2	S2_1	S1_2
0.4021	0.4062	0.3044	0.4430

With this result we see that for the subject 1, the accuracy of the score is bad as the scores are quite equal for S1_1 and S1_2. Even though for the Subject 2 the model seems to be more accurate, the two scores are still to close predict whether a test image is Subject1 or 2.

Therefore, we might try this approach with high definition images to see if the model is more accurate.