Politechnika Śląska

Data visualization

Bi-clustering laboratory

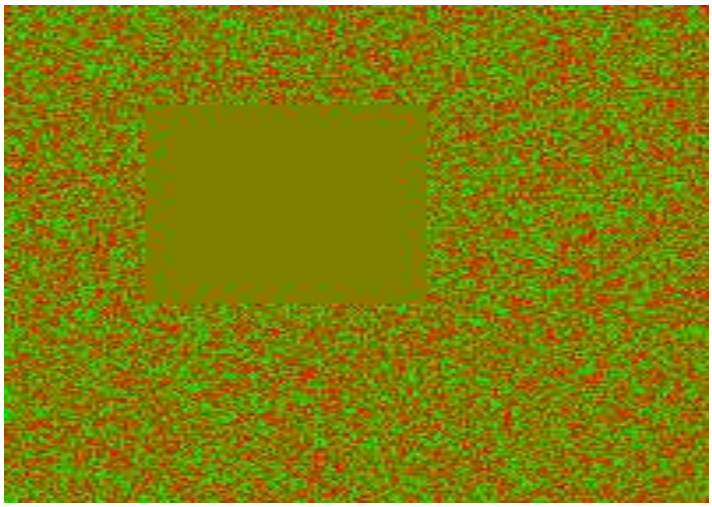
Ankit Rathi

Uwacu Jean Remy

Task – Perform a bi-clustering using NMF-LSE method(int the way described on lecture) and compare result with ground truth.

we have used various examples presented in the data file to perform our task –

Example 1 –



1.png

Description—

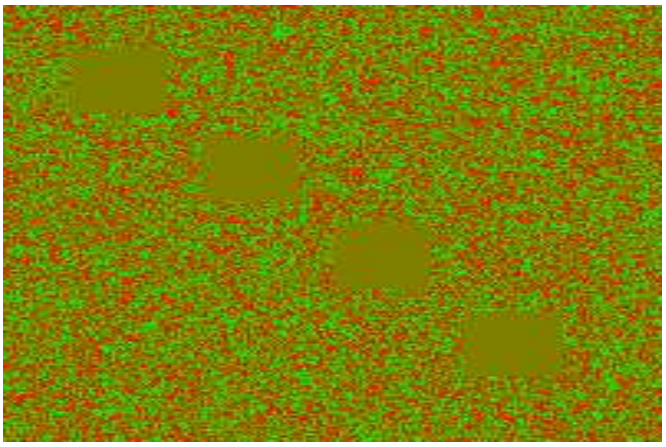
```
Constant values with average
Single bi-cluster (80 - 80 points)
Matrix size 200 - 200 points
Background values randomly chosen from range <0 – 100>
Bi-cluster with constant values set to 50
No noise within the bi-cluster
Data ID: 1
```

Output - -

Jaccard index -- 0.2863674584822367

We have got a low jaccard index in respect of how cluster is visible to us, this probably is due to cluster being equal to mean values of points, it's hard to differentiate for the algorithm.

Example –2



2.png

Description –

```
Constant values with average
4 bi-clusters (75 - 75 points) with exclusive rows and columns
Matrix size 200 - 200 points
Background values randomly chosen from range <0 – 100>
Bi-clusters with constant values set to 50
No noise within the bi-cluster
Data ID: 2
```

Output –

After Initialization, LSE: 11307.892378943214

Iter: 1, LSE: 5684.83685915778, Difference: -5541.48229992372

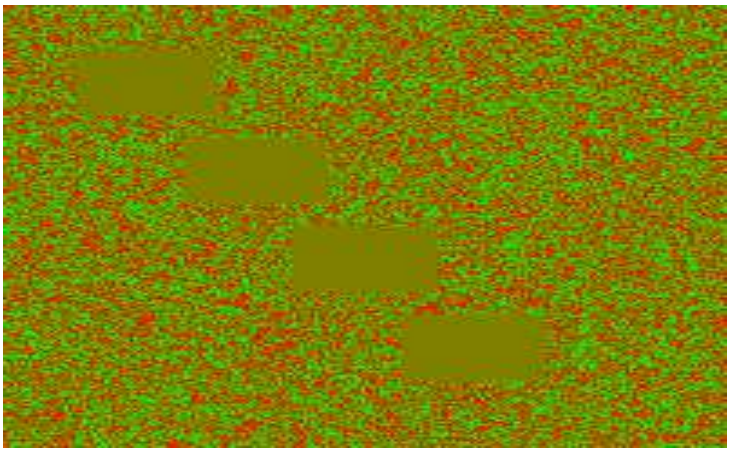
Iter: 2, LSE: 5634.587882835018, Difference: -50.248976322762246

Jaccard index –

0.1065, 0.11936090225563908, 0.11240561825119753, 0.13514610389610388

As we can see there were four clusters in the 2.png so we have got 4 Jaccard index, and the value that we got for it is low, so again it's the same result as 1.png , just the difference is higher number of clusters.

Example 3 –



3.png

Description –

```
Constant values with average
4 bi-clusters (75 - 100 points) overlap on columns 25%
Matrix size 200 - 200 points
Background values randomly chosen from range <0 – 100>
Bi-clusters with constant values set to 50
No noise within the bi-cluster
Data ID: 3
```

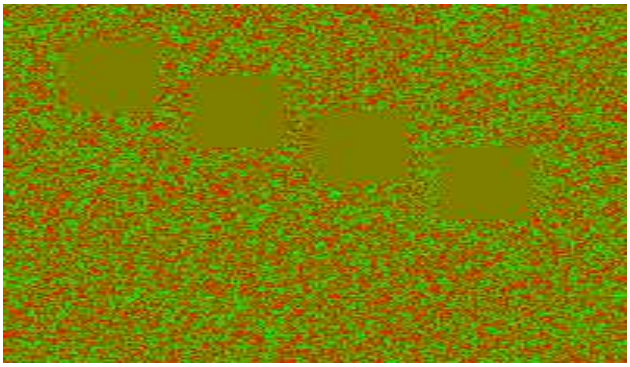
Output –

Jaccard index –

0.139597556593604, 0.17478070175438598, 0.1542368350708367, 0.13373476063038953

Again , it's the same result as the previous parts just with higher number of cluster and we have got a very weak Jaccard index value.

Example – 7



7.png

Description –

```
Constant values with average
4 bi-clusters (100 - 75 points) overlap on rows 50%
Matrix size 200 - 200 points
Background values randomly chosen from range <0 – 100>
Bi-clusters with constant values set to 50
No noise within the bi-cluster
Data ID: 7
```

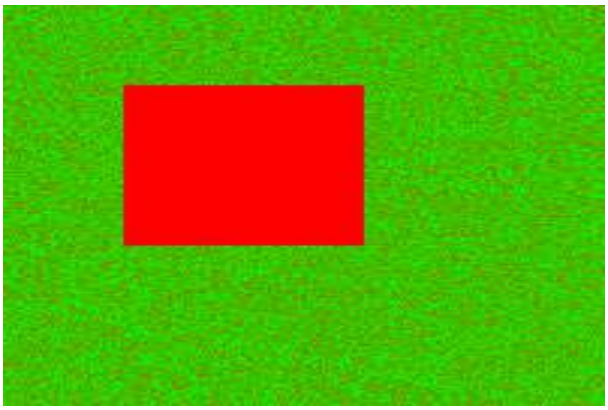
Output –

Jaccard Index –

0.14482306684141547, 0.19130434782608696, 0.13270617981705957, 0.16796212549462974

It's the same situation with this we have weak jaccard index value and it's hard to differentiate the algorithm.

Example –10



10.png

Description –

```
Constant values with upregulated values
Single bi-cluster (80 - 80 points)
Matrix size 200 - 200 points
Background values randomly chosen from range <0 – 100>
Bi-cluster with constant values set to 200
No noise within the bi-cluster
Data ID: 10
```

Output - -

After Initialization, LSE: 19140.22105313805

Iter: 1, LSE: 7059.417811084739, Difference: -12080.803242053313

Iter: 2, LSE: 6614.101649745424, Difference: -445.31616133931493

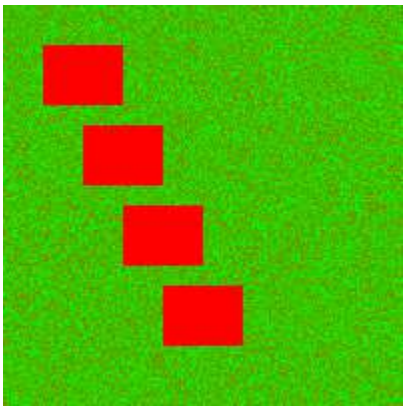
Iter: 3, LSE: 6612.9519655368285, Difference: -1.1496842085953176

Iter: 4, LSE: 6612.949120757566, Difference: -0.002844779262886732

Jaccard index – [1.0]

In this example the cluster was very different from the background in the terms of values, and we can see that algorithm have dealt with the problem fairly easily, and we have received the output we have desired for.

Example 13 - -



13.png

Description –

```
Constant values with upregulated values
4 bi-clusters (75 - 100 points) overlap on columns 50%
Matrix size 200 - 200 points
Background values randomly chosen from range <0 – 100>
Bi-clusters with constant values set to 200
No noise within the bi-cluster
Data ID: 13
```

Output - -

After Initialization, LSE: 17429.401155067764

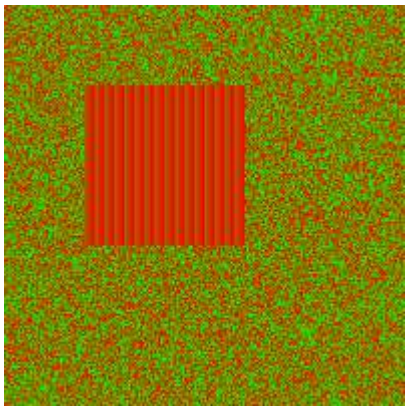
Iter: 1, LSE: 10356.62265242349, Difference: -7072.778502644274

Iter: 2, LSE: 10273.94942012216, Difference: -82.67323230132934

Jaccard Index -- 0.6041666666666666, 0.4, 0.0, 0.0

After taking example with less regular shapes of rectangles we noticed that because they were overlapping 50% on columns, only 1 received good result.

Example – 19



19.png

Description –

```
Constant values on rows
Single bi-cluster (80 - 80 points)
Matrix size 200 - 200 points
Background values randomly chosen from range <0 – 100>
Bi-cluster with constant values on rows set to one of { 60, 70, 80, 90, 100}
No noise within the bi-cluster
Data ID: 19
```

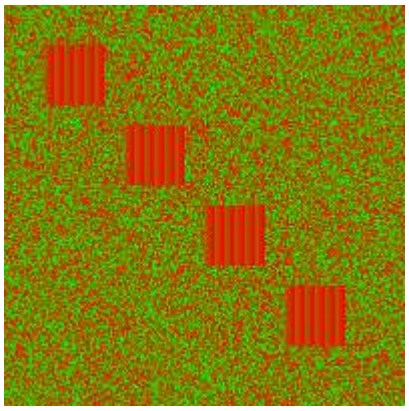
Output –

Jaccard index –

0.94375

we took noisy image. The algorithm performed well, and we have received optimal result for the cluster.

Example – 20



20.png

Description –

```
Constant values on rows
4 bi-clusters (75 - 75 points) with exclusive rows and columns
Matrix size 200 - 200 points
Background values randomly chosen from range <0 – 100>
Bi-clusters with constant values on rows set to one of { 60, 70, 80, 90, 100}
No noise within the bi-cluster
Data ID: 20
```

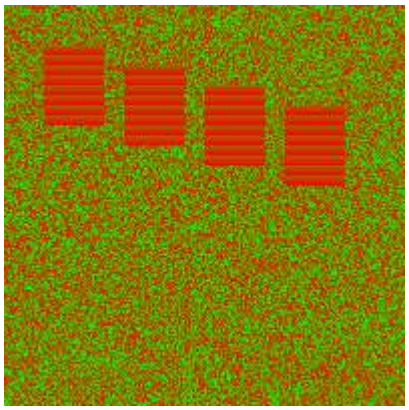
Output –

Jaccard index –

0.084, 0.053, 0.03, 0.089

We have chosen a more noisy image this time, and as we can see algorithm didn't worked nicely at all and we have received poor result for all the clusters present.

Example – 35



35.png

Description –

```
Constant values on columns
4 bi-clusters (100 - 75 points) overlap on rows 75%
Matrix size 200 - 200 points
Background values randomly chosen from range <0 – 100>
Bi-clusters with constant values on columns set to one of { 60, 70, 80, 90,
100}
No noise within the bi-cluster
Data ID: 35
```

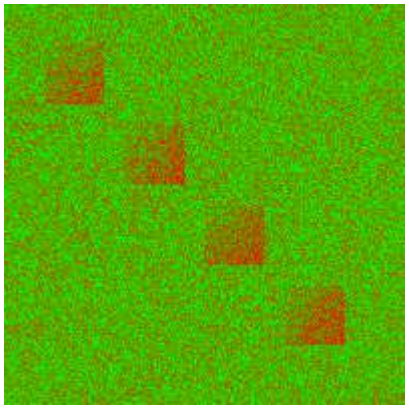
Output –

Jaccard index –

[0.07178014766201804, 0.15231990231990233, 0.2231692677070828, 0.03571428571428571]

Again the algorithm didn't worked nicely, we have received poor cluster.

Example 38 –



38.png

Description –

```
Plaid values
4 bi-clusters (75 - 75 points) with exclusive rows and columns
Matrix size 200 - 200 points
Background values randomly chosen from range <0 – 100>
Bi-clusters with plaid values
No noise within the bi-cluster
Data ID: 38
```

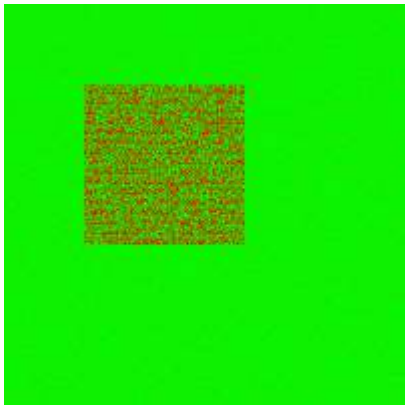
Output –

Jaccard Index –

0.010763888888888889, 0.009051441677588467, 0.02782104534440983, 0.04278197208876687

Again , it's the same situation algorithm isn't working nicely.

Example – 46



46.png

Description –

```
Scaled values
Single bi-cluster (80 - 80 points)
Matrix size 200 - 200 points
Background values randomly chosen from range <0 – 100>
Bi-cluster with scaled values where scale where taken from one of { 6, 7, 8,
9, 10}
No noise within the bi-cluster
Data ID: 46
```

Output –

After Initialization, LSE: 15200.242512234674

Iter: 1, LSE: 6679.223357960478, Difference: -8521.019154274196

Iter: 2, LSE: 6563.480018810064, Difference: -115.7433391504137

Iter: 3, LSE: 6563.162667473623, Difference: -0.3173513364417886

Iter: 4, LSE: 6563.161772359636, Difference: -0.0008951139861892443

Iter: 5, LSE: 6563.161769834791, Difference: -2.5248455131077208e-06

Iter: 6, LSE: 6563.161769827669, Difference: -7.122253009583801e-09

Jaccard index – [1.0]

we took noisy image. The algorithm performed well, and we have received optimal result for the cluster.

CONCLUSION -- After reading the algorithm for different image it's safe to say that if there is high contrast it's safe for algorithm to solve it , whereas for low contrast and images where clusters are overlapping algorithm is working poorly.

Comparison -- After the implementation of the algorithm we checked its performance on a few examples provided by the tutor. Every time we perform updates of W and H until we reach a maximal number of iterations or differences in LSE between consecutive iterations is smaller than a threshold.

Code –

```
'''import pandas as pd
import numpy as np
from sklearn.preprocessing import normalize

# calculate LSE
def lse(V, W, H):
    return np.linalg.norm(V - np.matmul(W, H))

#UPDATE THE RULE

def update_rules(V, W, H):

    WtV = np.matmul(W.T, V)
    WtWH = np.matmul(np.matmul(W.T, W), H)

    H = H * WtV / WtWH

    VHt = np.matmul(V, H.T)
```

```
    WHHt = np.matmul(np.matmul(W, H), H.T)

    W = W * VHt / WHHt

    return W, H

def init_WH(k):
    # k denotes the number of clusters
    # Init W and H matrices with 0.1 value
    #W, H = np.full([V.shape[0], k], 0.1), np.full([k, V.shape[1]], 0.1)

    W, H = np.random.uniform(0, 1, (V.shape[0], k)), np.random.uniform(0, 1, (k, V.shape[1]))
    return W, H

#calculate I and J
def calc_IJ(W, H):

    H_norm = normalize(H.T, axis=0).T
    W_norm = normalize(W, axis=0).T

    H_mean = np.mean(H_norm)
    W_mean = np.mean(W_norm)

    I = [np.where(H_norm[i] >= H_mean)[0]for i in range(H_norm.shape[0])]
    J = [np.where(W_norm[i] >= W_mean)[0]for i in range(W_norm.shape[0])]

    idx = np.argsort([ix[0] for ix in I])

    I = [I[i] for i in idx]
```

```
    J = [J[j] for j in idx]


    return I, J


def Jaccard(I1, J1, I2, J2):
    Jacc = []
    for i in range (len(I1)):
        i1, i2, j1, j2 = set(I1[i]), set(I2[i]), set(J1[i]), set(J2[i])
        Jacc.append(((len(i1.intersection(i2)) / len(i1.union(i2))) + (len(j1.intersection(j2)) /
len(j1.union(j2))))/2)


    return Jacc


np.random.seed(0)
dataset_no = 2


max_iter = 10000
threshold = 1e-6
'''
k = pd.read_csv('/Users/rathi/Downloads/data/1.vmatrix', delimiter='\t',
nrows=1,header=None).values[0][0]
df = pd.read_csv('/Users/rathi/Downloads/data/1.vmatrix', delimiter='\t', header = None, skiprows=[i
for i in range (k+1)], decimal =',')
GT = np.array(pd.read_csv('/Users/rathi/Downloads/data/1.vmatrix', sep="\t", nrows=k, skiprows=1,
decimal=",", header=None)) V = df.values.astype(np.float)
V = df.values.astype(np.float)


k = pd.read_csv('/Users/rathi/Downloads/data/2.vmatrix', delimiter='\t',
nrows=1,header=None).values[0][0]
df = pd.read_csv('/Users/rathi/Downloads/data/2.vmatrix', delimiter='\t', header = None, skiprows=[i
for i in range (k+1)], decimal =',')
```

```
GT = np.array(pd.read_csv('/Users/rathi/Downloads/data/2.vmatrix', sep="\t", nrows=k, skiprows=1, decimal=",", header=None))

V = df.values.astype(np.float)


k = pd.read_csv('/Users/rathi/Downloads/data/3.vmatrix', delimiter='\t', nrows=1,header=None).values[0][0]

df = pd.read_csv('/Users/rathi/Downloads/data/3.vmatrix', delimiter='\t', header = None, skiprows=[i for i in range (k+1)], decimal =',')

GT = np.array(pd.read_csv('/Users/rathi/Downloads/data/3.vmatrix', sep="\t", nrows=k, skiprows=1, decimal=",", header=None))V = df.values.astype(np.float)

V = df.values.astype(np.float)


k = pd.read_csv('/Users/rathi/Downloads/data/7.vmatrix', delimiter='\t', nrows=1,header=None).values[0][0]

df = pd.read_csv('/Users/rathi/Downloads/data/7.vmatrix', delimiter='\t', header = None, skiprows=[i for i in range (k+1)], decimal =',')

GT = np.array(pd.read_csv('/Users/rathi/Downloads/data/7.vmatrix', sep="\t", nrows=k, skiprows=1, decimal=",", header=None))

V = df.values.astype(np.float)


k = pd.read_csv('/Users/rathi/Downloads/data/10.vmatrix', delimiter='\t', nrows=1,header=None).values[0][0]

df = pd.read_csv('/Users/rathi/Downloads/data/10.vmatrix', delimiter='\t', header = None, skiprows=[i for i in range (k+1)], decimal =',')

GT = np.array(pd.read_csv('/Users/rathi/Downloads/data/10.vmatrix', sep="\t", nrows=k, skiprows=1, decimal=",", header=None))

V = df.values.astype(np.float)'''


k = pd.read_csv('/Users/rathi/Downloads/data/46.vmatrix', delimiter='\t', nrows=1,header=None).values[0][0]

df = pd.read_csv('/Users/rathi/Downloads/data/46.vmatrix', delimiter='\t', header = None, skiprows=[i for i in range (k+1)], decimal =',')
```

```
GT = np.array(pd.read_csv('/Users/rathi/Downloads/data/46.vmatrix', sep="\t", nrows=k, skiprows=1, decimal=",", header=None))

V = df.values.astype(np.float)


# init w and h


W, H = init_WH(k)


#main loop updation


prev = lse(V, W, H)

print(f'After Initialization, LSE:[prev')


for i in range(max_iter):
   W, H  =  update_rules(V, W, H)
   diff = lse(V, W, H) - prev
   prev = lse(V, W, H)
   print(f'Iter: [i+1], LSE: [1se(V, W, H)], Differnace: [diff]')
   if diff > threshold:
      print('LSE differance is less than threshold')
      break


#ground truth


I, J = calc_IJ(W, H)

GT_I = GT[0:k, 1:GT[0, 0]+1]

GT_J = GT[0:k, GT[0, 0]+2:GT.shape[1]]


print(Jaccard(I, J, GT_I, GT_J))'''
```