Omer Moussa

900171920

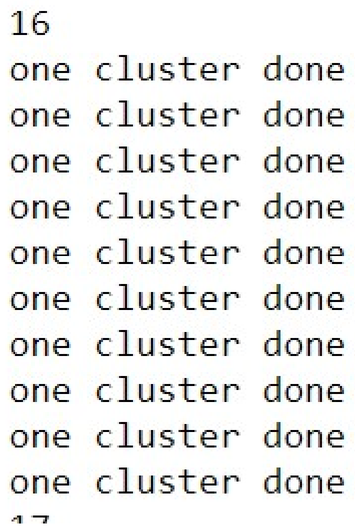
Q1]

1- Code is done by:
   1. Reading the data
   2. Calculate probabilities function
   3. E step
   4. M step
   5. Combine both, initialize and loop for some number of iterations
   6. Print output

   The code is a vectorized implementation and is attached.

   The program takes a long time to run because of the huge dataset and the cost of calculating the inverse and the covariance matrix

2- Done by reading the mnist training dataset, remove the label column, and call the E_M function with 10 clusters. Here is an image for the program while running.



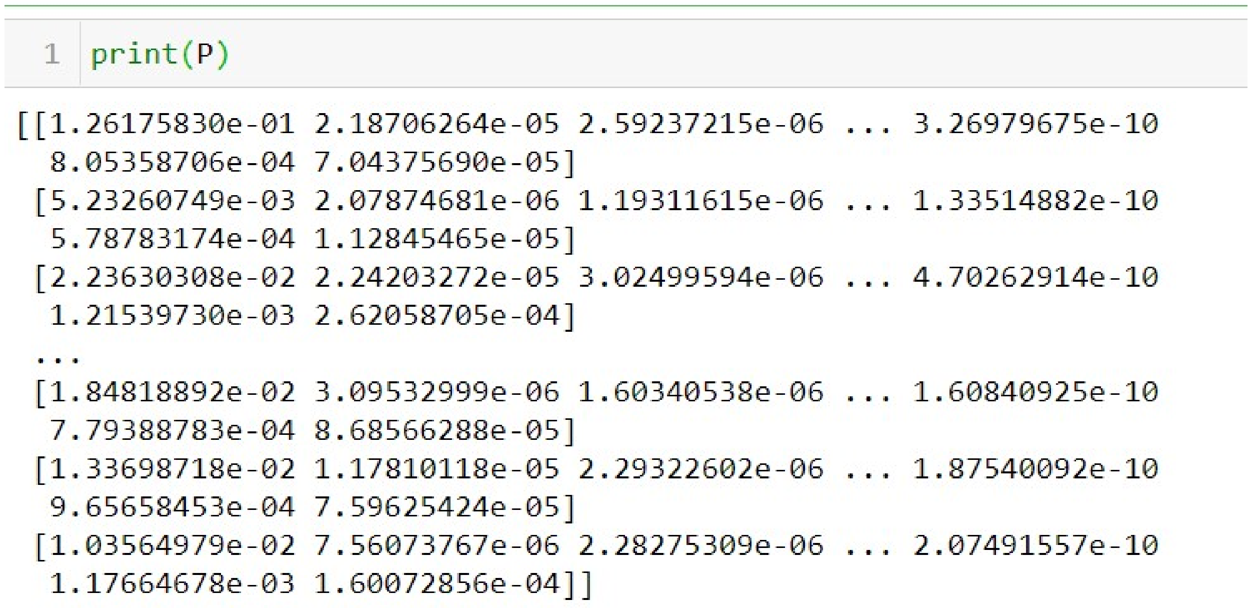
```
16
one cluster done
one cluster done
one cluster done
one cluster done
one cluster done
one cluster done
one cluster done
one cluster done
one cluster done
one cluster done
```

3- This is done by choosing the argmax of the probabilities Array:

```
        for i in range(P.shape[0]):
                Labels.append(np.argmax(P[i]))
```

   Can be found in the code.

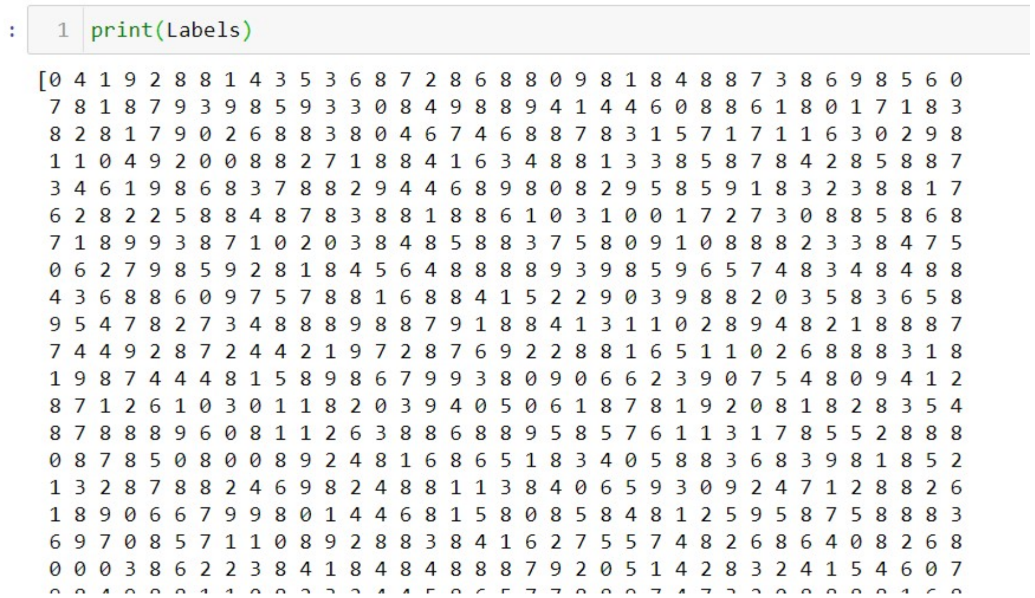
Probabilities look like :



```
print(P)
```

```
[[1.26175830e-01 2.18706264e-05 2.59237215e-06 ... 3.26979675e-10
  8.05358706e-04 7.04375690e-05]
 [5.23260749e-03 2.07874681e-06 1.19311615e-06 ... 1.33514882e-10
  5.78783174e-04 1.12845465e-05]
 [2.23630308e-02 2.24203272e-05 3.02499594e-06 ... 4.70262914e-10
  1.21539730e-03 2.62058705e-04]
 ...
 [1.84818892e-02 3.09532999e-06 1.60340538e-06 ... 1.60840925e-10
  7.79388783e-04 8.68566288e-05]
 [1.33698718e-02 1.17810118e-05 2.29322602e-06 ... 1.87540092e-10
  9.65658453e-04 7.59625424e-05]
 [1.03564979e-02 7.56073767e-06 2.28275309e-06 ... 2.07491557e-10
  1.17664678e-03 1.60072856e-04]]
```

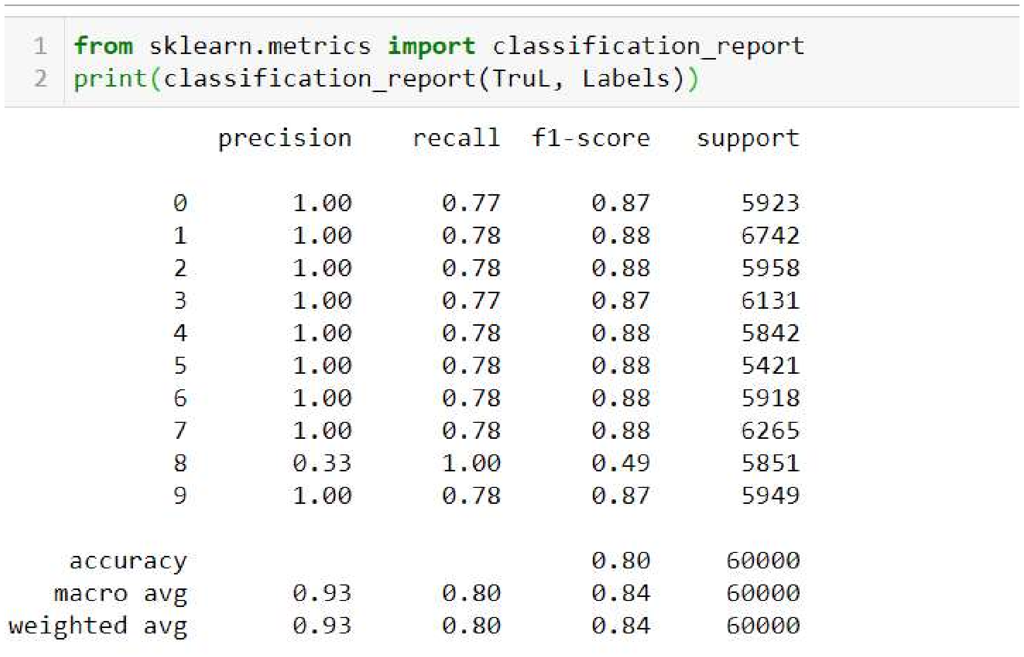
Cluster assignments looks as follow:



```
print(Labels)
```

```
[0 4 1 9 2 8 8 1 4 3 5 3 6 8 7 2 8 6 8 8 0 9 8 1 8 4 8 8 7 3 8 6 9 8 5 6 0
 7 8 1 8 7 9 3 9 8 5 9 3 3 0 8 4 9 8 8 9 4 1 4 4 6 0 8 8 6 1 8 0 1 7 1 8 3
 8 2 8 1 7 9 0 2 6 8 8 3 8 0 4 6 7 4 6 8 8 7 8 3 1 5 7 1 7 1 1 6 3 0 2 9 8
 1 1 0 4 9 2 0 0 8 8 2 7 1 8 8 4 1 6 3 4 8 8 1 3 3 8 5 8 7 8 4 2 8 5 8 8 7
 3 4 6 1 9 8 6 8 3 7 8 8 2 9 4 4 6 8 9 8 0 8 2 9 5 8 5 9 1 8 3 2 3 8 8 1 7
 6 2 8 2 2 5 8 8 4 8 7 8 3 8 8 1 8 8 6 1 0 3 1 0 0 1 7 2 7 3 0 8 8 5 8 6 8
 7 1 8 9 9 3 8 7 1 0 2 0 3 8 4 8 5 8 8 3 7 5 8 0 9 1 0 8 8 8 2 3 3 8 4 7 5
 0 6 2 7 9 8 5 9 2 8 1 8 4 5 6 4 8 8 8 8 9 3 9 8 5 9 6 5 7 4 8 3 4 8 4 8 8
 4 3 6 8 8 6 0 9 7 5 7 8 8 1 6 8 8 4 1 5 2 2 9 0 3 9 8 8 2 0 3 5 8 3 6 5 8
 9 5 4 7 8 2 7 3 4 8 8 8 9 8 8 7 9 1 8 8 4 1 3 1 1 0 2 8 9 4 8 2 1 8 8 8 7
 7 4 4 9 2 8 7 2 4 4 2 1 9 7 2 8 7 6 9 2 2 8 8 1 6 5 1 1 0 2 6 8 8 8 3 1 8
 1 9 8 7 4 4 4 8 1 5 8 9 8 6 7 9 9 3 8 0 9 0 6 6 2 3 9 0 7 5 4 8 0 9 4 1 2
 8 7 1 2 6 1 0 3 0 1 1 8 2 0 3 9 4 0 5 0 6 1 8 7 8 1 9 2 0 8 1 8 2 8 3 5 4
 8 7 8 8 8 9 6 0 8 1 1 2 6 3 8 8 6 8 8 9 5 8 5 7 6 1 1 3 1 7 8 5 5 2 8 8 8
 0 8 7 8 5 0 8 0 0 8 9 2 4 8 1 6 8 6 5 1 8 3 4 0 5 8 8 3 6 8 3 9 8 1 8 5 2
 1 3 2 8 7 8 8 2 4 6 9 8 2 4 8 8 1 1 3 8 4 0 6 5 9 3 0 9 2 4 7 1 2 8 8 2 6
 1 8 9 0 6 6 7 9 9 8 0 1 4 4 6 8 1 5 8 0 8 5 8 4 8 1 2 5 9 5 8 7 5 8 8 8 3
 6 9 7 0 8 5 7 1 1 0 8 9 2 8 8 3 8 4 1 6 2 7 5 5 7 4 8 2 6 8 6 4 0 8 2 6 8
 0 0 0 3 8 6 2 2 3 8 4 1 8 4 8 4 8 8 8 7 9 2 0 5 1 4 2 8 3 2 4 1 5 4 6 0 7
```

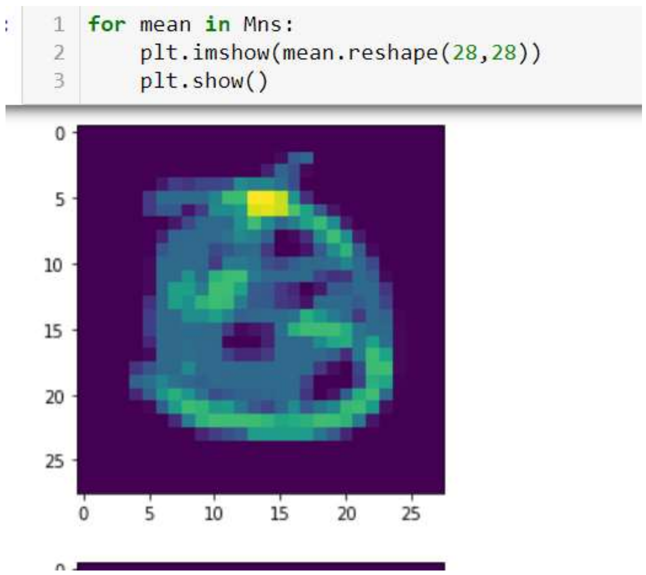
Then we run the classification report:



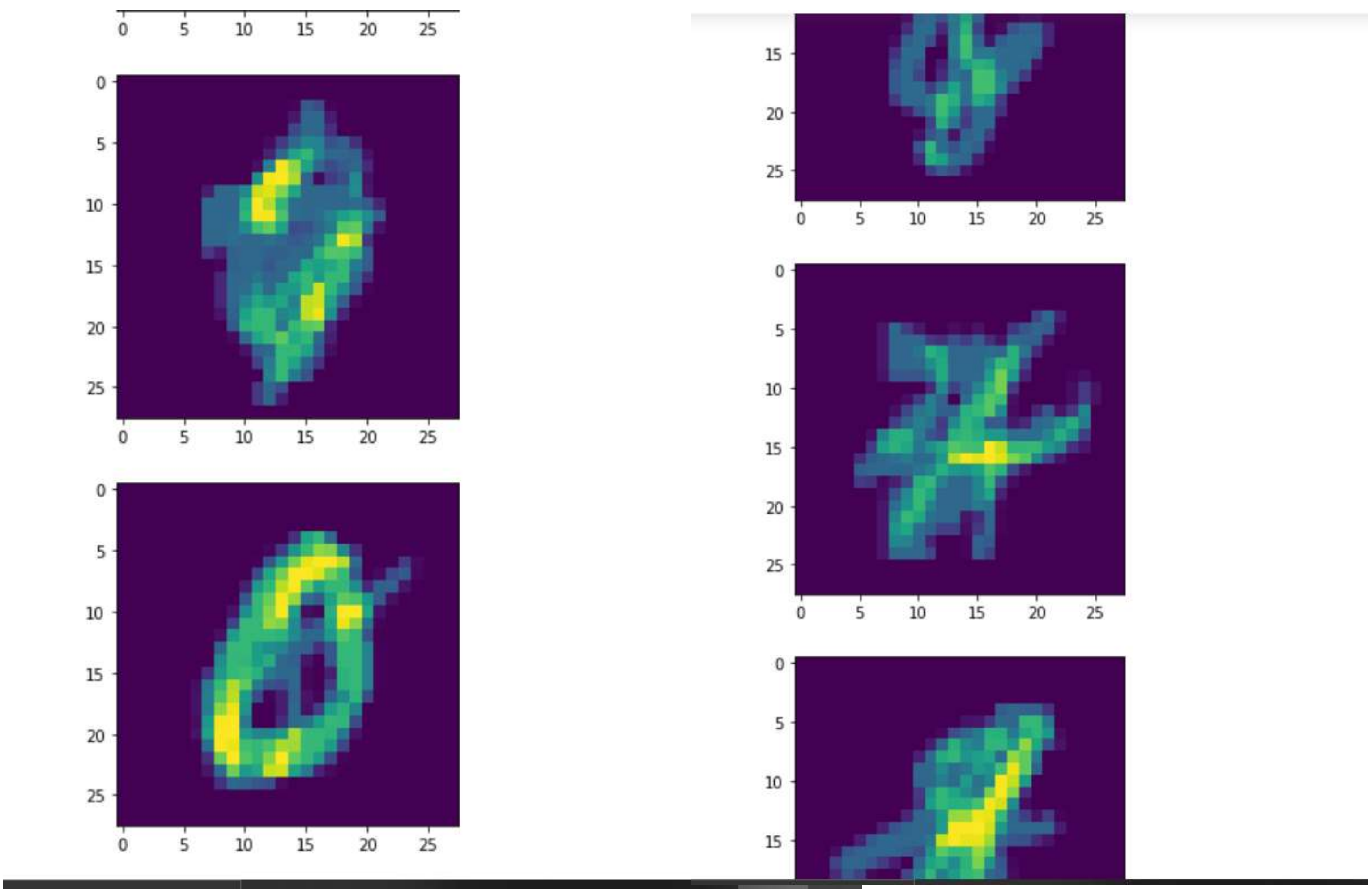
```
from sklearn.metrics import classification_report
print(classification_report(TruL, Labels))
```

```
              precision    recall  f1-score   support

           0       1.00      0.77      0.87      5923
           1       1.00      0.78      0.88      6742
           2       1.00      0.78      0.88      5958
           3       1.00      0.77      0.87      6131
           4       1.00      0.78      0.88      5842
           5       1.00      0.78      0.88      5421
           6       1.00      0.78      0.88      5918
           7       1.00      0.78      0.88      6265
           8       0.33      1.00      0.49      5851
           9       1.00      0.78      0.87      5949

    accuracy                           0.80     60000
   macro avg       0.93      0.80      0.84     60000
weighted avg       0.93      0.80      0.84     60000
```

4- Using imshow of matplotlib, the cluster look something like (not all are shown in the screenshot):



```
for mean in Mns:
    plt.imshow(mean.reshape(28,28))
    plt.show()
```



5-

The implementation of Affinity Prbo is attached. The steps were to follow the equaitions studied in class:

1- Define self similarity measure
2- Measure similarity matrix
3- Update responsibility matrix
4- Update availability matrix
5- Update self availability matrix
6- Choose examplers that are unique.

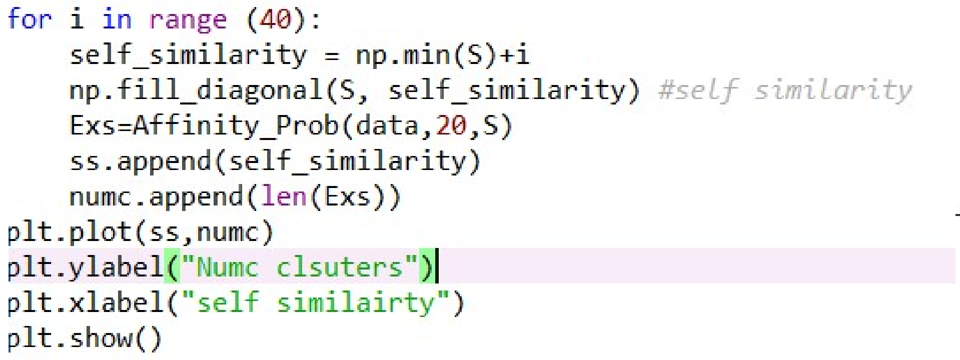
$$r(i,k) \leftarrow s(i,k) - \max_{k' \neq k}\{a(i,k') + s(i,k')\}$$

$$s(i,k) = -\|x_i - x_k\|^2$$ similarity between i and k

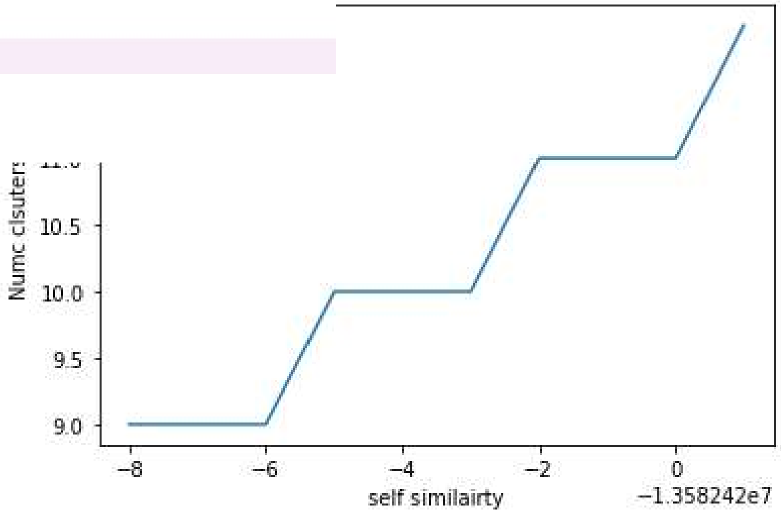
$$a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' \neq i,k} \max\{0, r(i',k)\}\right\}$$

$$a(k,k) \leftarrow \sum_{i' \neq k} \max\{0, r(i',k)\}$$

6- I varied the value of self similarity from the minimum of S matrix over a smaller dataset to check the effect of the self similarity change and plotted the relation of it with the number of clusters. As, we know the number of clusters was minimum when self similarity was initialized to the minimum value of Simalirites. The plot is shown below along with the code. The smaller dataset started with 9 clusters and increased.

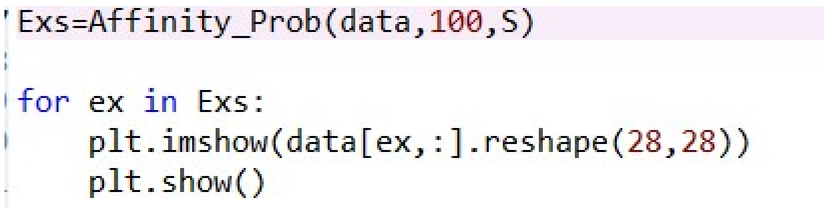


```
for i in range (40):
    self_similarity = np.min(S)+i
    np.fill_diagonal(S, self_similarity) #self similarity
    Exs=Affinity_Prob(data,20,S)
    ss.append(self_similarity)
    numc.append(len(Exs))
plt.plot(ss,numc)
plt.ylabel("Numc clsuters")
plt.xlabel("self similairty")
plt.show()
```



7-

The minimum number of clusters obtained from the dataset (min num of examplers)was 65. I plotted them using the code:



```
Exs=Affinity_Prob(data,100,S)

for ex in Exs:
    plt.imshow(data[ex,:].reshape(28,28))
    plt.show()
```

Some examples of the examplers are: