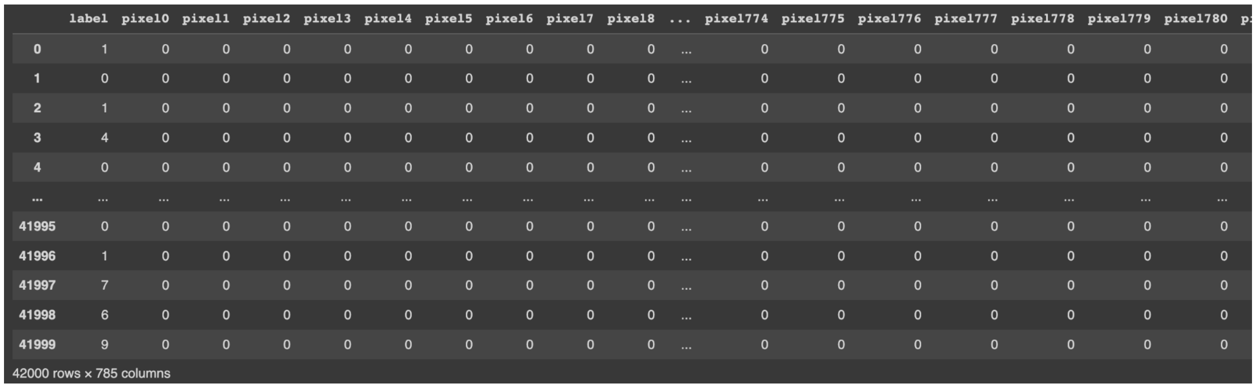
Machine Learning with Big Data
M22MA003

Question 2 :-

Imported the dataset using Google drive.

Edge Pixels have mostly zero values.

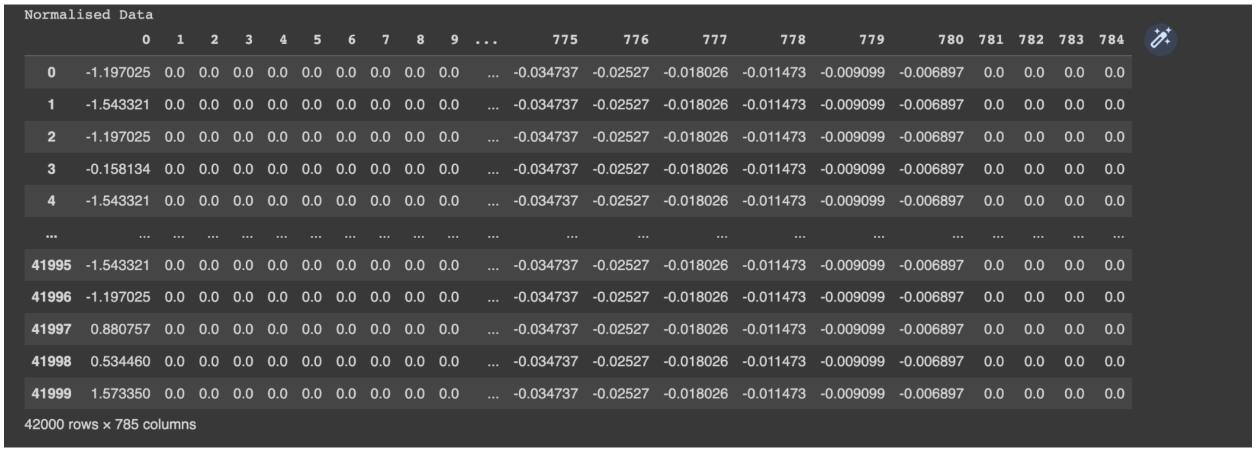


	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
41995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
41996	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
41997	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
41998	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
41999	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0

42000 rows × 785 columns

Preprocess the features using appropriate technique(s), and store them in a file.

Using StandardScaler library, standardized the data.



Normalised Data

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	784
0	-1.197025	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.034737	-0.02527	-0.018026	-0.011473	-0.009099	-0.006897	0.0	0.0	0.0	0.0
1	-1.543321	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.034737	-0.02527	-0.018026	-0.011473	-0.009099	-0.006897	0.0	0.0	0.0	0.0
2	-1.197025	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.034737	-0.02527	-0.018026	-0.011473	-0.009099	-0.006897	0.0	0.0	0.0	0.0
3	-0.158134	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.034737	-0.02527	-0.018026	-0.011473	-0.009099	-0.006897	0.0	0.0	0.0	0.0
4	-1.543321	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.034737	-0.02527	-0.018026	-0.011473	-0.009099	-0.006897	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
41995	-1.543321	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.034737	-0.02527	-0.018026	-0.011473	-0.009099	-0.006897	0.0	0.0	0.0	0.0
41996	-1.197025	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.034737	-0.02527	-0.018026	-0.011473	-0.009099	-0.006897	0.0	0.0	0.0	0.0
41997	0.880757	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.034737	-0.02527	-0.018026	-0.011473	-0.009099	-0.006897	0.0	0.0	0.0	0.0
41998	0.534460	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.034737	-0.02527	-0.018026	-0.011473	-0.009099	-0.006897	0.0	0.0	0.0	0.0
41999	1.573350	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.034737	-0.02527	-0.018026	-0.011473	-0.009099	-0.006897	0.0	0.0	0.0	0.0

42000 rows × 785 columns

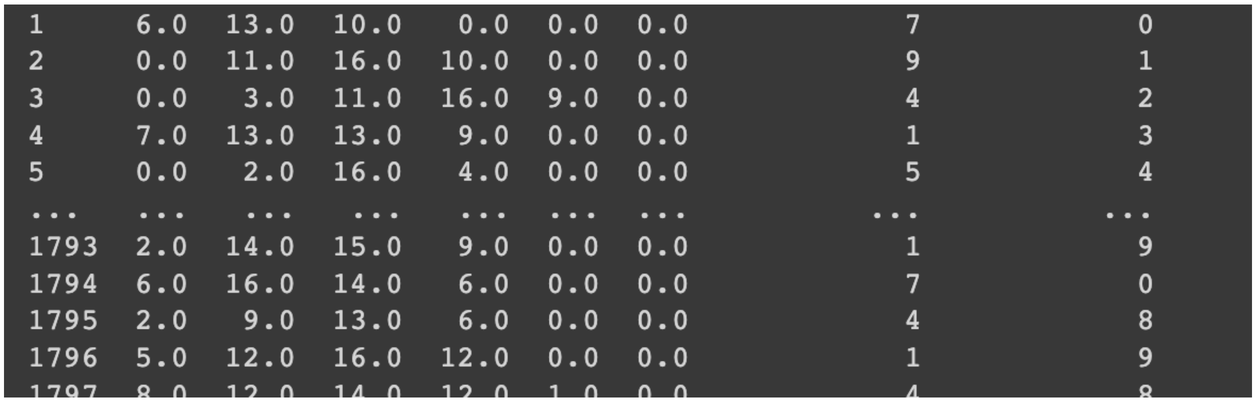
Saved the preprocessed file in google drive. Link
The CURE algorithm implementation is now done using this preprocessed data.

(1) Implement the BFR and CURE clustering algorithms on this data assuming that you can store `K1' samples in the main memory at a time. One may use any existing libraries for performing the initial k-means clustering in case of BFR, and agglomerative clustering in case of

CURE. After this, every step needs to be implemented from scratch.
(2) For the CURE algorithm, keep the number of clusters as 10.

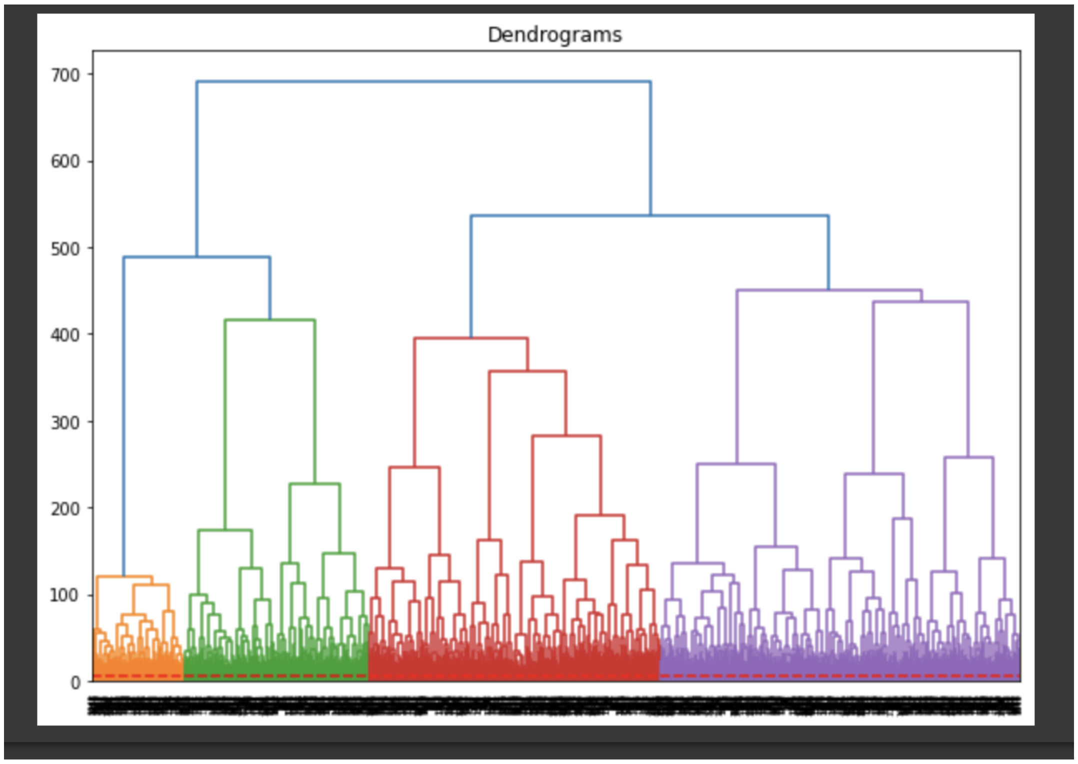
In the implemented method, I have not used any existing libraries to find the initial clusters, instead the hierarchical clustering is also implemented from scratch. Reason being that the existing library of hierarchical/agglomerative clustering will not provide us with the cluster centroids but only the predicted labels. It was also observed that there was too much difference between the predicted values of labels and true labels when hierarchical clustering was used.

To confirm the high difference when existing library for hierarchical clustering is used:-



```
1      6.0  13.0  10.0   0.0  0.0  0.0            7            0
2      0.0  11.0  16.0  10.0  0.0  0.0            9            1
3      0.0   3.0  11.0  16.0  9.0  0.0            4            2
4      7.0  13.0  13.0   9.0  0.0  0.0            1            3
5      0.0   2.0  16.0   4.0  0.0  0.0            5            4
...    ...   ...   ...   ...  ...  ...          ...          ...
1793   2.0  14.0  15.0   9.0  0.0  0.0            1            9
1794   6.0  16.0  14.0   6.0  0.0  0.0            7            0
1795   2.0   9.0  13.0   6.0  0.0  0.0            4            8
1796   5.0  12.0  16.0  12.0  0.0  0.0            1            9
1797   8.0  12.0  14.0  12.0  1.0  0.0            4            8
```

Dendrogram created using hierarchy:-

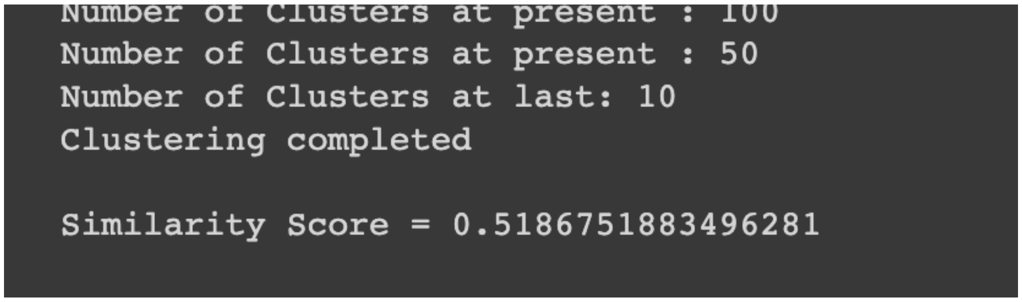


Implementation of Hierarchical Clustering:-

1. Take each input point as a cluster.
2. Find distances between two instances of the input data points or clusters. Basically, it is finding the distance between from one centroid point to every other centroid point.
3. Merge the clusters which are closer to each other in the given range.
4. Loop the process until we reach our desired number of clusters.

(3) After clustering, calculate the percentage of samples from each class and convert it into probability values. Using these, calculate the entropy of each cluster. Also calculate the total entropy of all the clusters by summing the entropy of individual clusters.

I have calculated the similarity score between the two outputs, i.e., the predicted label value and the true label values. One can use the entropy method as well to find the probabilities and then calculate the uncertainty of the predicted labels values.

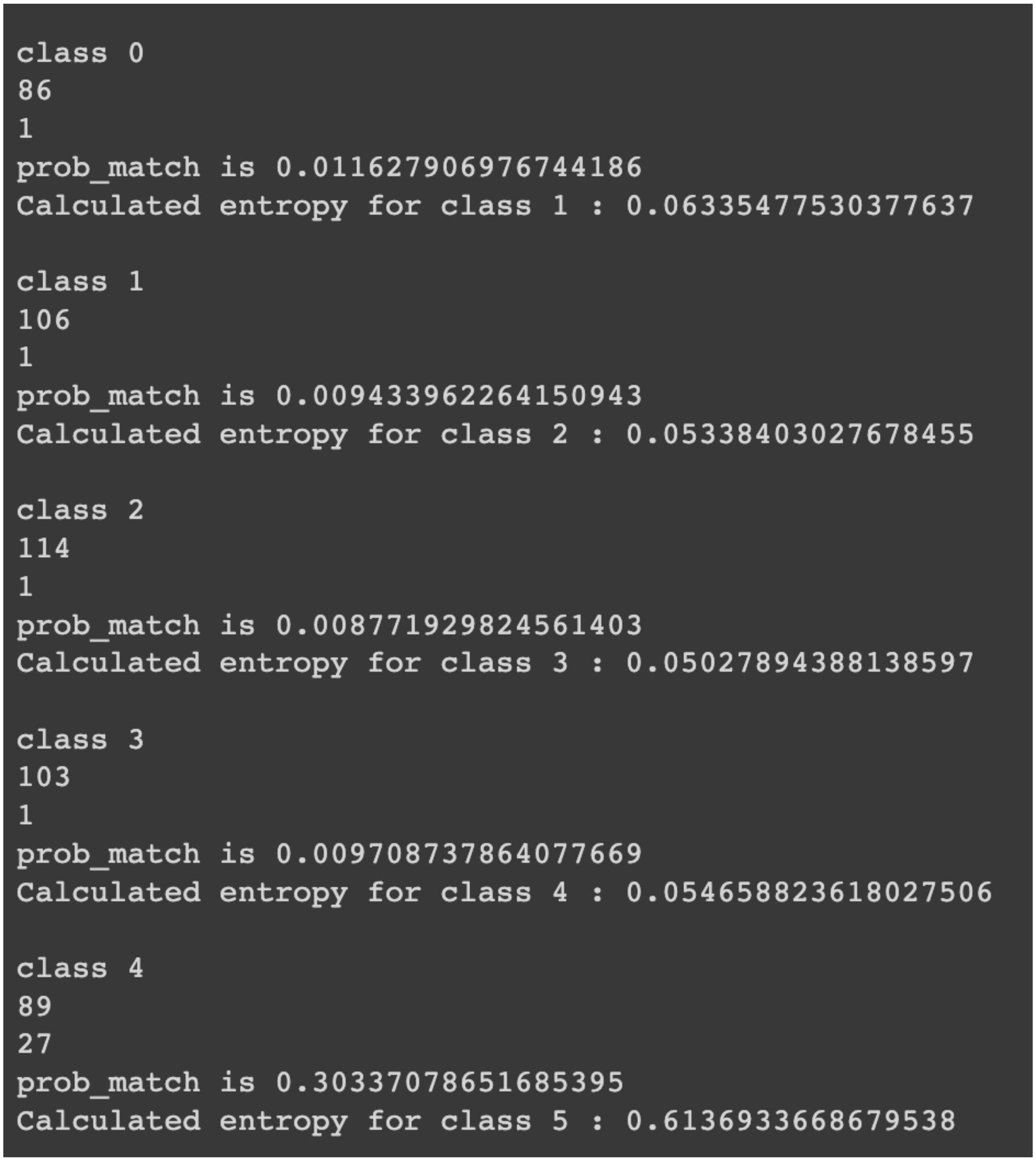


```
Number of Clusters at present : 100
Number of Clusters at present : 50
Number of Clusters at last: 10
Clustering completed

Similarity Score = 0.5186751883496281
```

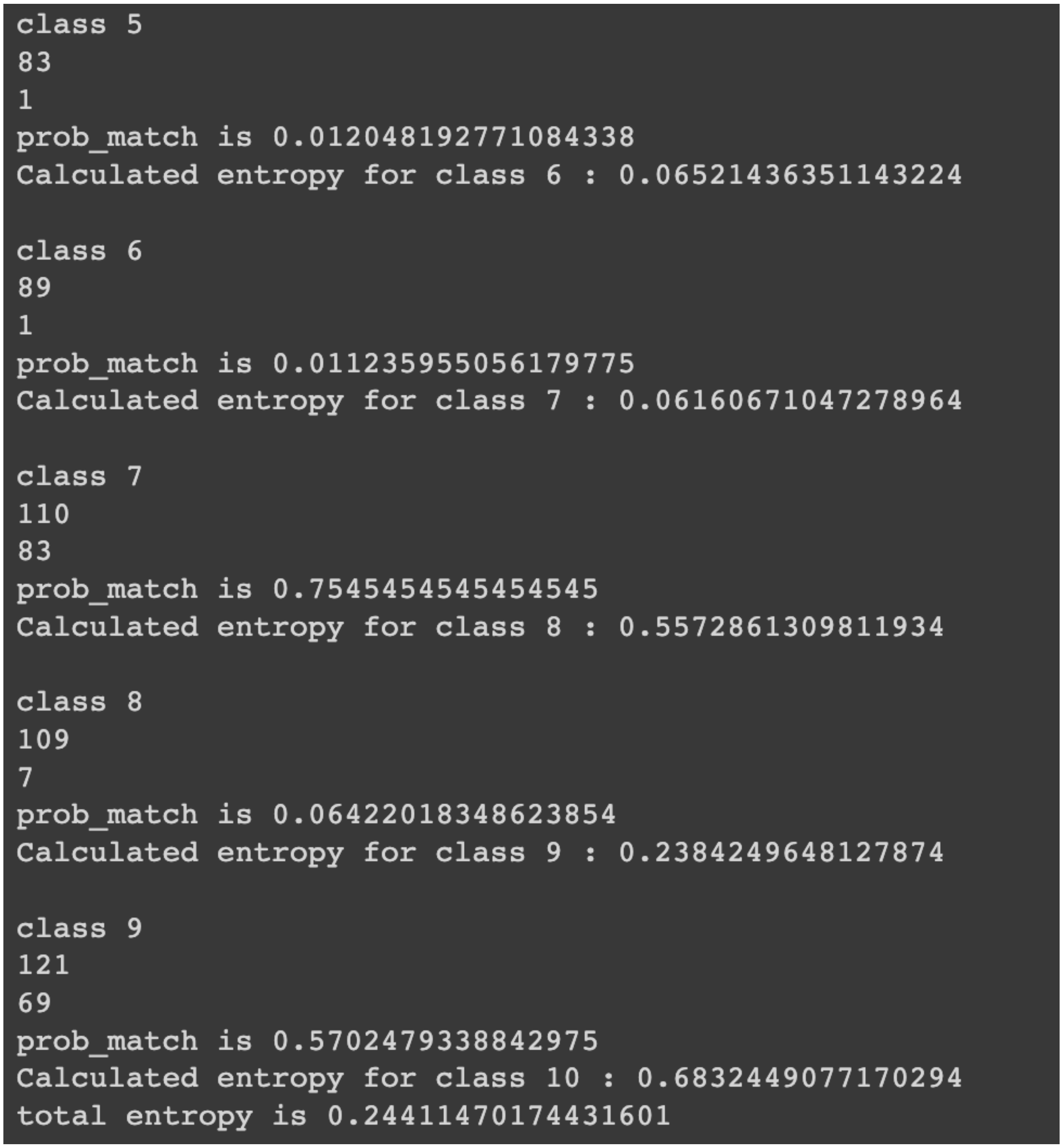
To calculate the entropy of each cluster, the predicted and actual labels are compared to find how many data points were correctly predicted and therefore the probability of the correctly predicted labels are found. Furthermore, entropy is calculated on top of these predicted values.

Later, the total entropy is calculated using the previously calculated cluster wise entropies.

First Usecase : Using K=1000

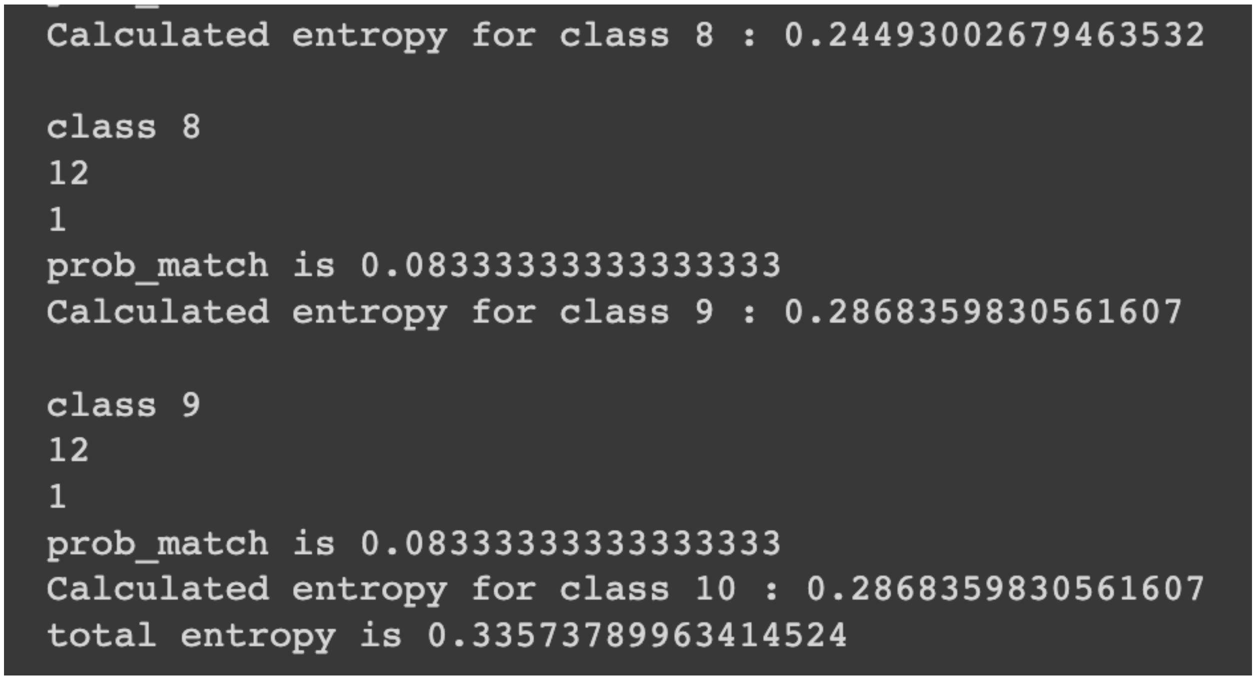


```
class 0
86
1
prob_match is 0.011627906976744186
Calculated entropy for class 1 : 0.06335477530377637

class 1
106
1
prob_match is 0.009433962264150943
Calculated entropy for class 2 : 0.05338403027678455

class 2
114
1
prob_match is 0.008771929824561403
Calculated entropy for class 3 : 0.05027894388138597

class 3
103
1
prob_match is 0.009708737864077669
Calculated entropy for class 4 : 0.054658823618027506

class 4
89
27
prob_match is 0.30337078651685395
Calculated entropy for class 5 : 0.6136933668679538
```



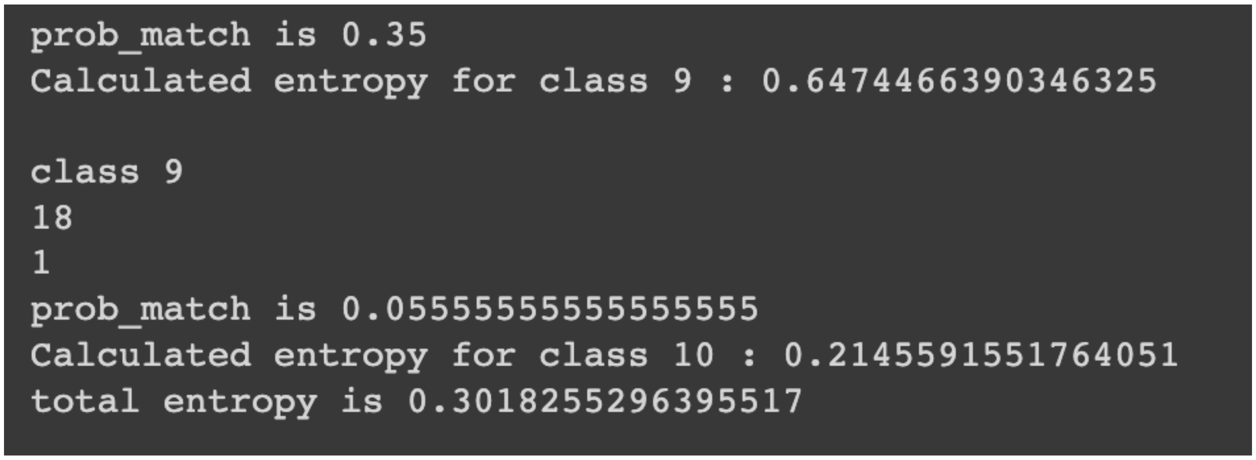
```
class 5
83
1
prob_match is 0.012048192771084338
Calculated entropy for class 6 : 0.06521436351143224

class 6
89
1
prob_match is 0.011235955056179775
Calculated entropy for class 7 : 0.06160671047278964

class 7
110
83
prob_match is 0.7545454545454545
Calculated entropy for class 8 : 0.5572861309811934

class 8
109
7
prob_match is 0.06422018348623854
Calculated entropy for class 9 : 0.2384249648127874

class 9
121
69
prob_match is 0.5702479338842975
Calculated entropy for class 10 : 0.6832449077170294
total entropy is 0.24411470174431601
```

Second Usecase : Using K=100



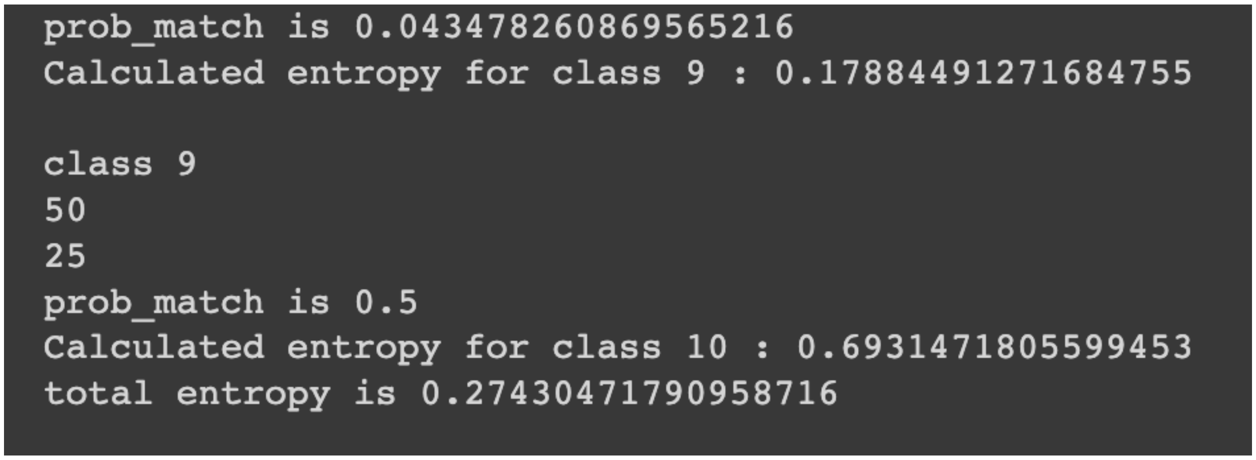
```
Calculated entropy for class 8 : 0.24493002679463532

class 8
12
1
prob_match is 0.08333333333333333
Calculated entropy for class 9 : 0.2868359830561607

class 9
12
1
prob_match is 0.08333333333333333
Calculated entropy for class 10 : 0.2868359830561607
total entropy is 0.33573789963414524
```

Third Usecase : Using K=300



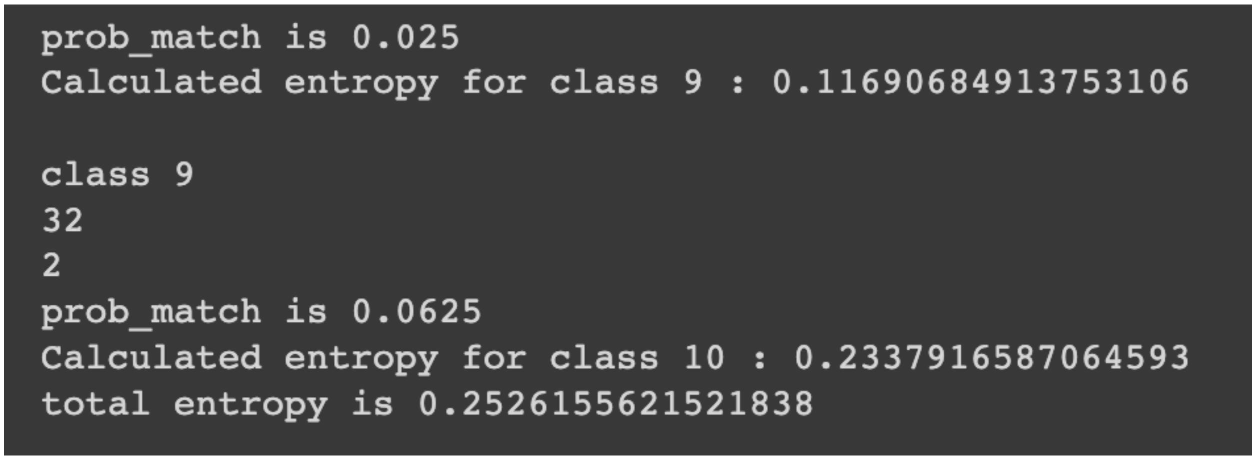
```
prob_match is 0.35
Calculated entropy for class 9 : 0.6474466390346325

class 9
18
1
prob_match is 0.05555555555555555
Calculated entropy for class 10 : 0.2145591551764051
total entropy is 0.3018255296395517
```

Fourth Usecase : Using K=400



```
prob_match is 0.043478260869565216
Calculated entropy for class 9 : 0.17884491271684755

class 9
50
25
prob_match is 0.5
Calculated entropy for class 10 : 0.6931471805599453
total entropy is 0.27430471790958716
```

Fifth Usecase : Using K=500



```
prob_match is 0.025
Calculated entropy for class 9 : 0.11690684913753106

class 9
32
2
prob_match is 0.0625
Calculated entropy for class 10 : 0.2337916587064593
total entropy is 0.2526155621521838
```

- **We can observe that as we are increasing the number of data points in RAM or initial clustering there is a decrease in the entropy/uncertainty value.**
- **Larger the number of data points in initial clustering, better the representatives to be obtained.**

Google Colab Link : [Link]

References: https://github.com/Kchu/CURE-cluster-python