ECE 219 Project 2

Tanner Dulay, Loïc Maxwell, Zoë Tucker

February 3, 2021

1 Question 1

In this assignment, we explore various methods for clustering of text data. To begin, we use the“20 Newsgroups” dataset, taking the two well-separated classes previously defined (“computer technology” and “recreational activity”). As this project uses unsupervised learning, we do not distinguish between training and test data, and instead use the entire dataset throughout the project.

To start our process, we build the TF-IDF matrix. We use `min_df` $= 3$, exclude English stopwords/punctuation, and remove headers and footers, but we do not perform stemming or lemmatization. The resulting TF-IDF matrix is $\mathbf{7882} \times \mathbf{21909}$.

2 Question 2

Next, we apply k-means clustering directly to the TF-IDF data. As we are interested in two classes of documents, we set $k = 2$. The below contingency matrix (Figure 1) shows the results of this.

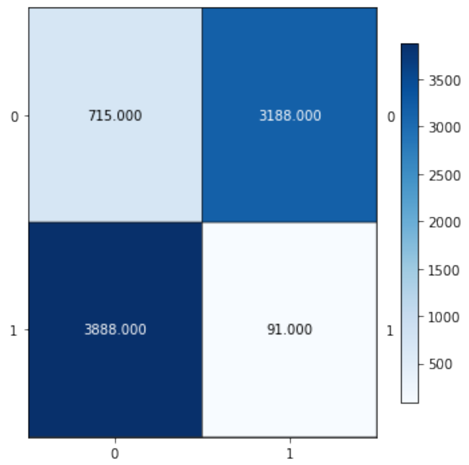


Figure 1: Contingency Matrix for K-Means on TF-IDF Matrix

3 Question 3

We then find the metrics we need to evaluate our classification results. We find the following values:

Homogeneity score	0.5600025412974119
Completeness score	0.5716557539454379
V_measure score	0.5657691483417876
Adjusted Rand Index score	0.6327475635535726
Adjusted Mutual Information Score	0.5657289846874574

4 Question 4

As the metrics found in Question 3 do not demonstrate a good clustering result, we now try dimensionality reduction to help our data better fit the assumptions made by k-means algorithms.

We begin by finding the percent of variance the top r principal components can retain for various values of r. Figure 2 shows our findings.

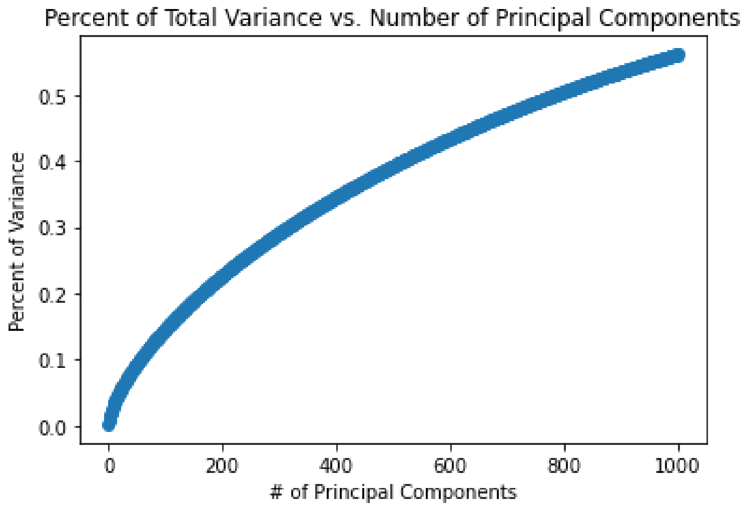


Figure 2: Principal Component Results

5 Question 5

Next, we will try to experimentally find the value of r (i.e. the number of dimensions) that we can reduce our data to in order to obtain the best clustering results. Testing $r \in \{1, 2, 3, 5, 10, 20, 50, 100, 300\}$ using both NMF and SVD, we plot the values of the five metrics found in Question 3 for each r. The results are shown in Figures 3 and 4.

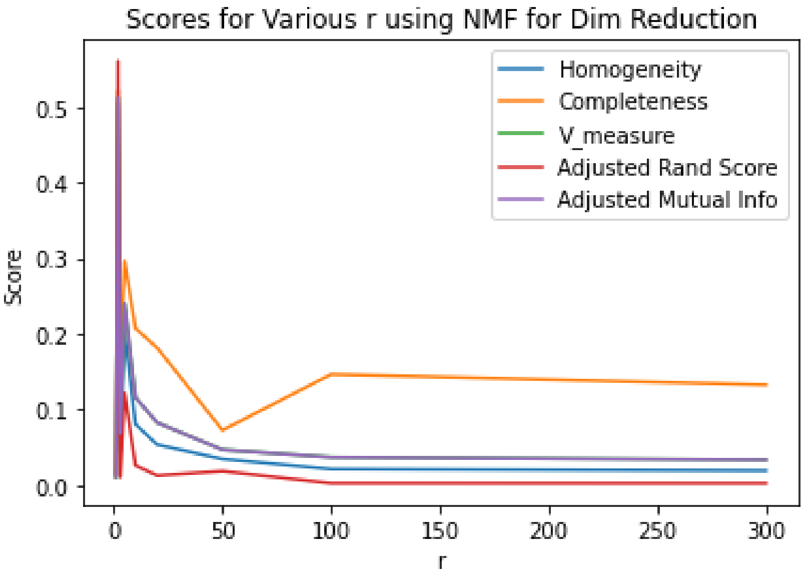


Figure 3: NMF Results

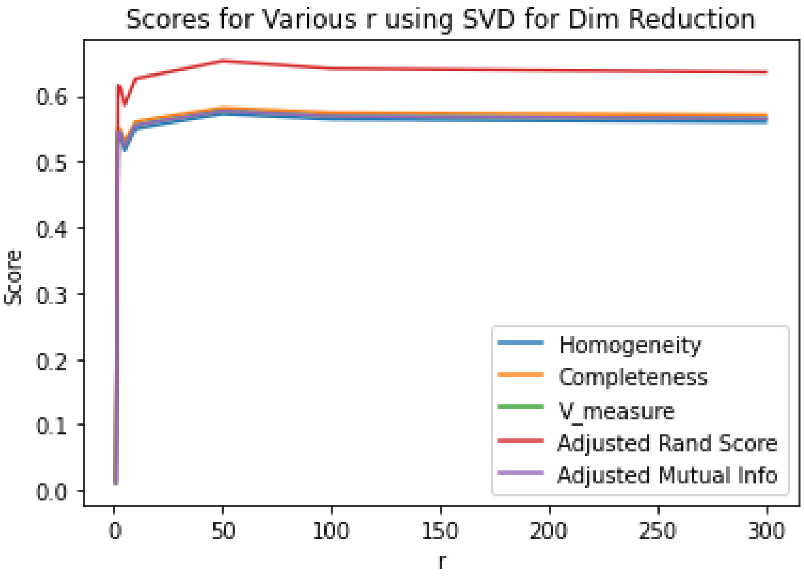


Figure 4: SVD Results

6 Question 6

As we can see, the behavior of the measures as r increases is non-monotonic.

This is due to a trade-off between a larger value of r giving a more faithful representation of the data, and a smaller value of r being better suited for k-means clustering. The higher the dimensionality of a dataset, the more difficult it is to meaningfully evaluate the "distance" between data points, which is what k-means relies on. Thus, we expect there to be an optimal value of r that balances these two concerns and maximizes our scores. In our case, we find that $r = 2$ for NMF, and $r = 50$ for SVD.

7 Question 7

Now, we want to visualize our clustering results. We do this by using k-means clustering with our dataset reduced by NMF and SVD with the optimal values of r found in Question 6, then projecting the data into 2-dimensional space with SVD and coloring the points according to their label. Figures 5 and 6 show the results for NMF with $r = 2$. Figures 7 and 8 show the results for SVD with $r = 50$.

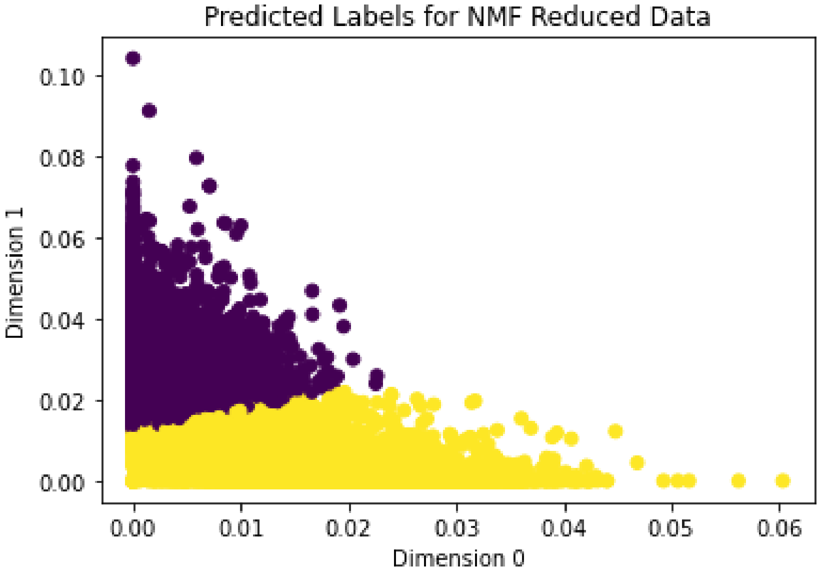


Figure 5: Predicted labels for NMF reduced data

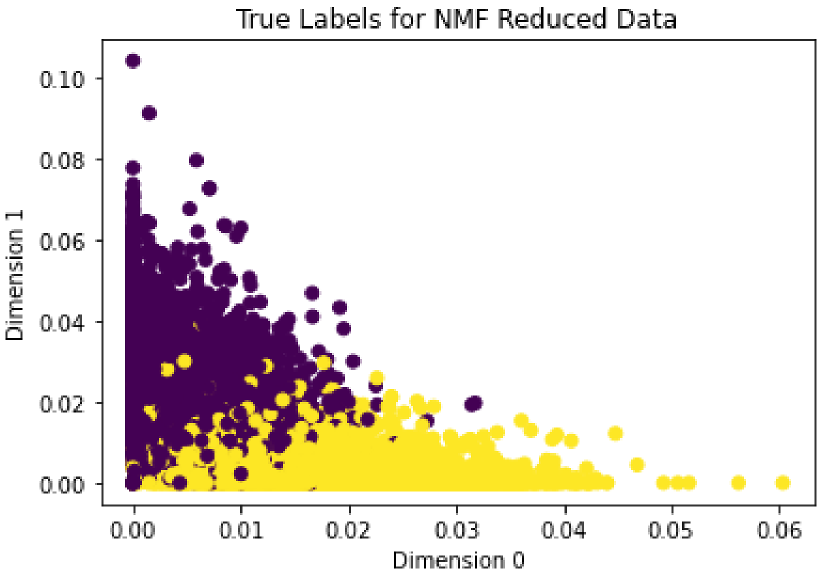


Figure 6: Ground truth labels for NMF reduced data

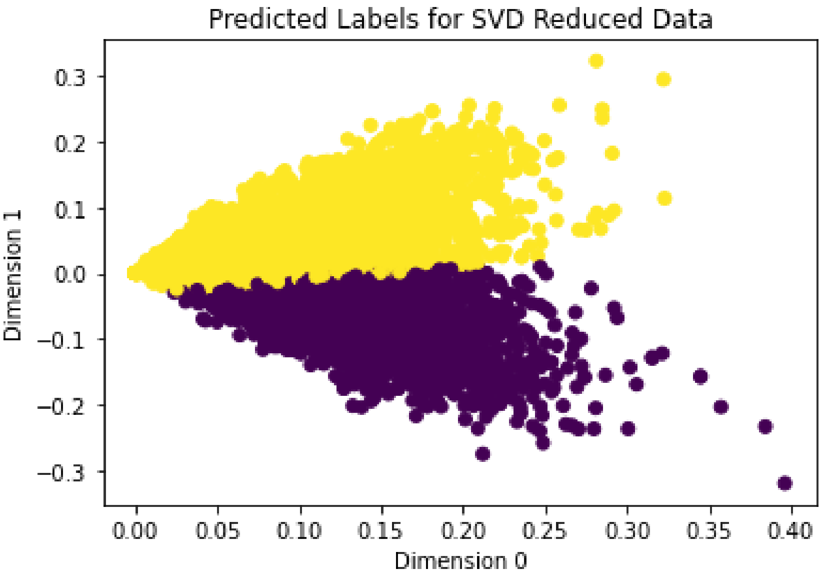


Figure 7: Predicted labels for SVD reduced data

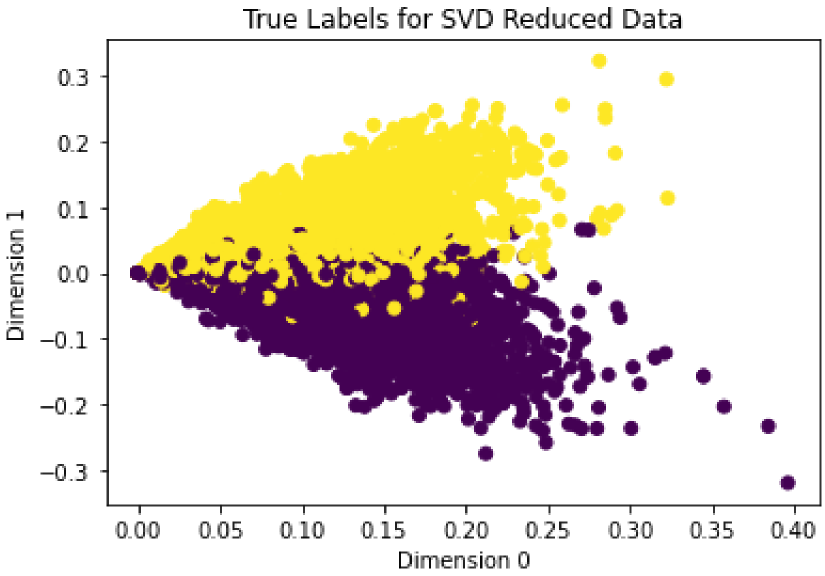


Figure 8: Ground truth labels for SVD reduced data

8 Question 8

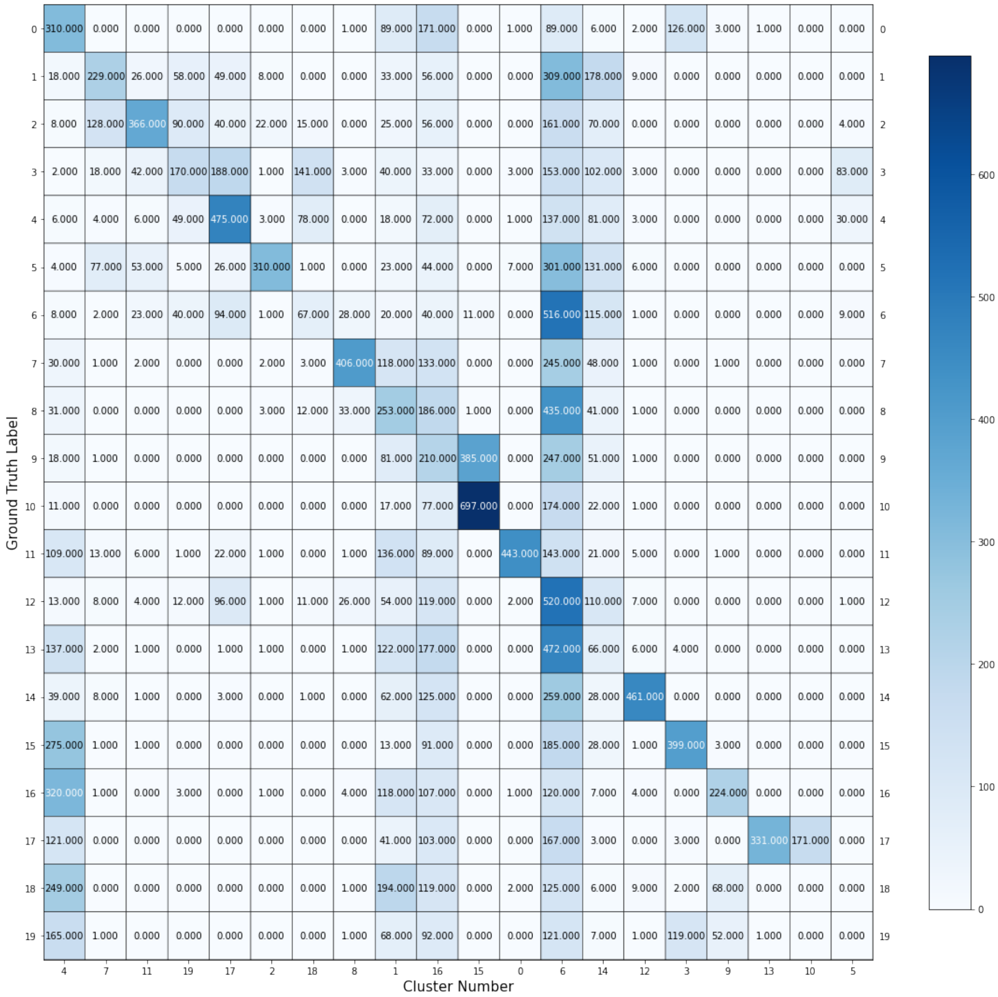
Because the NMF reduced data was already in 2D, we did not reduce it further with SVD. However, if we were to do SVD reduction, the resulting matrix would still be positive, as it is in Figures 5 and 6. SVD is an orthogonal projection that preserves the angles between feature vectors.

The 2D visualization from Figure 8 shows that the SVD reduced data looks somewhat linearly separable around a value of 0 for Dimension 1, which stems from the second principal component of the TF-IDF matrix. This suggests that this principal component holds important information pertaining to the class that the data belongs to. Although this analysis is restricted to two dimensions, this also suggests that this type of distribution would be good for KMeans clustering. A similar type of separation can be found at the diagonal between dimensions 0 and 1 for the NMF reduced data in Figure 6. This would also make it ideal for KMeans clusering.

However, for both types of reductions, because there is no way to perfectly linearly separate the data there are some better options out there for properly classifying this data. Perhaps a classifier that can learn non-linear behavior in the data, such as a neural network with ReLU/eLU, can perform better on this dataset.

9 Question 9

Using SVD with r=50, which gave us the best results on the two classes, we got the following contingency matrix for KMeans on the entire dataset of 20 classes:



	4	7	11	19	17	2	18	8	1	16	15	0	6	14	12	3	9	13	10	5	
0	310.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	89.000	171.000	0.000	1.000	89.000	6.000	2.000	126.000	3.000	1.000	0.000	0.000	0
1	18.000	229.000	26.000	58.000	49.000	8.000	0.000	0.000	33.000	56.000	0.000	0.000	309.000	178.000	9.000	0.000	0.000	0.000	0.000	0.000	1
2	8.000	128.000	366.000	90.000	40.000	22.000	15.000	0.000	25.000	56.000	0.000	0.000	161.000	70.000	0.000	0.000	0.000	0.000	0.000	4.000	2
3	2.000	18.000	42.000	170.000	188.000	1.000	141.000	3.000	40.000	33.000	0.000	3.000	153.000	102.000	3.000	0.000	0.000	0.000	0.000	83.000	3
4	6.000	4.000	6.000	49.000	475.000	3.000	78.000	0.000	18.000	72.000	0.000	1.000	137.000	81.000	3.000	0.000	0.000	0.000	0.000	30.000	4
5	4.000	77.000	53.000	5.000	26.000	310.000	1.000	0.000	23.000	44.000	0.000	7.000	301.000	131.000	6.000	0.000	0.000	0.000	0.000	0.000	5
6	8.000	2.000	23.000	40.000	94.000	1.000	67.000	28.000	20.000	40.000	11.000	0.000	516.000	115.000	1.000	0.000	0.000	0.000	0.000	9.000	6
7	30.000	1.000	2.000	0.000	0.000	2.000	3.000	406.000	118.000	133.000	0.000	0.000	245.000	48.000	1.000	0.000	1.000	0.000	0.000	0.000	7
8	31.000	0.000	0.000	0.000	0.000	3.000	12.000	33.000	253.000	186.000	1.000	0.000	435.000	41.000	1.000	0.000	0.000	0.000	0.000	0.000	8
9	18.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	81.000	210.000	385.000	0.000	247.000	51.000	1.000	0.000	0.000	0.000	0.000	0.000	9
10	11.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	17.000	77.000	697.000	0.000	174.000	22.000	1.000	0.000	0.000	0.000	0.000	0.000	10
11	109.000	13.000	6.000	1.000	22.000	1.000	0.000	1.000	136.000	89.000	0.000	443.000	143.000	21.000	5.000	0.000	1.000	0.000	0.000	0.000	11
12	13.000	8.000	4.000	12.000	96.000	1.000	11.000	26.000	54.000	119.000	0.000	2.000	520.000	110.000	7.000	0.000	0.000	0.000	0.000	1.000	12
13	137.000	2.000	1.000	0.000	1.000	1.000	0.000	1.000	122.000	177.000	0.000	0.000	472.000	66.000	6.000	4.000	0.000	0.000	0.000	0.000	13
14	39.000	8.000	1.000	0.000	3.000	0.000	1.000	0.000	62.000	125.000	0.000	0.000	259.000	28.000	461.000	0.000	0.000	0.000	0.000	0.000	14
15	275.000	1.000	1.000	0.000	0.000	0.000	0.000	0.000	13.000	91.000	0.000	0.000	185.000	28.000	1.000	399.000	3.000	0.000	0.000	0.000	15
16	320.000	1.000	0.000	3.000	0.000	1.000	0.000	4.000	118.000	107.000	0.000	1.000	120.000	7.000	4.000	0.000	224.000	0.000	0.000	0.000	16
17	121.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	41.000	103.000	0.000	0.000	167.000	3.000	0.000	3.000	0.000	331.000	171.000	0.000	17
18	249.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	194.000	119.000	0.000	2.000	125.000	6.000	9.000	2.000	68.000	0.000	0.000	0.000	18
19	165.000	1.000	0.000	0.000	0.000	0.000	0.000	1.000	68.000	92.000	0.000	0.000	121.000	7.000	1.000	119.000	52.000	1.000	0.000	0.000	19

Figure 9: Contingency Matrix for KMeans on 20 Class Data

The respective scores for this clustering are as follows:

Homogeneity score	0.31630458145750995
Completeness score	0.3682552023183464
V_measure score	0.3403086491478565
Adjusted Rand Index score	0.10379799750286293
Adjusted Mutual Information Score	0.3380068278889245

10 Question 10

Using the Kullback-Leibler Divergence loss metric for NMF and r=10, we were able to get the following scores:

Homogeneity score	0.37714629232162333
Completeness score	0.3900505160910544
V_measure score	0.3834898799077068
Adjusted Rand Index score	0.18833101982676798
Adjusted Mutual Information Score	0.38146535004967785

There is indeed a noticeable improvement over the performance using ordinary SVD with r=50 (from Q9).

11 Question 11

We compared the performance of KMeans on the dataset reduced with UMAP using various values for n_components. The plots are shown below:

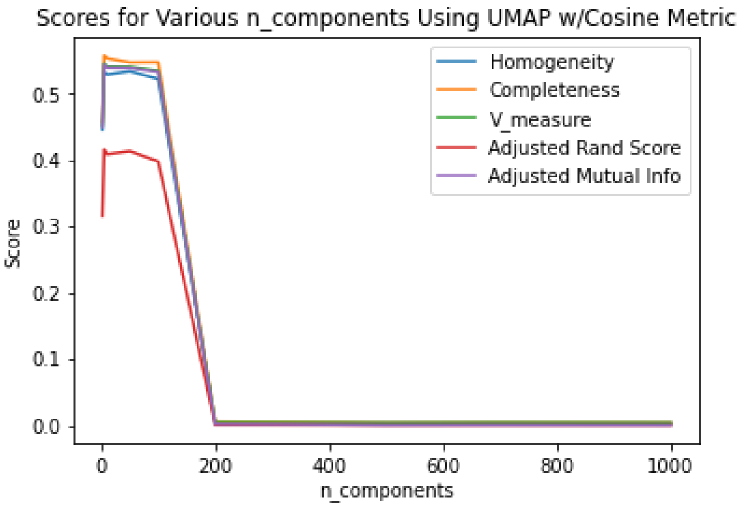


Figure 10: Metric scores for Cosine Metric vs. n_components

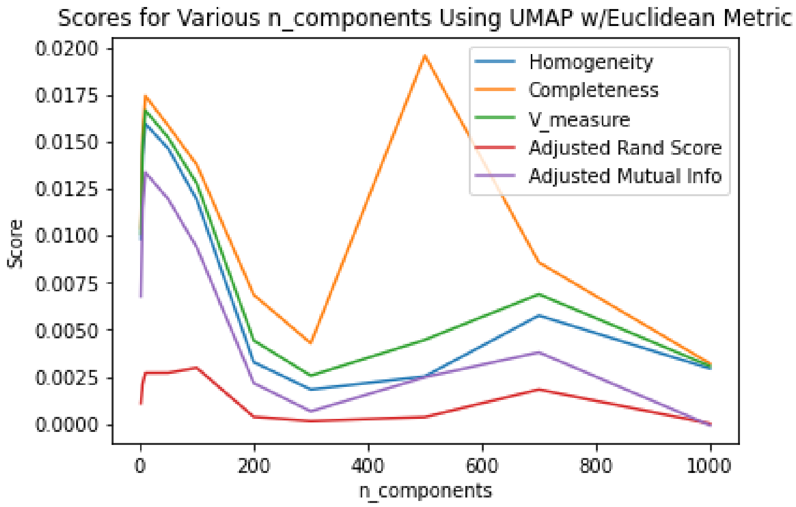


Figure 11: Metric scores for Euclidean Metric vs. n_components

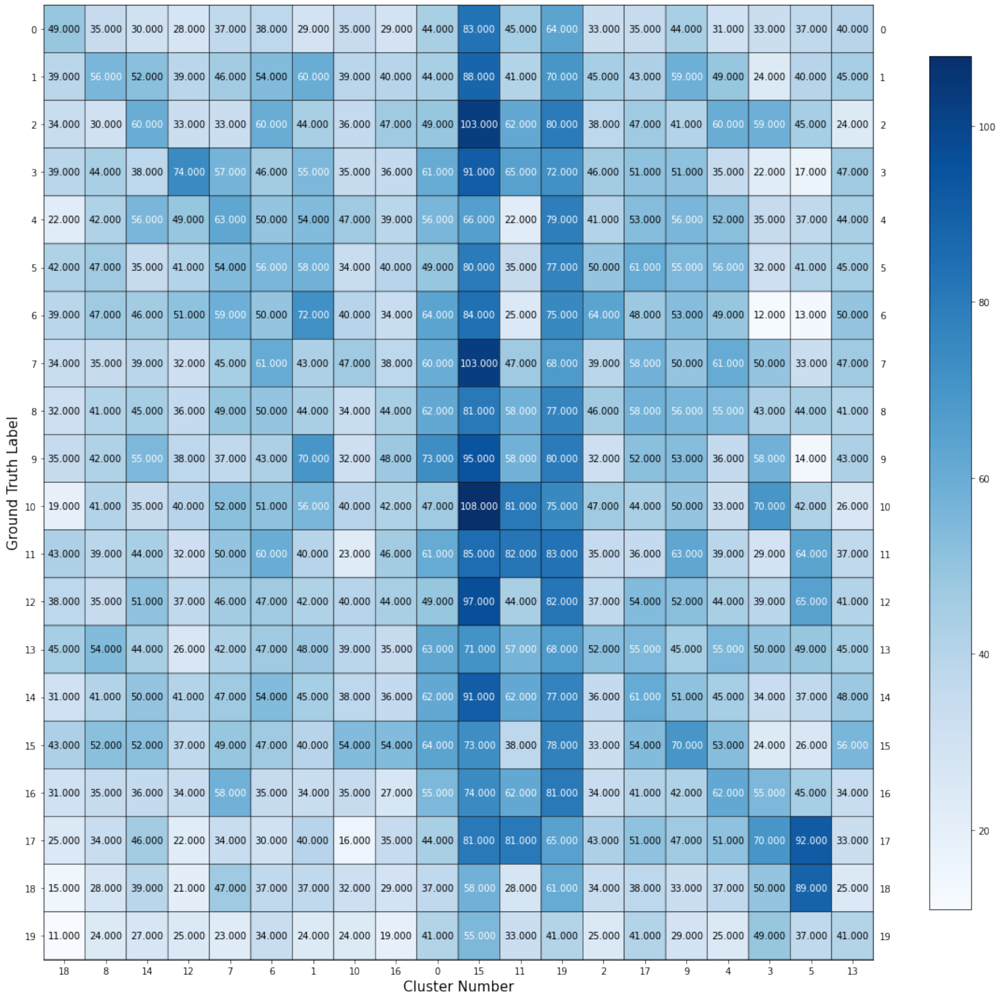
The best scores using both metrics came from **n_components=5**:

Cosine Metric	
Homogeneity score	0.5347926404764849
Completeness score	0.5549487009354954
V_measure score	0.5446842655666289
Adjusted Rand Index score	0.4167210932677909
Adjusted Mutual Information Score	0.5431756896373161

Euclidean metric	
Homogeneity score	0.008428056566878686
Completeness score	0.008490790660743977
V_measure score	0.008459307306639803
Adjusted Rand Index score	0.0014762057832842827
Adjusted Mutual Information Score	0.005247811170211466

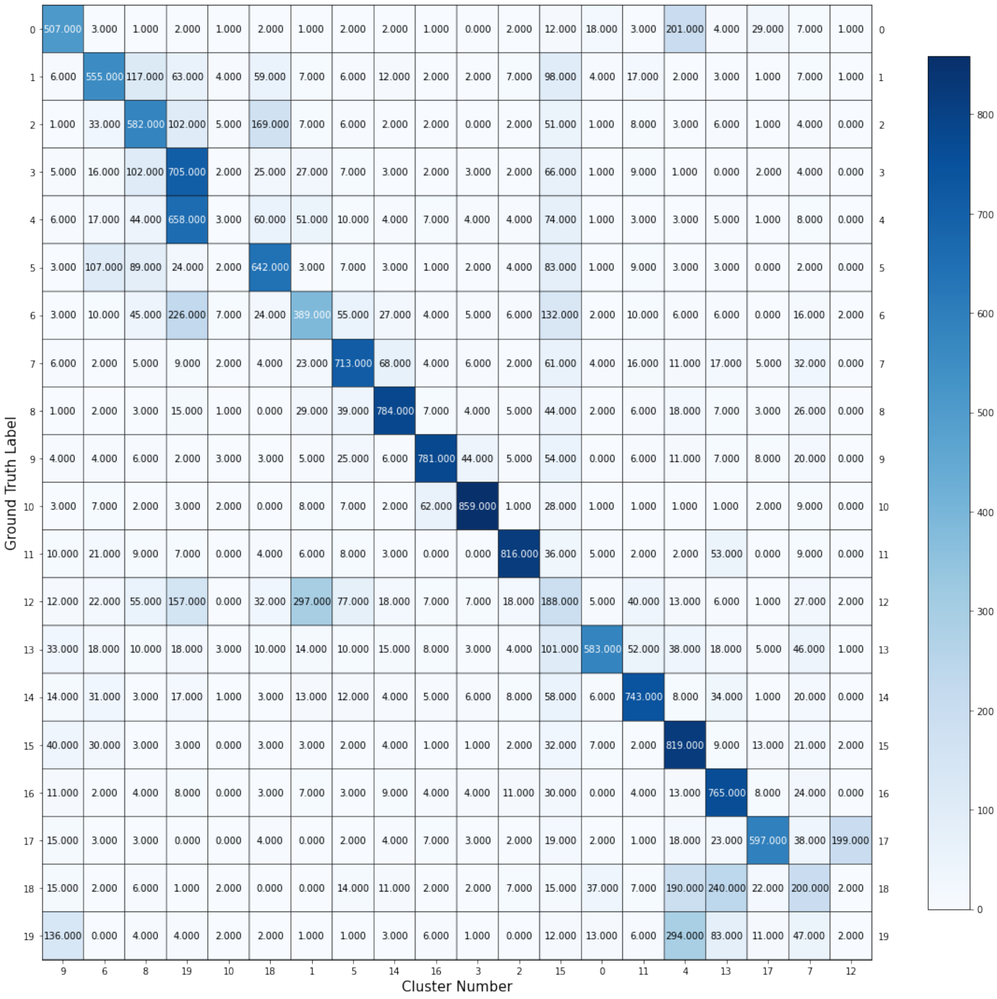
12 Question 12

Using n_components = 5 as determined above, we get the below contingency matrices – Figure 12 corresponds to the Euclidean metric, and Figure 13 corresponds to the cosine metric.



	18	8	14	12	7	6	1	10	16	0	15	11	19	2	17	9	4	3	5	13	
0	49.000	35.000	30.000	28.000	37.000	38.000	29.000	35.000	29.000	44.000	83.000	45.000	64.000	33.000	35.000	44.000	31.000	33.000	37.000	40.000	0
1	39.000	56.000	52.000	39.000	46.000	54.000	60.000	39.000	40.000	44.000	88.000	41.000	70.000	45.000	43.000	59.000	49.000	24.000	40.000	45.000	1
2	34.000	30.000	60.000	33.000	33.000	60.000	44.000	36.000	47.000	49.000	103.000	62.000	80.000	38.000	47.000	41.000	60.000	59.000	45.000	24.000	2
3	39.000	44.000	38.000	74.000	57.000	46.000	55.000	35.000	36.000	61.000	91.000	65.000	72.000	46.000	51.000	51.000	35.000	22.000	17.000	47.000	3
4	22.000	42.000	56.000	49.000	63.000	50.000	54.000	47.000	39.000	56.000	66.000	22.000	79.000	41.000	53.000	56.000	52.000	35.000	37.000	44.000	4
5	42.000	47.000	35.000	41.000	54.000	56.000	58.000	34.000	40.000	49.000	80.000	35.000	77.000	50.000	61.000	55.000	56.000	32.000	41.000	45.000	5
6	39.000	47.000	46.000	51.000	59.000	50.000	72.000	40.000	34.000	64.000	84.000	25.000	75.000	64.000	48.000	53.000	49.000	12.000	13.000	50.000	6
7	34.000	35.000	39.000	32.000	45.000	61.000	43.000	47.000	38.000	60.000	103.000	47.000	68.000	39.000	58.000	50.000	61.000	50.000	33.000	47.000	7
8	32.000	41.000	45.000	36.000	49.000	50.000	44.000	34.000	44.000	62.000	81.000	58.000	77.000	46.000	58.000	56.000	55.000	43.000	44.000	41.000	8
9	35.000	42.000	55.000	38.000	37.000	43.000	70.000	32.000	48.000	73.000	95.000	58.000	80.000	32.000	52.000	53.000	36.000	58.000	14.000	43.000	9
10	19.000	41.000	35.000	40.000	52.000	51.000	56.000	40.000	42.000	47.000	108.000	81.000	75.000	47.000	44.000	50.000	33.000	70.000	42.000	26.000	10
11	43.000	39.000	44.000	32.000	50.000	60.000	40.000	23.000	46.000	61.000	85.000	82.000	83.000	35.000	36.000	63.000	39.000	29.000	64.000	37.000	11
12	38.000	35.000	51.000	37.000	46.000	47.000	42.000	40.000	44.000	49.000	97.000	44.000	82.000	37.000	54.000	52.000	44.000	39.000	65.000	41.000	12
13	45.000	54.000	44.000	26.000	42.000	47.000	48.000	39.000	35.000	63.000	71.000	57.000	68.000	52.000	55.000	45.000	55.000	50.000	49.000	45.000	13
14	31.000	41.000	50.000	41.000	47.000	54.000	45.000	38.000	36.000	62.000	91.000	62.000	77.000	36.000	61.000	51.000	45.000	34.000	37.000	48.000	14
15	43.000	52.000	52.000	37.000	49.000	47.000	40.000	54.000	54.000	64.000	73.000	38.000	78.000	33.000	54.000	70.000	53.000	24.000	26.000	56.000	15
16	31.000	35.000	36.000	34.000	58.000	35.000	34.000	35.000	27.000	55.000	74.000	62.000	81.000	34.000	41.000	42.000	62.000	55.000	45.000	34.000	16
17	25.000	34.000	46.000	22.000	34.000	30.000	40.000	16.000	35.000	44.000	81.000	81.000	65.000	43.000	51.000	47.000	51.000	70.000	92.000	33.000	17
18	15.000	28.000	39.000	21.000	47.000	37.000	37.000	32.000	29.000	37.000	58.000	28.000	61.000	34.000	38.000	33.000	37.000	50.000	89.000	25.000	18
19	11.000	24.000	27.000	25.000	23.000	34.000	24.000	24.000	19.000	41.000	55.000	33.000	41.000	25.000	41.000	29.000	25.000	49.000	37.000	41.000	19

Figure 12: Confusion matrix for UMAP-reduced Kmeans with Euclidean metric



	9	6	8	19	10	18	1	5	14	16	3	2	15	0	11	4	13	17	7	12	
0	507.000	3.000	1.000	2.000	1.000	2.000	1.000	2.000	2.000	1.000	0.000	2.000	12.000	18.000	3.000	201.000	4.000	29.000	7.000	1.000	0
1	6.000	555.000	117.000	63.000	4.000	59.000	7.000	6.000	12.000	2.000	2.000	7.000	98.000	4.000	17.000	2.000	3.000	1.000	7.000	1.000	1
2	1.000	33.000	582.000	102.000	5.000	169.000	7.000	6.000	2.000	2.000	0.000	2.000	51.000	1.000	8.000	3.000	6.000	1.000	4.000	0.000	2
3	5.000	16.000	102.000	705.000	2.000	25.000	27.000	7.000	3.000	2.000	3.000	2.000	66.000	1.000	9.000	1.000	0.000	2.000	4.000	0.000	3
4	6.000	17.000	44.000	658.000	3.000	60.000	51.000	10.000	4.000	7.000	4.000	4.000	74.000	1.000	3.000	3.000	5.000	1.000	8.000	0.000	4
5	3.000	107.000	89.000	24.000	2.000	642.000	3.000	7.000	3.000	1.000	2.000	4.000	83.000	1.000	9.000	3.000	3.000	0.000	2.000	0.000	5
6	3.000	10.000	45.000	226.000	7.000	24.000	389.000	55.000	27.000	4.000	5.000	6.000	132.000	2.000	10.000	6.000	6.000	0.000	16.000	2.000	6
7	6.000	2.000	5.000	9.000	2.000	4.000	23.000	713.000	68.000	4.000	6.000	2.000	61.000	4.000	16.000	11.000	17.000	5.000	32.000	0.000	7
8	1.000	2.000	3.000	15.000	1.000	0.000	29.000	39.000	784.000	7.000	4.000	5.000	44.000	2.000	6.000	18.000	7.000	3.000	26.000	0.000	8
9	4.000	4.000	6.000	2.000	3.000	3.000	5.000	25.000	6.000	781.000	44.000	5.000	54.000	0.000	6.000	11.000	7.000	8.000	20.000	0.000	9
10	3.000	7.000	2.000	3.000	2.000	0.000	8.000	7.000	2.000	62.000	859.000	1.000	28.000	1.000	1.000	1.000	1.000	2.000	9.000	0.000	10
11	10.000	21.000	9.000	7.000	0.000	4.000	6.000	8.000	3.000	0.000	0.000	816.000	36.000	5.000	2.000	2.000	53.000	0.000	9.000	0.000	11
12	12.000	22.000	55.000	157.000	0.000	32.000	297.000	77.000	18.000	7.000	7.000	18.000	188.000	5.000	40.000	13.000	6.000	1.000	27.000	2.000	12
13	33.000	18.000	10.000	18.000	3.000	10.000	14.000	10.000	15.000	8.000	3.000	4.000	101.000	583.000	52.000	38.000	18.000	5.000	46.000	1.000	13
14	14.000	31.000	3.000	17.000	1.000	3.000	13.000	12.000	4.000	5.000	6.000	8.000	58.000	6.000	743.000	8.000	34.000	1.000	20.000	0.000	14
15	40.000	30.000	3.000	3.000	0.000	3.000	3.000	2.000	4.000	1.000	1.000	2.000	32.000	7.000	2.000	819.000	9.000	13.000	21.000	2.000	15
16	11.000	2.000	4.000	8.000	0.000	3.000	7.000	3.000	9.000	4.000	4.000	11.000	30.000	0.000	4.000	13.000	765.000	8.000	24.000	0.000	16
17	15.000	3.000	3.000	0.000	0.000	4.000	0.000	2.000	4.000	7.000	3.000	2.000	19.000	2.000	1.000	18.000	23.000	597.000	38.000	199.000	17
18	15.000	2.000	6.000	1.000	2.000	0.000	0.000	14.000	11.000	2.000	2.000	7.000	15.000	37.000	7.000	190.000	240.000	22.000	200.000	2.000	18
19	136.000	0.000	4.000	4.000	2.000	2.000	1.000	1.000	3.000	6.000	1.000	0.000	12.000	13.000	6.000	294.000	83.000	11.000	47.000	2.000	19

Figure 13: Confusion matrix for UMAP-reduced Kmeans with cosine metric

The cosine matrix is obviously much better, which corresponds with the greatly improved scores using this metric.

Examining the cosine metric more closely, we see that most categories were well identified, but some were more likely to be confused with each other. Tracing the connections, we can see that points with ground truth label 19 were likely to be identified with points labeled 0 or 15, and points from label 12 were misidentified as 1 or 6. In the original dataset, this means that points from `talk.religion.misc` were identified as `alt.atheism` or `soc.religion.christian`, and points from `sci.electronics` were identified as `comp.graphics` or `misc.forsale`. The religion groups form an intuitive grouping, and the confusion between the latter three groups perhaps indicates that selling hardware and electronics was common on these newsgroups.

13 Question 13

Using the best hyperparameter combination from Q11/12, n_components=5 and the cosine metric, we reduced the Tfidf matrix with UMAP. The scores for agglomerative clustering using 'ward' and 'single' linkage on this data are shown below:

Ward linkage	
Homogeneity score	0.524473300498607
Completeness score	0.5479912976714623
V_measure score	0.53597443687123
Adjusted Rand Index score	0.3940501791969448
Adjusted Mutual Information Score	0.5344316766896055

Single linkage	
Homogeneity score	0.022917457355135236
Completeness score	0.3464156710750703
V_measure score	0.04299081646294147
Adjusted Rand Index score	0.00046761208355549543
Adjusted Mutual Information Score	0.03748616963750802

It's clear that ward linkage performs significantly better on the dataset. Single linkage uses the minimum distance between all points in each cluster to determine whether to merge them or not, so it's a lot more sensitive to outliers and tends to result in uneven cluster sizes. Ward linkage on the other hand, minimizes the variance between pairs of clusters, which can produce more regular cluster sizes and hence better results for our particular dataset.

14 Question 14

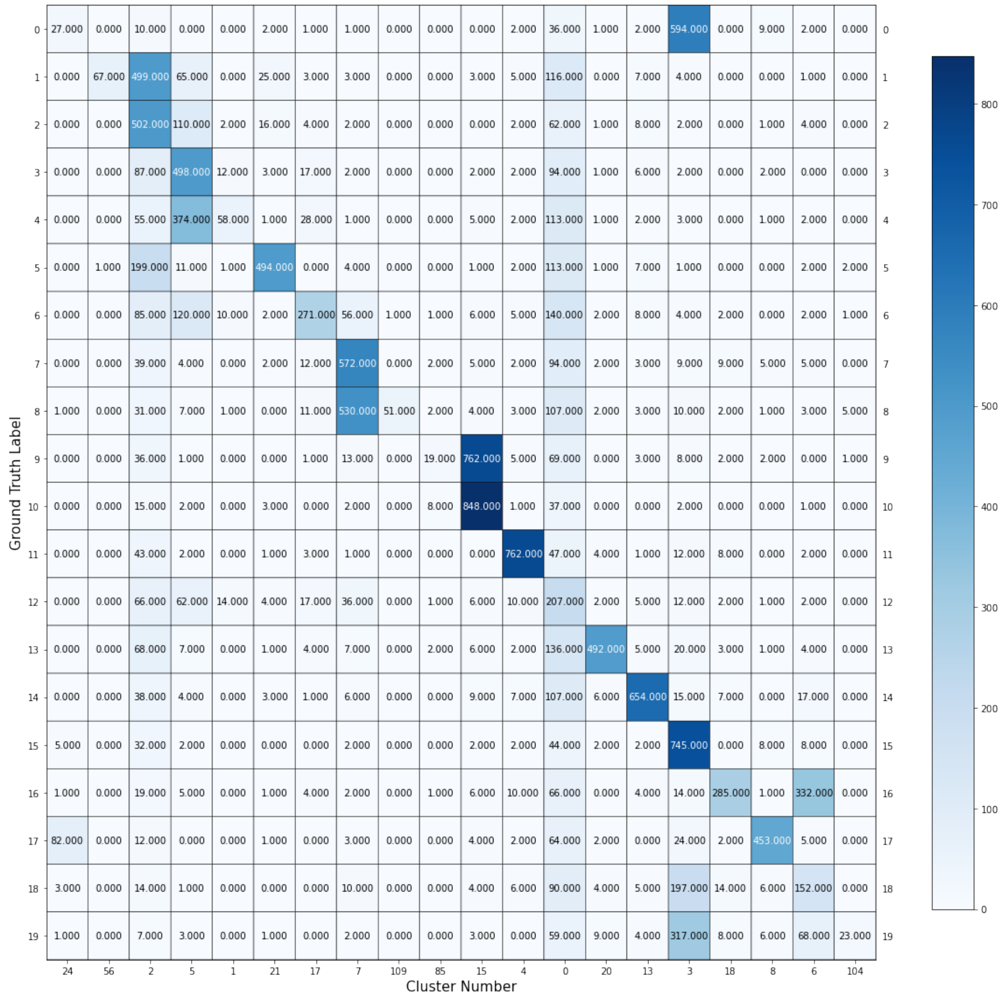
In this section we investigate the DBSCAN and HDBSCAN clustering algorithms. We kept UMAP with n_components=5 and the cosine metric as the dimensionality reduction algorithm. The HDBSCAN min_cluster_size was set to 100, and other hyperparameters were tuned to improve the performance of each of these algorithms. We performed hyperparameter tuning on DBSCAN and HDBSCAN for epsilon values of [0.01,0.05,0.1,0.3,0.5,0.7,0.9,1], metric types of [Euclidean, cosine, Manhattan] (if applicable), and minimum sample values of [10, 20, 50, 70, 100]. The following table depicts the scores for each clustering algorithm for the hyperparameter combination that resulted in the best average score across all 5 clustering metrics:

DBSCAN: Epsilon = 0.5, Metric = Manhattan, Minimum samples = 10	
Homogeneity score	0.5407607595568981
Completeness score	0.45471096739764455
V_measure score	0.49401673890037173
Adjusted Rand Index score	0.2967260971944622
Adjusted Mutual Information Score	0.47920988791262803

HDBSCAN: Epsilon = 0.3, Metric = Euclidean, Minimum samples = 10	
Homogeneity score	0.43889528158878366
Completeness score	0.5288819318072416
V_measure score	0.47970500064413385
Adjusted Rand Index score	0.2565568616115164
Adjusted Mutual Information Score	0.4780526456871693

15 Question 15

DBSCAN was the better performing clustering algorithm. The associated contingency matrix is shown below:



	24	56	2	5	1	21	17	7	109	85	15	4	0	20	13	3	18	8	6	104	
0	27.000	0.000	10.000	0.000	0.000	2.000	1.000	1.000	0.000	0.000	0.000	2.000	36.000	1.000	2.000	594.000	0.000	9.000	2.000	0.000	0
1	0.000	67.000	499.000	65.000	0.000	25.000	3.000	3.000	0.000	0.000	3.000	5.000	116.000	0.000	7.000	4.000	0.000	0.000	1.000	0.000	1
2	0.000	0.000	502.000	110.000	2.000	16.000	4.000	2.000	0.000	0.000	0.000	2.000	62.000	1.000	8.000	2.000	0.000	1.000	4.000	0.000	2
3	0.000	0.000	87.000	498.000	12.000	3.000	17.000	2.000	0.000	0.000	2.000	2.000	94.000	1.000	6.000	2.000	0.000	2.000	0.000	0.000	3
4	0.000	0.000	55.000	374.000	58.000	1.000	28.000	1.000	0.000	0.000	5.000	2.000	113.000	1.000	2.000	3.000	0.000	1.000	2.000	0.000	4
5	0.000	1.000	199.000	11.000	1.000	494.000	0.000	4.000	0.000	0.000	1.000	2.000	113.000	1.000	7.000	1.000	0.000	0.000	2.000	2.000	5
6	0.000	0.000	85.000	120.000	10.000	2.000	271.000	56.000	1.000	1.000	6.000	5.000	140.000	2.000	8.000	4.000	2.000	0.000	2.000	1.000	6
7	0.000	0.000	39.000	4.000	0.000	2.000	12.000	572.000	0.000	2.000	5.000	2.000	94.000	2.000	3.000	9.000	9.000	5.000	5.000	0.000	7
8	1.000	0.000	31.000	7.000	1.000	0.000	11.000	530.000	51.000	2.000	4.000	3.000	107.000	2.000	3.000	10.000	2.000	1.000	3.000	5.000	8
9	0.000	0.000	36.000	1.000	0.000	0.000	1.000	13.000	0.000	19.000	762.000	5.000	69.000	0.000	3.000	8.000	2.000	2.000	0.000	1.000	9
10	0.000	0.000	15.000	2.000	0.000	3.000	0.000	2.000	0.000	8.000	848.000	1.000	37.000	0.000	0.000	2.000	0.000	0.000	1.000	0.000	10
11	0.000	0.000	43.000	2.000	0.000	1.000	3.000	1.000	0.000	0.000	0.000	762.000	47.000	4.000	1.000	12.000	8.000	0.000	2.000	0.000	11
12	0.000	0.000	66.000	62.000	14.000	4.000	17.000	36.000	0.000	1.000	6.000	10.000	207.000	2.000	5.000	12.000	2.000	1.000	2.000	0.000	12
13	0.000	0.000	68.000	7.000	0.000	1.000	4.000	7.000	0.000	2.000	6.000	2.000	136.000	492.000	5.000	20.000	3.000	1.000	4.000	0.000	13
14	0.000	0.000	38.000	4.000	0.000	3.000	1.000	6.000	0.000	0.000	9.000	7.000	107.000	6.000	654.000	15.000	7.000	0.000	17.000	0.000	14
15	5.000	0.000	32.000	2.000	0.000	0.000	0.000	2.000	0.000	0.000	2.000	2.000	44.000	2.000	2.000	745.000	0.000	8.000	8.000	0.000	15
16	1.000	0.000	19.000	5.000	0.000	1.000	4.000	2.000	0.000	1.000	6.000	10.000	66.000	0.000	4.000	14.000	285.000	1.000	332.000	0.000	16
17	82.000	0.000	12.000	0.000	0.000	1.000	0.000	3.000	0.000	0.000	4.000	2.000	64.000	2.000	0.000	24.000	2.000	453.000	5.000	0.000	17
18	3.000	0.000	14.000	1.000	0.000	0.000	0.000	10.000	0.000	0.000	4.000	6.000	90.000	4.000	5.000	197.000	14.000	6.000	152.000	0.000	18
19	1.000	0.000	7.000	3.000	0.000	1.000	0.000	2.000	0.000	0.000	3.000	0.000	59.000	9.000	4.000	317.000	8.000	6.000	68.000	23.000	19

Figure 14: Contingency Matrix for DBSCAN on 20 Class Data

The contingency matrix indicates that there must be at least 96 clusters since that's the largest number shown on the x-axis ticks. By looking at the number of unique labels in the trained DBSCAN model, we are able to determine that there are actually 168 clusters formed by the algorithm (excluding the -1 labels). This number is high because of our low value for the mimimum samples hyperparameter, so most of these cluters contain few data points. However, most of the data points were captured by the larger clusters shown in the contingency matrix, which resulted in acceptable clustering metric scores.

A label of -1 indicates that the data point was classified as noise, and was not assigned to any cluster. This cluster label is represented as cluster number 0 in the contingency matrix. We can tell because this cluster contains many data points from each of the 20 classes.

16 Question 16

We used the "read_csv" function of the Pandas library to acquire the BBC dataset as a dataframe. After converting the written class labels to numerical target values for each document, we then used our established pipeline to vectorize (excluding stopwords/punctuation) then normalize the data and analyze the dimensions of the resulting TF-IDF matrix. As with the 20 Newsgroups dataset, we then reduced dimensionality with UMAP and searched for the optimal parameter configuration for multiple clustering algorithms. Starting with KMeans, we tested how our five aforementioned accuracy metrics vary with the number of principle components used in clustering for both Euclidean (Figure 15) and Cosine distance metrics (Figure 16). The best combination was n_components = 10 with the Euclidean distance metric.

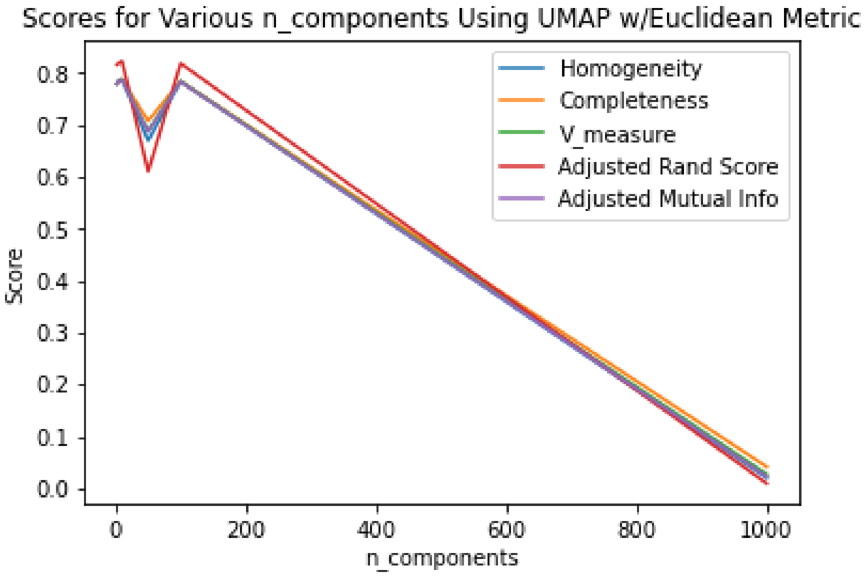


Figure 15: Variation in accuracy metrics by number of components for Kmeans Clustering and Euclidean distance

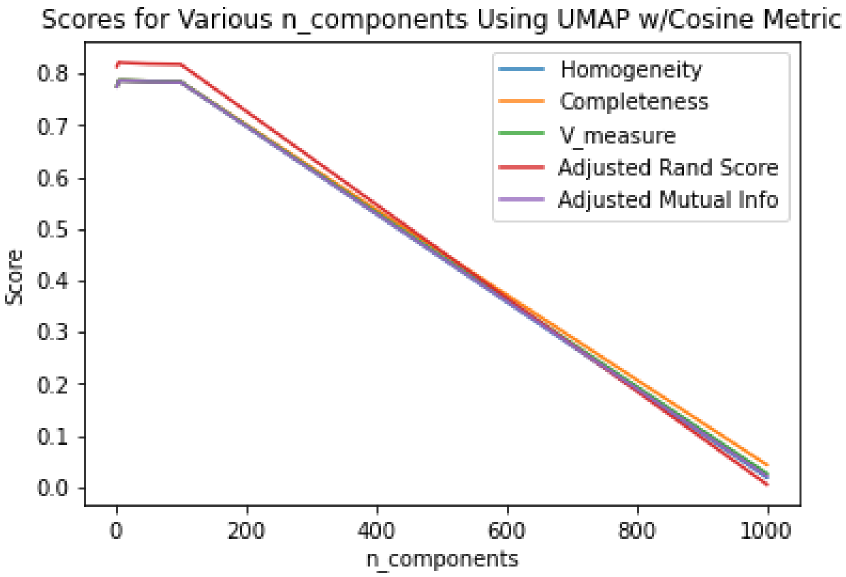


Figure 16: Variation in accuracy metrics by number of components for Kmeans Clustering and Cosine distance

We then tested the accuracy of the DBSCAN clustering algorithm for both distance metrics. We cycled through values from .01 to 1 for epsillon and from 1 to 50 for the minimum number of points per cluster to find the best combination of hyperparameters for our analysis. We used 3-dimensional surface plots to visualize how these parameters in concert affected our five accuracy metrics and the estimated number of clusters generated by DBSCAN (Figure 17).

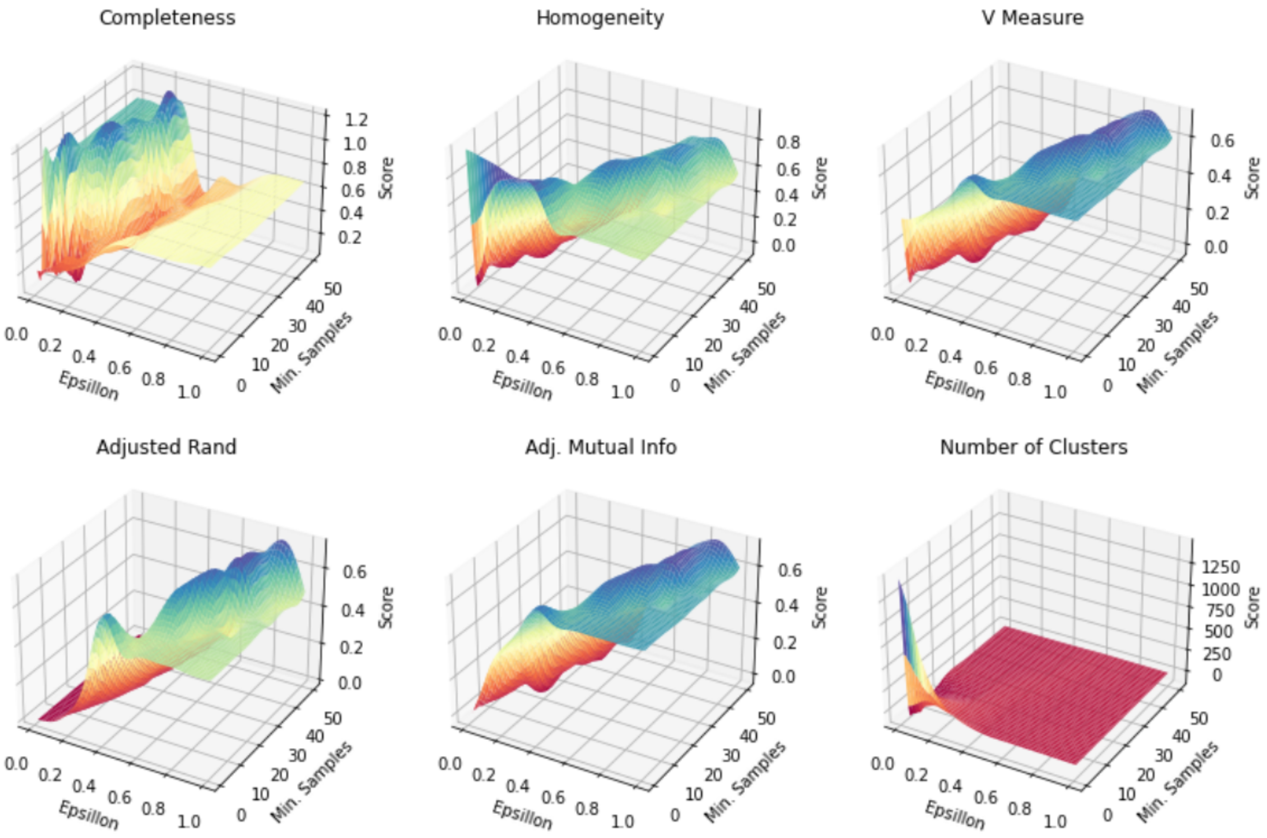


Figure 17: Variation in accuracy metrics corresponding to changes in epsilon and minimum cluster metrics for DBSCAN clustering of UMAP reduced data with Cosine distance metric (Euclidean metric results visually similar)

Avoiding the low values of epsilon and minimum samples that resulted in excessively high numbers of estimated clusters, we found the combination of parameters that maximized the average of the accuracy metrics to be 0.8 for epsilon and 35 for minimum samples. We then compared the results of each of our protocols (Kmeans with Euclidean/Cosine, DBSCAN with Euclidean/Cosine) to find the best reduction and clustering technique for our data. We report the five accuracy metrics for each protocol below:

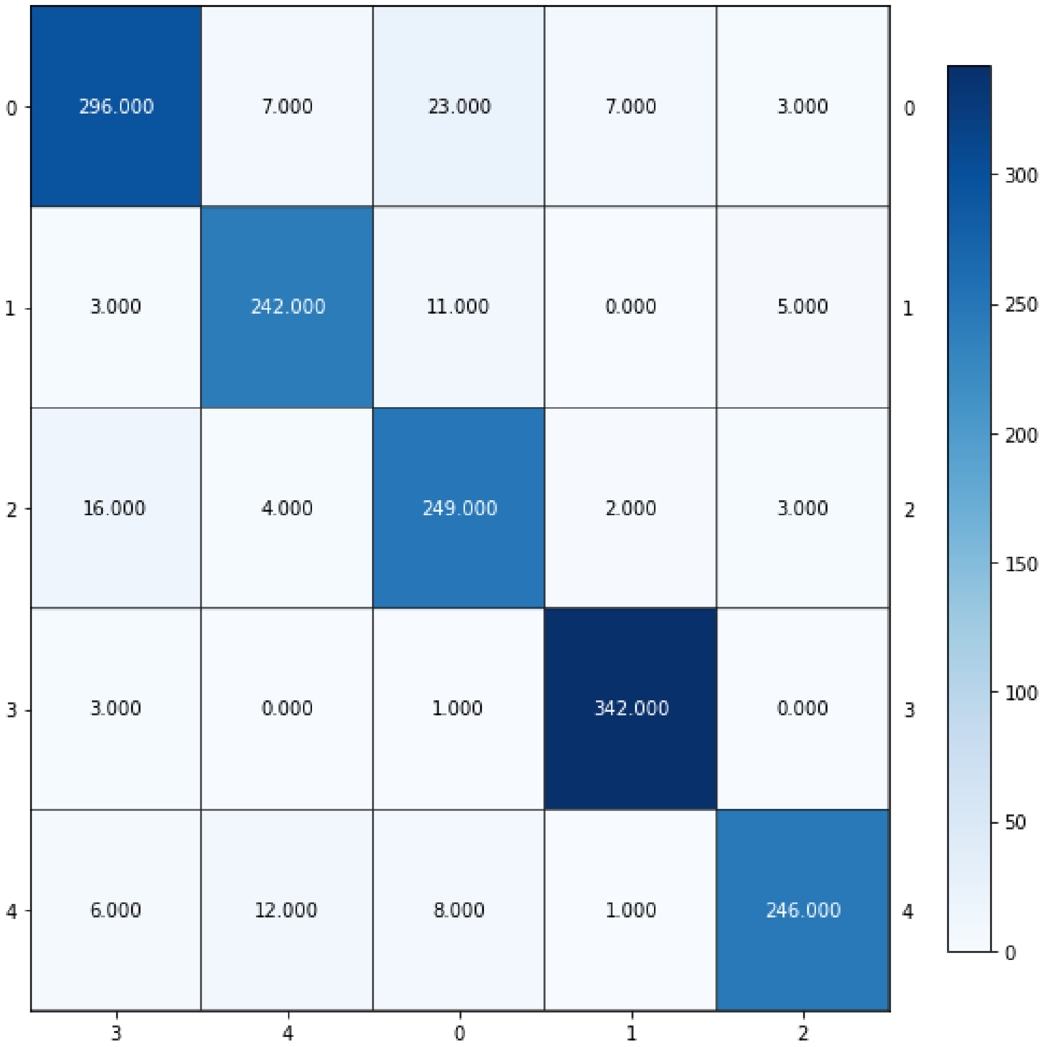
Accuracy Metrics for DBSCAN, Metric = Euclidean	
Homogeneity score	0.7896799513244787
Completeness score	0.667782538810624
V_measure score	0.7236336939202158
Adjusted Rand Index score	0.7108732847743878
Adjusted Mutual Information Score	0.7221232251101021

Accuracy Metrics for DBSCAN, Metric = Cosine	
Homogeneity score	0.7896799513244787
Completeness score	0.667782538810624
V_measure score	0.7236336939202158
Adjusted Rand Index score	0.7108732847743878
Adjusted Mutual Information Score	0.7221232251101021

Accuracy Metrics for KMeans, Metric = Euclidean	
Homogeneity score	0.7874360944663693
Completeness score	0.7874929063651126
V_measure score	0.7874644993910623
Adjusted Rand Index score	0.8229763601775667
Adjusted Mutual Information Score	0.786746935945491

Accuracy Metrics for KMeans, Metric = Cosine	
Homogeneity score	0.7863911146778318
Completeness score	0.7863053396249526
V_measure score	0.7863482248123015
Adjusted Rand Index score	0.8206748429226303
Adjusted Mutual Information Score	0.785626959948192

The best scoring protocol was Kmeans with Euclidean distance metric and n_components set to 10. The contingency matrix for this clustering is shown in Figure 18 below



	3	4	0	1	2	
0	296.000	7.000	23.000	7.000	3.000	0
1	3.000	242.000	11.000	0.000	5.000	1
2	16.000	4.000	249.000	2.000	3.000	2
3	3.000	0.000	1.000	342.000	0.000	3
4	6.000	12.000	8.000	1.000	246.000	4

Figure 18: Contingency Matrix for optimal clustering of BBC News data

Our pipeline and parameter selections produce a fairly good clustering through both DBSCAN and KMeans. One factor that may affect the accuracy of our analysis is the relatively small size of the BBC dataset when compared to the 20 Newsgroups dataset. With more data, our clustering would likely become more accurate.

Project_2

February 3, 2021

[88]:

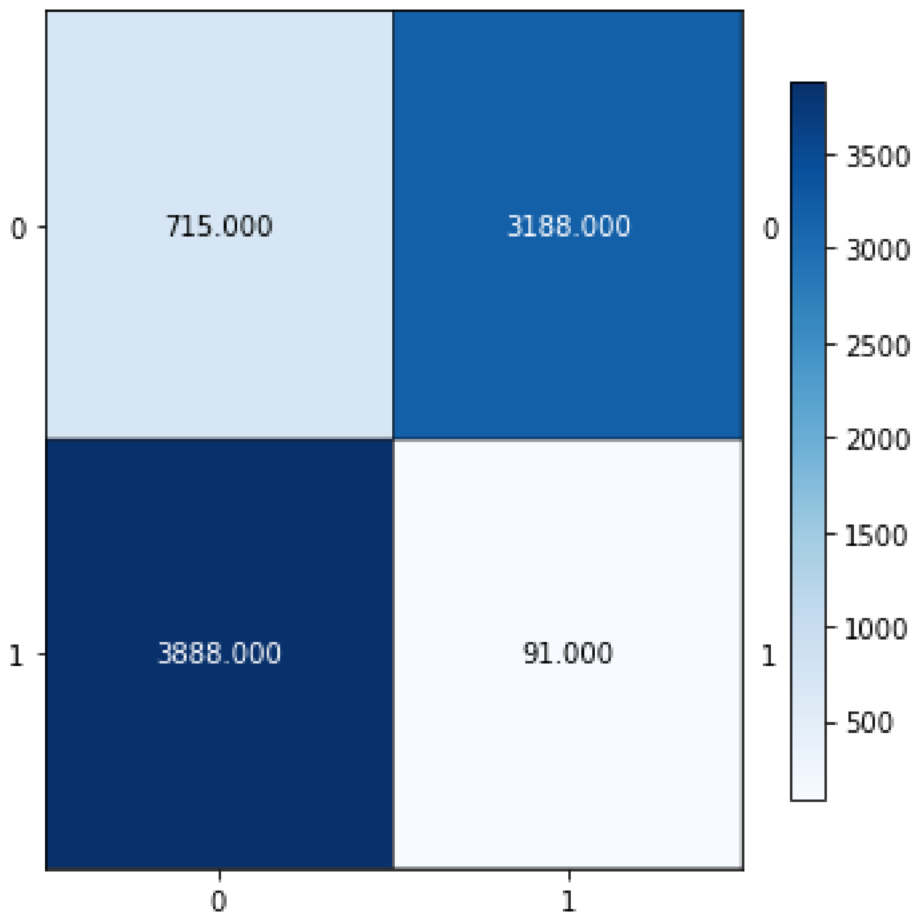
```
#Q1

import nltk
from nltk import pos_tag
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from string import punctuation
from nltk.corpus import stopwords
from matplotlib import pyplot as plt
from sklearn.datasets import fetch_20newsgroups

categories = ['comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
'rec.autos', 'rec.motorcycles',
'rec.sport.baseball', 'rec.sport.hockey']

#Vectorizer
analyzer = CountVectorizer().build_analyzer()
tfidf_transformer = TfidfTransformer()

combined_stopwords = set.union(set(stopwords.words('english')),set(punctuation))


def stem_rmv_punc_nolem(doc):
    return (word for word in analyzer(doc) if word not in combined_stopwords and␣
 ↪not word.isdigit())

#Download dataset
dataset = fetch_20newsgroups(subset='all', categories = categories, shuffle =␣
 ↪True, random_state = 0, remove=('headers','footers'))

count_vect = CountVectorizer(min_df=3,analyzer=stem_rmv_punc_nolem,␣
 ↪stop_words='english')

X_counts = count_vect.fit_transform(dataset.data)
X_tfidf = tfidf_transformer.fit_transform(X_counts)
```

```
print("X_tfidf shape: ", X_tfidf.shape)
```

```
X_tfidf shape:  (7882, 21909)
```

[89]:

```
#Q2
#Get labels to verify k means accuracy

from sklearn.cluster import KMeans
from plotmat import plot_mat
from sklearn.metrics.cluster import contingency_matrix

dataset_targets_bin = dataset.target.copy()
for i in range(len(dataset_targets_bin)):
    if dataset.target[i] in [0,1,2,3]:
        dataset_targets_bin[i] = 0
    else:
        dataset_targets_bin[i] = 1

kmeans_clf = KMeans(n_clusters=2, max_iter=1500, random_state=0, n_init=50)
kmeans_clf.fit(X_tfidf)
km_cont_mat = contingency_matrix(dataset_targets_bin, kmeans_clf.labels_)

plot_mat(km_cont_mat, size=(5,5))
```

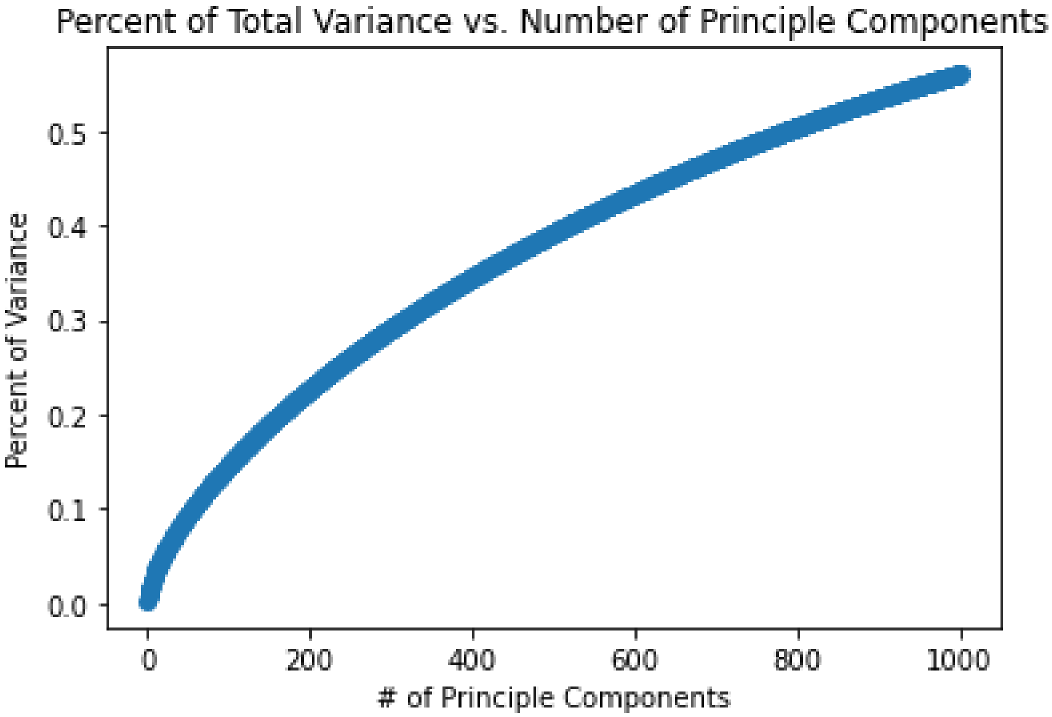


[90]:
```
#Q3

from sklearn import metrics
print("Homogeneity score: ", metrics.homogeneity_score(dataset_targets_bin,␣
 ↪kmeans_clf.labels_))
print("Completeness score: ", metrics.completeness_score(dataset_targets_bin,␣
 ↪kmeans_clf.labels_))
print("V_measure score: ", metrics.v_measure_score(dataset_targets_bin,␣
 ↪kmeans_clf.labels_))
print("Adjusted Rand Index score: ", metrics.
 ↪adjusted_rand_score(dataset_targets_bin, kmeans_clf.labels_))
print("Adjusted Mutual Information Score: ", metrics.
 ↪adjusted_mutual_info_score(dataset_targets_bin, kmeans_clf.labels_))
```

```
Homogeneity score:  0.5600025412974119
Completeness score:  0.5716557539454379
V_measure score:  0.5657691483417876
Adjusted Rand Index score:  0.6327475635535726
Adjusted Mutual Information Score:  0.5657289846874574
```

[91]:

```python
#Q4

from sklearn.decomposition import TruncatedSVD, NMF
import numpy as np

variances = []
r_vals = np.arange(1, 1001)

trunc_svd = TruncatedSVD(n_components=1000, random_state=0)
X_svd = trunc_svd.fit_transform(X_tfidf)

variances = trunc_svd.explained_variance_ratio_

plt.scatter(r_vals, np.cumsum(variances))
plt.xlabel("# of Principle Components")
plt.ylabel("Percent of Variance")
plt.title("Percent of Total Variance vs. Number of Principle Components")
plt.show()
```



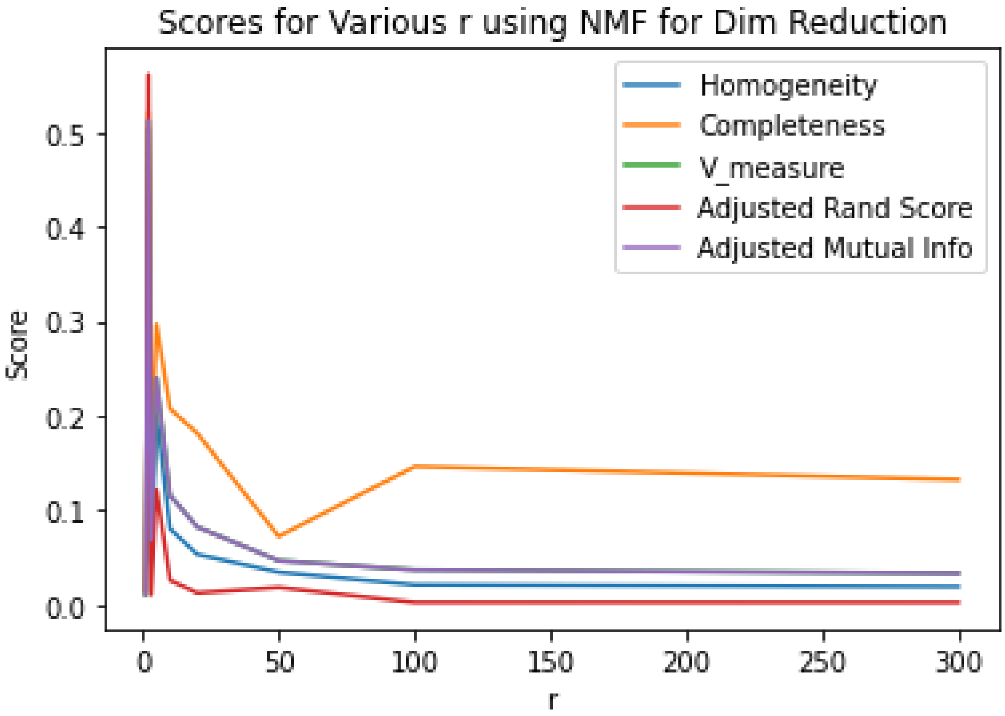
[92]:

```python
#Q5
r_vals_5 = [1,2,3,5,10,20,50,100,300]
metric_scores_svd = []
```

```
metric_scores_nmf = []

for r in r_vals_5:
    nmf_model = NMF(n_components=r, init='random', random_state=0, max_iter=1000)
    nmf_red = nmf_model.fit_transform(X_tfidf)
    svd_red = X_svd[:,:r]

    kmeans_clf.fit(nmf_red)
    metric_scores_nmf.append((metrics.homogeneity_score(dataset_targets_bin,␣
 ↪kmeans_clf.labels_),
                              metrics.completeness_score(dataset_targets_bin,␣
 ↪kmeans_clf.labels_),
                              metrics.v_measure_score(dataset_targets_bin,␣
 ↪kmeans_clf.labels_),
                              metrics.adjusted_rand_score(dataset_targets_bin,␣
 ↪kmeans_clf.labels_),
                              metrics.
 ↪adjusted_mutual_info_score(dataset_targets_bin, kmeans_clf.labels_)))
    kmeans_clf.fit(svd_red)
    metric_scores_svd.append((metrics.homogeneity_score(dataset_targets_bin,␣
 ↪kmeans_clf.labels_),
                              metrics.completeness_score(dataset_targets_bin,␣
 ↪kmeans_clf.labels_),
                              metrics.v_measure_score(dataset_targets_bin,␣
 ↪kmeans_clf.labels_),
                              metrics.adjusted_rand_score(dataset_targets_bin,␣
 ↪kmeans_clf.labels_),
                              metrics.
 ↪adjusted_mutual_info_score(dataset_targets_bin, kmeans_clf.labels_)))
```

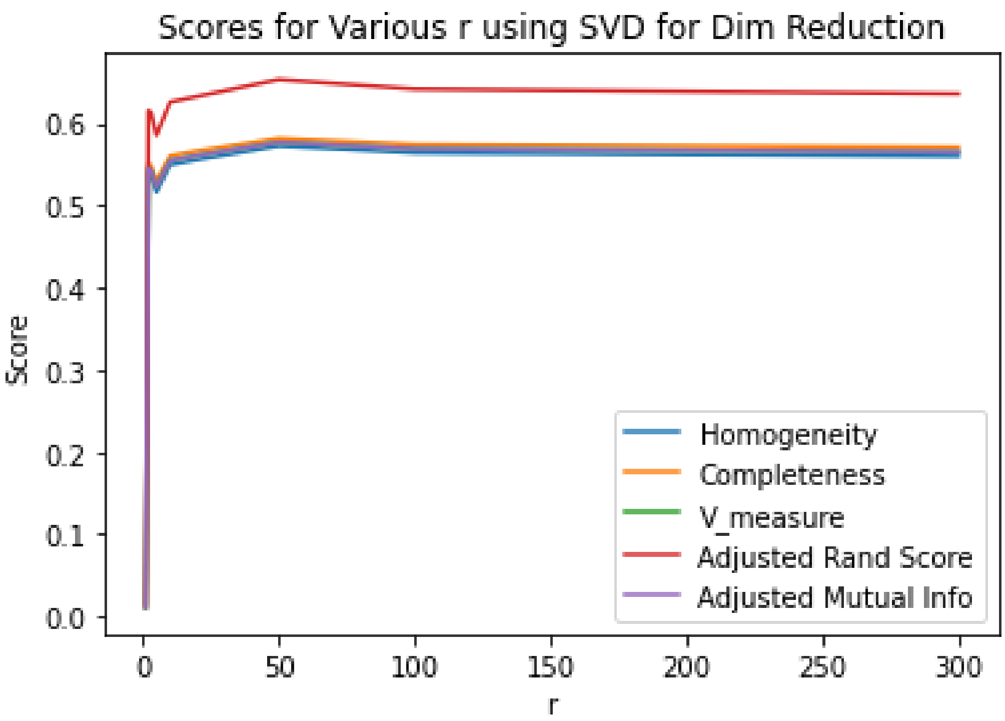
[93]:

```
plt.plot(r_vals_5, metric_scores_nmf)
plt.legend(("Homogeneity","Completeness","V_measure", "Adjusted Rand Score",␣
 ↪"Adjusted Mutual Info"))
plt.xlabel("r")
plt.ylabel("Score")
plt.title("Scores for Various r using NMF for Dim Reduction")
plt.show()
```



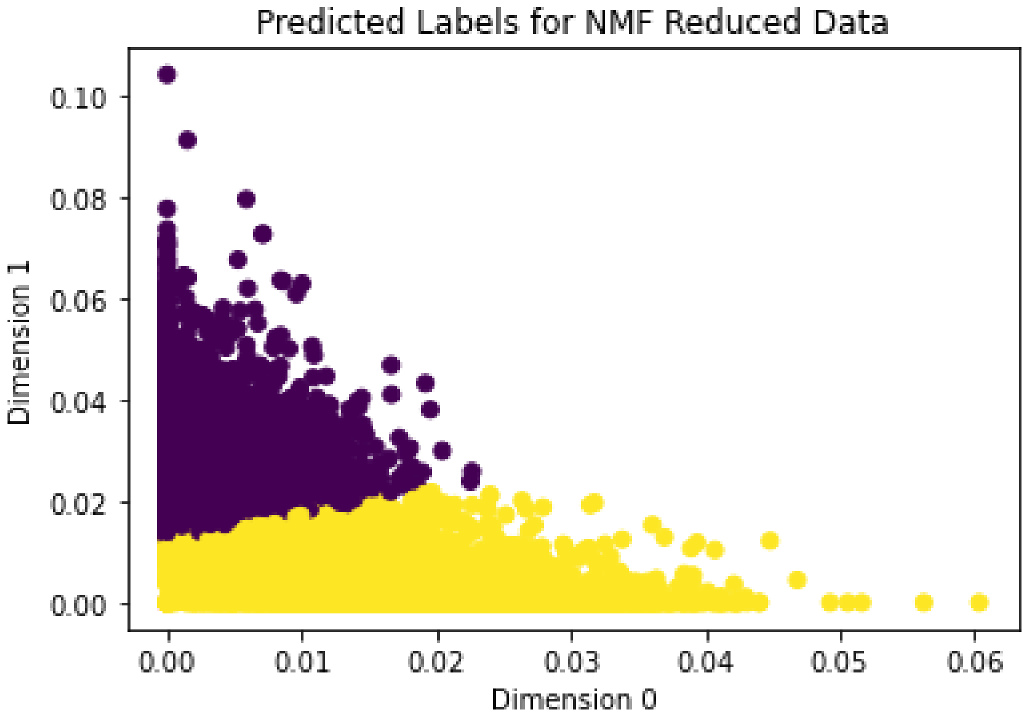
[94]:

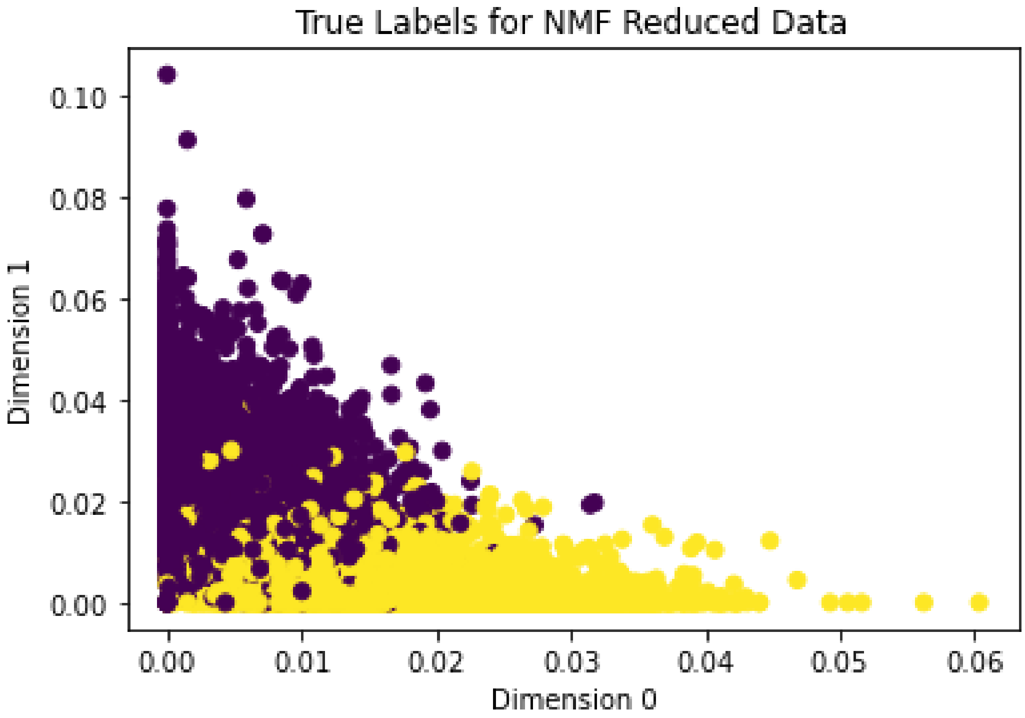
```
plt.plot(r_vals_5, metric_scores_svd)
plt.legend(("Homogeneity","Completeness","V_measure", "Adjusted Rand Score",␣
 ↪"Adjusted Mutual Info"))
plt.xlabel("r")
plt.ylabel("Score")
plt.title("Scores for Various r using SVD for Dim Reduction")
plt.show()
```

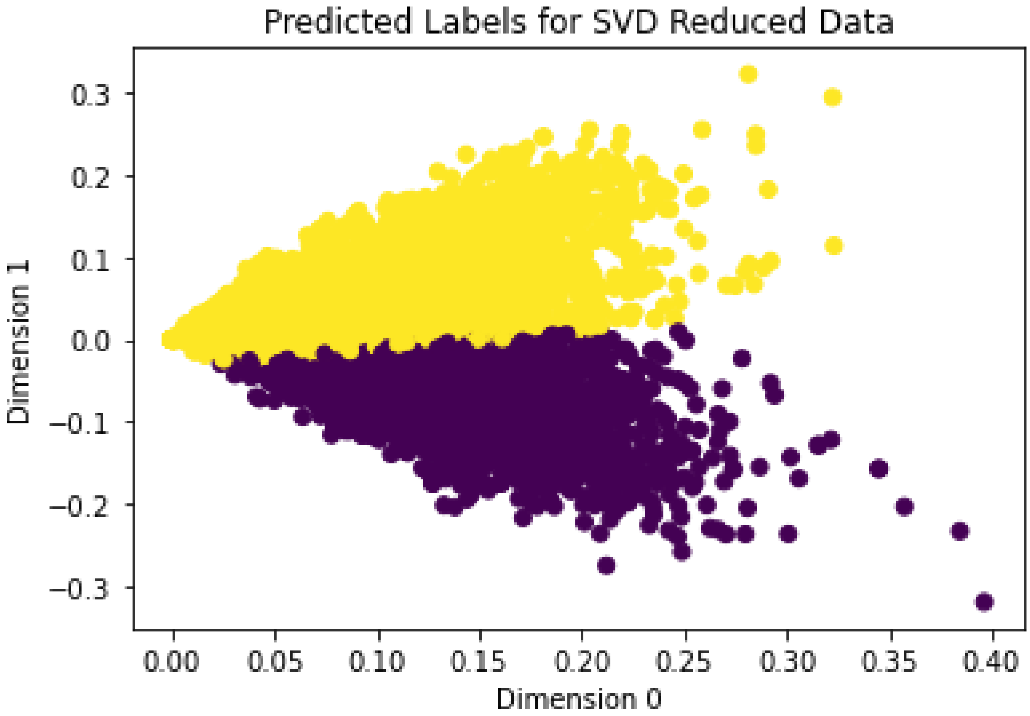


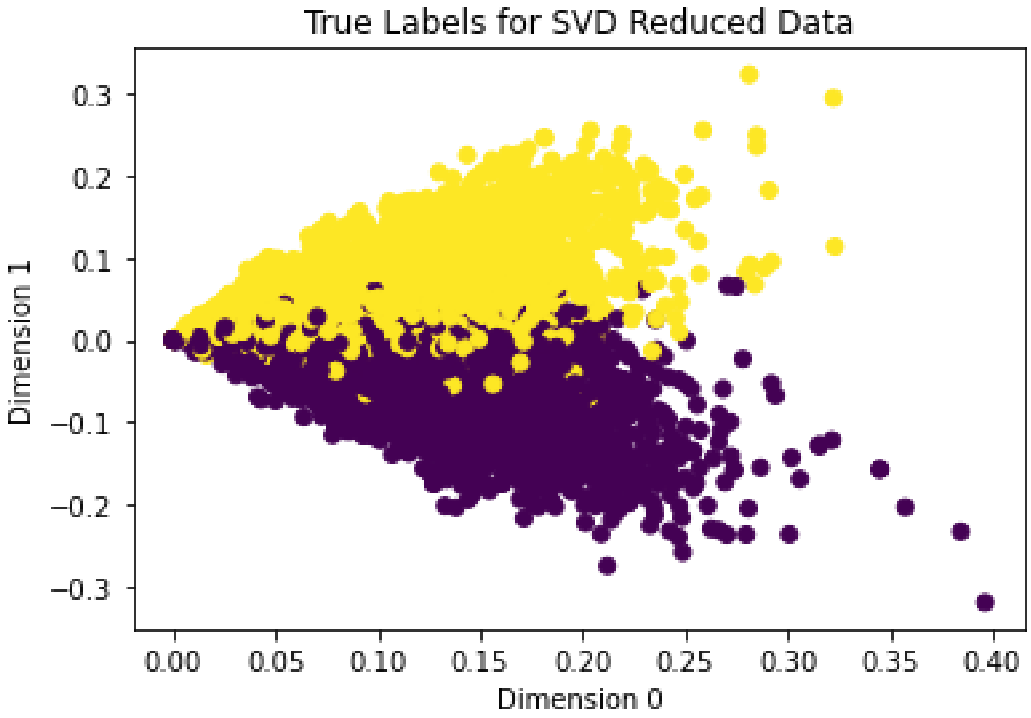
[95]:

```
#Q7

nmf_best_k = 2
svd_best_k = 50

trunc_svd_q3 = TruncatedSVD(n_components=2, random_state=0)

nmf_model = NMF(n_components=nmf_best_k, init='random', random_state=0,␣
 ↪max_iter=1000)
nmf_red = nmf_model.fit_transform(X_tfidf)
nmf_red_SVD = nmf_red#trunc_svd_q3.fit_transform(nmf_red)
kmeans_clf.fit(nmf_red_SVD)
nmf_labels = kmeans_clf.labels_

svd_red = X_svd[:,:svd_best_k]
svd_red_SVD = trunc_svd_q3.fit_transform(svd_red)
kmeans_clf.fit(svd_red_SVD)
svd_labels = kmeans_clf.labels_

plt.scatter(nmf_red_SVD[:,0], nmf_red_SVD[:,1], c=nmf_labels)
plt.title("Predicted Labels for NMF Reduced Data")
plt.xlabel("Dimension 0")
plt.ylabel("Dimension 1")
plt.show()

plt.scatter(nmf_red_SVD[:,0], nmf_red_SVD[:,1], c=dataset_targets_bin)
plt.title("True Labels for NMF Reduced Data")
plt.xlabel("Dimension 0")
plt.ylabel("Dimension 1")
plt.show()

plt.scatter(svd_red_SVD[:,0], svd_red_SVD[:,1], c=svd_labels)
plt.title("Predicted Labels for SVD Reduced Data")
plt.xlabel("Dimension 0")
plt.ylabel("Dimension 1")
plt.show()

plt.scatter(svd_red_SVD[:,0], svd_red_SVD[:,1], c=dataset_targets_bin)
plt.title("True Labels for SVD Reduced Data")
plt.xlabel("Dimension 0")
plt.ylabel("Dimension 1")
plt.show()
```




True Labels for NMF Reduced Data
Dimension 1
0.10
0.08
0.06
0.04
0.02
0.00
0.00
0.01
0.02
0.03
0.04
0.05
0.06
Dimension 0

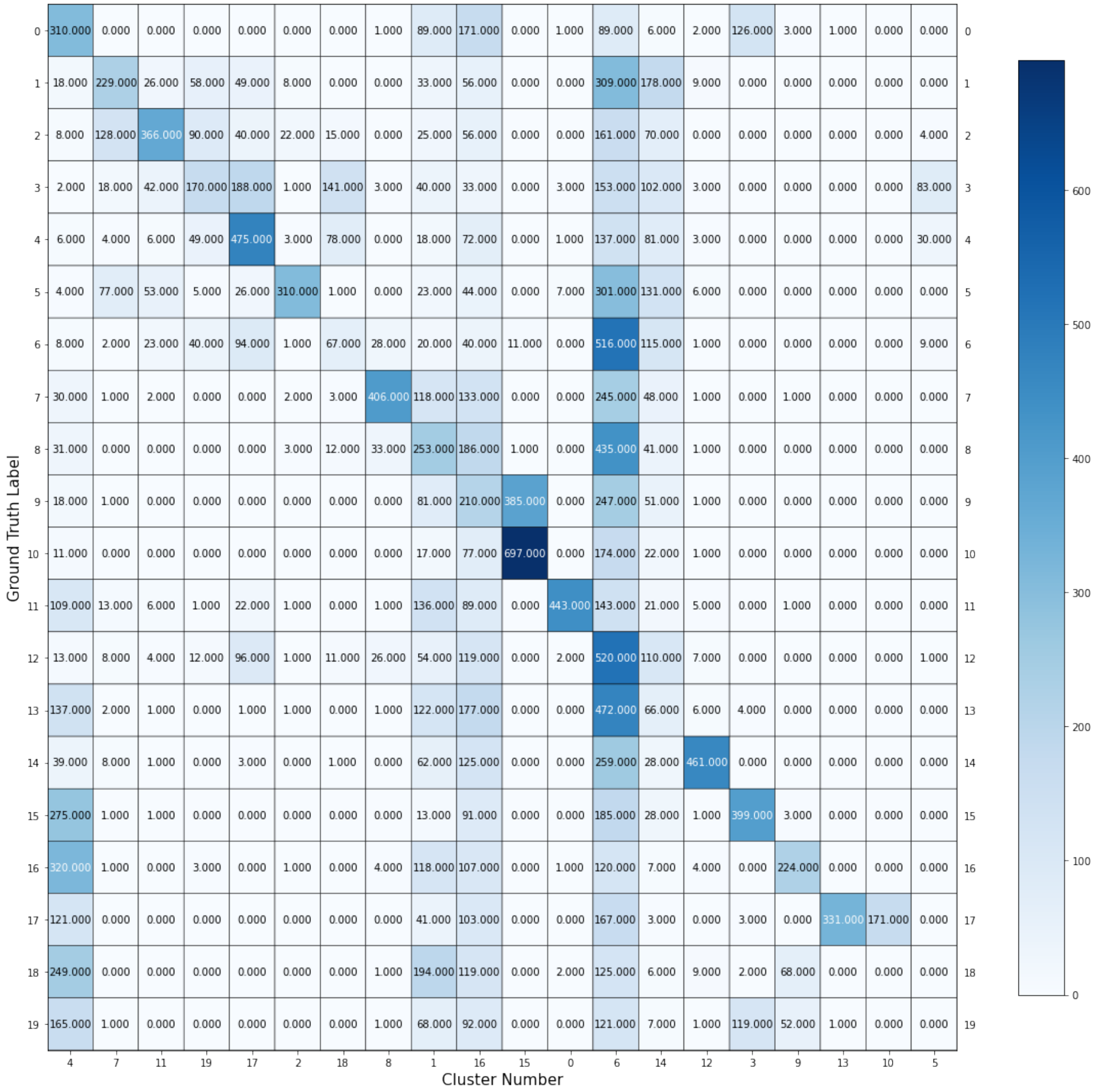

Predicted Labels for SVD Reduced Data
Dimension 1
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Dimension 0



[96]:

```
#Q9
#get all 20 categories this time
full_dataset = fetch_20newsgroups(subset='all', shuffle = True, random_state =␣
 ↪0, remove=('headers','footers'))
full_count_vect = CountVectorizer(min_df=3,analyzer=stem_rmv_punc_nolem,␣
 ↪stop_words='english')
full_X_counts = full_count_vect.fit_transform(full_dataset.data)
full_X_tfidf = tfidf_transformer.fit_transform(full_X_counts)

#using SVD with r=50 - best results from earlier
full_trunc_svd = TruncatedSVD(n_components=1000, random_state=0)
full_X_svd = full_trunc_svd.fit_transform(full_X_tfidf)
full_svd_red = full_X_svd[:,:50]

#k-means w/20 clusters
full_kmeans_clf = KMeans(n_clusters=20, max_iter=1500, random_state=0, n_init=50)
full_kmeans_clf.fit(full_svd_red)

#metrics!
from scipy.optimize import linear_sum_assignment
from sklearn.metrics import confusion_matrix

cm = contingency_matrix(full_dataset.target, full_kmeans_clf.labels_)
```

```
rows, cols=linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
 ↪size=(15,15), xlabel="Cluster Number", ylabel="Ground Truth Label")

print("Homogeneity score: ", metrics.homogeneity_score(full_dataset.target,␣
 ↪full_kmeans_clf.labels_))
print("Completeness score: ", metrics.completeness_score(full_dataset.target,␣
 ↪full_kmeans_clf.labels_))
print("V_measure score: ", metrics.v_measure_score(full_dataset.target,␣
 ↪full_kmeans_clf.labels_))
print("Adjusted Rand Index score: ", metrics.adjusted_rand_score(full_dataset.
 ↪target, full_kmeans_clf.labels_))
print("Adjusted Mutual Information Score: ",metrics.
 ↪adjusted_mutual_info_score(full_dataset.target, full_kmeans_clf.labels_))
```



	4	7	11	19	17	2	18	8	1	16	15	0	6	14	12	3	9	13	10	5
0	310.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	89.000	171.000	0.000	1.000	89.000	6.000	2.000	126.000	3.000	1.000	0.000	0.000
1	18.000	229.000	26.000	58.000	49.000	8.000	0.000	0.000	33.000	56.000	0.000	0.000	309.000	178.000	9.000	0.000	0.000	0.000	0.000	0.000
2	8.000	128.000	366.000	90.000	40.000	22.000	15.000	0.000	25.000	56.000	0.000	0.000	161.000	70.000	0.000	0.000	0.000	0.000	0.000	4.000
3	2.000	18.000	42.000	170.000	188.000	1.000	141.000	3.000	40.000	33.000	0.000	3.000	153.000	102.000	3.000	0.000	0.000	0.000	0.000	83.000
4	6.000	4.000	6.000	49.000	475.000	3.000	78.000	0.000	18.000	72.000	0.000	1.000	137.000	81.000	3.000	0.000	0.000	0.000	0.000	30.000
5	4.000	77.000	53.000	5.000	26.000	310.000	1.000	0.000	23.000	44.000	0.000	7.000	301.000	131.000	6.000	0.000	0.000	0.000	0.000	0.000
6	8.000	2.000	23.000	40.000	94.000	1.000	67.000	28.000	20.000	40.000	11.000	0.000	516.000	115.000	1.000	0.000	0.000	0.000	0.000	9.000
7	30.000	1.000	2.000	0.000	0.000	2.000	3.000	406.000	118.000	133.000	0.000	0.000	245.000	48.000	1.000	0.000	1.000	0.000	0.000	0.000
8	31.000	0.000	0.000	0.000	0.000	3.000	12.000	33.000	253.000	186.000	1.000	0.000	435.000	41.000	1.000	0.000	0.000	0.000	0.000	0.000
9	18.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	81.000	210.000	385.000	0.000	247.000	51.000	1.000	0.000	0.000	0.000	0.000	0.000
10	11.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	17.000	77.000	697.000	0.000	174.000	22.000	1.000	0.000	0.000	0.000	0.000	0.000
11	109.000	13.000	6.000	1.000	22.000	1.000	0.000	1.000	136.000	89.000	0.000	443.000	143.000	21.000	5.000	0.000	1.000	0.000	0.000	0.000
12	13.000	8.000	4.000	12.000	96.000	1.000	11.000	26.000	54.000	119.000	0.000	2.000	520.000	110.000	7.000	0.000	0.000	0.000	0.000	1.000
13	137.000	2.000	1.000	0.000	1.000	1.000	0.000	1.000	122.000	177.000	0.000	0.000	472.000	66.000	6.000	4.000	0.000	0.000	0.000	0.000
14	39.000	8.000	1.000	0.000	3.000	0.000	1.000	0.000	62.000	125.000	0.000	0.000	259.000	28.000	461.000	0.000	0.000	0.000	0.000	0.000
15	275.000	1.000	1.000	0.000	0.000	0.000	0.000	0.000	13.000	91.000	0.000	0.000	185.000	28.000	1.000	399.000	3.000	0.000	0.000	0.000
16	320.000	1.000	0.000	3.000	0.000	1.000	0.000	4.000	118.000	107.000	0.000	1.000	120.000	7.000	4.000	0.000	224.000	0.000	0.000	0.000
17	121.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	41.000	103.000	0.000	0.000	167.000	3.000	0.000	3.000	0.000	331.000	171.000	0.000
18	249.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	194.000	119.000	0.000	2.000	125.000	6.000	9.000	2.000	68.000	0.000	0.000	0.000
19	165.000	1.000	0.000	0.000	0.000	0.000	0.000	1.000	68.000	92.000	0.000	0.000	121.000	7.000	1.000	119.000	52.000	1.000	0.000	0.000

```
Homogeneity score:  0.31630458145750995
Completeness score:  0.3682552023183464
V_measure score:  0.3403086491478565
Adjusted Rand Index score:  0.10379799750286293
Adjusted Mutual Information Score:  0.3380068278889245
```

[97]:

```
#Q10

r_vals = [2,5,10,30,50,100]

for i in r_vals:
    nmf_KL = NMF(n_components=i, init='random', random_state=0, max_iter=1000,␣
 ↪beta_loss = "kullback-leibler", solver='mu')
    nmf_red_KL = nmf_KL.fit_transform(full_X_tfidf)
    kmeans_clf_KL = KMeans(n_clusters=20, max_iter=1500, random_state=0,␣
 ↪n_init=50)
    kmeans_clf_KL.fit(nmf_red_KL)
    print()
    print("r = " + str(i))
    print("Homogeneity score: ", metrics.homogeneity_score(full_dataset.target,␣
 ↪kmeans_clf_KL.labels_))
    print("Completeness score: ", metrics.completeness_score(full_dataset.
 ↪target, kmeans_clf_KL.labels_))
    print("V_measure score: ", metrics.v_measure_score(full_dataset.target,␣
 ↪kmeans_clf_KL.labels_))
    print("Adjusted Rand Index score: ", metrics.
 ↪adjusted_rand_score(full_dataset.target, kmeans_clf_KL.labels_))
    print("Adjusted Mutual Information Score: ",metrics.
 ↪adjusted_mutual_info_score(full_dataset.target, kmeans_clf_KL.labels_))
```

```
r = 2
Homogeneity score:  0.19676527887644293
Completeness score:  0.2136834606423116
V_measure score:  0.2048756966527202
Adjusted Rand Index score:  0.0604569445452653
Adjusted Mutual Information Score:  0.20217480540664073

r = 5
Homogeneity score:  0.35641214262487414
Completeness score:  0.38144347646289517
V_measure score:  0.3685032226345344
Adjusted Rand Index score:  0.15027690007707672
Adjusted Mutual Information Score:  0.3663744867466823

r = 10
Homogeneity score:  0.37714629232162333
Completeness score:  0.3900505160910544
V_measure score:  0.3834898799077068
Adjusted Rand Index score:  0.18833101982676798
Adjusted Mutual Information Score:  0.38146535004967785

r = 30
Homogeneity score:  0.36866650441353743
Completeness score:  0.42527782891459665
V_measure score:  0.39495386265506316
Adjusted Rand Index score:  0.12617792572471453
Adjusted Mutual Information Score:  0.3928569966801935

r = 50
Homogeneity score:  0.30640759897200137
Completeness score:  0.47873972223954575
V_measure score:  0.3736610566220568
Adjusted Rand Index score:  0.05415577061481132
Adjusted Mutual Information Score:  0.37118024804272814

r = 100
Homogeneity score:  0.1924960903730588
Completeness score:  0.406934904386975
V_measure score:  0.26135911828245
Adjusted Rand Index score:  0.028607969901381668
Adjusted Mutual Information Score:  0.25806962570480285
```

[153]:

```
#Q11

import umap

n_vals = [2,5,10,50,100,200,300,500,700,1000]
metric_scores_euc = []
metric_scores_cos = []


for i in n_vals:
    print("working on reducing for n = " + str(i))
    reducer_euc = umap.UMAP(n_components=i, init='random', metric="euclidean")
    embedding_euc = reducer_euc.fit_transform(full_X_tfidf)
    reducer_cos = umap.UMAP(n_components=i, init='random', metric="cosine")
    embedding_cos = reducer_cos.fit_transform(full_X_tfidf)

    print("working on kmeans for n = " + str(i))
    kmeans_clf_euc = KMeans(n_clusters=20, max_iter=1500, random_state=0,␣
 ↪n_init=50)
    kmeans_clf_euc.fit(embedding_euc)
    kmeans_clf_cos = KMeans(n_clusters=20, max_iter=1500, random_state=0,␣
 ↪n_init=50)
    kmeans_clf_cos.fit(embedding_cos)

    metric_scores_euc.append((metrics.homogeneity_score(full_dataset.target,␣
 ↪kmeans_clf_euc.labels_),
                              metrics.completeness_score(full_dataset.target,␣
 ↪kmeans_clf_euc.labels_),
                              metrics.v_measure_score(full_dataset.target,␣
 ↪kmeans_clf_euc.labels_),
                              metrics.adjusted_rand_score(full_dataset.target,␣
 ↪kmeans_clf_euc.labels_),
                              metrics.adjusted_mutual_info_score(full_dataset.
 ↪target, kmeans_clf_euc.labels_)))

    metric_scores_cos.append((metrics.homogeneity_score(full_dataset.target,␣
 ↪kmeans_clf_cos.labels_),
                              metrics.completeness_score(full_dataset.target,␣
 ↪kmeans_clf_cos.labels_),
                              metrics.v_measure_score(full_dataset.target,␣
 ↪kmeans_clf_cos.labels_),
                              metrics.adjusted_rand_score(full_dataset.target,␣
 ↪kmeans_clf_cos.labels_),
                              metrics.adjusted_mutual_info_score(full_dataset.
 ↪target, kmeans_clf_cos.labels_)))


plt.plot(n_vals, metric_scores_euc)
plt.legend(("Homogeneity","Completeness","V_measure", "Adjusted Rand Score",␣
 ↪"Adjusted Mutual Info"))
plt.xlabel("n_components")
plt.ylabel("Score")
plt.title("Scores for Various n_components Using UMAP w/Euclidean Metric")
plt.show()

plt.plot(n_vals, metric_scores_cos)
plt.legend(("Homogeneity","Completeness","V_measure", "Adjusted Rand Score",␣
 ↪"Adjusted Mutual Info"))
plt.xlabel("n_components")
plt.ylabel("Score")
plt.title("Scores for Various n_components Using UMAP w/Cosine Metric")
plt.show()
```

```
working on reducing for n = 2
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-153-abf5f6750690> in <module>()
     11     print("working on reducing for n = " + str(i))
     12     reducer_euc = umap.UMAP(n_components=i, init='random',␣
 ↪metric="euclidean")
---> 13     embedding_euc = reducer_euc.fit_transform(full_X_tfidf)
     14     reducer_cos = umap.UMAP(n_components=i, init='random', metric="cosine")
     15     embedding_cos = reducer_cos.fit_transform(full_X_tfidf)

C:\Users\loicm\anaconda3\envs\ece219_2\lib\site-packages\umap\umap_.py in␣
 ↪fit_transform(self, X, y)
   2012             Embedding of the training data in low-dimensional space.
   2013         """
-> 2014         self.fit(X, y)
   2015         return self.embedding_
   2016

C:\Users\loicm\anaconda3\envs\ece219_2\lib\site-packages\umap\umap_.py in␣
 ↪fit(self, X, y)
   1799                 self.low_memory,
   1800                 use_pynndescent=True,
-> 1801                 verbose=self.verbose,
   1802             )
   1803

C:\Users\loicm\anaconda3\envs\ece219_2\lib\site-packages\umap\umap_.py in␣
 ↪nearest_neighbors(X, n_neighbors, metric, metric_kwds, angular, random_state,␣
 ↪low_memory, use_pynndescent, verbose)
    298                 max_candidates=60,
    299                 low_memory=low_memory,
--> 300                 verbose=verbose,
    301             )
    302             knn_indices, knn_dists = nnd.neighbor_graph

C:\Users\loicm\anaconda3\envs\ece219_2\lib\site-packages\pynndescent\pynndescent_
 ↪py in __init__(self, data, metric, metric_kwds, n_neighbors, n_trees,␣
 ↪leaf_size, pruning_degree_multiplier, diversify_prob, n_search_trees,␣
 ↪tree_init, init_graph, random_state, low_memory, max_candidates, n_iters,␣
 ↪delta, n_jobs, compressed, verbose)
    789                 current_random_state,
    790                 self.n_jobs,
--> 791                 self._angular_trees,
    792             )
    793             leaf_array = rptree_leaf_array(self._rp_forest)
```

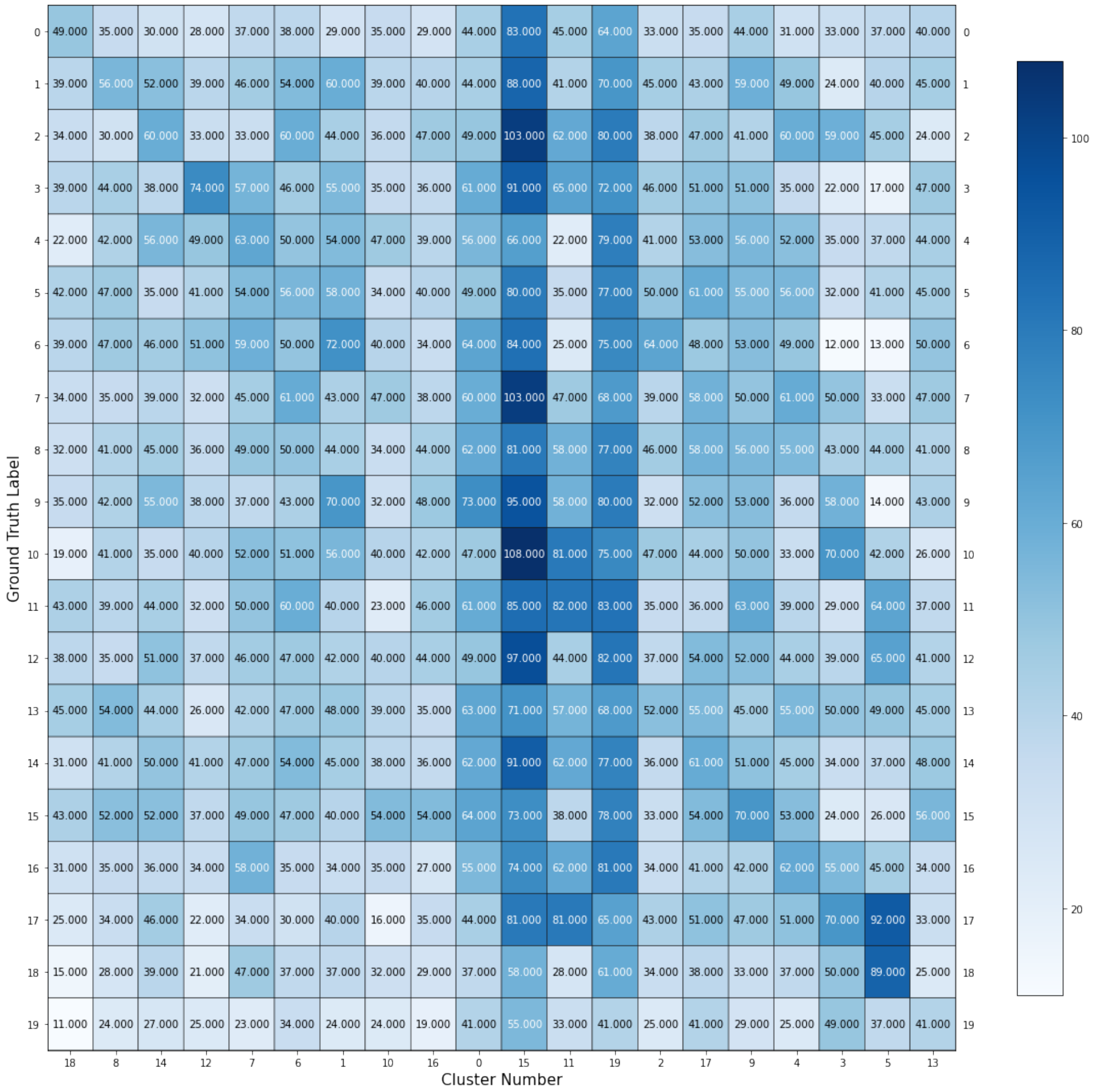
```
C:\Users\loicm\anaconda3\envs\ece219_2\lib\site-packages\pynndescent\rp_trees.py␣
 ↪in make_forest(data, n_neighbors, n_trees, leaf_size, rng_state, random_state,
 ↪n_jobs, angular)
    994                         angular,
    995                     )
--> 996                     for i in range(n_trees)
    997                 )
    998         else:

C:\Users\loicm\anaconda3\envs\ece219_2\lib\site-packages\joblib\parallel.py in␣
 ↪__call__(self, iterable)
   1059
   1060                 with self._backend.retrieval_context():
-> 1061                     self.retrieve()
   1062                 # Make sure that we get a last message telling us we are done
   1063                 elapsed_time = time.time() - self._start_time

C:\Users\loicm\anaconda3\envs\ece219_2\lib\site-packages\joblib\parallel.py in␣
 ↪retrieve(self)
    938                 try:
    939                     if getattr(self._backend, 'supports_timeout', False):
--> 940                         self._output.extend(job.get(timeout=self.timeout))
    941                     else:
    942                         self._output.extend(job.get())

C:\Users\loicm\anaconda3\envs\ece219_2\lib\multiprocessing\pool.py in get(self,␣
 ↪timeout)
    649
    650     def get(self, timeout=None):
--> 651         self.wait(timeout)
    652         if not self.ready():
    653             raise TimeoutError

C:\Users\loicm\anaconda3\envs\ece219_2\lib\multiprocessing\pool.py in wait(self,␣
 ↪timeout)
    646
    647     def wait(self, timeout=None):
--> 648         self._event.wait(timeout)
    649
    650     def get(self, timeout=None):

C:\Users\loicm\anaconda3\envs\ece219_2\lib\threading.py in wait(self, timeout)
    550                 signaled = self._flag
    551                 if not signaled:
--> 552                     signaled = self._cond.wait(timeout)
    553                 return signaled
    554
```

```
C:\Users\loicm\anaconda3\envs\ece219_2\lib\threading.py in wait(self, timeout)
    294         try:    # restore state no matter what (e.g., KeyboardInterrupt)
    295             if timeout is None:
--> 296                 waiter.acquire()
    297                 gotit = True
    298             else:

KeyboardInterrupt:
```

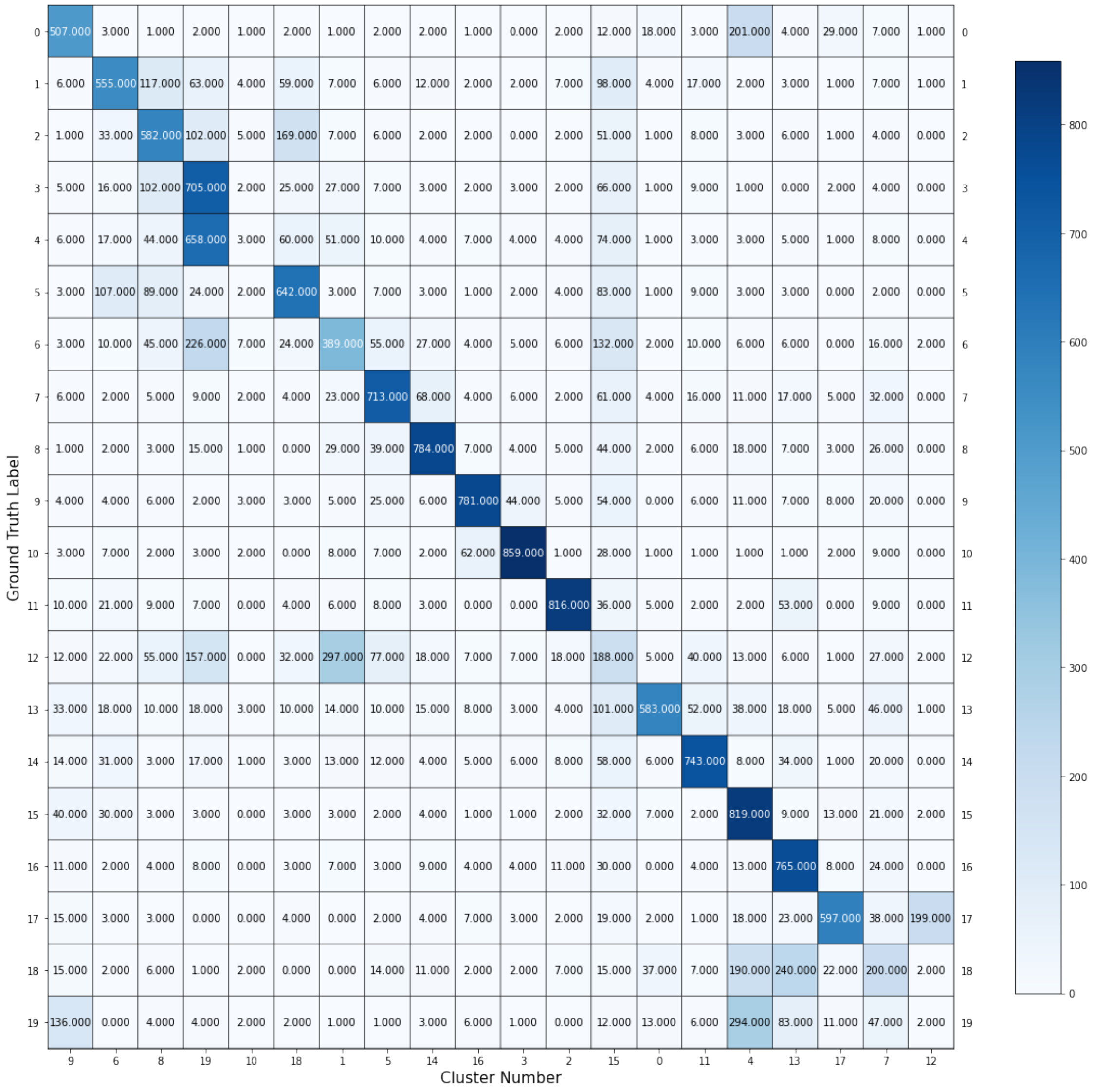
[143]:
```
#Q12
#Best n_components = 5 for both euclidean and cosine metrics

reducer_euc = umap.UMAP(n_components=5, init='random', metric="euclidean",␣
 ↪random_state=42)
embedding_euc = reducer_euc.fit_transform(full_X_tfidf)
reducer_cos = umap.UMAP(n_components=5, init='random', metric="cosine",␣
 ↪random_state=42)
embedding_cos = reducer_cos.fit_transform(full_X_tfidf)

kmeans_clf_euc = KMeans(n_clusters=20, max_iter=1500, random_state=0, n_init=50)
kmeans_clf_euc.fit(embedding_euc)
kmeans_clf_cos = KMeans(n_clusters=20, max_iter=1500, random_state=0, n_init=50)
kmeans_clf_cos.fit(embedding_cos)

print("Performance of UMAP reduced data for n_components=5")
print()
print("Euclidean metric")
print("Homogeneity score: ", metrics.homogeneity_score(full_dataset.target,␣
 ↪kmeans_clf_euc.labels_))
print("Completeness score: ", metrics.completeness_score(full_dataset.target,␣
 ↪kmeans_clf_euc.labels_))
print("V_measure score: ", metrics.v_measure_score(full_dataset.target,␣
 ↪kmeans_clf_euc.labels_))
print("Adjusted Rand Index score: ", metrics.adjusted_rand_score(full_dataset.
 ↪target, kmeans_clf_euc.labels_))
print("Adjusted Mutual Information Score: ",metrics.
 ↪adjusted_mutual_info_score(full_dataset.target, kmeans_clf_euc.labels_))

print()
print("Cosine metric")
print("Homogeneity score: ", metrics.homogeneity_score(full_dataset.target,␣
 ↪kmeans_clf_cos.labels_))
print("Completeness score: ", metrics.completeness_score(full_dataset.target,␣
 ↪kmeans_clf_cos.labels_))
print("V_measure score: ", metrics.v_measure_score(full_dataset.target,␣
 ↪kmeans_clf_cos.labels_))
```

```
print("Adjusted Rand Index score: ", metrics.adjusted_rand_score(full_dataset.
 ↪target, kmeans_clf_cos.labels_))
print("Adjusted Mutual Information Score: ",metrics.
 ↪adjusted_mutual_info_score(full_dataset.target, kmeans_clf_cos.labels_))

cm = contingency_matrix(full_dataset.target, kmeans_clf_euc.labels_)
rows, cols=linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
 ↪size=(15,15), xlabel="Cluster Number", ylabel="Ground Truth Label")

cm = contingency_matrix(full_dataset.target, kmeans_clf_cos.labels_)
rows, cols=linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
 ↪size=(15,15), xlabel="Cluster Number", ylabel="Ground Truth Label")
```

```
Performance of UMAP reduced data for n_components=5

Euclidean metric
Homogeneity score:  0.008428056566878686
Completeness score:  0.008490790660743977
V_measure score:  0.008459307306639803
Adjusted Rand Index score:  0.0014762057832842827
Adjusted Mutual Information Score:  0.00524781170211466

Cosine metric
Homogeneity score:  0.5347926404764849
Completeness score:  0.5549487009354954
V_measure score:  0.5446842655666289
Adjusted Rand Index score:  0.4167210932677909
Adjusted Mutual Information Score:  0.5431756896373161
```



	18	8	14	12	7	6	1	10	16	0	15	11	19	2	17	9	4	3	5	13	
0	49.000	35.000	30.000	28.000	37.000	38.000	29.000	35.000	29.000	44.000	83.000	45.000	64.000	33.000	35.000	44.000	31.000	33.000	37.000	40.000	0
1	39.000	56.000	52.000	39.000	46.000	54.000	60.000	39.000	40.000	44.000	88.000	41.000	70.000	45.000	43.000	59.000	49.000	24.000	40.000	45.000	1
2	34.000	30.000	60.000	33.000	33.000	60.000	44.000	36.000	47.000	49.000	103.000	62.000	80.000	38.000	47.000	41.000	60.000	59.000	45.000	24.000	2
3	39.000	44.000	38.000	74.000	57.000	46.000	55.000	35.000	36.000	61.000	91.000	65.000	72.000	46.000	51.000	51.000	35.000	22.000	17.000	47.000	3
4	22.000	42.000	56.000	49.000	63.000	50.000	54.000	47.000	39.000	56.000	66.000	22.000	79.000	41.000	53.000	56.000	52.000	35.000	37.000	44.000	4
5	42.000	47.000	35.000	41.000	54.000	56.000	58.000	34.000	40.000	49.000	80.000	35.000	77.000	50.000	61.000	55.000	56.000	32.000	41.000	45.000	5
6	39.000	47.000	46.000	51.000	59.000	50.000	72.000	40.000	34.000	64.000	84.000	25.000	75.000	64.000	48.000	53.000	49.000	12.000	13.000	50.000	6
7	34.000	35.000	39.000	32.000	45.000	61.000	43.000	47.000	38.000	60.000	103.000	47.000	68.000	39.000	58.000	50.000	61.000	50.000	33.000	47.000	7
8	32.000	41.000	45.000	36.000	49.000	50.000	44.000	34.000	44.000	62.000	81.000	58.000	77.000	46.000	58.000	56.000	55.000	43.000	44.000	41.000	8
9	35.000	42.000	55.000	38.000	37.000	43.000	70.000	32.000	48.000	73.000	95.000	58.000	80.000	32.000	52.000	53.000	36.000	58.000	14.000	43.000	9
10	19.000	41.000	35.000	40.000	52.000	51.000	56.000	40.000	42.000	47.000	108.000	81.000	75.000	47.000	44.000	50.000	33.000	70.000	42.000	26.000	10
11	43.000	39.000	44.000	32.000	50.000	60.000	40.000	23.000	46.000	61.000	85.000	82.000	83.000	35.000	36.000	63.000	39.000	29.000	64.000	37.000	11
12	38.000	35.000	51.000	37.000	46.000	47.000	42.000	40.000	44.000	49.000	97.000	44.000	82.000	37.000	54.000	52.000	44.000	39.000	65.000	41.000	12
13	45.000	54.000	44.000	26.000	42.000	47.000	48.000	39.000	35.000	63.000	71.000	57.000	68.000	52.000	55.000	45.000	55.000	50.000	49.000	45.000	13
14	31.000	41.000	50.000	41.000	47.000	54.000	45.000	38.000	36.000	62.000	91.000	62.000	77.000	36.000	61.000	51.000	45.000	34.000	37.000	48.000	14
15	43.000	52.000	52.000	37.000	49.000	47.000	40.000	54.000	54.000	64.000	73.000	38.000	78.000	33.000	54.000	70.000	53.000	24.000	26.000	56.000	15
16	31.000	35.000	36.000	34.000	58.000	35.000	34.000	35.000	27.000	55.000	74.000	62.000	81.000	34.000	41.000	42.000	62.000	55.000	45.000	34.000	16
17	25.000	34.000	46.000	22.000	34.000	30.000	40.000	16.000	35.000	44.000	81.000	81.000	65.000	43.000	51.000	47.000	51.000	70.000	92.000	33.000	17
18	15.000	28.000	39.000	21.000	47.000	37.000	37.000	32.000	29.000	37.000	58.000	28.000	61.000	34.000	38.000	33.000	37.000	50.000	89.000	25.000	18
19	11.000	24.000	27.000	25.000	23.000	34.000	24.000	24.000	19.000	41.000	55.000	33.000	41.000	25.000	41.000	29.000	25.000	49.000	37.000	41.000	19



	9	6	8	19	10	18	1	5	14	16	3	2	15	0	11	4	13	17	7	12
0	507.000	3.000	1.000	2.000	1.000	2.000	1.000	2.000	2.000	1.000	0.000	2.000	12.000	18.000	3.000	201.000	4.000	29.000	7.000	1.000
1	6.000	555.000	117.000	63.000	4.000	59.000	7.000	6.000	12.000	2.000	2.000	7.000	98.000	4.000	17.000	2.000	3.000	1.000	7.000	1.000
2	1.000	33.000	582.000	102.000	5.000	169.000	7.000	6.000	2.000	2.000	0.000	2.000	51.000	1.000	8.000	3.000	6.000	1.000	4.000	0.000
3	5.000	16.000	102.000	705.000	2.000	25.000	27.000	7.000	3.000	2.000	3.000	2.000	66.000	1.000	9.000	1.000	0.000	2.000	4.000	0.000
4	6.000	17.000	44.000	658.000	3.000	60.000	51.000	10.000	4.000	7.000	4.000	4.000	74.000	1.000	3.000	3.000	5.000	1.000	8.000	0.000
5	3.000	107.000	89.000	24.000	2.000	642.000	3.000	7.000	3.000	1.000	2.000	4.000	83.000	1.000	9.000	3.000	3.000	0.000	2.000	0.000
6	3.000	10.000	45.000	226.000	7.000	24.000	389.000	55.000	27.000	4.000	5.000	6.000	132.000	2.000	10.000	6.000	6.000	0.000	16.000	2.000
7	6.000	2.000	5.000	9.000	2.000	4.000	23.000	713.000	68.000	4.000	6.000	2.000	61.000	4.000	16.000	11.000	17.000	5.000	32.000	0.000
8	1.000	2.000	3.000	15.000	1.000	0.000	29.000	39.000	784.000	7.000	4.000	5.000	44.000	2.000	6.000	18.000	7.000	3.000	26.000	0.000
9	4.000	4.000	6.000	2.000	3.000	3.000	5.000	25.000	6.000	781.000	44.000	5.000	54.000	0.000	6.000	11.000	7.000	8.000	20.000	0.000
10	3.000	7.000	2.000	3.000	2.000	0.000	8.000	7.000	2.000	62.000	859.000	1.000	28.000	1.000	1.000	1.000	1.000	2.000	9.000	0.000
11	10.000	21.000	9.000	7.000	0.000	4.000	6.000	8.000	3.000	0.000	0.000	816.000	36.000	5.000	2.000	2.000	53.000	0.000	9.000	0.000
12	12.000	22.000	55.000	157.000	0.000	32.000	297.000	77.000	18.000	7.000	7.000	18.000	188.000	5.000	40.000	13.000	6.000	1.000	27.000	2.000
13	33.000	18.000	10.000	18.000	3.000	10.000	14.000	10.000	15.000	8.000	3.000	4.000	101.000	583.000	52.000	38.000	18.000	5.000	46.000	1.000
14	14.000	31.000	3.000	17.000	1.000	3.000	13.000	12.000	4.000	5.000	6.000	8.000	58.000	6.000	743.000	8.000	34.000	1.000	20.000	0.000
15	40.000	30.000	3.000	3.000	0.000	3.000	3.000	2.000	4.000	1.000	1.000	2.000	32.000	7.000	2.000	819.000	9.000	13.000	21.000	2.000
16	11.000	2.000	4.000	8.000	0.000	3.000	7.000	3.000	9.000	4.000	4.000	11.000	30.000	0.000	4.000	13.000	765.000	8.000	24.000	0.000
17	15.000	3.000	3.000	0.000	0.000	4.000	0.000	2.000	4.000	7.000	3.000	2.000	19.000	2.000	1.000	18.000	23.000	597.000	38.000	199.000
18	15.000	2.000	6.000	1.000	2.000	0.000	0.000	14.000	11.000	2.000	2.000	7.000	15.000	37.000	7.000	190.000	240.000	22.000	200.000	2.000
19	136.000	0.000	4.000	4.000	2.000	2.000	1.000	1.000	3.000	6.000	1.000	0.000	12.000	13.000	6.000	294.000	83.000	11.000	47.000	2.000

[142]:

```
#Q13
#using same umap parameters as above

from sklearn.cluster import AgglomerativeClustering

umap_best_n = 5
umap_best_metric = 'cosine'

umap_reducer = umap.UMAP(n_components=umap_best_n, init='random',␣
 ↪metric=umap_best_metric, random_state=42)

red_umap = umap_reducer.fit_transform(full_X_tfidf)
```

```python
agg_clust_ward = AgglomerativeClustering(n_clusters=20, linkage='ward')
agg_clust_single = AgglomerativeClustering(n_clusters=20, linkage='single')

agg_clust_ward.fit(red_umap)
agg_clust_single.fit(red_umap)

print("Ward Linkage Scores")
print("Homogeneity score: ", metrics.homogeneity_score(full_dataset.target,␣
 ↪agg_clust_ward.labels_))
print("Completeness score: ", metrics.completeness_score(full_dataset.target,␣
 ↪agg_clust_ward.labels_))
print("V_measure score: ", metrics.v_measure_score(full_dataset.target,␣
 ↪agg_clust_ward.labels_))
print("Adjusted Rand Index score: ", metrics.adjusted_rand_score(full_dataset.
 ↪target, agg_clust_ward.labels_))
print("Adjusted Mutual Information Score: ",metrics.
 ↪adjusted_mutual_info_score(full_dataset.target, agg_clust_ward.labels_))

print()
print("Single Linkage Scores")
print("Homogeneity score: ", metrics.homogeneity_score(full_dataset.target,␣
 ↪agg_clust_single.labels_))
print("Completeness score: ", metrics.completeness_score(full_dataset.target,␣
 ↪agg_clust_single.labels_))
print("V_measure score: ", metrics.v_measure_score(full_dataset.target,␣
 ↪agg_clust_single.labels_))
print("Adjusted Rand Index score: ", metrics.adjusted_rand_score(full_dataset.
 ↪target, agg_clust_single.labels_))
print("Adjusted Mutual Information Score: ",metrics.
 ↪adjusted_mutual_info_score(full_dataset.target, agg_clust_single.labels_))
```

```
Ward Linkage Scores
Homogeneity score:  0.524473300498607
Completeness score:  0.5479912976714623
V_measure score:  0.53597443687123
Adjusted Rand Index score:  0.3940501791969448
Adjusted Mutual Information Score:  0.5344316766896055

Single Linkage Scores
Homogeneity score:  0.022917457355135236
Completeness score:  0.3464156710750703
V_measure score:  0.04299081646294147
Adjusted Rand Index score:  0.00046761208355549543
Adjusted Mutual Information Score:  0.03748616963750802
```

```
[140]: #Q14 with more hyperparams

       eps_vals = [.01,.05,.1,.3,.5,.7,.9,1]
       metric_types = [ 'euclidean', 'manhattan', 'cosine']
       min_samples = [10, 20, 50, 70, 100]

       db_max_score = -100
       hdb_max_score = -100

       for eps in eps_vals:
           for metric_t in metric_types:
               for min_samp in min_samples:
                   print("Current Iteration: eps={:f}, metric={:s}, min_samples={:d}".
        ↪format(eps, metric_t, min_samp))
                   DBSCAN_clust = DBSCAN(min_samples=min_samp, metric=metric_t, eps=eps)

                   #HDB does not support cosine metric
                   if(metric_t != 'cosine'):
                       HDBSCAN_clust = HDBSCAN(min_cluster_size=100,␣
        ↪min_samples=min_samp, cluster_selection_epsilon=eps)

                   DBSCAN_clust.fit(red_umap)
                   HDBSCAN_clust.fit(red_umap)

                   homog_score = metrics.homogeneity_score(full_dataset.target,␣
        ↪DBSCAN_clust.labels_)
                   comp_score = metrics.completeness_score(full_dataset.target,␣
        ↪DBSCAN_clust.labels_)
                   v_score = metrics.v_measure_score(full_dataset.target, DBSCAN_clust.
        ↪labels_)
                   rand_score = metrics.adjusted_rand_score(full_dataset.target,␣
        ↪DBSCAN_clust.labels_)
                   adj_mut_score = metrics.adjusted_mutual_info_score(full_dataset.
        ↪target, DBSCAN_clust.labels_)

                   score_avg = (homog_score + comp_score + v_score + rand_score +␣
        ↪adj_mut_score)/5

                   labels = DBSCAN_clust.labels_
                   n_clust = len(set(labels)) - (1 if -1 in labels else 0)

                   if score_avg > db_max_score:
                       db_max_score = score_avg
                       db_max_params = (eps, metric_t, min_samp)
                       db_max_metrics = (homog_score, comp_score, v_score, rand_score,␣
        ↪adj_mut_score)
```

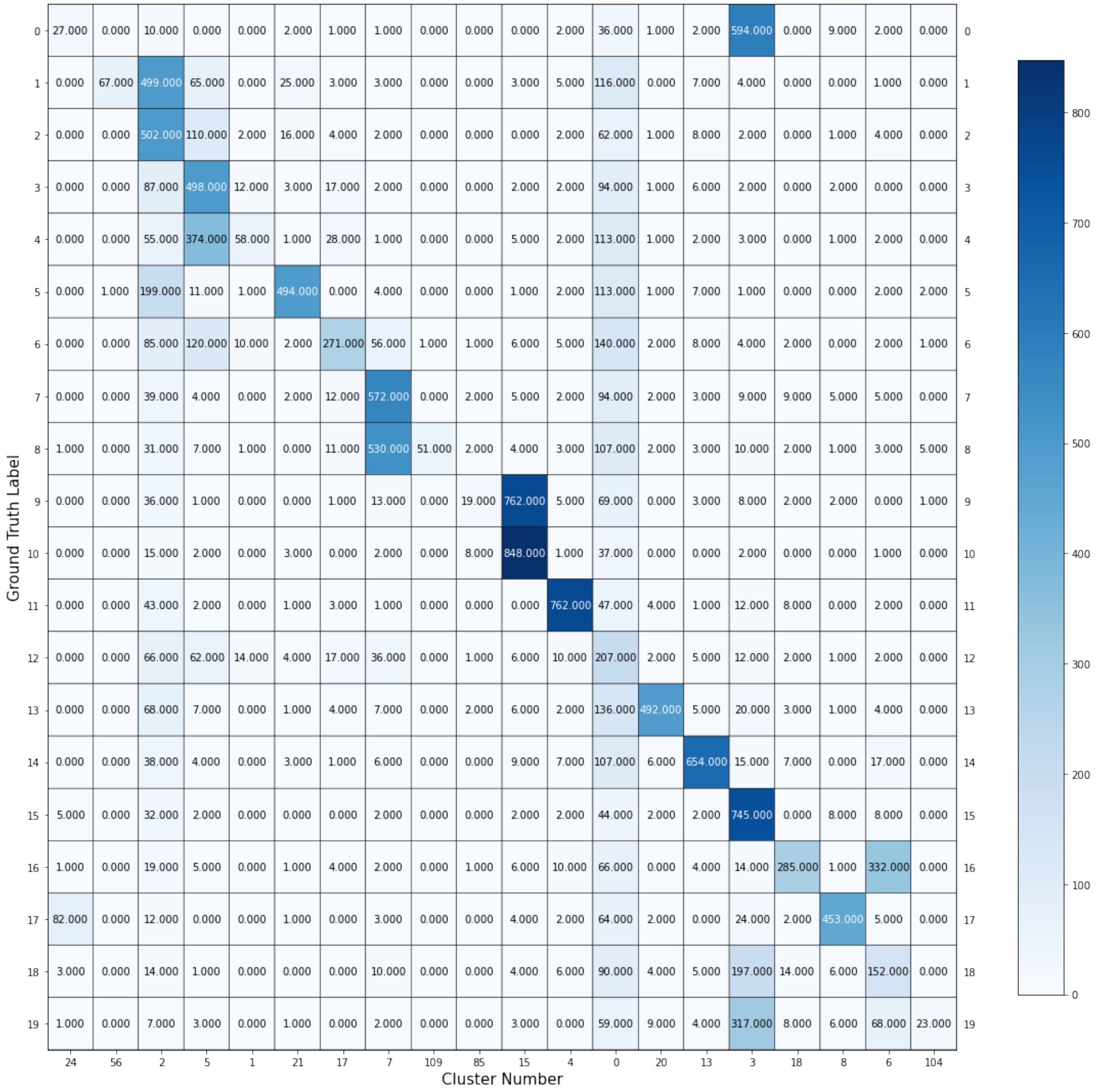
```
            homog_score = metrics.homogeneity_score(full_dataset.target,␣
 ↪HDBSCAN_clust.labels_)
            comp_score = metrics.completeness_score(full_dataset.target,␣
 ↪HDBSCAN_clust.labels_)
            v_score = metrics.v_measure_score(full_dataset.target, HDBSCAN_clust.
 ↪labels_)
            rand_score = metrics.adjusted_rand_score(full_dataset.target,␣
 ↪HDBSCAN_clust.labels_)
            adj_mut_score = metrics.adjusted_mutual_info_score(full_dataset.
 ↪target, HDBSCAN_clust.labels_)

            score_avg = (homog_score + comp_score + v_score + rand_score +␣
 ↪adj_mut_score)/5

            labels = HDBSCAN_clust.labels_
            n_clust = len(set(labels)) - (1 if -1 in labels else 0)

            if metric_t != 'cosine' and score_avg > hdb_max_score:
                hdb_max_score = score_avg
                hdb_max_params = (eps, metric_t, min_samp)
                hdb_max_metrics = (homog_score, comp_score, v_score, rand_score,␣
 ↪adj_mut_score)

print()
print("DBSCAN Best Parameters: eps={:f}, metric={:s}, min_samples={:d}".
 ↪format(db_max_params[0], db_max_params[1], db_max_params[2]))
print()
print("Best DBSCAN Scores")
print("Homogeneity score: ", db_max_metrics[0])
print("Completeness score: ", db_max_metrics[1])
print("V_measure score: ", db_max_metrics[2])
print("Adjusted Rand Index score: ", db_max_metrics[3])
print("Adjusted Mutual Information Score: ", db_max_metrics[4])

print()
print("HDBSCAN Best Parameters: eps={:f}, metric={:s}, min_samples={:d}".
 ↪format(hdb_max_params[0], hdb_max_params[1], hdb_max_params[2]))
print()
print("Best HDBSCAN Scores")
print("Homogeneity score: ", hdb_max_metrics[0])
print("Completeness score: ", hdb_max_metrics[1])
print("V_measure score: ", hdb_max_metrics[2])
print("Adjusted Rand Index score: ", hdb_max_metrics[3])
print("Adjusted Mutual Information Score: ", hdb_max_metrics[4])
```

```
Current Iteration: eps=0.010000, metric=euclidean, min_samples=10
Current Iteration: eps=0.010000, metric=euclidean, min_samples=20
Current Iteration: eps=0.010000, metric=euclidean, min_samples=50
Current Iteration: eps=0.010000, metric=euclidean, min_samples=70
Current Iteration: eps=0.010000, metric=euclidean, min_samples=100
Current Iteration: eps=0.010000, metric=manhattan, min_samples=10
Current Iteration: eps=0.010000, metric=manhattan, min_samples=20
Current Iteration: eps=0.010000, metric=manhattan, min_samples=50
Current Iteration: eps=0.010000, metric=manhattan, min_samples=70
Current Iteration: eps=0.010000, metric=manhattan, min_samples=100
Current Iteration: eps=0.010000, metric=cosine, min_samples=10
Current Iteration: eps=0.010000, metric=cosine, min_samples=20
Current Iteration: eps=0.010000, metric=cosine, min_samples=50
Current Iteration: eps=0.010000, metric=cosine, min_samples=70
Current Iteration: eps=0.010000, metric=cosine, min_samples=100
Current Iteration: eps=0.050000, metric=euclidean, min_samples=10
Current Iteration: eps=0.050000, metric=euclidean, min_samples=20
Current Iteration: eps=0.050000, metric=euclidean, min_samples=50
Current Iteration: eps=0.050000, metric=euclidean, min_samples=70
Current Iteration: eps=0.050000, metric=euclidean, min_samples=100
Current Iteration: eps=0.050000, metric=manhattan, min_samples=10
Current Iteration: eps=0.050000, metric=manhattan, min_samples=20
Current Iteration: eps=0.050000, metric=manhattan, min_samples=50
Current Iteration: eps=0.050000, metric=manhattan, min_samples=70
Current Iteration: eps=0.050000, metric=manhattan, min_samples=100
Current Iteration: eps=0.050000, metric=cosine, min_samples=10
Current Iteration: eps=0.050000, metric=cosine, min_samples=20
Current Iteration: eps=0.050000, metric=cosine, min_samples=50
Current Iteration: eps=0.050000, metric=cosine, min_samples=70
Current Iteration: eps=0.050000, metric=cosine, min_samples=100
Current Iteration: eps=0.100000, metric=euclidean, min_samples=10
Current Iteration: eps=0.100000, metric=euclidean, min_samples=20
Current Iteration: eps=0.100000, metric=euclidean, min_samples=50
Current Iteration: eps=0.100000, metric=euclidean, min_samples=70
Current Iteration: eps=0.100000, metric=euclidean, min_samples=100
Current Iteration: eps=0.100000, metric=manhattan, min_samples=10
Current Iteration: eps=0.100000, metric=manhattan, min_samples=20
Current Iteration: eps=0.100000, metric=manhattan, min_samples=50
Current Iteration: eps=0.100000, metric=manhattan, min_samples=70
Current Iteration: eps=0.100000, metric=manhattan, min_samples=100
Current Iteration: eps=0.100000, metric=cosine, min_samples=10
Current Iteration: eps=0.100000, metric=cosine, min_samples=20
Current Iteration: eps=0.100000, metric=cosine, min_samples=50
Current Iteration: eps=0.100000, metric=cosine, min_samples=70
Current Iteration: eps=0.100000, metric=cosine, min_samples=100
Current Iteration: eps=0.300000, metric=euclidean, min_samples=10
Current Iteration: eps=0.300000, metric=euclidean, min_samples=20
Current Iteration: eps=0.300000, metric=euclidean, min_samples=50
Current Iteration: eps=0.300000, metric=euclidean, min_samples=70
Current Iteration: eps=0.300000, metric=euclidean, min_samples=100
Current Iteration: eps=0.300000, metric=manhattan, min_samples=10
Current Iteration: eps=0.300000, metric=manhattan, min_samples=20
Current Iteration: eps=0.300000, metric=manhattan, min_samples=50
Current Iteration: eps=0.300000, metric=manhattan, min_samples=70
Current Iteration: eps=0.300000, metric=manhattan, min_samples=100
Current Iteration: eps=0.300000, metric=cosine, min_samples=10
Current Iteration: eps=0.300000, metric=cosine, min_samples=20
Current Iteration: eps=0.300000, metric=cosine, min_samples=50
Current Iteration: eps=0.300000, metric=cosine, min_samples=70
Current Iteration: eps=0.300000, metric=cosine, min_samples=100
Current Iteration: eps=0.500000, metric=euclidean, min_samples=10
Current Iteration: eps=0.500000, metric=euclidean, min_samples=20
Current Iteration: eps=0.500000, metric=euclidean, min_samples=50
Current Iteration: eps=0.500000, metric=euclidean, min_samples=70
Current Iteration: eps=0.500000, metric=euclidean, min_samples=100
Current Iteration: eps=0.500000, metric=manhattan, min_samples=10
Current Iteration: eps=0.500000, metric=manhattan, min_samples=20
Current Iteration: eps=0.500000, metric=manhattan, min_samples=50
Current Iteration: eps=0.500000, metric=manhattan, min_samples=70
Current Iteration: eps=0.500000, metric=manhattan, min_samples=100
Current Iteration: eps=0.500000, metric=cosine, min_samples=10
Current Iteration: eps=0.500000, metric=cosine, min_samples=20
Current Iteration: eps=0.500000, metric=cosine, min_samples=50
Current Iteration: eps=0.500000, metric=cosine, min_samples=70
Current Iteration: eps=0.500000, metric=cosine, min_samples=100
Current Iteration: eps=0.700000, metric=euclidean, min_samples=10
Current Iteration: eps=0.700000, metric=euclidean, min_samples=20
Current Iteration: eps=0.700000, metric=euclidean, min_samples=50
Current Iteration: eps=0.700000, metric=euclidean, min_samples=70
Current Iteration: eps=0.700000, metric=euclidean, min_samples=100
Current Iteration: eps=0.700000, metric=manhattan, min_samples=10
Current Iteration: eps=0.700000, metric=manhattan, min_samples=20
Current Iteration: eps=0.700000, metric=manhattan, min_samples=50
Current Iteration: eps=0.700000, metric=manhattan, min_samples=70
Current Iteration: eps=0.700000, metric=manhattan, min_samples=100
Current Iteration: eps=0.700000, metric=cosine, min_samples=10
Current Iteration: eps=0.700000, metric=cosine, min_samples=20
Current Iteration: eps=0.700000, metric=cosine, min_samples=50
Current Iteration: eps=0.700000, metric=cosine, min_samples=70
Current Iteration: eps=0.700000, metric=cosine, min_samples=100
Current Iteration: eps=0.900000, metric=euclidean, min_samples=10
Current Iteration: eps=0.900000, metric=euclidean, min_samples=20
Current Iteration: eps=0.900000, metric=euclidean, min_samples=50
Current Iteration: eps=0.900000, metric=euclidean, min_samples=70
Current Iteration: eps=0.900000, metric=euclidean, min_samples=100
Current Iteration: eps=0.900000, metric=manhattan, min_samples=10
Current Iteration: eps=0.900000, metric=manhattan, min_samples=20
Current Iteration: eps=0.900000, metric=manhattan, min_samples=50
Current Iteration: eps=0.900000, metric=manhattan, min_samples=70
Current Iteration: eps=0.900000, metric=manhattan, min_samples=100
Current Iteration: eps=0.900000, metric=cosine, min_samples=10
Current Iteration: eps=0.900000, metric=cosine, min_samples=20
Current Iteration: eps=0.900000, metric=cosine, min_samples=50
Current Iteration: eps=0.900000, metric=cosine, min_samples=70
Current Iteration: eps=0.900000, metric=cosine, min_samples=100
Current Iteration: eps=1.000000, metric=euclidean, min_samples=10
Current Iteration: eps=1.000000, metric=euclidean, min_samples=20
Current Iteration: eps=1.000000, metric=euclidean, min_samples=50
Current Iteration: eps=1.000000, metric=euclidean, min_samples=70
Current Iteration: eps=1.000000, metric=euclidean, min_samples=100
Current Iteration: eps=1.000000, metric=manhattan, min_samples=10
Current Iteration: eps=1.000000, metric=manhattan, min_samples=20
Current Iteration: eps=1.000000, metric=manhattan, min_samples=50
Current Iteration: eps=1.000000, metric=manhattan, min_samples=70
Current Iteration: eps=1.000000, metric=manhattan, min_samples=100
Current Iteration: eps=1.000000, metric=cosine, min_samples=10
Current Iteration: eps=1.000000, metric=cosine, min_samples=20
Current Iteration: eps=1.000000, metric=cosine, min_samples=50
Current Iteration: eps=1.000000, metric=cosine, min_samples=70
Current Iteration: eps=1.000000, metric=cosine, min_samples=100

DBSCAN Best Parameters: eps=0.500000, metric=manhattan, min_samples=10

Best DBSCAN Scores
Homogeneity score:  0.5407607595568981
Completeness score:  0.45471096739764455
V_measure score:  0.49401673890037173
Adjusted Rand Index score:  0.2967260971944622
Adjusted Mutual Information Score:  0.47920988791262803

HDBSCAN Best Parameters: eps=0.300000, metric=euclidean, min_samples=10

Best HDBSCAN Scores
Homogeneity score:  0.43889528158878366
Completeness score:  0.5288819318072416
V_measure score:  0.47970500064413385
Adjusted Rand Index score:  0.2565568616115164
Adjusted Mutual Information Score:  0.4780526456871693
```

[141]:

```
# Q15

DBSCAN_clust = DBSCAN(min_samples=10, eps=0.5, metric='manhattan')
DBSCAN_clust.fit(red_umap)
```

```
cm = contingency_matrix(full_dataset.target, DBSCAN_clust.labels_)
rows, cols=linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
 ↪size=(15,15), xlabel="Cluster Number", ylabel="Ground Truth Label")

#when label is -1, DBSCAN thinks the point is noise - so it might estimate fewer␣
 ↪clusters than we actually have!
labels = DBSCAN_clust.labels_
n_clust = len(set(labels)) - (1 if -1 in labels else 0)
print("Estimated number of clusters: ", n_clust)
```



	24	56	2	5	1	21	17	7	109	85	15	4	0	20	13	3	18	8	6	104
0	27.000	0.000	10.000	0.000	0.000	2.000	1.000	1.000	0.000	0.000	0.000	2.000	36.000	1.000	2.000	594.000	0.000	9.000	2.000	0.000
1	0.000	67.000	499.000	65.000	0.000	25.000	3.000	3.000	0.000	0.000	3.000	5.000	116.000	0.000	7.000	4.000	0.000	0.000	1.000	0.000
2	0.000	0.000	502.000	110.000	2.000	16.000	4.000	2.000	0.000	0.000	0.000	2.000	62.000	1.000	8.000	2.000	0.000	1.000	4.000	0.000
3	0.000	0.000	87.000	498.000	12.000	3.000	17.000	2.000	0.000	0.000	2.000	2.000	94.000	1.000	6.000	2.000	0.000	2.000	0.000	0.000
4	0.000	0.000	55.000	374.000	58.000	1.000	28.000	1.000	0.000	0.000	5.000	2.000	113.000	1.000	2.000	3.000	0.000	1.000	2.000	0.000
5	0.000	1.000	199.000	11.000	1.000	494.000	0.000	4.000	0.000	0.000	1.000	2.000	113.000	1.000	7.000	1.000	0.000	0.000	2.000	2.000
6	0.000	0.000	85.000	120.000	10.000	2.000	271.000	56.000	1.000	1.000	6.000	5.000	140.000	2.000	8.000	4.000	2.000	0.000	2.000	1.000
7	0.000	0.000	39.000	4.000	0.000	2.000	12.000	572.000	0.000	2.000	5.000	2.000	94.000	2.000	3.000	9.000	9.000	5.000	5.000	0.000
8	1.000	0.000	31.000	7.000	1.000	0.000	11.000	530.000	51.000	2.000	4.000	3.000	107.000	2.000	3.000	10.000	2.000	1.000	3.000	5.000
9	0.000	0.000	36.000	1.000	0.000	0.000	1.000	13.000	0.000	19.000	762.000	5.000	69.000	0.000	3.000	8.000	2.000	2.000	0.000	1.000
10	0.000	0.000	15.000	2.000	0.000	3.000	0.000	2.000	0.000	8.000	848.000	1.000	37.000	0.000	0.000	2.000	0.000	0.000	1.000	0.000
11	0.000	0.000	43.000	2.000	0.000	1.000	3.000	1.000	0.000	0.000	0.000	762.000	47.000	4.000	1.000	12.000	8.000	0.000	2.000	0.000
12	0.000	0.000	66.000	62.000	14.000	4.000	17.000	36.000	0.000	1.000	6.000	10.000	207.000	2.000	5.000	12.000	2.000	1.000	2.000	0.000
13	0.000	0.000	68.000	7.000	0.000	1.000	4.000	7.000	0.000	2.000	6.000	2.000	136.000	492.000	5.000	20.000	3.000	1.000	4.000	0.000
14	0.000	0.000	38.000	4.000	0.000	3.000	1.000	6.000	0.000	0.000	9.000	7.000	107.000	6.000	654.000	15.000	7.000	0.000	17.000	0.000
15	5.000	0.000	32.000	2.000	0.000	0.000	0.000	2.000	0.000	0.000	2.000	2.000	44.000	2.000	2.000	745.000	0.000	8.000	8.000	0.000
16	1.000	0.000	19.000	5.000	0.000	1.000	4.000	2.000	0.000	1.000	6.000	10.000	66.000	0.000	4.000	14.000	285.000	1.000	332.000	0.000
17	82.000	0.000	12.000	0.000	0.000	1.000	0.000	3.000	0.000	0.000	4.000	2.000	64.000	2.000	0.000	24.000	2.000	453.000	5.000	0.000
18	3.000	0.000	14.000	1.000	0.000	0.000	0.000	10.000	0.000	0.000	4.000	6.000	90.000	4.000	5.000	197.000	14.000	6.000	152.000	0.000
19	1.000	0.000	7.000	3.000	0.000	1.000	0.000	2.000	0.000	0.000	3.000	0.000	59.000	9.000	4.000	317.000	8.000	6.000	68.000	23.000

```
Estimated number of clusters:  168
```

[126]:

```
#Q16

#get data and vectorize

#Using the larger training dataset
import pandas
from sklearn.cluster import DBSCAN
from sklearn.metrics.cluster import contingency_matrix
from scipy.optimize import linear_sum_assignment
from plotmat import plot_mat
import numpy as np
import umap.umap_ as umap

bbc = pandas.read_csv('BBC_News_Train.csv')
categories = (bbc.Category.unique()).tolist()
targets = []
for i in bbc.Category:
    for j in range(0,len(categories)):
        if i == categories[j]:
            targets.append(j)
bbc['target'] = targets

X_counts = count_vect.fit_transform(bbc.Text)
bbc_X_tfidf = tfidf_transformer.fit_transform(X_counts)
print("bbc_X_tfidf shape: ", bbc_X_tfidf.shape)
```

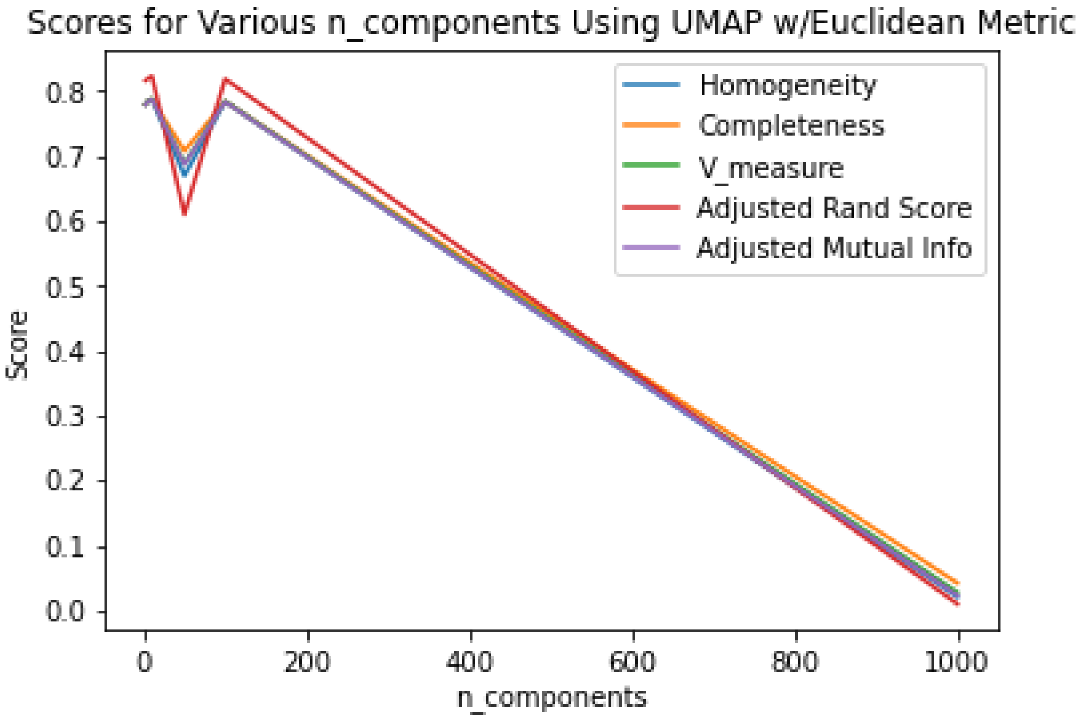
```
bbc_X_tfidf shape:  (1490, 10197)
```

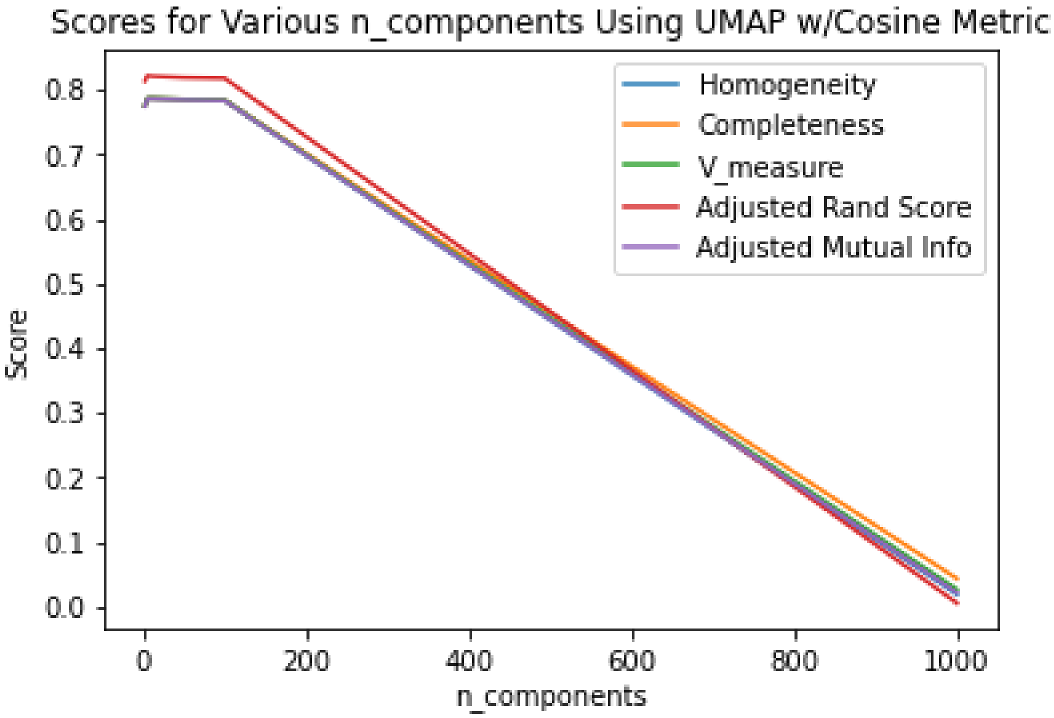
[150]:

```
#Find best number of components and metric for UMAP reduction, using KMeans␣
 ↪scores as evaluation

# Find best N and distance metric for KMeans

n_vals = [2,5,10,50,100,1000]
metric_scores_euc = []
metric_scores_cos = []


for i in n_vals:
    print("working on reducing for n = " + str(i))
    reducer_euc = umap.UMAP(n_components=i, init='random', metric="euclidean",␣
 ↪random_state=0)
    embedding_euc = reducer_euc.fit_transform(bbc_X_tfidf)
    reducer_cos = umap.UMAP(n_components=i, init='random', metric="cosine",␣
 ↪random_state=0)
    embedding_cos = reducer_cos.fit_transform(bbc_X_tfidf)
    col_mean = np.nanmean(embedding_cos, axis=0)
    inds = np.where(np.isnan(embedding_cos))
    embedding_cos[inds] = np.take(col_mean, inds[1])

    print("working on kmeans for n = " + str(i))
    kmeans_clf_euc = KMeans(n_clusters=5, max_iter=1500, random_state=0,␣
 ↪n_init=50)
    kmeans_clf_euc.fit(embedding_euc)
    kmeans_clf_cos = KMeans(n_clusters=5, max_iter=1500, random_state=0,␣
 ↪n_init=50)
    kmeans_clf_cos.fit(embedding_cos)

    metric_scores_euc.append((metrics.homogeneity_score(bbc.target,␣
 ↪kmeans_clf_euc.labels_),
                              metrics.completeness_score(bbc.target,␣
 ↪kmeans_clf_euc.labels_),
                              metrics.v_measure_score(bbc.target, kmeans_clf_euc.
 ↪labels_),
                              metrics.adjusted_rand_score(bbc.target,␣
 ↪kmeans_clf_euc.labels_),
                              metrics.adjusted_mutual_info_score(bbc.target,␣
 ↪kmeans_clf_euc.labels_)))

    metric_scores_cos.append((metrics.homogeneity_score(bbc.target,␣
 ↪kmeans_clf_cos.labels_),
                              metrics.completeness_score(bbc.target,␣
 ↪kmeans_clf_cos.labels_),
                              metrics.v_measure_score(bbc.target, kmeans_clf_cos.
 ↪labels_),
                              metrics.adjusted_rand_score(bbc.target,␣
 ↪kmeans_clf_cos.labels_),
                              metrics.adjusted_mutual_info_score(bbc.target,␣
 ↪kmeans_clf_cos.labels_)))


# Plot Metrics against n_components
plt.plot(n_vals, metric_scores_euc)
plt.legend(("Homogeneity","Completeness","V_measure", "Adjusted Rand Score",␣
 ↪"Adjusted Mutual Info"))
plt.xlabel("n_components")
plt.ylabel("Score")
plt.title("Scores for Various n_components Using UMAP w/Euclidean Metric")
plt.show()

plt.plot(n_vals, metric_scores_cos)
plt.legend(("Homogeneity","Completeness","V_measure", "Adjusted Rand Score",␣
 ↪"Adjusted Mutual Info"))
plt.xlabel("n_components")
plt.ylabel("Score")
plt.title("Scores for Various n_components Using UMAP w/Cosine Metric")
plt.show()

# Find best values and print metrics

avs_Kmeans_cos = []
for i in range(len(metric_scores_cos)):
    a =␣
 ↪((metric_scores_cos[i][0]+metric_scores_cos[i][1]+metric_scores_cos[i][2]+metric_scores_cos[i
 ↪5)
    avs_Kmeans_cos.append(a)

avs_Kmeans_euc = []
for i in range(len(metric_scores_euc)):
    a =␣
 ↪((metric_scores_euc[i][0]+metric_scores_euc[i][1]+metric_scores_euc[i][2]+metric_scores_euc[i
 ↪5)
    avs_Kmeans_euc.append(a)

print('Best Kmeans Cos: ', max(avs_Kmeans_cos))
print('Best Kmeans Euc: ', max(avs_Kmeans_euc))

c = avs_Kmeans_euc.index(max(avs_Kmeans_euc))
b = avs_Kmeans_cos.index(max(avs_Kmeans_cos))

print()
print("Best KMeans EUC Scores")
print("Homogeneity score: ", metric_scores_euc[c][0])
print("Completeness score: ", metric_scores_euc[c][1])
print("V_measure score: ", metric_scores_euc[c][2])
print("Adjusted Rand Index score: ", metric_scores_euc[c][3])
print("Adjusted Mutual Information Score: ", metric_scores_euc[c][4])
print("Best N_Components: ", n_vals[c])

umap_reducer = umap.UMAP(n_components = n_vals[c], init = 'random', metric =␣
 ↪'euclidean', random_state=0)
bbc_umap = umap_reducer.fit_transform(bbc_X_tfidf)

#k-means w/5 clusters
bbc_kmeans_clf = KMeans(n_clusters=5, max_iter=1500, random_state=0, n_init=50)
bbc_kmeans_clf.fit(bbc_umap)

#metrics!
```

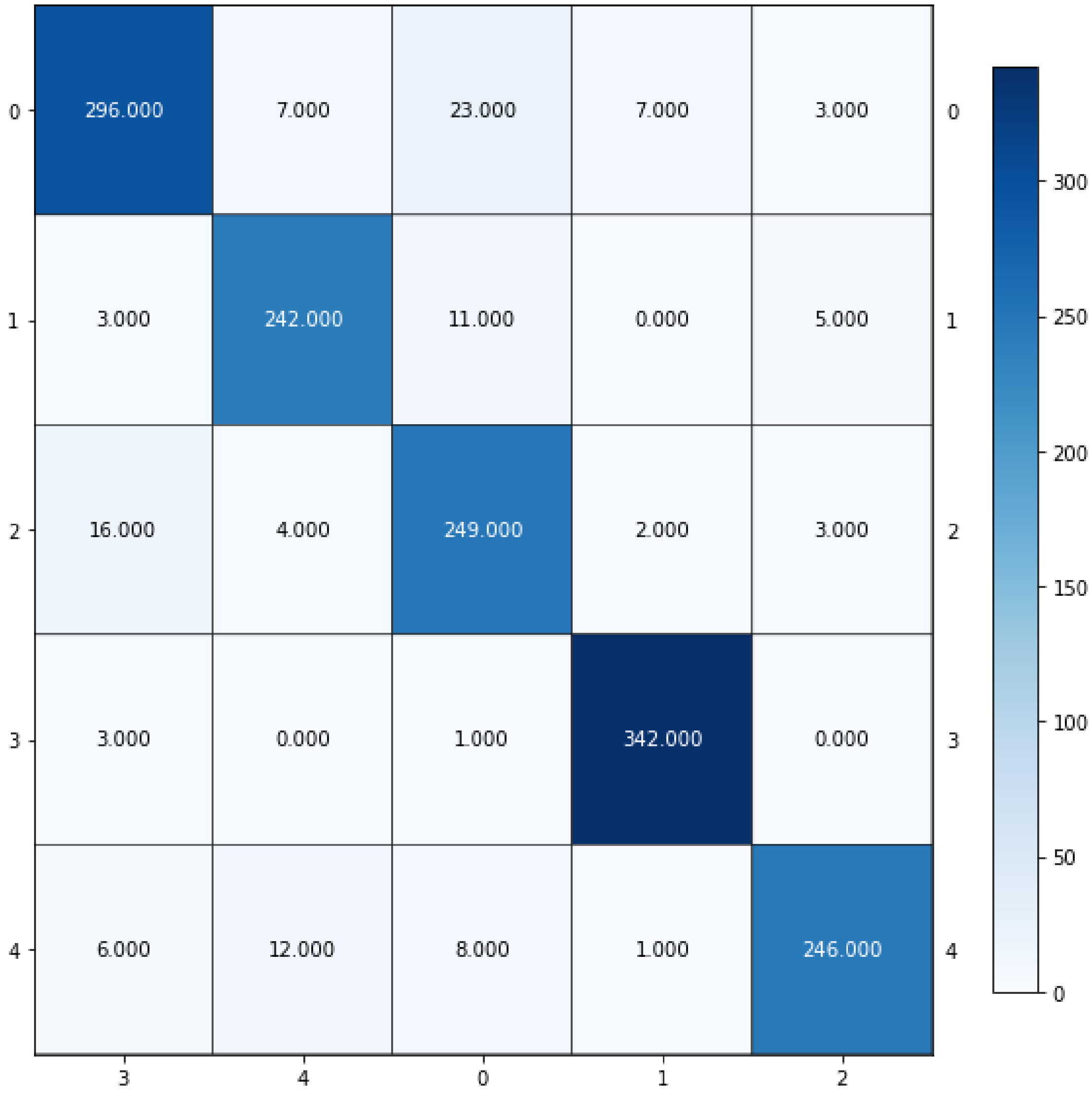
```
from scipy.optimize import linear_sum_assignment
from sklearn.metrics import confusion_matrix

cm = contingency_matrix(bbc.target, bbc_kmeans_clf.labels_)
rows, cols=linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
 ↪size=(8,8))

print()
print("Best KMeans COS Scores")
print("Homogeneity score: ", metric_scores_cos[b][0])
print("Completeness score: ", metric_scores_cos[b][1])
print("V_measure score: ", metric_scores_cos[b][2])
print("Adjusted Rand Index score: ", metric_scores_cos[b][3])
print("Adjusted Mutual Information Score: ", metric_scores_cos[b][4])
print("Best N_Components: ", n_vals[b])

umap_reducer = umap.UMAP(n_components = n_vals[c], init = 'random', metric =␣
 ↪'cosine', random_state=0)
bbc_umap = umap_reducer.fit_transform(bbc_X_tfidf)

#k-means w/5 clusters
bbc_kmeans_clf = KMeans(n_clusters=5, max_iter=1500, random_state=0, n_init=50)
bbc_kmeans_clf.fit(bbc_umap)

#metrics!
from scipy.optimize import linear_sum_assignment
from sklearn.metrics import confusion_matrix

cm = contingency_matrix(bbc.target, bbc_kmeans_clf.labels_)
rows, cols=linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
 ↪size=(8,8))
```

```
working on reducing for n = 2
working on kmeans for n = 2
working on reducing for n = 5
working on kmeans for n = 5
working on reducing for n = 10
working on kmeans for n = 10
working on reducing for n = 50
working on kmeans for n = 50
working on reducing for n = 100
working on kmeans for n = 100
working on reducing for n = 1000
working on kmeans for n = 1000
```


Scores for Various n_components Using UMAP w/Euclidean Metric
Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
n_components
0
200
400
600
800
1000
Homogeneity
Completeness
V_measure
Adjusted Rand Score
Adjusted Mutual Info


Scores for Various n_components Using UMAP w/Cosine Metric
Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
n_components
0
200
400
600
800
1000
Homogeneity
Completeness
V_measure
Adjusted Rand Score
Adjusted Mutual Info

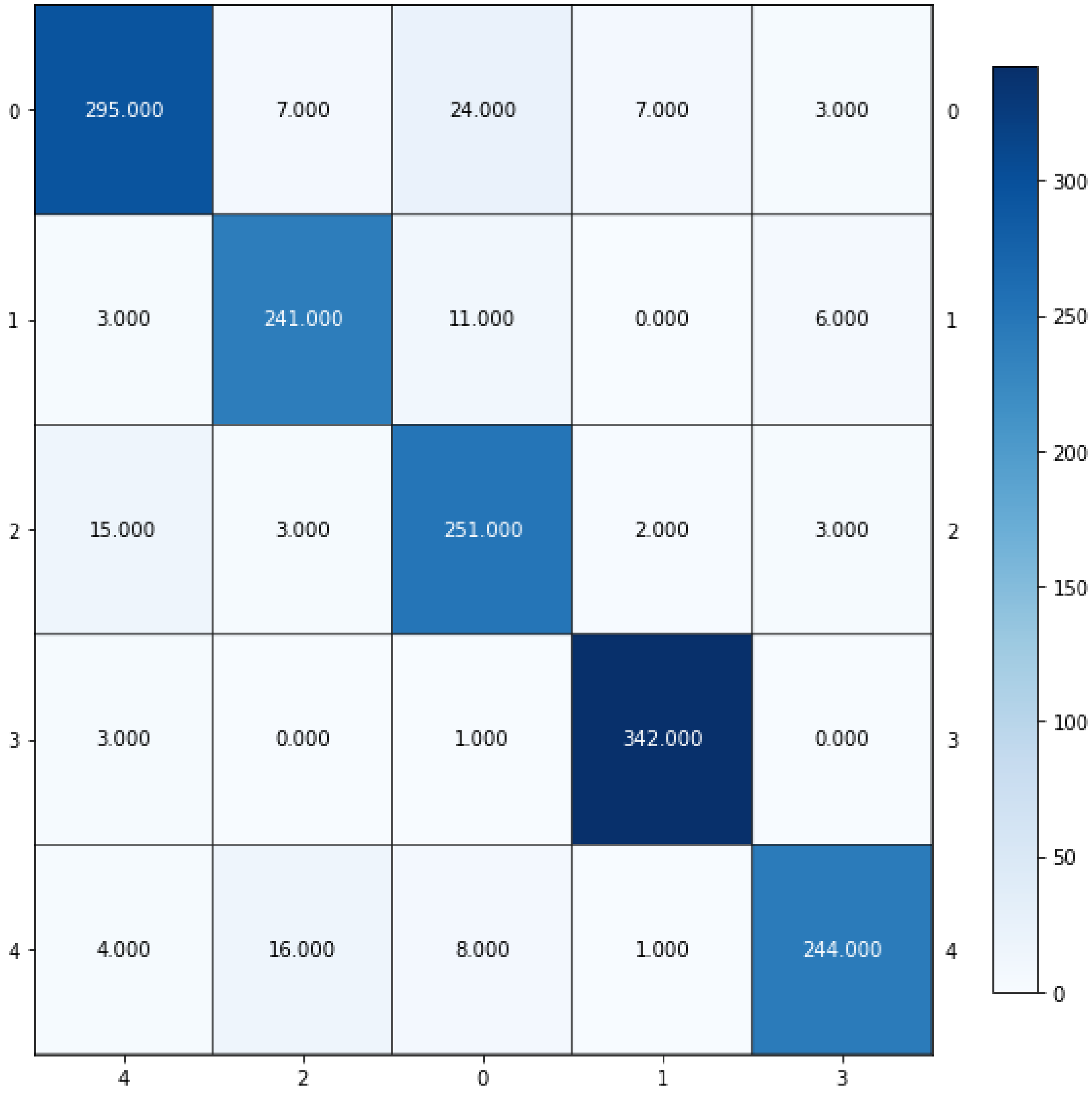
```
Best Kmeans Cos:  0.7930692963971817
Best Kmeans Euc:  0.7944233592691203

Best KMeans EUC Scores
Homogeneity score:  0.7874360944663693
Completeness score:  0.7874929063651126
V_measure score:  0.7874644993910623
Adjusted Rand Index score:  0.8229763601775667
Adjusted Mutual Information Score:  0.786746935945491
Best N_Components:  10
```



	3	4	0	1	2	
0	296.000	7.000	23.000	7.000	3.000	0
1	3.000	242.000	11.000	0.000	5.000	1
2	16.000	4.000	249.000	2.000	3.000	2
3	3.000	0.000	1.000	342.000	0.000	3
4	6.000	12.000	8.000	1.000	246.000	4

```
Best KMeans COS Scores
Homogeneity score:  0.7863911146778318
```

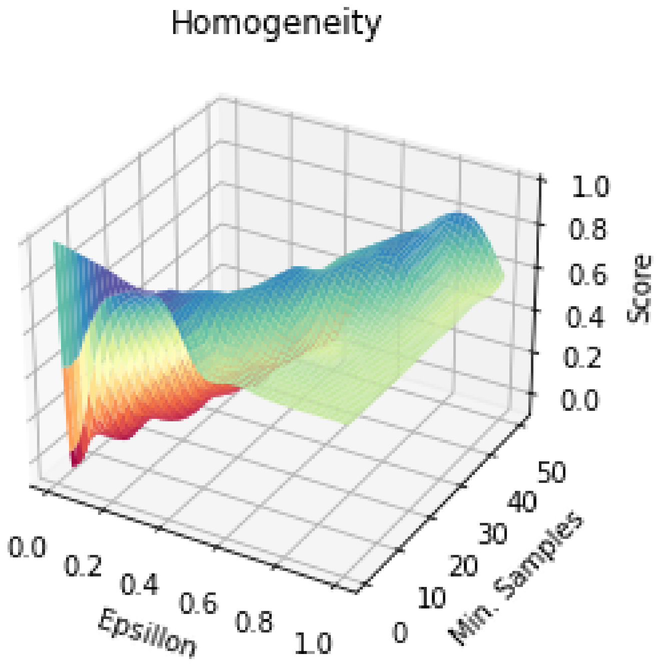
```
Completeness score:  0.7863053396249526
V_measure score:  0.7863482248123015
Adjusted Rand Index score:  0.8206748429226303
Adjusted Mutual Information Score:  0.785626959948192
Best N_Components:  10
```

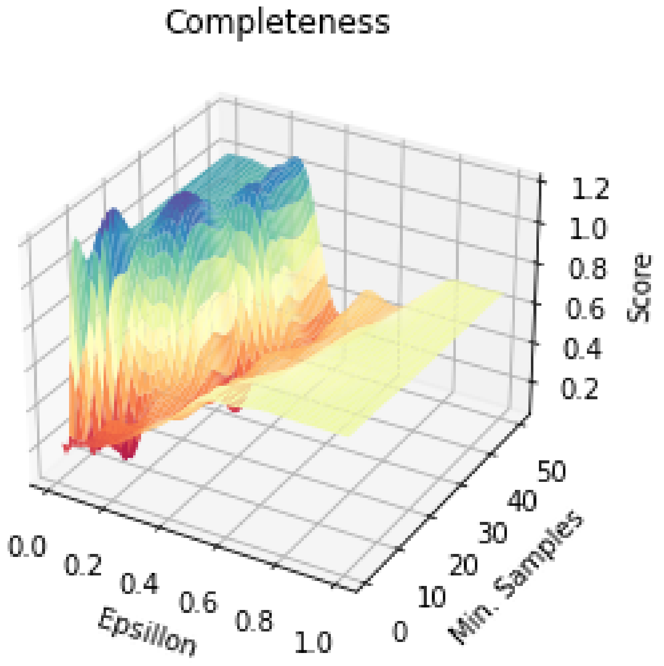


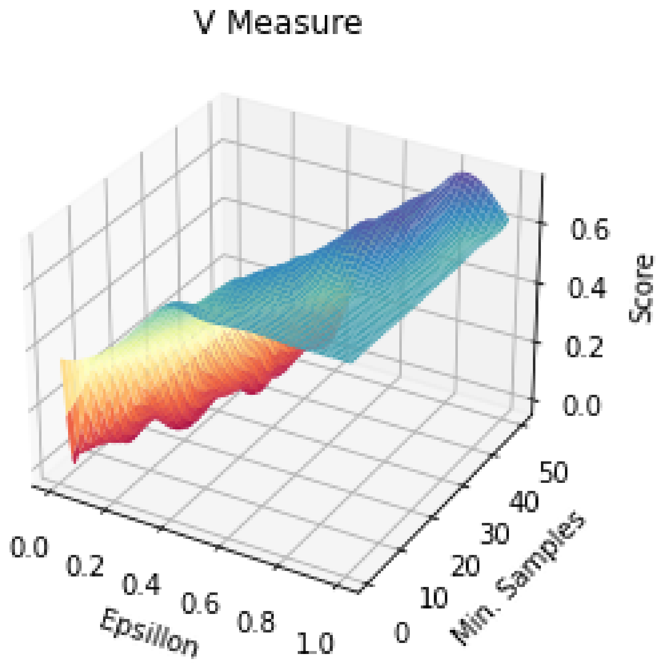
	4	2	0	1	3
0	295.000	7.000	24.000	7.000	3.000
1	3.000	241.000	11.000	0.000	6.000
2	15.000	3.000	251.000	2.000	3.000
3	3.000	0.000	1.000	342.000	0.000
4	4.000	16.000	8.000	1.000	244.000

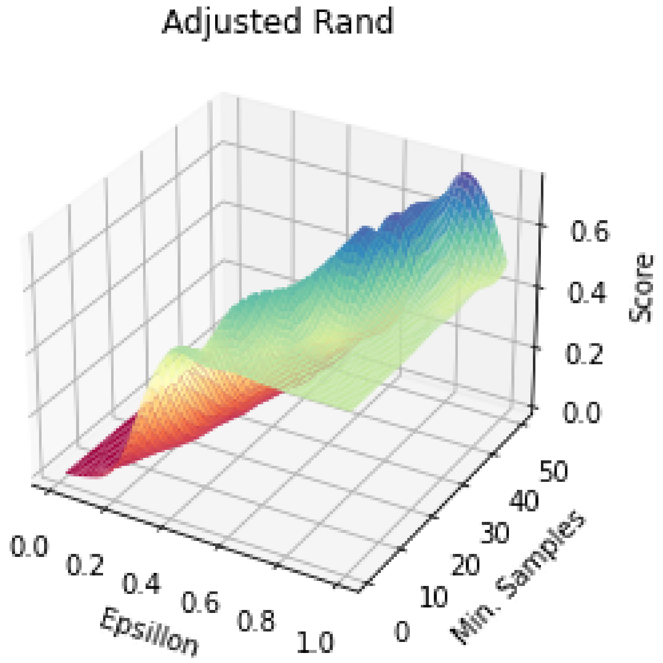
```
[151]: # Find best parameter combo in DBSCAN with COSINE metric
       # Generate 3D surface plots for each metric against epsillon and min samples␣
        ↪values

       umap_reducer = umap.UMAP(n_components = 5, init = 'random', metric = 'cosine',␣
        ↪random_state=0)
       bbc_umap_COS = umap_reducer.fit_transform(bbc_X_tfidf)
```

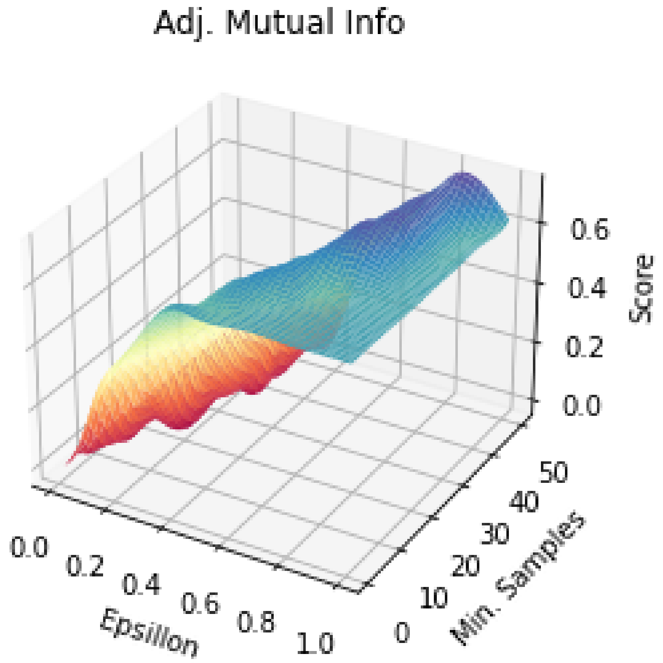
```
eps_vals = [.01,.05,.1,.2,.3,.4,.5,.6,.7,.8,.9,1]
samples = [1,3,5,10,20,30,35,40,50]
metric_scores_DBSCAN_COS = []

for eps in eps_vals:
    for sam in samples:
        DBSCAN_clust = DBSCAN(min_samples=sam, eps=eps)

        DBSCAN_clust.fit(bbc_umap)

        labels = DBSCAN_clust.labels_
        n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

        metric_scores_DBSCAN_COS.append([metrics.homogeneity_score(bbc.target,␣
 ↪DBSCAN_clust.labels_),
                                    metrics.completeness_score(bbc.target,␣
 ↪DBSCAN_clust.labels_),
                                    metrics.v_measure_score(bbc.target,␣
 ↪DBSCAN_clust.labels_),
                                    metrics.adjusted_rand_score(bbc.target,␣
 ↪DBSCAN_clust.labels_),
                                    metrics.adjusted_mutual_info_score(bbc.
 ↪target, DBSCAN_clust.labels_),
                                    n_clusters_,eps,sam])


# Visualize parameter effects on metrics
from mpl_toolkits.mplot3d import Axes3D
from scipy.interpolate import griddata
metricdata = np.array(metric_scores_DBSCAN_COS)
metric_scores = ['Homogeneity', 'Completeness', 'V Measure', 'Adjusted Rand',␣
 ↪'Adj. Mutual Info', 'Number of Clusters']

for i in range(0,6):
    plotdata = metricdata[:, [i,6,7]]
    z, x, y = zip(*plotdata)
    z = list(map(float, z))
    grid_x, grid_y = np.mgrid[min(x):max(x):100j, min(y):max(y):100j]
    grid_z = griddata((x, y), z, (grid_x, grid_y), method='cubic')

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.plot_surface(grid_x, grid_y, grid_z, cmap=plt.cm.Spectral)
    ax.set_xlabel('Epsillon')
    ax.set_ylabel('Min. Samples')
    ax.set_zlabel('Score')
    ax.set_title(str(metric_scores[i]))
    plt.show()

# Find best parameter combination for epsillon and minimum samples

avs = []
for i in range(len(metricdata)):
    a =␣
 ↪((metricdata[i][0]+metricdata[i][1]+metricdata[i][2]+metricdata[i][3]+metricdata[i][4])/
 ↪5)
    avs.append(a)

b = avs.index(max(avs))
print('Best Epsillon = ', metricdata[b][6])
print('Best N_Samples = ', metricdata[b][7])
print()
cm = contingency_matrix(bbc.target, DBSCAN_clust.labels_)
rows, cols=linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
 ↪size=(5,5))
print("Estimated Number of Clusters: ", metricdata[b][5])
print()
print("Best DBSCAN Scores for Cosine Distance Metric")
print("Homogeneity score: ", metricdata[b][0])
print("Completeness score: ", metricdata[b][1])
print("V_measure score: ", metricdata[b][2])
print("Adjusted Rand Index score: ", metricdata[b][3])
print("Adjusted Mutual Information Score: ", metricdata[b][4])
```

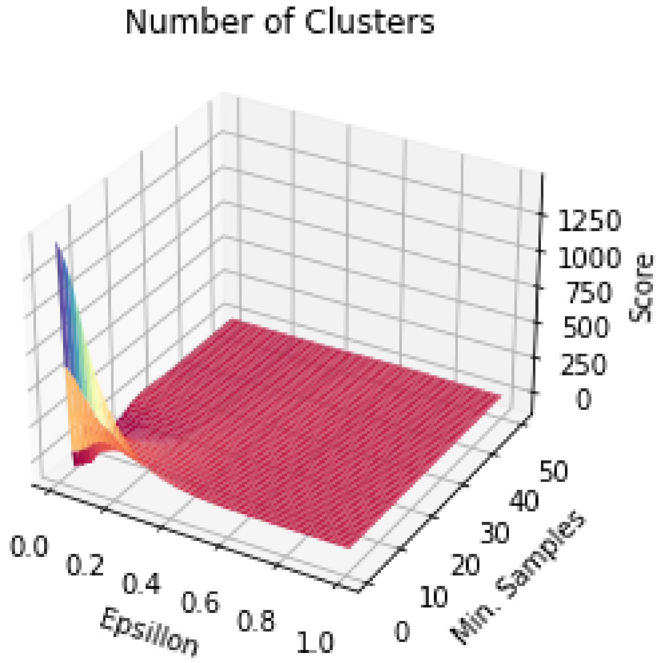

Homogeneity


Completeness

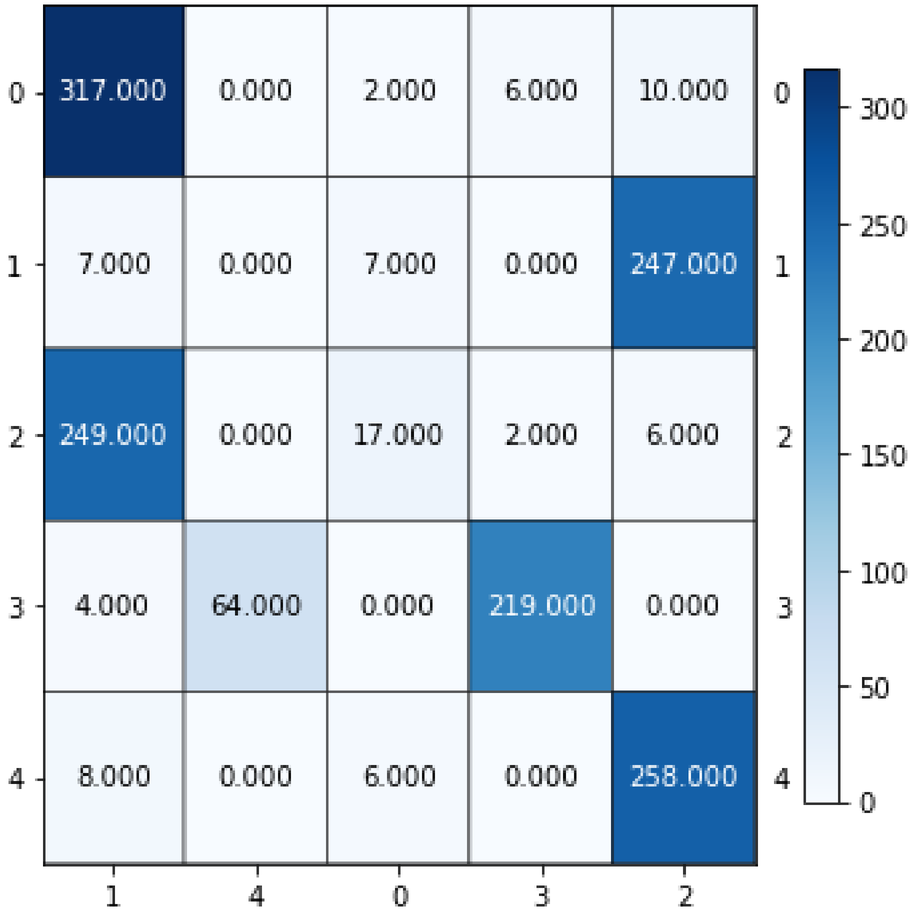

V Measure
Score
0.0
0.2
0.4
0.6
Epsilon
0.0
0.2
0.4
0.6
0.8
1.0
Min. Samples
0
10
20
30
40
50


Adjusted Rand
Score
0.0
0.2
0.4
0.6
Epsilon
0.0
0.2
0.4
0.6
0.8
1.0
Min. Samples
0
10
20
30
40
50





```
Best Epsillon =  0.9
Best N_Samples =  50.0
```

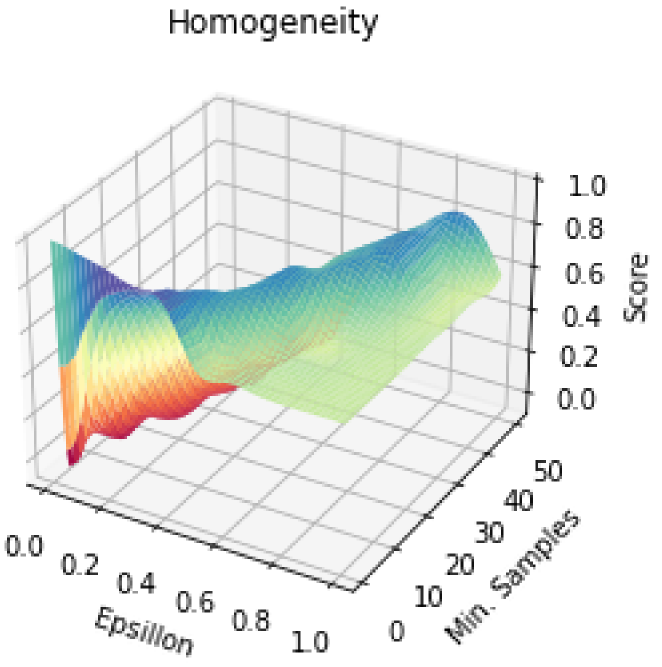


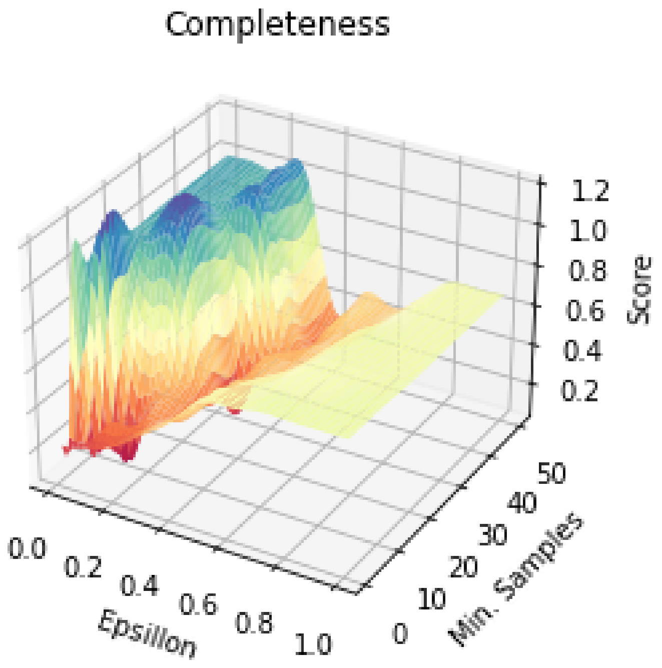
```
Estimated Number of Clusters:  7.0

Best DBSCAN Scores for Cosine Distance Metric
Homogeneity score:  0.7896799513244787
Completeness score:  0.667782538810624
V_measure score:  0.7236336939202158
Adjusted Rand Index score:  0.7108732847743878
Adjusted Mutual Information Score:  0.7221232251101021
```

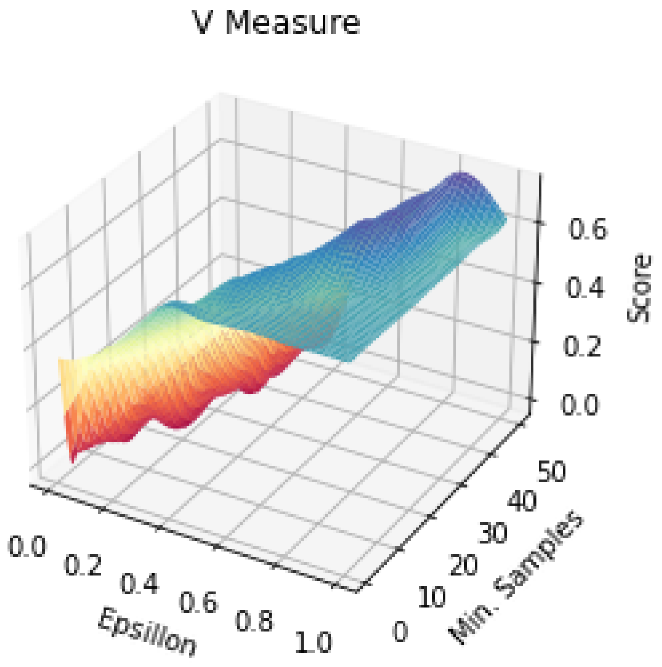
[152]:

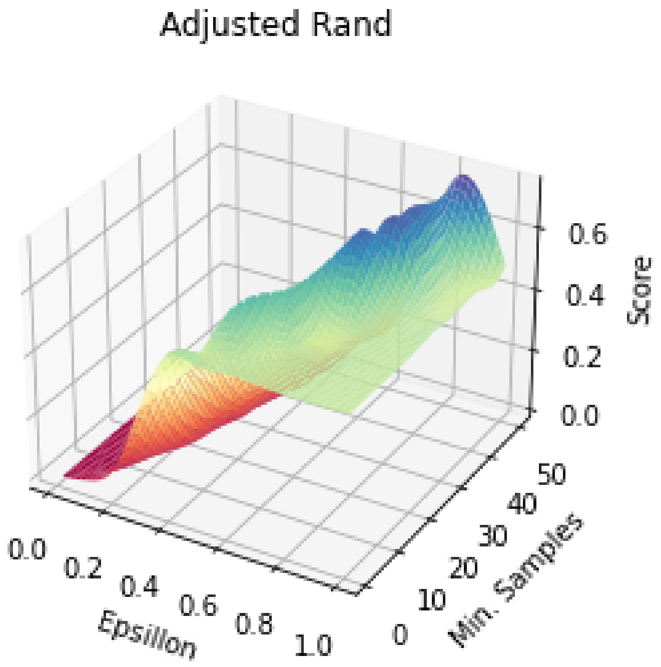
```
# Find best parameter combo in DBSCAN with EUCLIDEAN metric
# Generate 3D surface plots for each metric against epsillon and min samples␣
 ↪values

umap_reducer = umap.UMAP(n_components = 5, init = 'random', metric =␣
 ↪'euclidean', random_state=0)
bbc_umap_EUC = umap_reducer.fit_transform(bbc_X_tfidf)

eps_vals = [.01,.05,.1,.2,.3,.4,.5,.6,.7,.8,.9,1]
```

```
samples = [1,3,5,10,20,30,35,40,50]
metric_scores_DBSCAN_EUC = []

for eps in eps_vals:
    for sam in samples:
        DBSCAN_clust = DBSCAN(min_samples=sam, eps=eps)

        DBSCAN_clust.fit(bbc_umap)

        labels = DBSCAN_clust.labels_
        n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

        metric_scores_DBSCAN_EUC.append([metrics.homogeneity_score(bbc.target,␣
 ↪DBSCAN_clust.labels_),
                                         metrics.completeness_score(bbc.target,␣
 ↪DBSCAN_clust.labels_),
                                         metrics.v_measure_score(bbc.target,␣
 ↪DBSCAN_clust.labels_),
                                         metrics.adjusted_rand_score(bbc.target,␣
 ↪DBSCAN_clust.labels_),
                                         metrics.adjusted_mutual_info_score(bbc.
 ↪target, DBSCAN_clust.labels_),
                                         n_clusters_,eps,sam])


# Visualize parameter effects on metrics
from mpl_toolkits.mplot3d import Axes3D
from scipy.interpolate import griddata
metricdata = np.array(metric_scores_DBSCAN_EUC)
metric_scores = ['Homogeneity', 'Completeness', 'V Measure', 'Adjusted Rand',␣
 ↪'Adj. Mutual Info', 'Number of Clusters']

for i in range(0,6):
    plotdata = metricdata[:, [i,6,7]]
    z, x, y = zip(*plotdata)
    z = list(map(float, z))
    grid_x, grid_y = np.mgrid[min(x):max(x):100j, min(y):max(y):100j]
    grid_z = griddata((x, y), z, (grid_x, grid_y), method='cubic')

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.plot_surface(grid_x, grid_y, grid_z, cmap=plt.cm.Spectral)
    ax.set_xlabel('Epsillon')
    ax.set_ylabel('Min. Samples')
    ax.set_zlabel('Score')
    ax.set_title(str(metric_scores[i]))
    plt.show()

# Find best parameter combination for epsillon and minimum samples

avs = []
for i in range(len(metricdata)):
    a =␣
 ↪((metricdata[i][0]+metricdata[i][1]+metricdata[i][2]+metricdata[i][3]+metricdata[i][4])/
 ↪5)
    avs.append(a)

b = avs.index(max(avs))
print('Best Epsillon = ', metricdata[b][6])
print('Best N_Samples = ', metricdata[b][7])
print()
cm = contingency_matrix(bbc.target, DBSCAN_clust.labels_)
rows, cols=linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows,␣
 ↪size=(5,5))
print("Estimated Number of Clusters: ", metricdata[b][5])
print()
print("Best DBSCAN Scores for Euclidean Distance Metric")
print("Homogeneity score: ", metricdata[b][0])
print("Completeness score: ", metricdata[b][1])
print("V_measure score: ", metricdata[b][2])
print("Adjusted Rand Index score: ", metricdata[b][3])
print("Adjusted Mutual Information Score: ", metricdata[b][4])
```

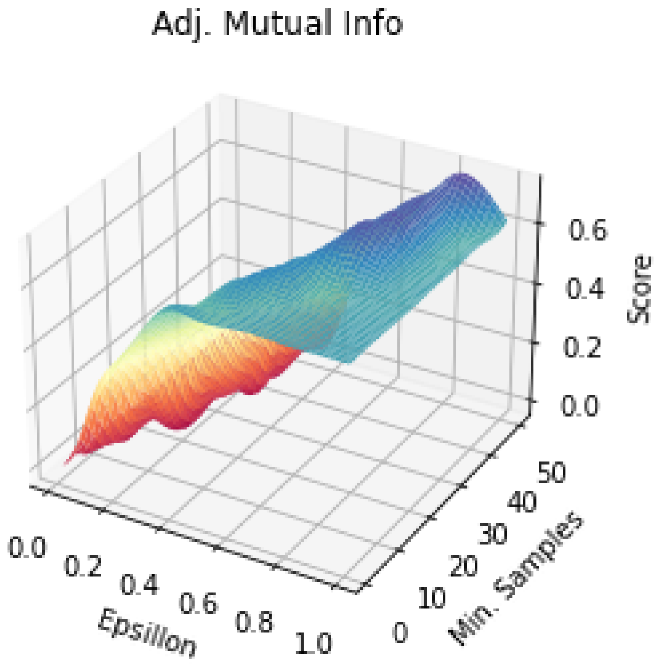



Completeness
Score
1.2
1.0
0.8
0.6
0.4
0.2
Epsilon
0.0
0.2
0.4
0.6
0.8
1.0
Min. Samples
0
10
20
30
40
50

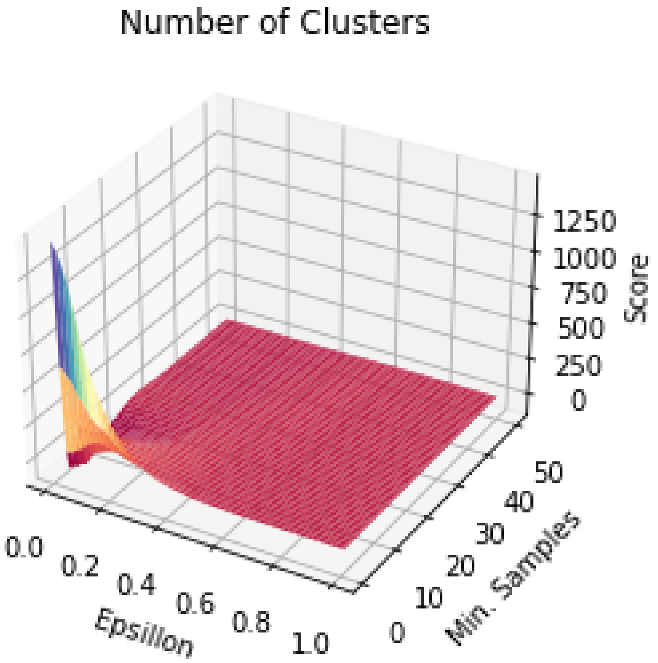

V Measure
Score
0.6
0.4
0.2
0.0
Epsilon
0.0
0.2
0.4
0.6
0.8
1.0
Min. Samples
0
10
20
30
40
50



Adjusted Rand

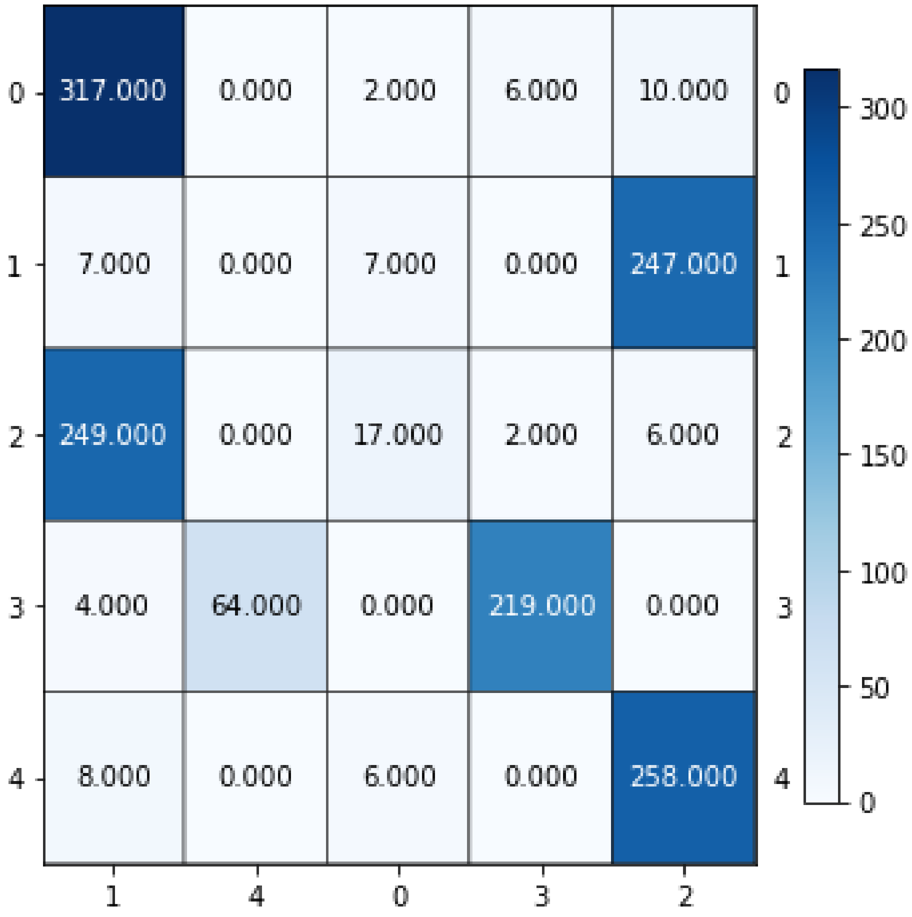


Adj. Mutual Info



Number of Clusters

```
Best Epsillon =  0.9
Best N_Samples =  50.0
```



```
Estimated Number of Clusters:  7.0

Best DBSCAN Scores for Euclidean Distance Metric
Homogeneity score:  0.7896799513244787
Completeness score:  0.667782538810624
V_measure score:  0.7236336939202158
Adjusted Rand Index score:  0.7108732847743878
Adjusted Mutual Information Score:  0.7221232251101021
```

[ ]: