PROJECT 2
REPORT

Clustering

Fabrice Harel-Canada (705221880)
Ganesha Srivallabha Durbha (405291327)
Boyuan He (004791432)

2021.02.02
21W-ECENGR219-1

Q1

Building the TF-IDF matrix.

Following the steps in Project 1, transform the documents into TF-IDF vectors.

Use min_df = 3, exclude the stopwords (no need to do stemming or lemmatization), and remove the headers and footers.

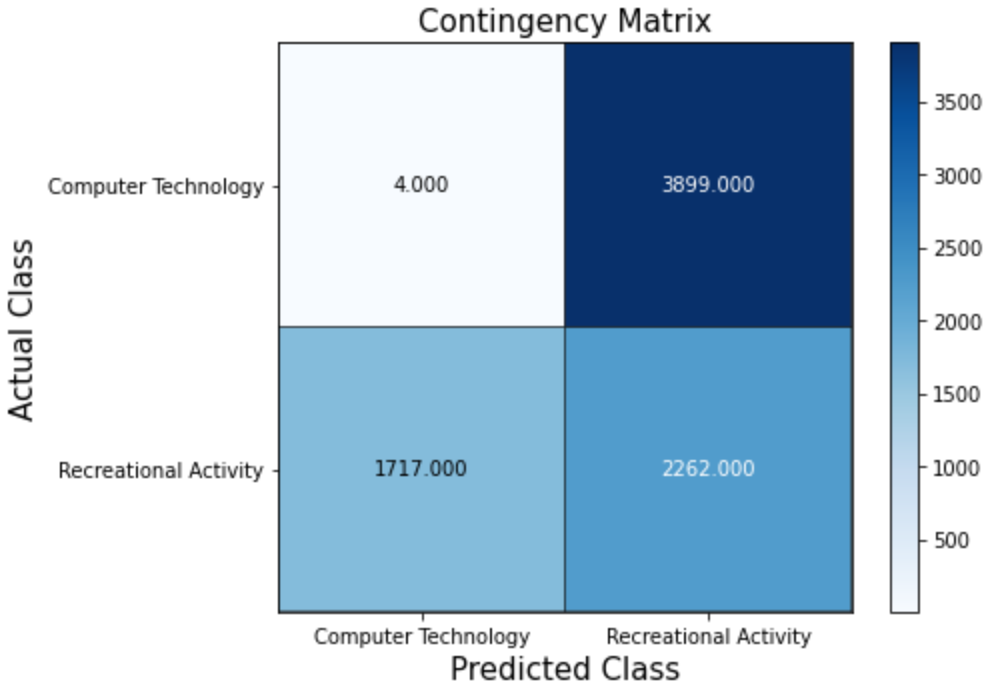
Report the dimensions of the TF-IDF matrix you get.

ANSWER

```
inputs_tfidf.shape: (7882, 27768)
```

Q2

Report the contingency table of your clustering result. You may use the provided plotmat.py to visualize the matrix.



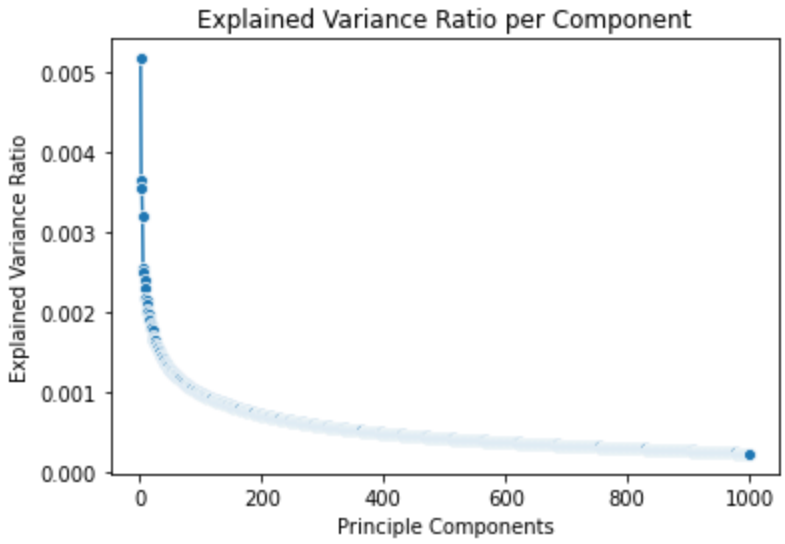
Q3

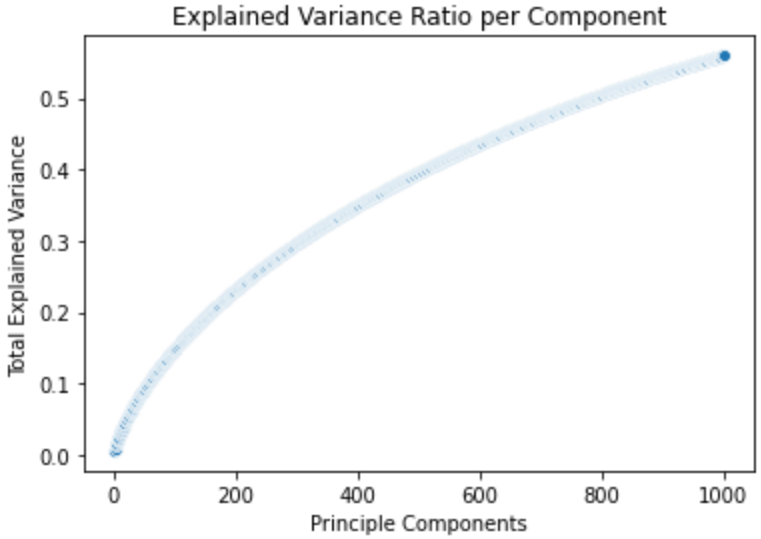
Report the 5 measures above for the K-means clustering results you get.

metric	**score**
homogeneity_score	0.253413
completeness_score	0.334677
v_measure_score	0.288430
adjusted_rand_score	0.180546
adjusted_mutual_info_score	0.288356

Q4

Report the plot of the percent of variance the top r principle components can retain v.s. r, for $r \in \{1 \ldots 1000\}$.





Q5

Let r be the dimension that we want to reduce the data to (i.e. n_components). Try $r \in \{1, 2, 3, 5, 10, 20, 50, 100, 300\}$, and plot the 5 measure scores v.s. r for both SVD and NMF.

Report a good choice of r for SVD and NMF respectively.

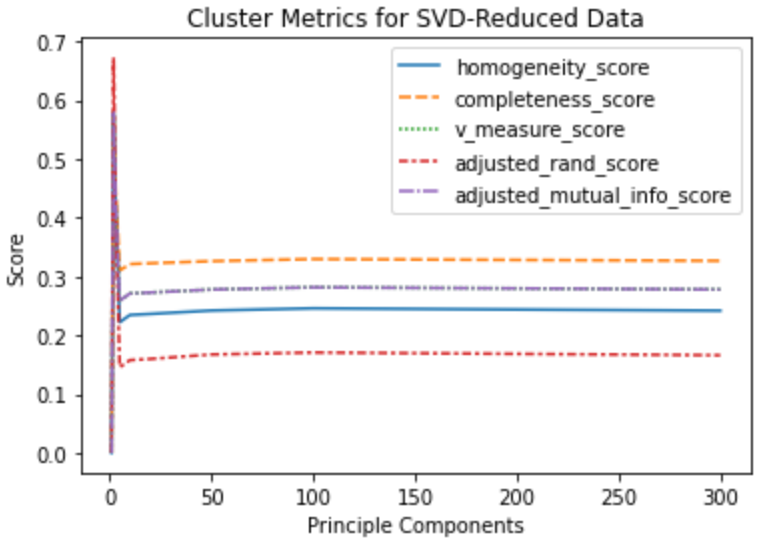
ANSWER

$r = 2$ is the best choice for both SVD and NMF.

See below for plots and figures.

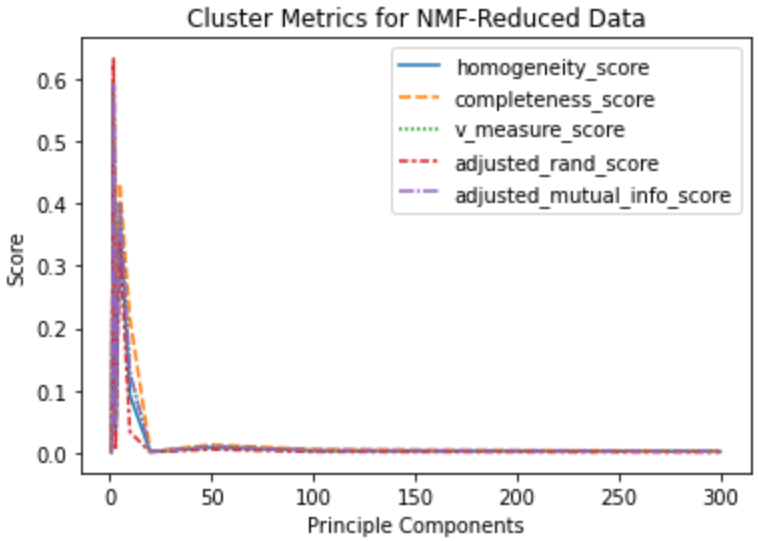
SVD-Reduced Data

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.000271	0.000275	0.000273	0.000296	0.000181
2	0.577167	0.579872	0.578517	0.672517	0.578478
3	0.413470	0.448835	0.430427	0.414009	0.430373
5	0.222218	0.310361	0.258995	0.145737	0.258916
10	0.234625	0.321219	0.271177	0.157799	0.271100
20	0.236404	0.322558	0.272841	0.159822	0.272765
50	0.242165	0.326171	0.277960	0.167420	0.277884
100	0.245914	0.329721	0.281717	0.170760	0.281641
300	0.241949	0.326734	0.278022	0.166176	0.277946



NMF-Reduced Data

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.000271	0.000275	0.000273	0.000296	0.000181
2	0.582552	0.599296	0.590806	0.634364	0.590768
3	0.027821	0.155780	0.047211	0.003928	0.047063
5	0.375401	0.433068	0.402178	0.341705	0.402119
10	0.092525	0.215844	0.129527	0.033742	0.129415
20	0.001860	0.002471	0.002122	0.001201	0.002018
50	0.008735	0.013233	0.010523	0.005683	0.010414
100	0.003617	0.005938	0.004496	0.001795	0.004382
300	0.002161	0.003253	0.002597	0.001114	0.002487



Q6

How do you explain the non-monotonic behavior of the measures as r increases?

ANSWER

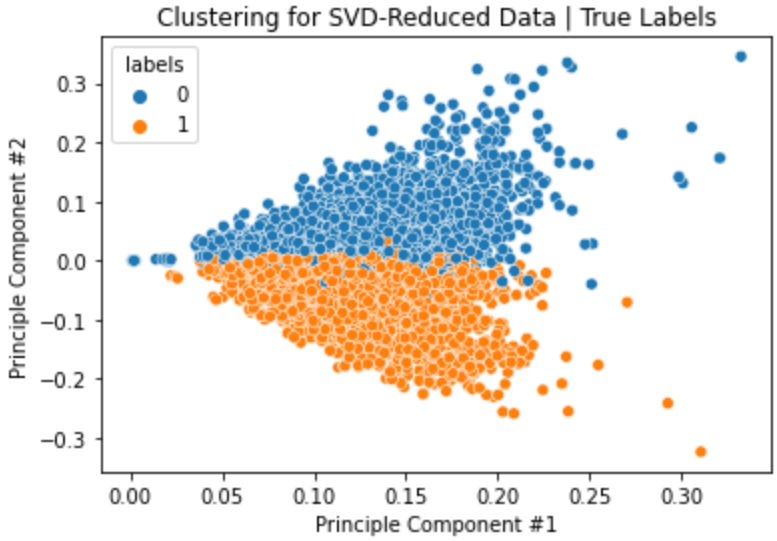
At $r = 1$, the distances are hard to differentiate from one another as there is only 1 dimension to "travel." Allowing for $r = 2$ provides for a larger space for the points to be projected upon and therefore easier to distinguish via K-Means. However, as r continues to increase and therefore the dimensionality increases, the default Euclidean distance measure becomes less and less capable of providing an intuitive sense of separation. The aptly named "Curse of Dimensionality" strikes and unless we use a different measure of distance, we are better off sticking with $r = 2$.

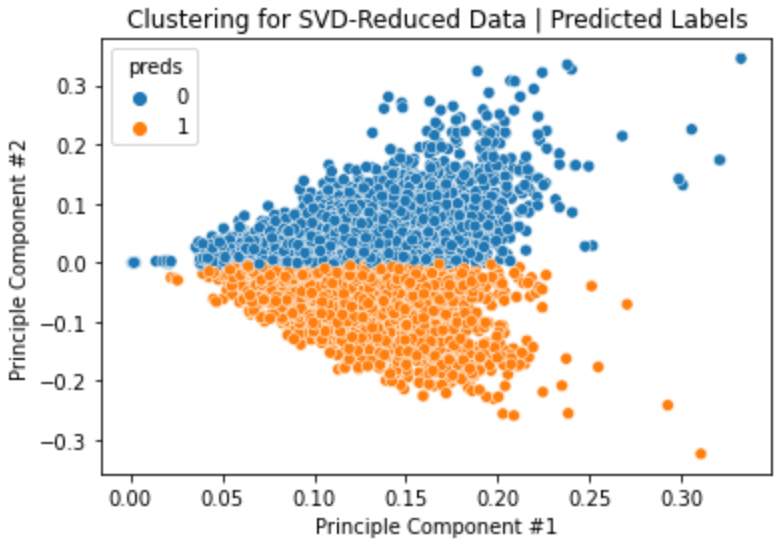
Q7

Visualize the clustering results for:

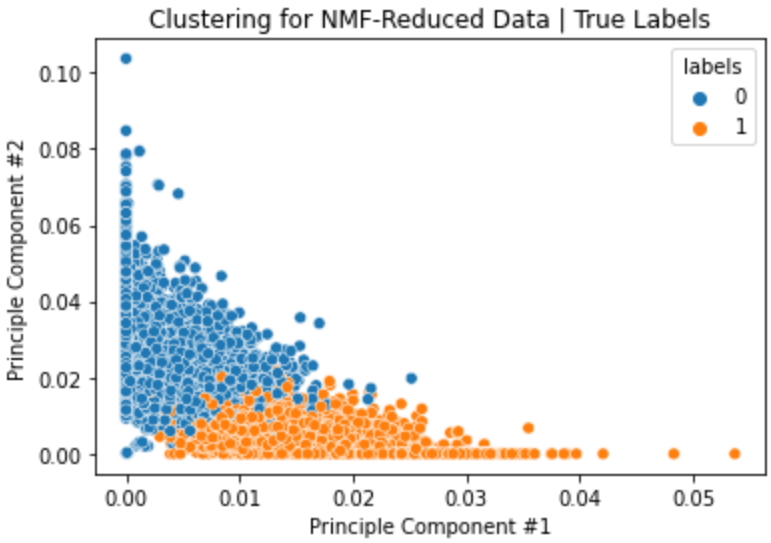
- SVD with your choice of r
- NMF with your choice of r

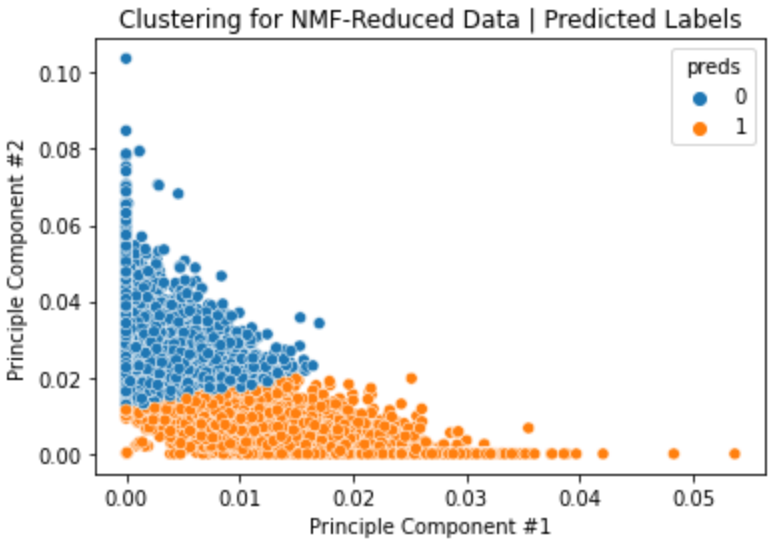
SVD-Reduced Data





NMF-Reduced Data





Q8

What do you observe in the visualization? How are the data points of the two classes distributed? Is the data distribution ideal for K-Means clustering?

ANSWER

KMeans makes 3 primary assumptions:

1. The feature distributions are spherical / round
2. The features have similar variance
3. There are approximately the same number of examples from each class.

If any of these assumptions are violated, KMeans will underperform. Unfortunately, the plots show that the feature distributions are not spherical. There are also no separable or distinct clusters for KMeans to assign. Much of the data overlaps in the reduced dimension. Ultimately, the data distribution is not ideal for KMeans clustering.

Q9

Multi-class classification using K-means with 20 clusters

Construct the TF-IDF matrix,including all 20 categories. Reduce its dimensionality properly using either NMF or SVD, and perform K-Means clustering with n_components=20 . Visualize the contingency matrix and report the five clustering metrics.

ANSWER

Documents from all 20 categories were loaded. To construct the TF-IDF matrix, excluded terms that had a document frequency less than 3 or were stopwords. The headers and footers were also removed. The resulting shape of the TF-IDF matrix was:

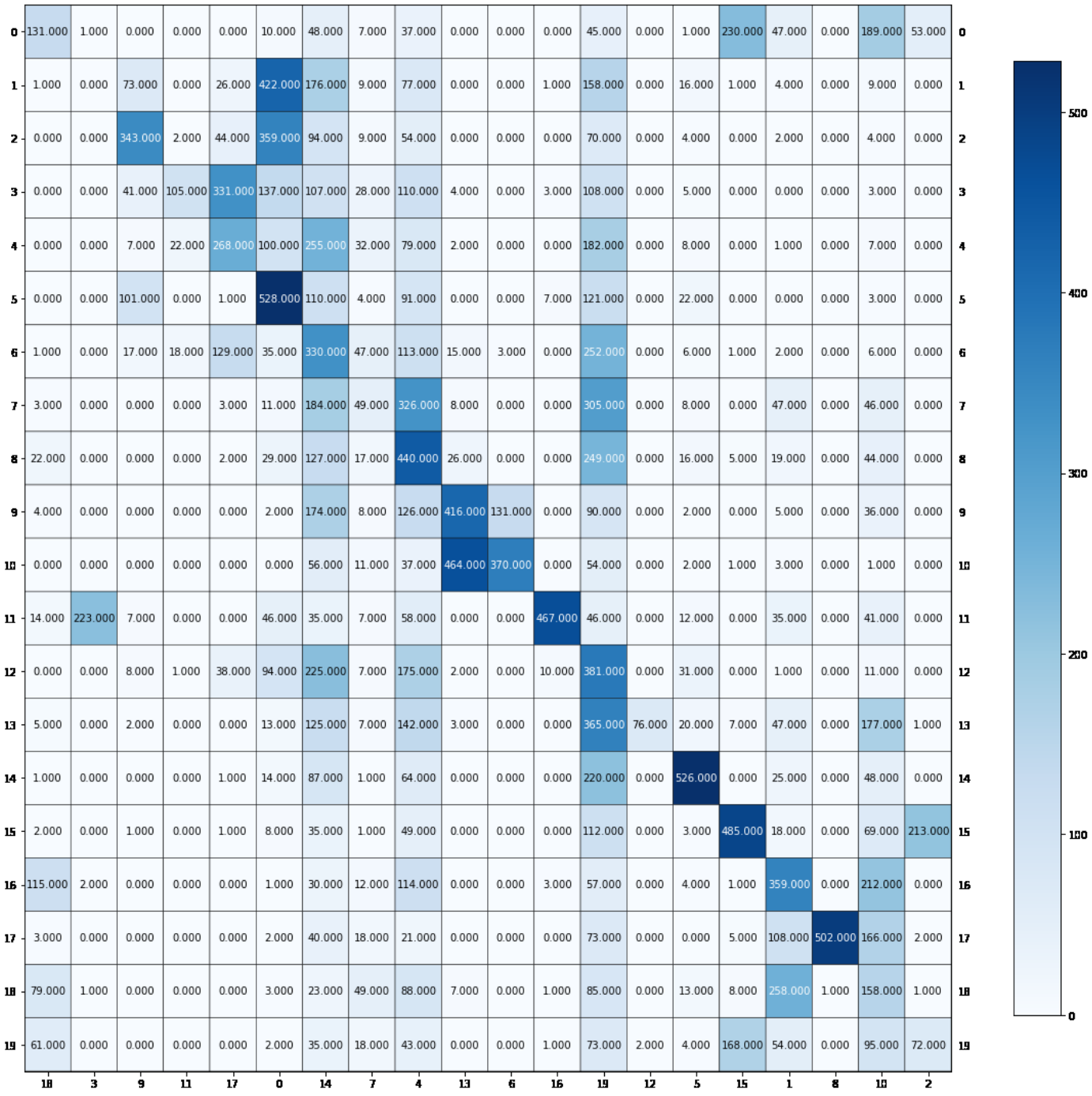
```
inputs_tfidf.shape: (18846, 52295)
```

The SVD method was used for feature reduction. Setting n_components = [1, 2, 3, 5, 10, 20, 50, 100, 300, 500, 800], the multi class classification's metrics were:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.027941	0.031138	0.029453	0.005799	0.026110
2	0.211032	0.224915	0.217753	0.064692	0.215137
3	0.232702	0.242342	0.237424	0.079573	0.234909
5	0.313221	0.332460	0.322554	0.123212	0.320297
10	0.339986	0.379278	0.358559	0.139547	0.356361
20	0.287669	0.383175	0.328624	0.092857	0.326112
50	0.310314	0.413422	0.354523	0.093870	0.352104
100	0.275215	0.381518	0.319763	0.083114	0.317164
300	0.299456	0.402700	0.343488	0.098837	0.340994
500	0.284961	0.451065	0.349270	0.087328	0.346626
800	0.297703	0.407515	0.344060	0.112155	0.341544

K-Means with 20 clusters, using SVD for dimension reduction. First column shows n_components.

We notice that n_components = 10 gives us the best overall clustering metrics, with the highest homogeneity, V-score, adjusted rand score, and adjusted mutual information score. The contingency matrix when the data was reduced to `n_components = 10` is:



Contingency table (SVD n_components = 10, Kmeans)

Q 10

Try Kullback-Leibler divergence for NMF and see whether it helps with the clustering of our text data. Report the five evaluation metrics.

ANSWER

On using Kullback-Leibler divergence as the distance measure in NMF reduction of feature dimension, the K-means clustering scores with increasing n_components are as follow:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.030556	0.034513	0.032414	0.006957	0.029031
2	0.198715	0.225536	0.211277	0.063138	0.208523
3	0.256536	0.278224	0.266940	0.093704	0.264456
5	0.350546	0.374438	0.362098	0.155914	0.359949
10	0.405617	0.414149	0.409838	0.210817	0.407913
20	0.382709	0.392057	0.387327	0.252586	0.385325
50	0.239701	0.342298	0.281956	0.046972	0.279203
100	0.199862	0.389404	0.264149	0.028596	0.260931
300	0.042846	0.231684	0.072318	0.002391	0.066781

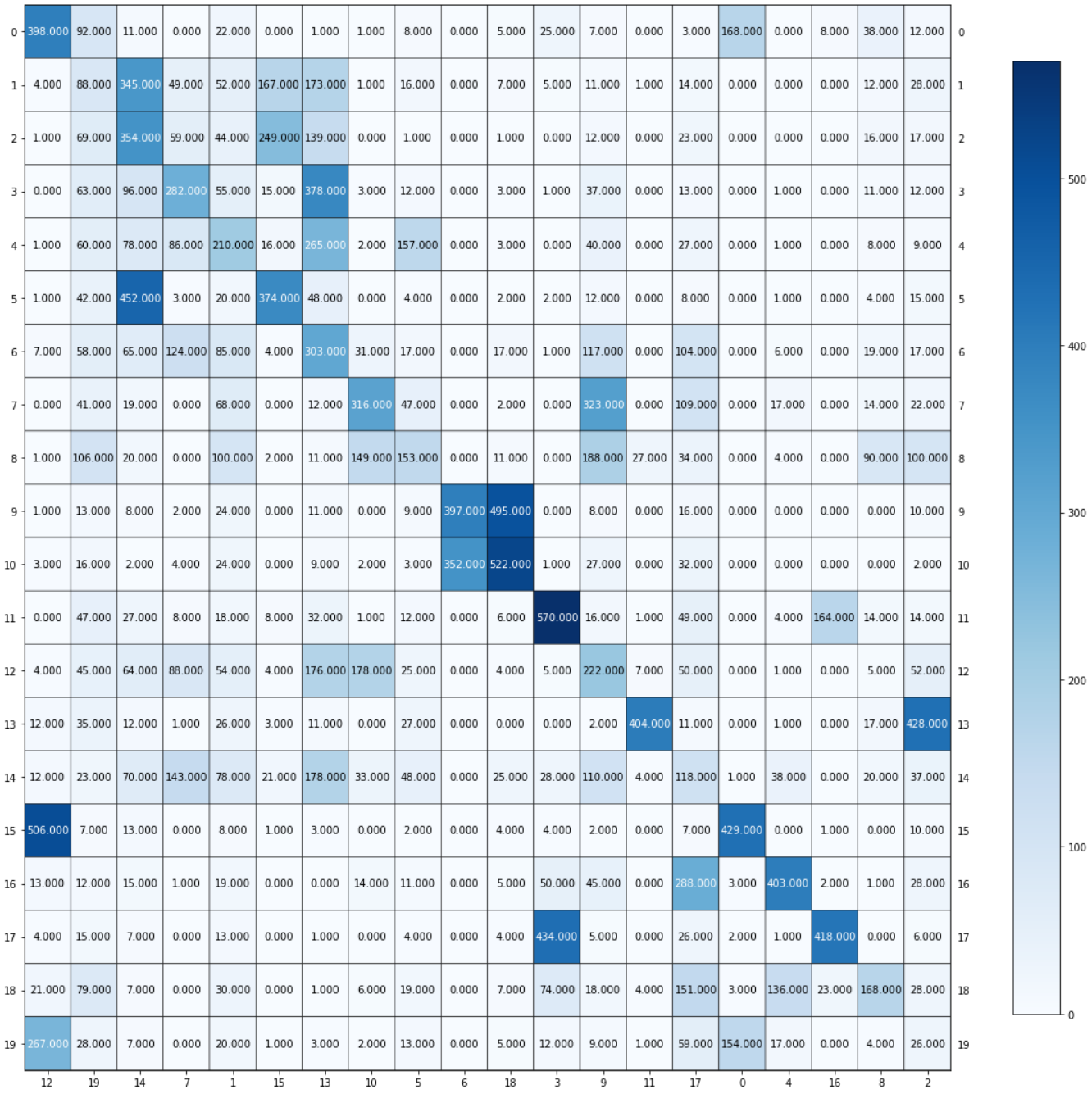
K-Means clustering scores on NMF reduction using KL- divergence as distance metric

The best performance was again achieved at `n_components = 10` and the five evaluation metrics are:

	metric	score
0	homogeneity_score	0.405617
1	completeness_score	0.414149
2	v_measure_score	0.409838
3	adjusted_rand_score	0.210817
4	adjusted_mutual_info_score	0.407913

Best K-means metrics with KL-divergence (at n_components = 10)

The contingency matrix in this case was:



Contingency matrix for the best K-means clustering using KL divergence metric in NMF method

Q 11

Use UMAP to reduce the dimensionality of the 20 categories TF-IDF matrix, and apply K-Means clustering with 20 clusters.

ANSWER

Using `'euclidean'` metric:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.313804	0.328794	0.321124	0.205228	0.318863
2	0.468820	0.486809	0.477645	0.369443	0.475912
3	0.489580	0.515532	0.502221	0.392948	0.500556
5	0.499953	0.532349	0.515643	0.392992	0.514013
10	0.497766	0.537320	0.516788	0.396259	0.515141
20	0.502224	0.543511	0.522052	0.402544	0.520413
50	0.508013	0.546018	0.526330	0.409757	0.524718
100	0.514402	0.542460	0.528059	0.419452	0.526467
300	0.515771	0.535938	0.525661	0.417134	0.524086

K-Means after reducing data by UMAP using `'euclidean'` metric

Using `'cosine'` metric:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.326141	0.337057	0.331509	0.220160	0.329310
2	0.475285	0.491008	0.483019	0.375309	0.481307
3	0.485451	0.509786	0.497321	0.393881	0.495632
5	0.501004	0.532950	0.516484	0.399789	0.514858
10	0.502372	0.530903	0.516244	0.407605	0.514623
20	0.510078	0.531100	0.520377	0.414975	0.518783
50	0.500182	0.541944	0.520227	0.394553	0.518588
100	0.498076	0.534279	0.515543	0.387900	0.513894
300	0.497215	0.537664	0.516649	0.394478	0.515015

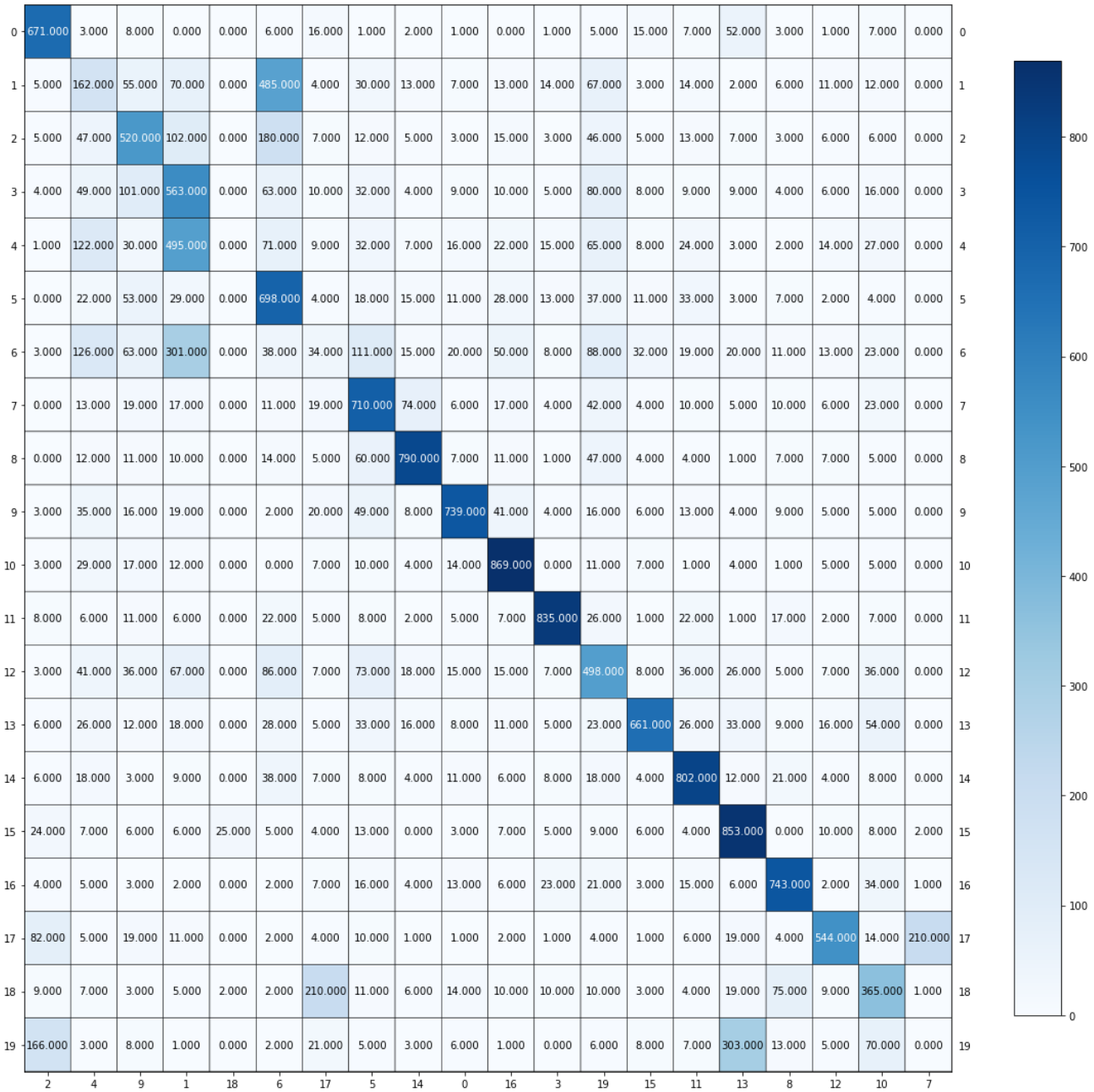
K-Means after reducing data by UMAP using `'cosine'` metric

Q 12

From the contingency matrices, identify the categories that are prone to be confused.

ANSWER

In case of ‘euclidean’ metric, the best `n_components = 5` (scores from the previous page)



Contingency matrix: K- means (20 clusters) UMAP reduction to `n_components=5` (`‘euclidean’`)

Analysis:

Recall the 20 categories being classified:

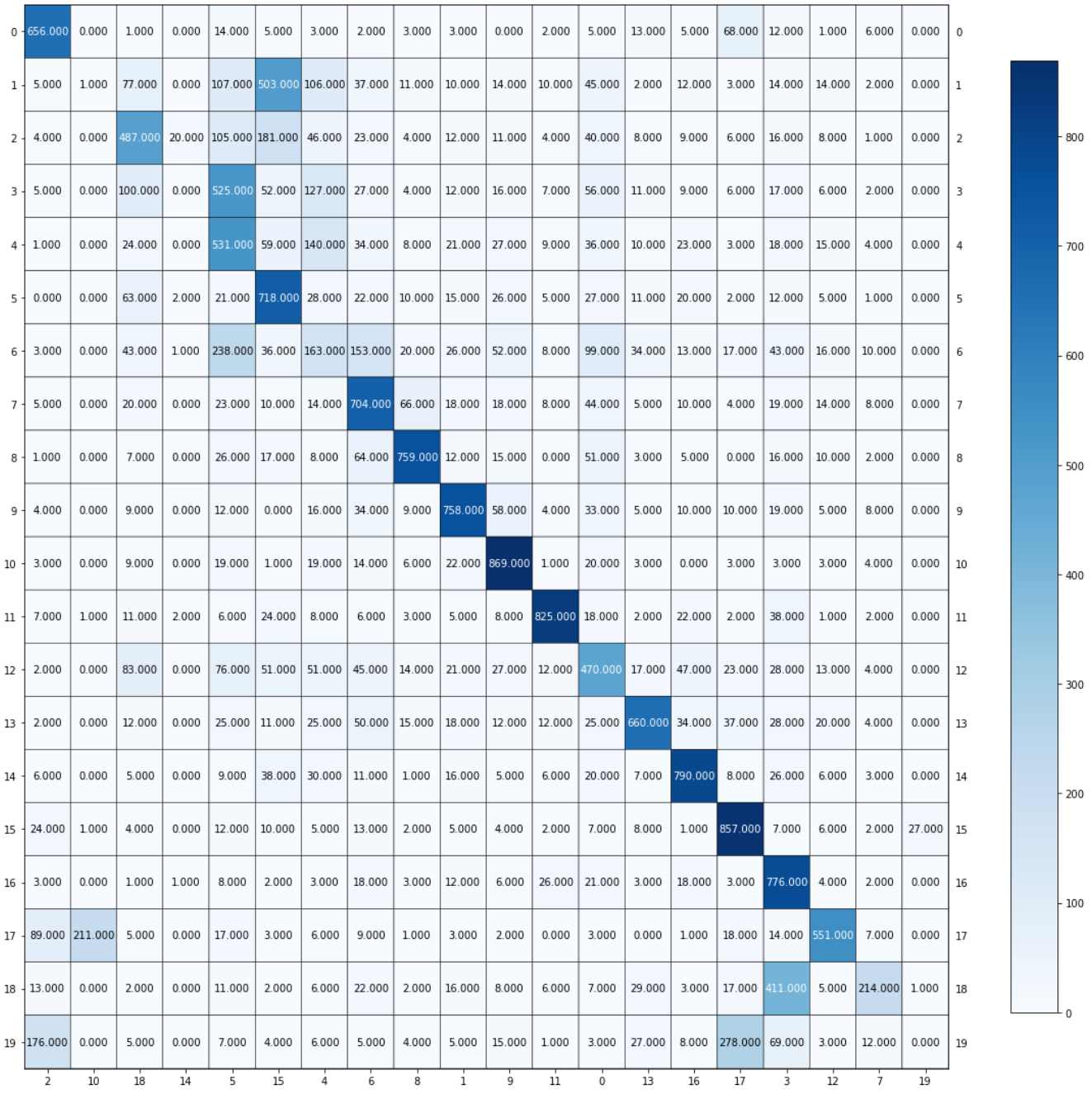
```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x',
'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball',
'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med',
'sci.space', 'soc.religion.christian', 'talk.politics.guns',
'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']
```

Since certain categories are close to each other like religion, atheism, and christianity; or IBM and mac hardware, the model is prone to get confused in clustering those categories that are close to each other. However, it must distinguish the other topics well enough. This is observed in the above matrix as:

1. **Christianity and talks about miscellaneous religions clustered together:** The cluster (column) labelled 13 has most of its documents from categories (rows) 15 and 19. This is an encouraging sign because categories 15 and 19 pertain to christianity and talks about miscellaneous religions. The topics being highly similar, were clustered together by the model.
2. **IBM PC hardware and Mac hardware clustered together:** The cluster (column) labelled 1 has most of its documents from categories (rows) 3 and 4. This is an encouraging sign because categories 3 and 4 pertain to IBM harware and Mac hardware. The topics being highly similar, were clustered together by the model.
3. **Atheism and religion clustered together:** The cluster (column) labelled 2 has most of its documents from categories (rows) 1 and 19. This is an encouraging sign because categories 1 and 19 pertain to atheism and miscellanous religions. The topics being related, were clustered together by the model.

Similar behaviour although with different labelling is observed on using the `'cosine'` metric, as seen in the contingency matrix below:

In the case of the `'cosine'` metric, best **`n_components = 5.`** (`n_components = 3` also gave similar results)



Contingency matrix: K- means (20 clusters) UMAP reduction to **`n_components=5`** (`'cosine'`)

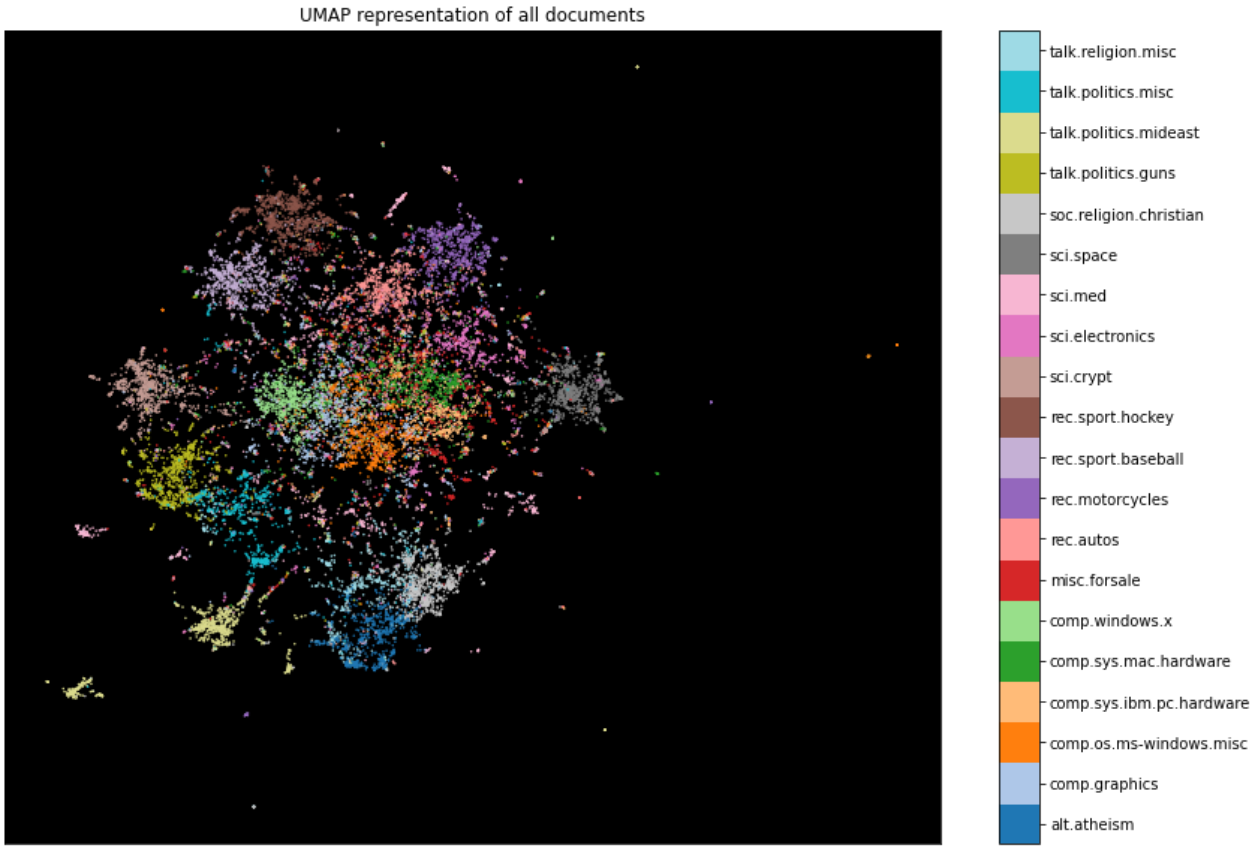
Q 13

Use UMAP to reduce the dimensionality properly, and perform Agglomerative clustering with n_clusters=20 . Compare the performance of “ward” and “single” linkage criteria. Report the five clustering evaluation metrics for each case.

ANSWER

Agglomerative clustering is a bottom-up hierarchical clustering technique where at each step, the pair of clusters that are closest to each other according to some distance metric are merged into one cluster. This process is iterated so that each iteration lowers the total number of clusters by 1. The point at which we stop the clustering is determined by the number of clusters we decide to group the data into.

To see the spread of data, a UMAP representation of all documents was plotted, colored according to the category they belong to.



UMAP representation of documents on 2D space

USING ‘WARD’ LINKAGE

Ward's method aims to minimize the total within-cluster variance. However, it need not give the optimal lowest within-cluster variance since it is constrained by the choices made in forming the previous clusters.

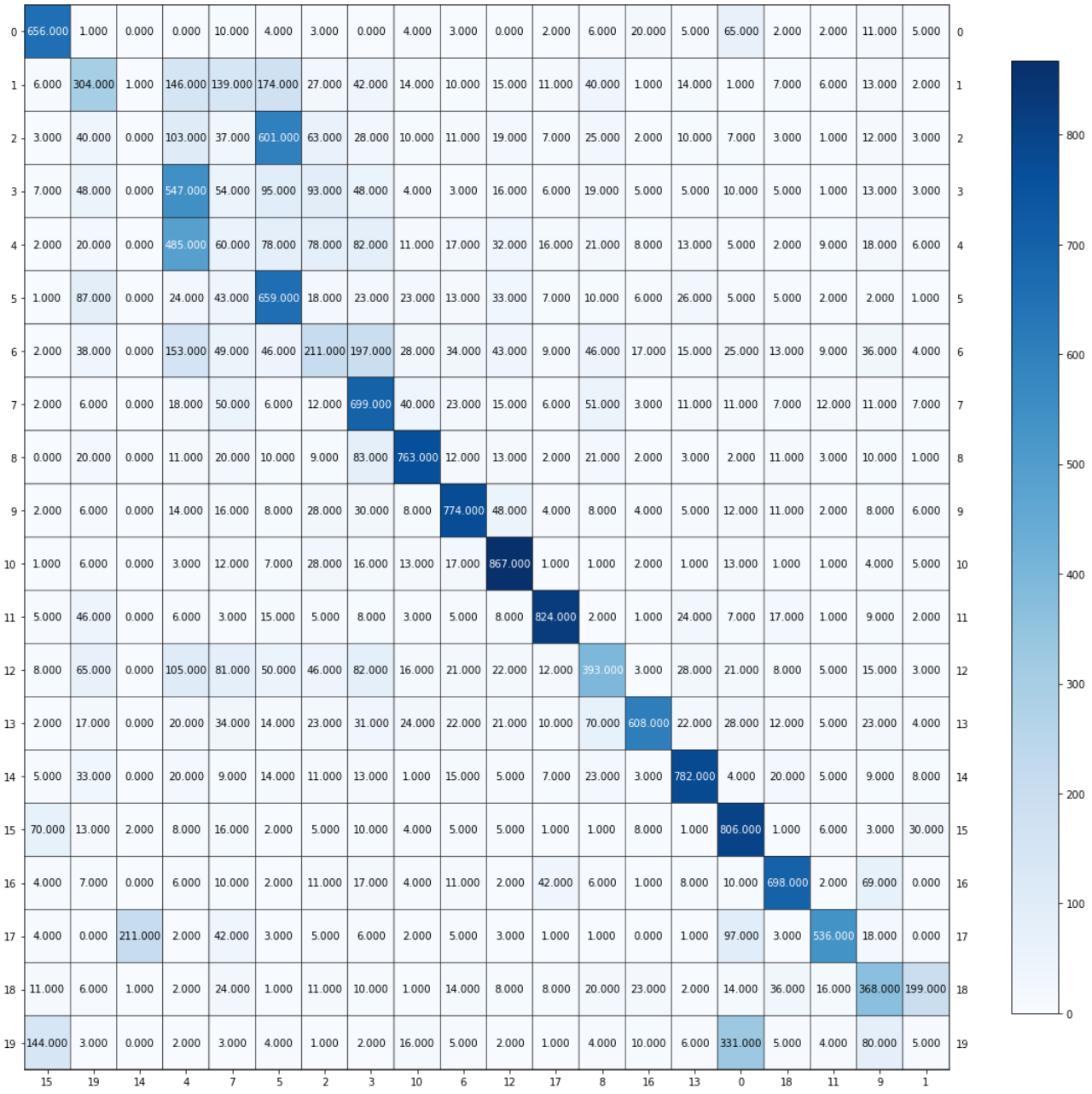
At any stage, those two clusters shall be merged that lead to the smallest increase in within-cluster variance after the merger. This can equivalently be described by saying that the two clusters merged have the lowest weighted centroid distance between all pairs of clusters. The metric is given by:

$\frac{n_A n_B}{n_A+n_B} \quad ||m_A - m_B||^2$ where m_j is the centroid of cluster j and n_j is the number of po

The ward method works well for the 'cloud' type of clusters which are dense about a centroid and relatively sparse as we move away from it. Since our UMAP representation of data shows similar behaviour, using the 'ward' criterion serves our clustering well, as seen in the scores below:

	metric	score
0	homogeneity_score	0.495834
1	completeness_score	0.510624
2	v_measure_score	0.503120
3	adjusted_rand_score	0.391852
4	adjusted_mutual_info_score	0.501491

Agglomerative clustering performed using `'ward'` linkage



Contingency matrix - Agglomerative clustering performed using `'ward'` linkage

USING SINGLE - LINKAGE CRITERIA

The single-linkage clustering criterion determines the distance between two clusters as the smallest distance between any pair of points, one coming from each of the clusters. Thus, the distance between two clusters A and B is given by:

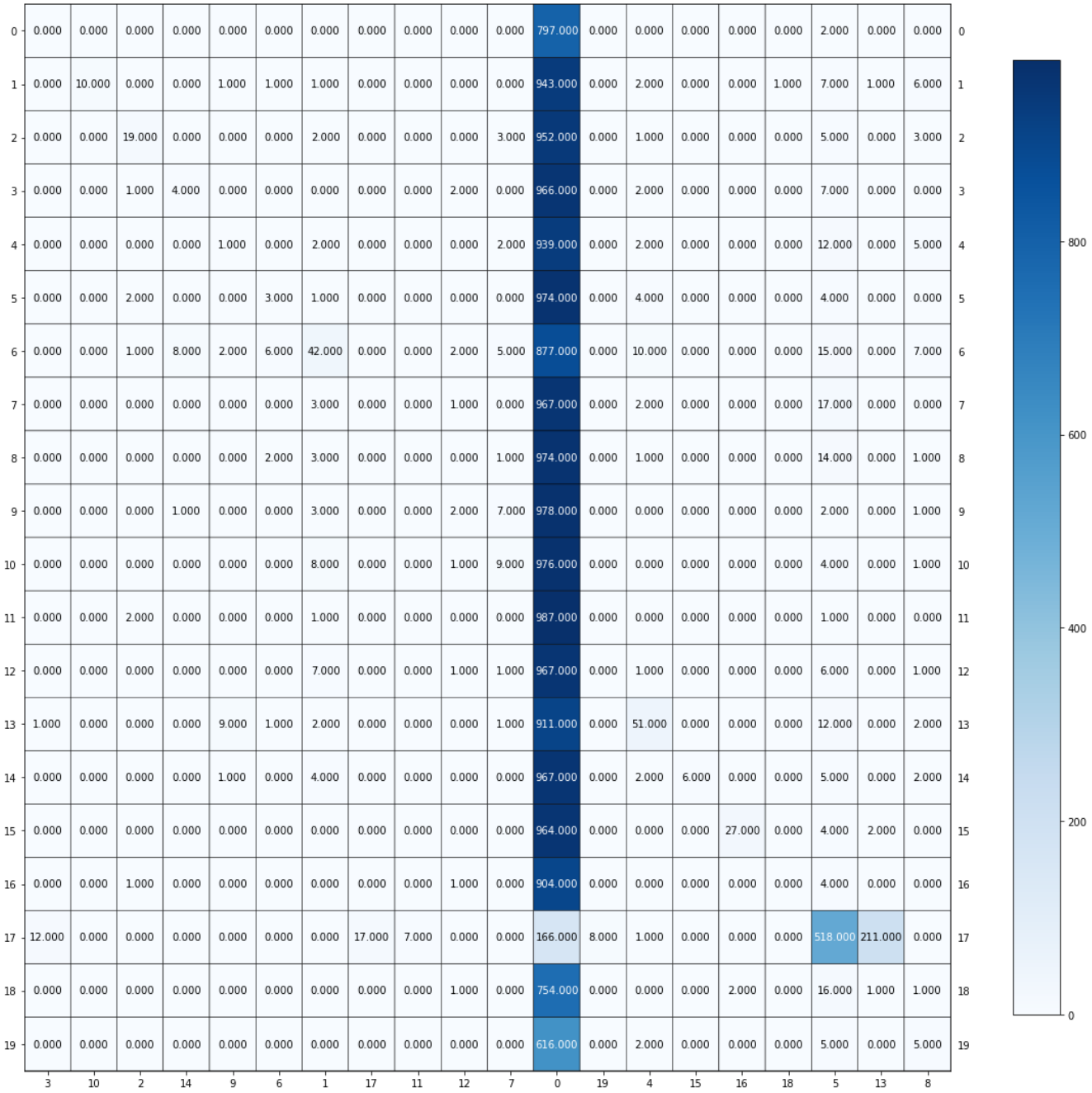
$$d_{AB} = \min \{|| a - b ||^2 : a \in A, b \in B\}$$

At each stage the two closest clusters in this regard are merged. Since all the distances between points in a cluster are not considered in making the merger decision, this clustering generally works only in the case of long 'chain' like clusters.

Since our document clusters do not have this representation (according to the UMAP representation above), single-linkage criterion has very poor behavior in clustering the given dataset, as seen in the scores below:

	metric	score
0	homogeneity_score	0.053183
1	completeness_score	0.435936
2	v_measure_score	0.094801
3	adjusted_rand_score	0.005671
4	adjusted_mutual_info_score	0.089461

Agglomerative clustering performed using `'single'` linkage



Contingency Matrix - Agglomerative clustering performed using `'single'` linkage

Q14 + Q15

Apply DBSCAN and HDBSCAN on UMAP-transformed 20-category data. Use min_cluster_size=100 . Experiment on the hyperparameters and report your findings in terms of the five clustering evaluation metrics.

ANSWER

DBSCAN and HDBSCAN:

They are density based clustering approaches. They form clusters based on highly dense areas and hence could take any shape. The advantage of both methods is that in forming clusters from given data points, they also assign some points to be noise. This is a useful feature, especially in large noisy datasets. The noisy points are those that are in regions of extremely sparse density (ie, there are no neighboring data points close to these noisy points).

The noisy points of the dataset are given the label -1

The basic principle is in identifying core samples (those that have at least a given number of neighbors within a set distance). All core samples that are closer to each other than the set distance and their neighbors form a cluster. Points that do not belong to any cluster are labelled as noise (with the label -1).

DBSCAN

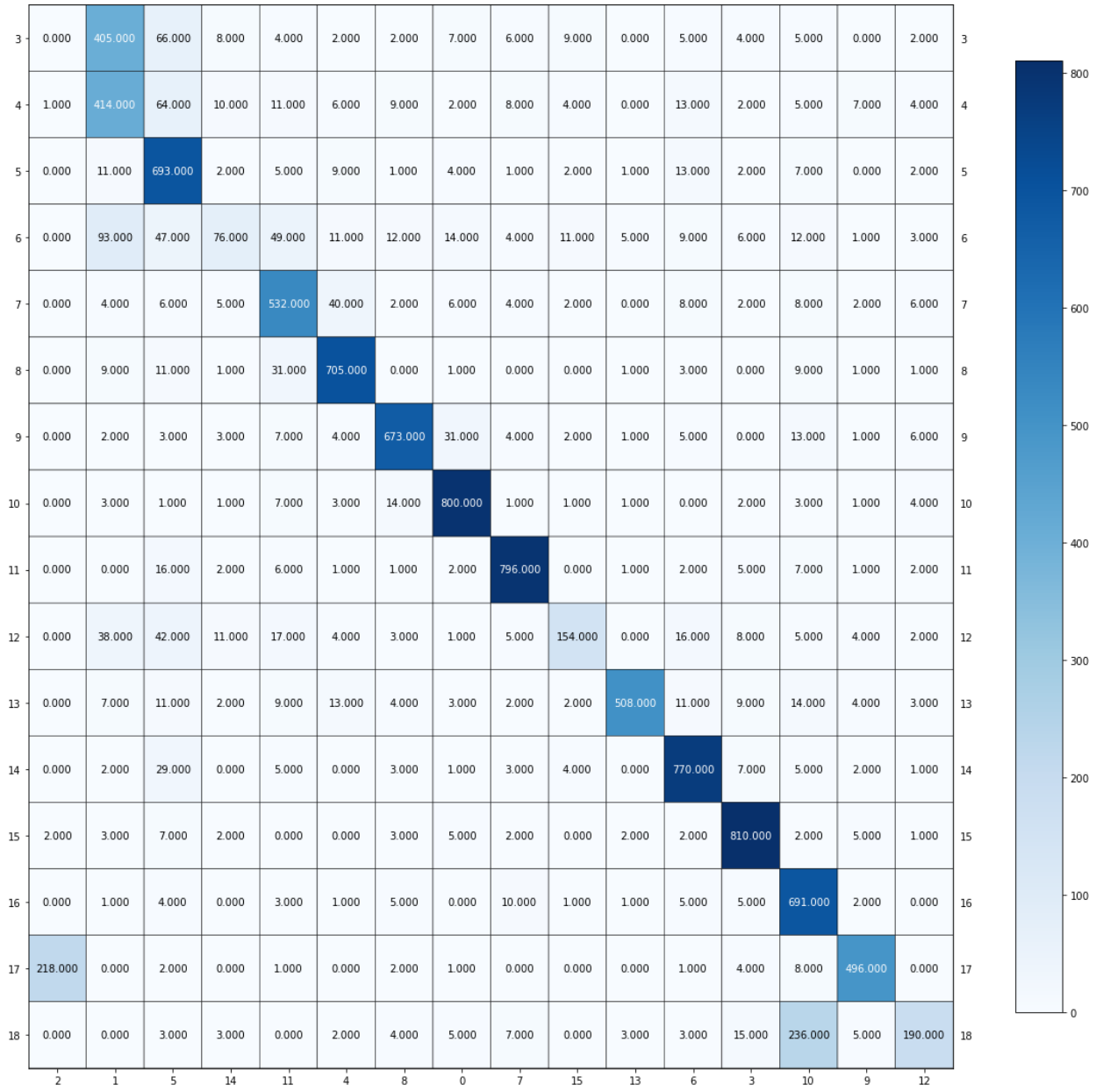
Let epsilon vary from 0.05 to 1, in steps of 0.05 and min_samples vary from 5 to 500 in steps of 10.

For DBSCAN, the best model obtained had `eps = 0.55 and min_samples = 115`. The other top performing models all have similar scores and are:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score	Epsilon	min_samples
511	0.456490	0.583557	0.512261	0.195460	0.510766	0.55	115
512	0.455876	0.584093	0.512081	0.182874	0.510587	0.55	125
565	0.456140	0.581712	0.511329	0.204580	0.509931	0.60	155
620	0.456960	0.579070	0.510819	0.214268	0.509423	0.65	205
621	0.454159	0.579352	0.509173	0.207271	0.507768	0.65	215

DBSCAN: Scores of top performing model hyperparameters (decreasing order)

The number of clusters as per the best DBSCAN model is 16. This is excluding the cluster labelled -1 which are all the noisy points. In plotting the below contingency matrix, noisy points were not considered since they are not a part of any cluster and are spread out in the sparse regions. Then, assigning each cluster to a class using `linear_sum_assignment`, the categories assigned to the clusters are:



Contingency Matrix: Best DBSCAN model (eps = 0.55, min_samples = 115) gives 16 clusters.

The categories not assigned to any cluster in the contingency matrix are 0, 2, 4, and 19. This is because of the matching optimization performed by `linear_sum_assignment`. However, simply plotting the non-permuted contingency matrix gives:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	67	0	0	0	711	3	4	4	1	1	1	4	0	2	1	0	0
1	479	8	60	0	3	4	378	8	9	4	4	6	5	1	0	2	2
2	372	2	92	0	2	2	483	7	3	3	2	2	4	1	0	8	2
3	457	7	405	0	4	2	66	5	6	2	0	5	4	2	0	8	9
4	403	2	414	1	2	6	64	13	8	9	7	5	11	4	0	10	4
5	235	4	11	0	2	9	693	13	1	1	0	7	5	2	1	2	2
6	622	14	93	0	6	11	47	9	4	12	1	12	49	3	5	76	11
7	363	6	4	0	2	40	6	8	4	2	2	8	532	6	0	5	2
8	223	1	9	0	0	705	11	3	0	0	1	9	31	1	1	1	0
9	239	31	2	0	0	4	3	5	4	673	1	13	7	6	1	3	2
10	157	800	3	0	2	3	1	0	1	14	1	3	7	4	1	1	1
11	149	2	0	0	5	1	16	2	796	1	1	7	6	2	1	2	0
12	674	1	38	0	8	4	42	16	5	3	4	5	17	2	0	11	154
13	388	3	7	0	9	13	11	11	2	4	4	14	9	3	508	2	2
14	155	1	2	0	7	0	29	770	3	3	2	5	5	1	0	0	4
15	151	5	3	2	810	0	7	2	2	3	5	2	0	1	2	2	0
16	181	0	1	0	5	1	4	5	10	5	2	691	3	0	1	0	1
17	207	1	0	218	4	0	2	1	0	2	496	8	1	0	0	0	0
18	299	5	0	0	15	2	3	3	7	4	5	236	0	190	3	3	0
19	173	0	0	0	370	2	3	9	0	2	3	57	2	5	2	0	0

Non permuted contingency matrix.
Best DBSCAN model: Rows- True categories, Columns - Clusters.

In the above figure, the 16 columns indicate the 16 clusters and the 20 rows are the ground categories.

- Column 0 has points from all ground labels and hence indicates the noisy points.
- Cluster (read column) 2 has grouped categories (read rows) 3 and 4. These categories are IBM hardware and Mac hardware and hence the model clustered these very similar labels together.
- Another cluster having highly similar topics is cluster (column) 4. It has most of the points

from categories (rows) 0, 15, and 19. These categories pertain to atheism, christianity, and miscellaneous religions. The closeness between the topics made it difficult for the model to make three clusters out of them.

- Custer 6 has grouped categories 1, 2, and 5 which are graphics, os ms-windows, and windows.x.

Such grouping led to the formation of 16 clusters instead of the expected 20 clusters.

HDBSCAN

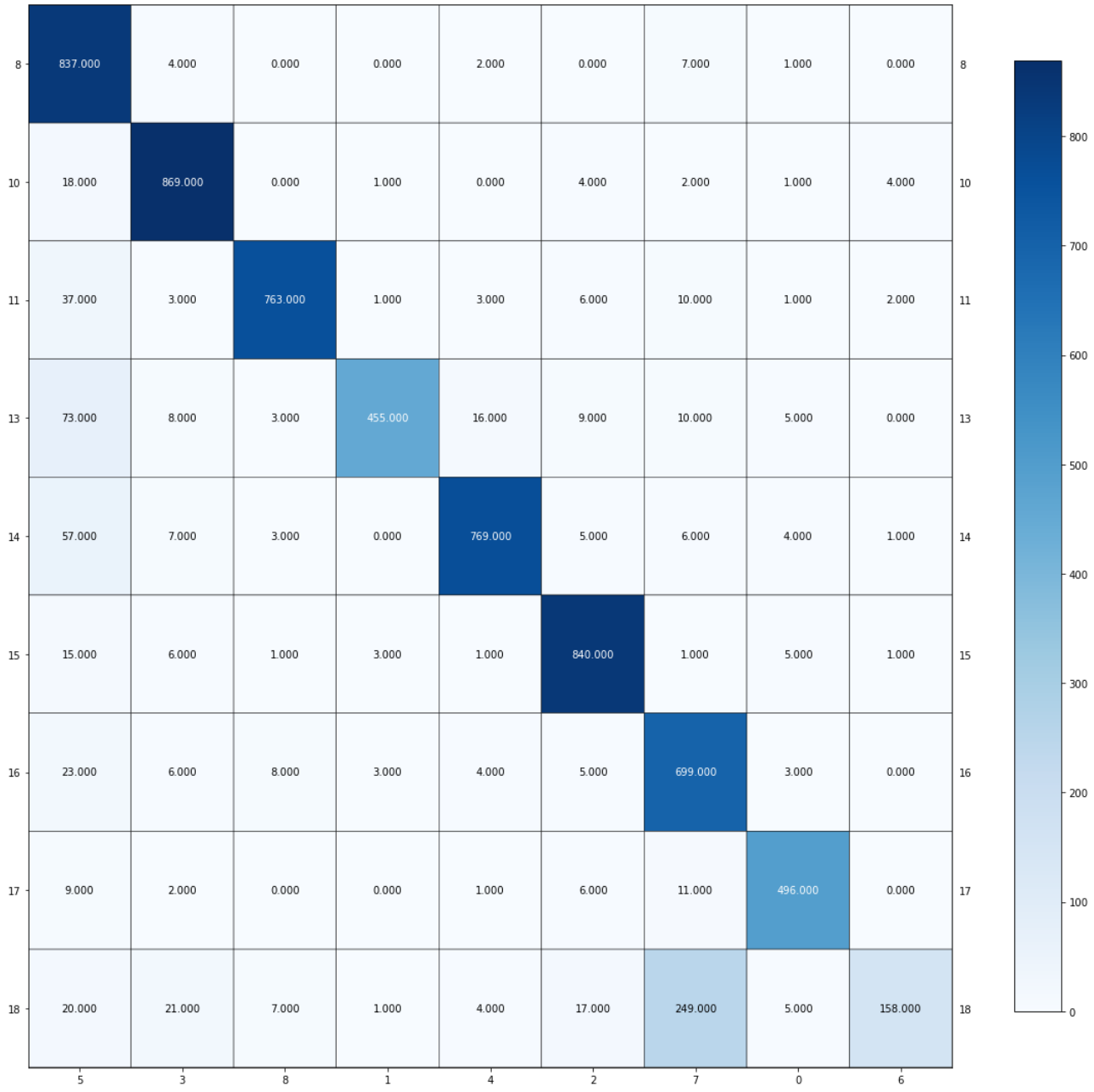
Hierarchical Density-Based Spatial Clustering of Applications with Noise - the HDBSCAN method performs a DBSCAN, but varies the epsilon (nearest distance limit). It results in finding clusters of varying density.

Let cluster_selection_epsilon vary from 0.05 to 1 and min_samples vary from 5 to 500.

The best HDBSCAN model predicted 9 clusters and was obtained with min_samples = 215 and cluster_selection_epsilon = 0.05. The other top performing model (all having very close scores) are:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score	Cluster_sel_Epsilon	min_samples
21	0.35603	0.572526	0.43904	0.160122	0.437981	0.05	215
71	0.35603	0.572526	0.43904	0.160122	0.437981	0.10	215
121	0.35603	0.572526	0.43904	0.160122	0.437981	0.15	215
171	0.35603	0.572526	0.43904	0.160122	0.437981	0.20	215
221	0.35603	0.572526	0.43904	0.160122	0.437981	0.25	215

HDBSCAN: Scores of top performing model hyperparameters



Permuted Contingency matrix - Best HDBSCAN model - has 9 clusters

The noisy points were not considered for plotting the above matrix because they are not part of any cluster. Also some categories were not assigned any clusters by the optimized matching performed by `linear_sum_assignment`. To get a look into the actual clustering, the non-permuted matrix is plotted below and the noise included.

	0	1	2	3	4	5	6	7	8	9
0	55	1	6	715	1	4	10	2	4	1
1	337	5	1	4	17	3	594	1	4	7
2	200	1	0	1	13	8	754	0	4	4
3	208	0	2	5	10	4	741	2	4	6
4	238	9	6	1	17	13	661	3	6	9
5	184	1	1	2	10	8	778	0	3	1
6	514	2	14	7	29	10	385	2	10	2
7	271	1	0	2	12	4	685	6	7	2
8	145	1	0	0	4	2	837	0	7	0
9	208	2	0	4	716	5	40	5	11	3
10	100	1	1	4	869	0	18	4	2	0
11	165	1	1	6	3	3	37	2	10	763
12	432	5	4	7	14	24	489	1	5	3
13	411	5	455	9	8	16	73	0	10	3
14	135	4	0	5	7	769	57	1	6	3
15	124	5	3	840	6	1	15	1	1	1
16	159	3	3	5	6	4	23	0	699	8
17	415	496	0	6	2	1	9	0	11	0
18	293	5	1	17	21	4	20	158	249	7
19	179	3	3	366	4	9	10	4	50	0

Non permuted Contingency matrix of best HDBSCAN

Rows: ground labels; Columns: clusters

The first column indicates noise as it has points from all ground labels.

Because the model forms only 9 clusters, all ‘computer’ related categories were grouped together (cluster 6). A surprising find is that category (row) 12 which deals with ‘electronics’ is also included in the same cluster having the computer topics. A major defect in this clustering is the grouping of ‘misc.forsale’, ‘rec.autos’, and ‘rec.motorcycles’ in the same cluster 6.

Other groups are appropriately clustered, as observed in clusters of DBSCAN model.

Q16

Data Acquisition

We load the data into a pandas dataframe from a csv file, and use factorize function to assign unique ID to each category.

Feature Engineering

We use Term Frequency-Inverse Document Frequency (TF-IDF) metric to build initial representation vectors for training. Then for dimension reduction, we evaluated three different dimension reduction methods (Truncated SVD, NMF and UMAP) with component numbers ranging from 1 to 1000. We evaluate the effectiveness of the dimension reduction using the K-mean clustering algorithm. The performance resulting from each dimension reduction data are shown in tables below.

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.094331	0.104362	0.099094	0.049817	0.095885
2	0.303752	0.346610	0.323769	0.193236	0.321323
3	0.448852	0.502422	0.474129	0.355048	0.472247
5	0.636503	0.685090	0.659903	0.572929	0.658711
10	0.633928	0.707318	0.668616	0.543747	0.667433
20	0.616071	0.693878	0.652664	0.514318	0.651419
50	0.635522	0.718284	0.674373	0.536805	0.673204
100	0.620452	0.698453	0.657146	0.517965	0.655918
300	0.715419	0.757440	0.735830	0.662944	0.734912
500	0.782276	0.799019	0.790559	0.780867	0.789844
800	0.721031	0.756067	0.738133	0.689256	0.737227

Truncated SVD

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.094331	0.104362	0.099094	0.049817	0.095885
2	0.247705	0.308859	0.274922	0.127401	0.272187
3	0.477529	0.493823	0.485539	0.343355	0.483772
5	0.663295	0.708868	0.685325	0.612784	0.684226
10	0.472978	0.627064	0.539229	0.319264	0.537447
20	0.254254	0.456657	0.326642	0.074724	0.323695
50	0.143036	0.427838	0.214395	0.049344	0.210228
100	0.068191	0.370604	0.115188	0.011561	0.109439
300	0.084514	0.349946	0.136147	0.015954	0.131116
500	0.032743	0.290324	0.058849	0.005426	0.052556
800	0.012738	0.241210	0.024199	0.001587	0.016873

NMF

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.685163	0.682540	0.683849	0.699950	0.682784
2	0.756216	0.757474	0.756845	0.789550	0.756023
3	0.762721	0.763487	0.763104	0.796711	0.762304
5	0.759920	0.760983	0.760451	0.794176	0.759642
10	0.762978	0.764017	0.763497	0.796813	0.762698
20	0.772143	0.772719	0.772431	0.804340	0.771663
50	0.761434	0.762417	0.761926	0.795680	0.761121
100	0.765329	0.766182	0.765755	0.799711	0.764964
300	0.770388	0.771026	0.770707	0.803880	0.769932
500	0.763265	0.764199	0.763732	0.798066	0.762934
800	0.769716	0.770296	0.770006	0.802935	0.769229

Euclidean UMAP

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.669491	0.668686	0.669088	0.677307	0.667972
2	0.777993	0.778654	0.778324	0.813780	0.777575
3	0.764136	0.765049	0.764592	0.800176	0.763797
5	0.761187	0.762158	0.761672	0.797965	0.760867
10	0.769021	0.769890	0.769456	0.804575	0.768677
20	0.768990	0.769854	0.769422	0.805443	0.768643
50	0.768743	0.769608	0.769175	0.803542	0.768396
100	0.765085	0.766117	0.765601	0.801170	0.764809
300	0.762515	0.763750	0.763132	0.797327	0.762331
500	0.770290	0.771010	0.770650	0.804821	0.769875
800	0.758685	0.759890	0.759287	0.793007	0.758474

Cosine UMAP

As we can see, Truncated SVD achieves the best score of 0.78 to 0.80 in all scoring categories; UMAP score is very close to that of Truncated SVD, and even surpass it in adjusted Rand score; NMF perform the worst, which might be caused by the non-negative weighing constrain making it fail to capture matrix.

Clustering

Here we use TruncatedSVD with 500 components for K-mean and cosine distance UMAP with 1000 components for Agglomerative, DBSCAN and HDBSCAN (these three work well with UMAP data).

For DBSCAN, we tested maximum neighbors distance ranging from 1 to 21 and core point neighbor count ranging from 5 to 200. For HDBSCAN, we set minimum cluster size to 100 and tested maximum neighbors distance ranging from 1 to 21 and core point neighbor count ranging from 5 to 500.

Results are shown in the evaluation section below.

Evaluation

	metric	score
0	homogeneity_score	0.782276
1	completeness_score	0.799019
2	v_measure_score	0.790559
3	adjusted_rand_score	0.780867
4	adjusted_mutual_info_score	0.789844

K-mean

	metric	score
0	homogeneity_score	0.760959
1	completeness_score	0.761969
2	v_measure_score	0.761464
3	adjusted_rand_score	0.796730
4	adjusted_mutual_info_score	0.760658

Agglomerative

For DBSCAN and HDBSCAN, there are 600 parameter combinations tested, so here we only report the score for the best parameter combination, detailed results can be found in DBSCAN_Q16.xlsx and HDBSCAN_Q16.xlsx.

	metric	score
0	homogeneity_score	0.728467
1	completeness_score	0.620127
2	v_measure_score	0.669945
3	adjusted_rand_score	0.645793
4	adjusted_mutual_info_score	0.668402

DBSCAN(eps = 0.95, min_samples = 75)

	metric	score
0	homogeneity_score	0.557704
1	completeness_score	0.786381
2	v_measure_score	0.652589
3	adjusted_rand_score	0.568314
4	adjusted_mutual_info_score	0.651539

HDBSCAN(min_cluster_size=100, min_samples = 15, cluster_selection_epsilon = 0.05)

As we can see, K-mean is the best performing algorithm. It outperforms DBSCAN/HDBSCAN and slightly outperforms Agglomerative by a very slim margin. We believe this is caused by the sparse nature of the dataset, which favors K-mean over DBSCAN/HDBSCAN. Moreover, K-mean and Agglomerative have the advantage of knowing the exact number of clusters, while DBSCAN/HDBSCAN doesn't.

Project 2: Clustering

ECE 219: Large-Scale Data Mining: Models and Algorithms [Winter 2021]

Prof. Vwani Roychowdhury

UCLA, Department of ECE

Due: 2021.02.23 11:59PM PT

In [37]:
```
from sklearn.datasets import fetch_20newsgroups
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import random
import string
import urllib
import zipfile
import sys

import warnings
warnings.filterwarnings('ignore')

np.set_printoptions(precision=4, suppress=True)

RANDOM_SEED = 42

np.random.seed(RANDOM_SEED)
random.seed(RANDOM_SEED)
```

Q1

Building the TF-IDF matrix.

Following the steps in Project 1, transform the documents into TF-IDF vectors.

Use min_df = 3, exclude the stopwords (no need to do stemming or lemmatization), and remove the headers and footers.

Report the dimensions of the TF-IDF matrix you get.

ANSWER

```
inputs_tfidf.shape: (7882, 27768)
```

In [4]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [21]:
```
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'
]
```

```
news_data = fetch_20newsgroups(
    subset='all',
    categories=categories,
    shuffle=True,
    random_state=RANDOM_SEED,
    remove=('header', 'footer')
)

inputs = news_data.data
labels = news_data.target // 4

target_names = ['Computer Technology', 'Recreational Activity']
```

In [6]:

```
tfidf_vectorizer = TfidfVectorizer(min_df=3, stop_words='english')

inputs_tfidf = tfidf_vectorizer.fit_transform(inputs)
print("inputs_tfidf.shape:", inputs_tfidf.shape)
```

```
inputs_tfidf.shape: (7882, 27768)
```

Q2

Report the contingency table of your clustering result. You may use the provided `plotmat.py` to visualize the matrix.

In [7]:

```
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import (
    contingency_matrix,
    homogeneity_score,
    v_measure_score,
    completeness_score,
    adjusted_rand_score,
    adjusted_mutual_info_score
)
from plotmat import plot_mat
```

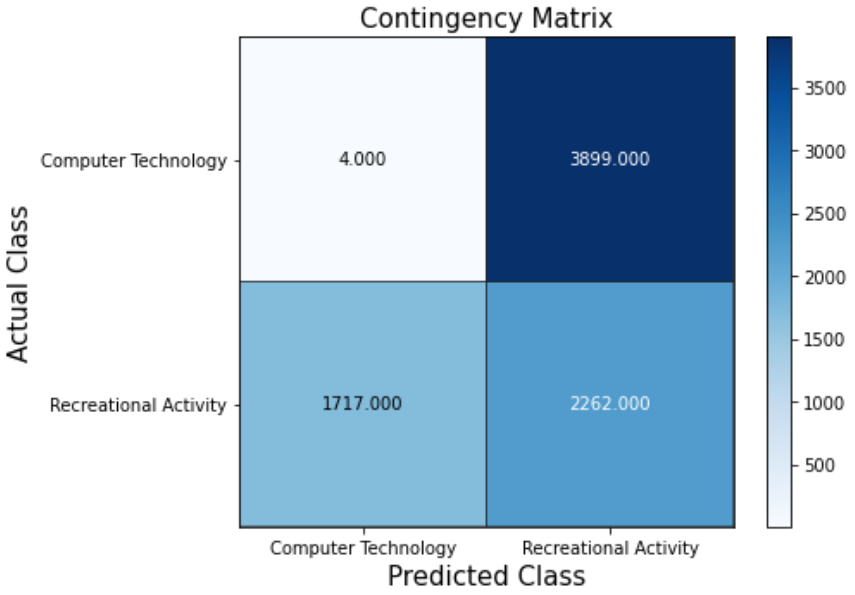
In [13]:

```
km = KMeans(
    n_clusters=2,
    random_state=0,
    max_iter=5000,
    n_init=50
)
preds = km.fit_predict(inputs_tfidf)
```

In [14]:

```
A = contingency_matrix(labels, preds)
print("Contingency table: \n", A)
plot_mat(
    mat=A,
    size=(7,5),
    xlabel='Predicted Class',
    ylabel='Actual Class',
    xticklabels=target_names,
    yticklabels=target_names,
    title='Contingency Matrix'
)
```

```
Contingency table:
 [[   4 3899]
 [1717 2262]]
```



Q3

Report the 5 measures above for the K-means clustering results you get.

In [15]:
```
def get_cluster_metrics(y_true, y_pred, metrics=None):
    if not metrics:
        metrics = [
            homogeneity_score,
            completeness_score,
            v_measure_score,
            adjusted_rand_score,
            adjusted_mutual_info_score
        ]
    d = {}
    for m in metrics:
        d[m.__name__] = m(y_true, y_pred)
    df = pd.DataFrame(d, index=[0]).T
    df.reset_index(inplace=True)
    df.rename(columns={'index':'metric', 0:'score'}, inplace=True)
    return df
```

In [16]:
```
get_cluster_metrics(labels, preds)
```

Out[16]:

	metric	score
0	homogeneity_score	0.253413
1	completeness_score	0.334677
2	v_measure_score	0.288430
3	adjusted_rand_score	0.180546
4	adjusted_mutual_info_score	0.288356

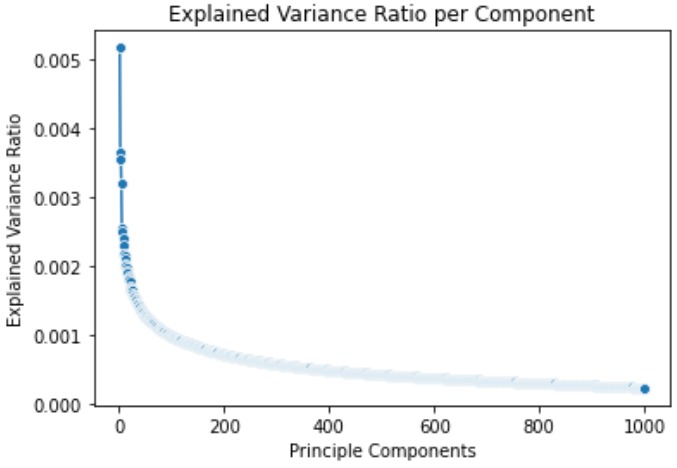
Q4

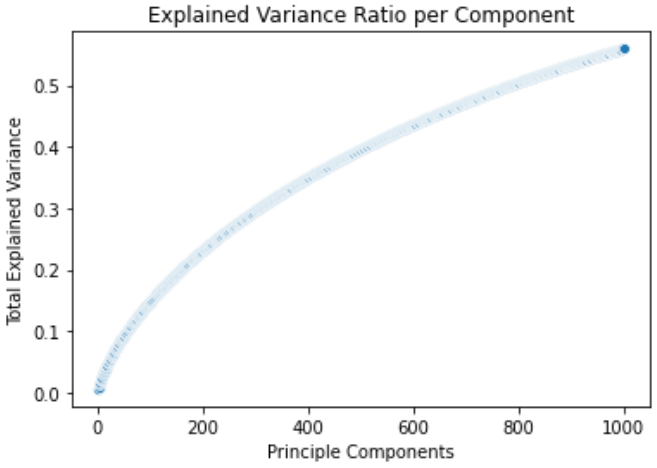
Report the plot of the percent of variance the top r principle components can retain v.s. r, for $r \in \{1...1000\}$.

In [17]:
```
from sklearn.decomposition import TruncatedSVD
```

In [22]:
```
svd = TruncatedSVD(n_components=1000,random_state=RANDOM_SEED)
inputs_svd = svd.fit_transform(inputs_tfidf)
```

In [55]:
```
xs = np.arange(1,1001)
ys = sorted(svd.explained_variance_ratio_, reverse=True)
ys_cum = np.cumsum(svd.explained_variance_ratio_)

g = sns.lineplot(
    x=xs,
    y=ys,
    marker='o'
)
g.set_xlabel('Principle Components')
g.set_ylabel('Explained Variance Ratio')
g.set_title('Explained Variance Ratio per Component')
plt.show()

g = sns.lineplot(
    x=xs,
    y=ys_cum,
    marker='o'
)
g.set_xlabel('Principle Components')
g.set_ylabel('Total Explained Variance')
g.set_title('Explained Variance Ratio per Component')
plt.show()
```





Q5

Let r be the dimension that we want to reduce the data to (i.e. n_components). Try $r \in \{1, 2, 3, 5, 10, 20, 50, 100, 300\}$, and plot the 5 measure scores v.s. r for both SVD and NMF.

Report a good choice of r for SVD and NMF respectively.

ANSWER

$r = 2$ is the best choice for both SVD and NMF.

In [89]:
```
from sklearn.decomposition import NMF
```

In [98]:
```
km = KMeans(
    n_clusters=2,
    random_state=0,
    max_iter=5000,
    n_init=50
)

n_components = [1, 2, 3, 5, 10, 20, 50, 100, 300]
```

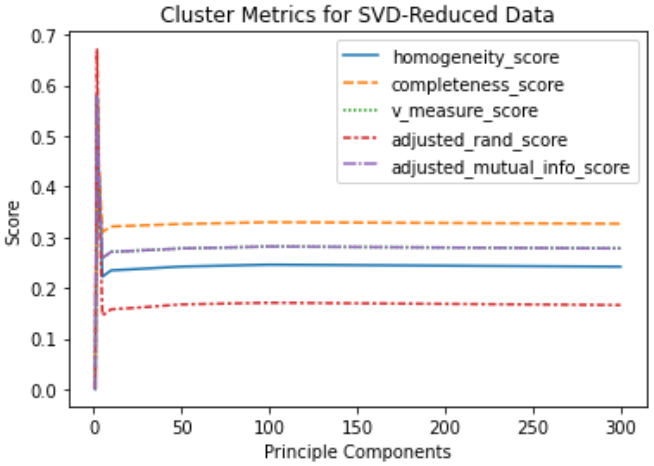
SVD-Reduced Data

In [99]:
```
scores = []
for r in n_components:
    inputs_svd = TruncatedSVD(n_components=r, random_state=RANDOM_SEED).fit_transform(inputs_tfidf)
    preds = km.fit_predict(inputs_svd)
    scores.append(get_cluster_metrics(labels, preds)['score'].tolist())
metrics = get_cluster_metrics(labels, preds)['metric'].tolist()
metrics_df = pd.DataFrame(scores, columns=metrics, index=n_components)
metrics_df
```

Out[99]:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.000271	0.000275	0.000273	0.000296	0.000181
2	0.577167	0.579872	0.578517	0.672517	0.578478
3	0.413470	0.448835	0.430427	0.414009	0.430373
5	0.222218	0.310361	0.258995	0.145737	0.258916
10	0.234625	0.321219	0.271177	0.157799	0.271100
20	0.236404	0.322558	0.272841	0.159822	0.272765
50	0.242165	0.326171	0.277960	0.167420	0.277884
100	0.245914	0.329721	0.281717	0.170760	0.281641
300	0.241949	0.326734	0.278022	0.166176	0.277946

In [10…
```
g = sns.lineplot(data=metrics_df)
g.set_xlabel('Principle Components')
g.set_ylabel('Score')
g.set_title('Cluster Metrics for SVD-Reduced Data')
plt.show()
```



NMF-Reduced Data

In [10…

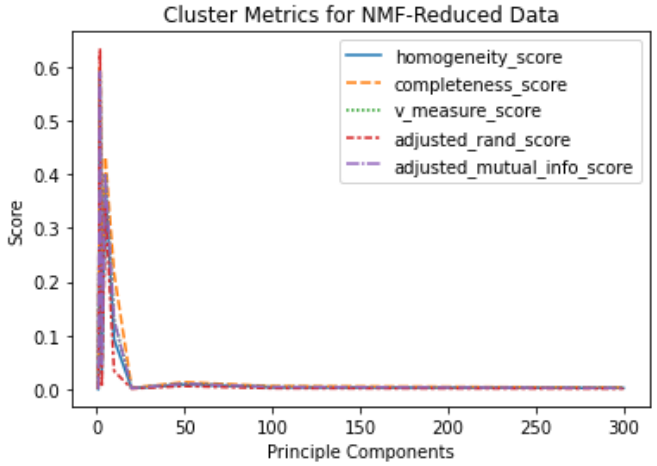
```
scores = []
for r in n_components:
    inputs_nmf = NMF(n_components=r, init='random', random_state=RANDOM_SEED).fit_transform(inputs_tfidf)
    preds = km.fit_predict(inputs_nmf)
    scores.append(get_cluster_metrics(labels, preds)['score'].tolist())
metrics = get_cluster_metrics(labels, preds)['metric'].tolist()
metrics_df = pd.DataFrame(scores, columns=metrics, index=n_components)
metrics_df
```

Out[101]:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.000271	0.000275	0.000273	0.000296	0.000181
2	0.582552	0.599296	0.590806	0.634364	0.590768
3	0.027821	0.155780	0.047211	0.003928	0.047063
5	0.375401	0.433068	0.402178	0.341705	0.402119
10	0.092525	0.215844	0.129527	0.033742	0.129415
20	0.001860	0.002471	0.002122	0.001201	0.002018
50	0.008735	0.013233	0.010523	0.005683	0.010414
100	0.003617	0.005938	0.004496	0.001795	0.004382
300	0.002161	0.003253	0.002597	0.001114	0.002487

In [10…

```
g = sns.lineplot(data=metrics_df)
g.set_xlabel('Principle Components')
g.set_ylabel('Score')
g.set_title('Cluster Metrics for NMF-Reduced Data')
plt.show()
```



Q6

How do you explain the non-monotonic behavior of the measures as r increases?

ANSWER

At $r = 1$, the distances are hard to differentiate from one another as there is only 1 dimension to "travel." Allowing for $r = 2$ provides for a larger space for the points to be projected upon and therefore easier to distinguish via K-Means. However, as r continues to increase and therefore the dimensionality increases, the default Euclidean distance measure becomes less and less capable of providing an intuitive sense of separation. The aptly named "Curse of Dimensionality" strikes and unless we use a different measure of distance, we are better off sticking with $r = 2$.

Q7

Visualize the clustering results for:

- SVD with your choice of r
- NMF with your choice of r

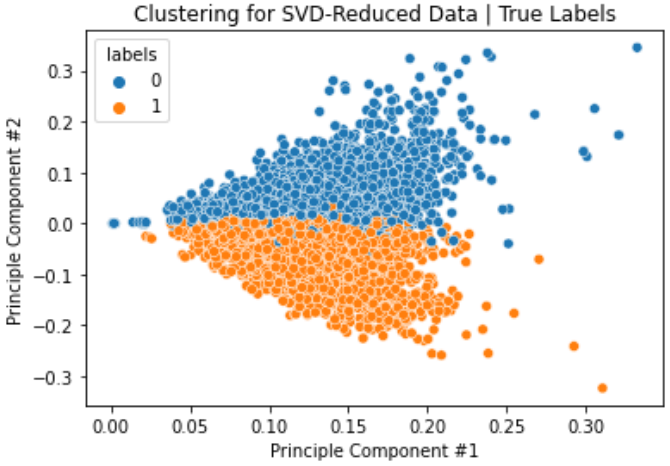
In [12…

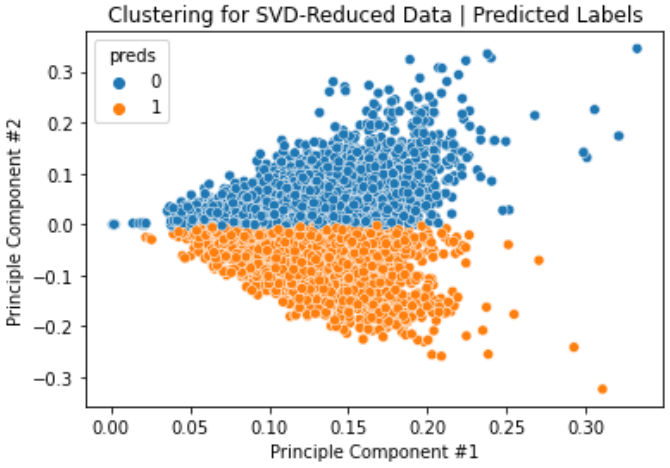
```
r = 2

inputs_svd = TruncatedSVD(n_components=r, random_state=RANDOM_SEED).fit_transform(inputs_tfidf)
km_svd = KMeans(n_clusters=2, random_state=RANDOM_SEED, max_iter=5000, n_init=50)
preds_svd = km_svd.fit_predict(inputs_svd)

inputs_nmf = NMF(n_components=r, init='random', random_state=RANDOM_SEED).fit_transform(inputs_tfidf)
km_nmf = KMeans(n_clusters=2, random_state=RANDOM_SEED, max_iter=5000, n_init=50)
preds_nmf = km_nmf.fit_predict(inputs_nmf)

svd_df = pd.DataFrame(inputs_svd, columns=['pc1', 'pc2'])
nmf_df = pd.DataFrame(inputs_nmf, columns=['pc1', 'pc2'])

svd_df['labels'] = labels
nmf_df['labels'] = labels

svd_df['preds'] = preds_svd
nmf_df['preds'] = preds_nmf
```

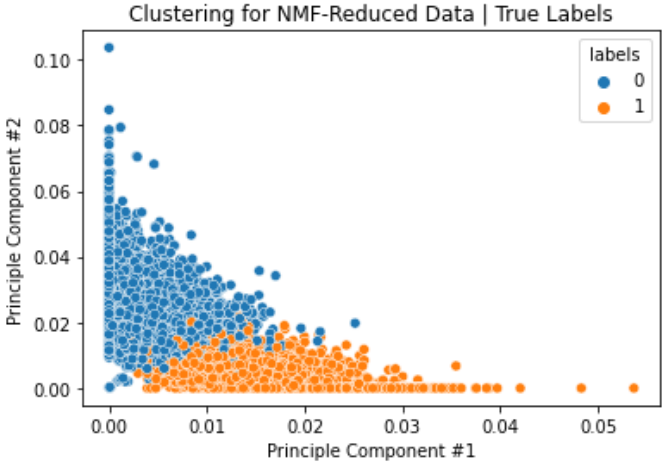
In [12…

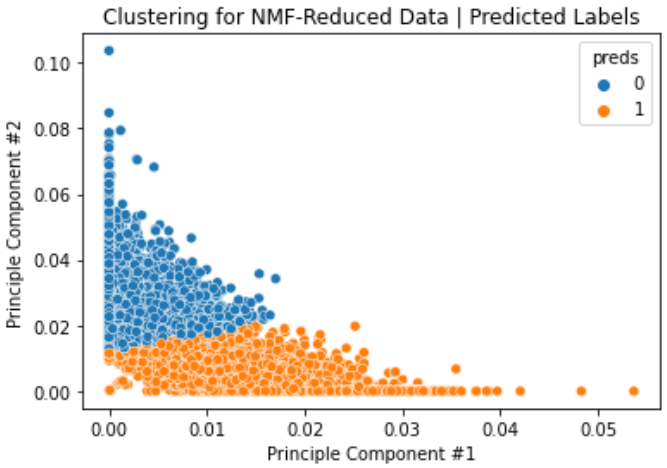
```
g = sns.scatterplot(x='pc1', y='pc2', hue='labels', data=svd_df)
g.set_xlabel('Principle Component #1')
g.set_ylabel('Principle Component #2')
g.set_title('Clustering for SVD-Reduced Data | True Labels')
plt.show()

g = sns.scatterplot(x='pc1', y='pc2', hue='preds', data=svd_df)
```

```
g.set_xlabel('Principle Component #1')
g.set_ylabel('Principle Component #2')
g.set_title('Clustering for SVD-Reduced Data | Predicted Labels')
plt.show()

g = sns.scatterplot(x='pc1', y='pc2', hue='labels', data=nmf_df)
g.set_xlabel('Principle Component #1')
g.set_ylabel('Principle Component #2')
g.set_title('Clustering for NMF-Reduced Data | True Labels')
plt.show()

g = sns.scatterplot(x='pc1', y='pc2', hue='preds', data=nmf_df)
g.set_xlabel('Principle Component #1')
g.set_ylabel('Principle Component #2')
g.set_title('Clustering for NMF-Reduced Data | Predicted Labels')
plt.show()
```









Q8

What do you observe in the visualization? How are the data points of the two classes distributed? Is the data distribution ideal for K-Means clustering?

ANSWER

KMeans makes 3 primary assumptions:

1. the feature distributions are spherical
2. the features have similar variance
3. there are approximately the same number of examples from each class.

If any of these assumptions are violated, KMeans will underperform. Unfortunately, the plots show that the feature distributions are not spherical. There are also no separable or distinct clusters for KMeans to assign. Much of the data overlaps in the reduced dimension.

Q9

In [ ]:

In [ ]:

In [45]:
```
from sklearn.datasets import fetch_20newsgroups
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
import random
import string
import urllib
import zipfile
import sys
from scipy.optimize import linear_sum_assignment
import warnings
warnings.filterwarnings('ignore')

np.set_printoptions(precision=4, suppress=True)

RANDOM_SEED = 42

np.random.seed(RANDOM_SEED)
random.seed(RANDOM_SEED)
```

9

Loaded documents of all 20 categories from 20Newsgroups.

min_df = 3, exculded stopwords, removed headers and footers.

SVD to reduce dimensions of TF-IDF matrix with n_components = 1000

In [84]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [85]:
```
news_data = fetch_20newsgroups(
    subset='all',
    shuffle=True,
    random_state=RANDOM_SEED,
    remove=('header', 'footer')
)

inputs = news_data.data
labels = news_data.target

target_names = news_data.target_names
```

In [86]:
```
print(target_names)
```

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']
```

In [87]:
```
tfidf_vectorizer = TfidfVectorizer(min_df=3, stop_words='english')

inputs_tfidf = tfidf_vectorizer.fit_transform(inputs)
print("inputs_tfidf.shape:", inputs_tfidf.shape)
```

```
inputs_tfidf.shape: (18846, 52295)
```

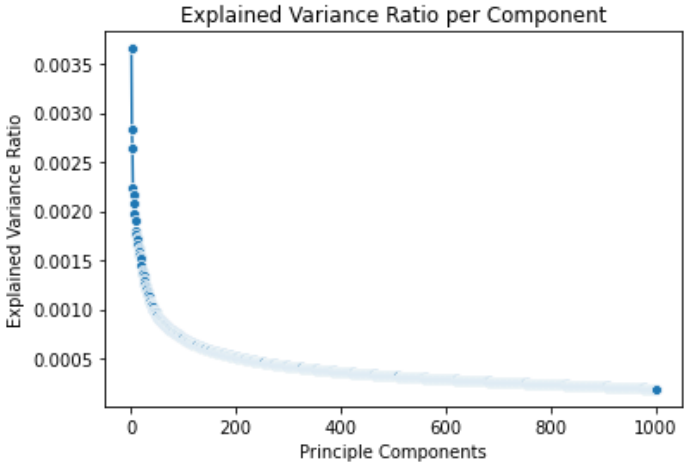
In [40]:
```
from sklearn.decomposition import TruncatedSVD
```

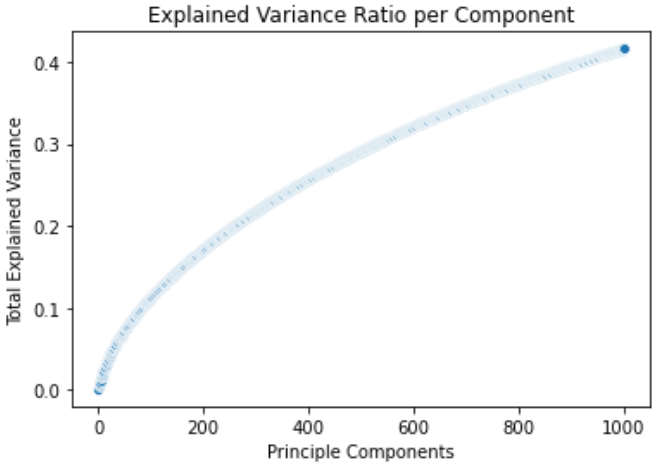
Visualized the variance explained by upto 1000 components. They seem to explain almost 50% of the variance in input

data, and each further component explains a very small fraction of the variance.

In [45]:
```
svd = TruncatedSVD(n_components=1000,random_state=RANDOM_SEED)
inputs_svd = svd.fit_transform(inputs_tfidf)
```

In [46]:
```
xs = np.arange(1,1001)
ys = sorted(svd.explained_variance_ratio_, reverse=True)
ys_cum = np.cumsum(svd.explained_variance_ratio_)

g = sns.lineplot(
    x=xs,
    y=ys,
    marker='o'
)
g.set_xlabel('Principle Components')
g.set_ylabel('Explained Variance Ratio')
g.set_title('Explained Variance Ratio per Component')
plt.show()

g = sns.lineplot(
    x=xs,
    y=ys_cum,
    marker='o'
)
g.set_xlabel('Principle Components')
g.set_ylabel('Total Explained Variance')
g.set_title('Explained Variance Ratio per Component')
plt.show()
```





In [50]:
```
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import (
```

```
    contingency_matrix,
    homogeneity_score,
    v_measure_score,
    completeness_score,
    adjusted_rand_score,
    adjusted_mutual_info_score
)
from plotmat import plot_mat
```

A function to display table of required scores:

In [51]:
```
def get_cluster_metrics(y_true, y_pred, metrics=None):
    if not metrics:
        metrics = [
            homogeneity_score,
            completeness_score,
            v_measure_score,
            adjusted_rand_score,
            adjusted_mutual_info_score
        ]
    d = {}
    for m in metrics:
        d[m.__name__] = m(y_true, y_pred)
    df = pd.DataFrame(d, index=[0]).T
    df.reset_index(inplace=True)
    df.rename(columns={'index':'metric', 0:'score'}, inplace=True)
    return df
```

Using n_components = [1, 2, 3, 5, 10, 20, 50, 100, 300, 500, 800] test the multi class classification:

In [39]:
```
km = KMeans(
    n_clusters= len(target_names),
    random_state=0,
    max_iter=5000,
    n_init=50
)
```

In [54]:
```
scores = []
n_components = [1, 2, 3, 5, 10, 20, 50, 100, 300, 500, 800]
for r in n_components:
    inputs_svd = TruncatedSVD(n_components=r, random_state=RANDOM_SEED).fit_transform(inputs_tfidf)
    preds = km.fit_predict(inputs_svd)
    scores.append(get_cluster_metrics(labels, preds)['score'].tolist())
metrics = get_cluster_metrics(labels, preds)['metric'].tolist()
metrics_df = pd.DataFrame(scores, columns=metrics, index=n_components)
metrics_df
```

Out[54]:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.027941	0.031138	0.029453	0.005799	0.026110
2	0.211032	0.224915	0.217753	0.064692	0.215137
3	0.232702	0.242342	0.237424	0.079573	0.234909
5	0.313221	0.332460	0.322554	0.123212	0.320297
10	0.339986	0.379278	0.358559	0.139547	0.356361
20	0.287669	0.383175	0.328624	0.092857	0.326112
50	0.310314	0.413422	0.354523	0.093870	0.352104
100	0.275215	0.381518	0.319763	0.083114	0.317164
300	0.299456	0.402700	0.343488	0.098837	0.340994
500	0.284961	0.451065	0.349270	0.087328	0.346626

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
800	0.297703	0.407515	0.344060	0.112155	0.341544

We see that n_components = 5 and 10 result in very good scores

Displaying the clustering scores and contingency matrix with n_components = 5

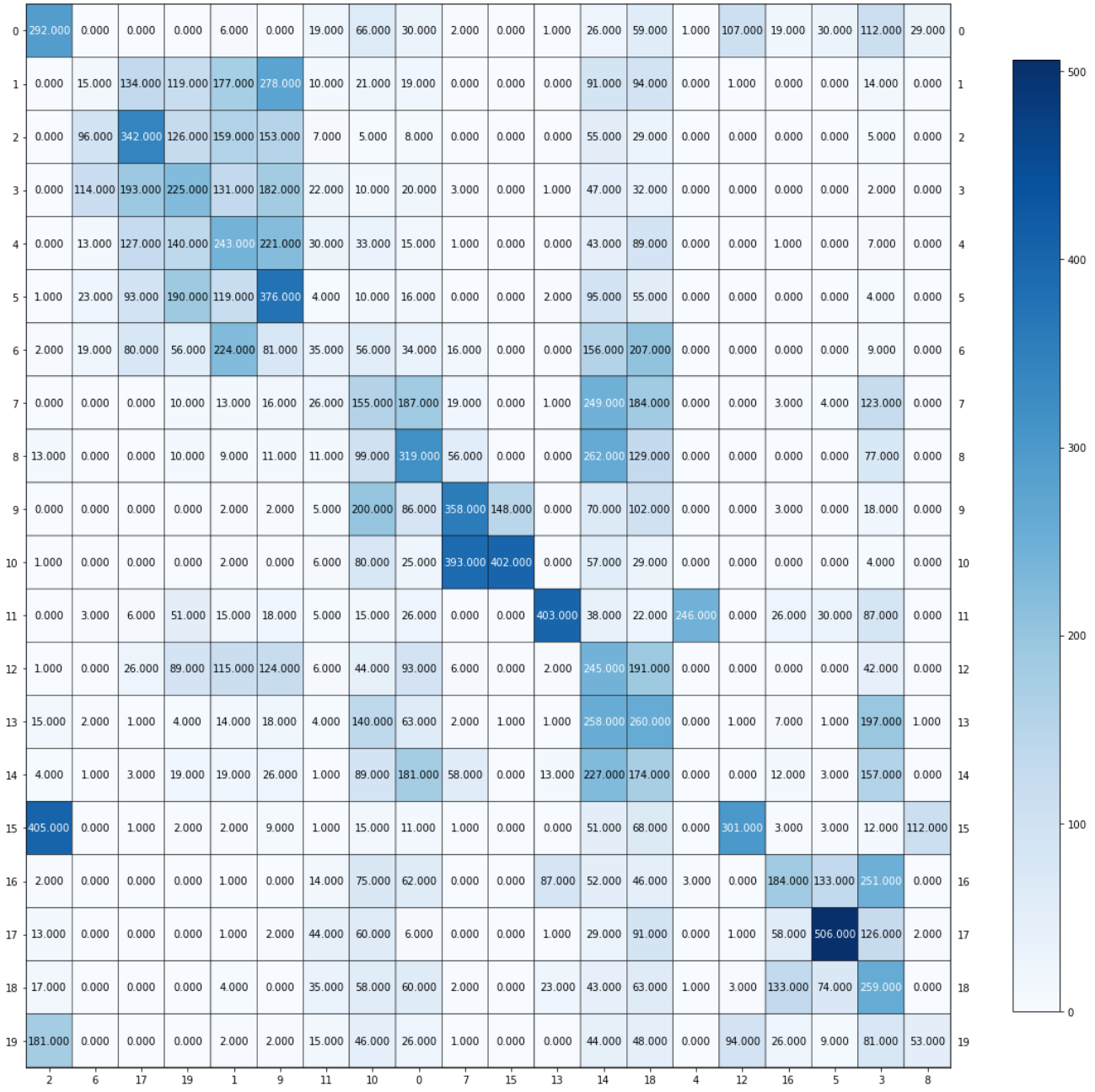
In [41]:
```
#n = 5
inputs_svd = TruncatedSVD(n_components=5, random_state=RANDOM_SEED).fit_transform(inputs_tfidf)
preds = km.fit_predict(inputs_svd)
get_cluster_metrics(labels, preds)
```

Out[41]:

	metric	score
0	homogeneity_score	0.313221
1	completeness_score	0.332460
2	v_measure_score	0.322554
3	adjusted_rand_score	0.123212
4	adjusted_mutual_info_score	0.320297

In [42]:
```
cm = contingency_matrix(labels, preds)
rows, cols = linear_sum_assignment(cm, maximize=True)
print("Contingency table (SVD n_components = 5, Kmeans)")
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, size=(15,15))
```

```
Contingency table (SVD n_components = 5, Kmeans)
```



Displaying the metrics and the contingency matrix with n_components = 10

In [43]:

```
#n = 10
inputs_svd = TruncatedSVD(n_components=10, random_state=RANDOM_SEED).fit_transform(inputs_tfidf)
preds = km.fit_predict(inputs_svd)
get_cluster_metrics(labels, preds)
```

Out[43]:

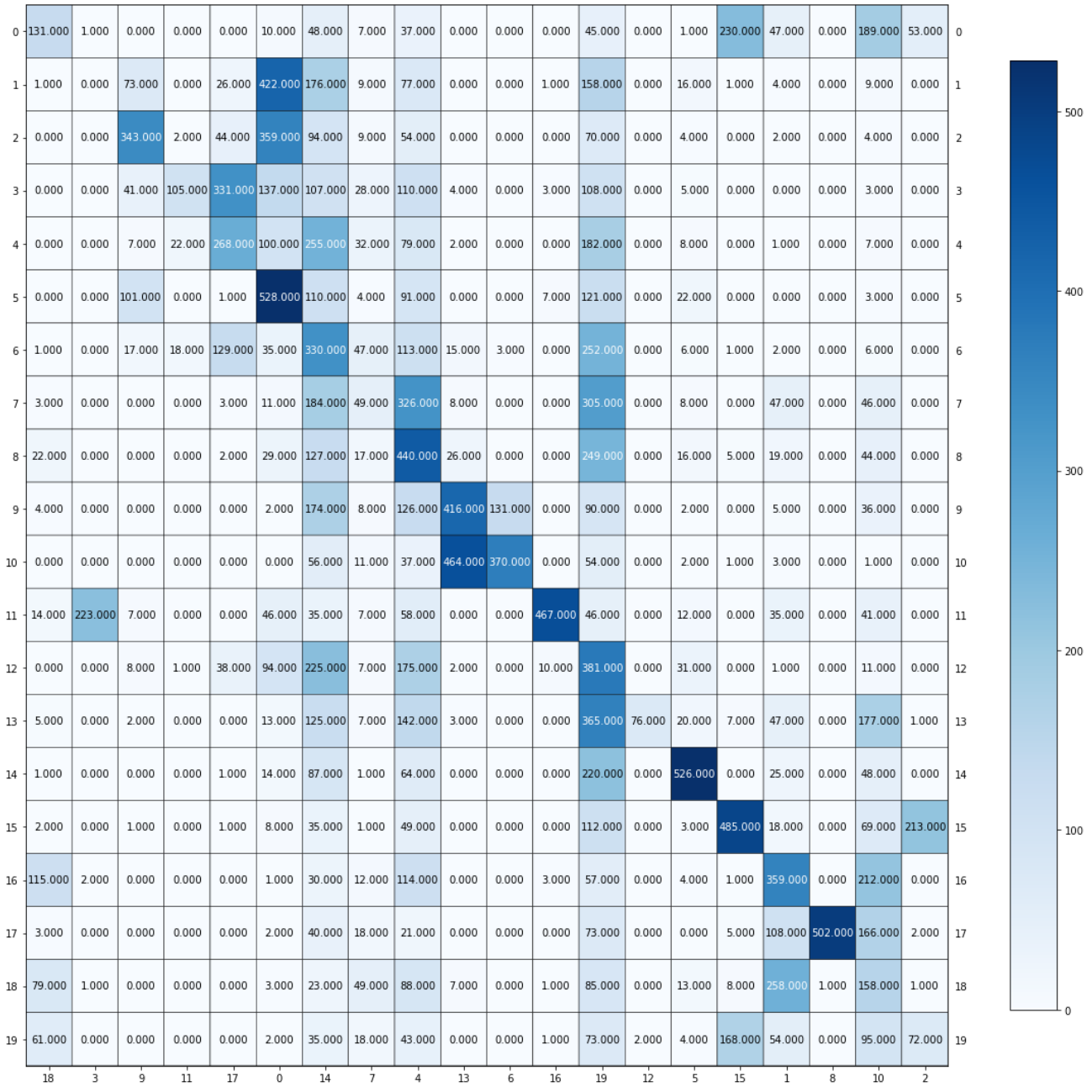
	metric	score
0	homogeneity_score	0.339986
1	completeness_score	0.379278
2	v_measure_score	0.358559
3	adjusted_rand_score	0.139547
4	adjusted_mutual_info_score	0.356361

In [44]:

```
cm = contingency_matrix(labels, preds)
rows, cols = linear_sum_assignment(cm, maximize=True)
```

```
print("Contingency table (SVD n_components = 10, Kmeans)")
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, size=(15,15))
```

Contingency table (SVD n_components = 10, Kmeans)



10

Using NMF decomposition, with Kullback-Leibler divergence as the distance measure

In [62]:
```
from sklearn.decomposition import NMF
```

In [26]:
```
km = KMeans(
    n_clusters=len(target_names),
    random_state=0,
    max_iter=5000,
    n_init=50
)
```

Checking which n_components gives better results when used NMF decompostion with KL divergence

In [64]:
```
scores = []
n_components = [1, 2, 3, 5, 10, 20, 50, 100, 300]
for r in n_components:
    inputs_nmf = NMF(n_components=r, init='random', solver = 'mu', beta_loss = 'kullback-leibler', random_
    preds = km.fit_predict(inputs_nmf)
    scores.append(get_cluster_metrics(labels, preds)['score'].tolist())
metrics = get_cluster_metrics(labels, preds)['metric'].tolist()
metrics_df = pd.DataFrame(scores, columns=metrics, index=n_components)
metrics_df
```

Out[64]:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.030556	0.034513	0.032414	0.006957	0.029031
2	0.198715	0.225536	0.211277	0.063138	0.208523
3	0.256536	0.278224	0.266940	0.093704	0.264456
5	0.350546	0.374438	0.362098	0.155914	0.359949
10	0.405617	0.414149	0.409838	0.210817	0.407913
20	0.382709	0.392057	0.387327	0.252586	0.385325
50	0.239701	0.342298	0.281956	0.046972	0.279203
100	0.199862	0.389404	0.264149	0.028596	0.260931
300	0.042846	0.231684	0.072318	0.002391	0.066781

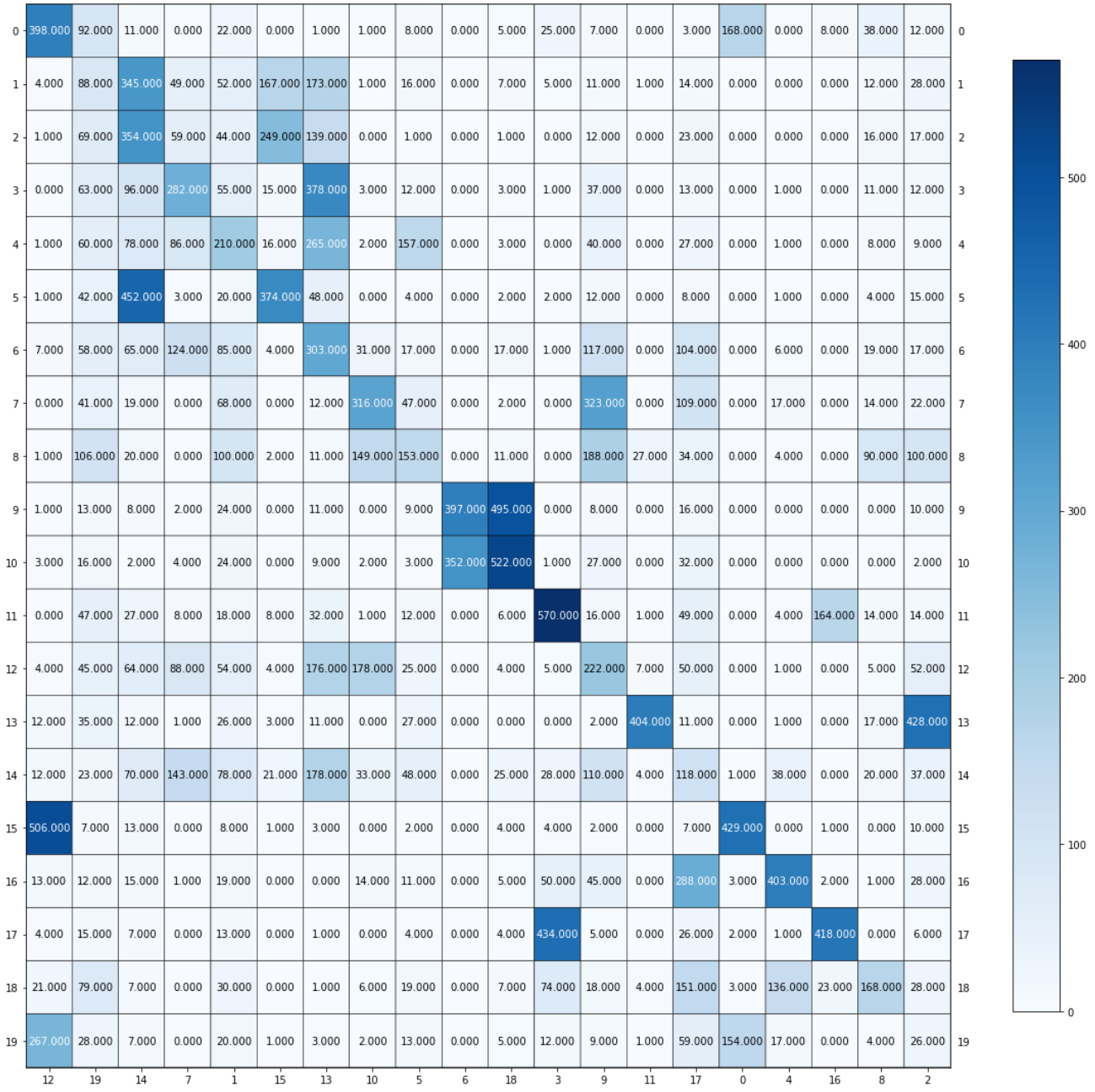
The best scores were obtained for n_components = 10. The contingency matrix in that case would be:

In [65]:
```
inputs_nmf = NMF(n_components=10, init='random', solver = 'mu', beta_loss = 'kullback-leibler', random_sta
preds = km.fit_predict(inputs_nmf)
get_cluster_metrics(labels, preds)
```

Out[65]:

	metric	score
0	homogeneity_score	0.405617
1	completeness_score	0.414149
2	v_measure_score	0.409838
3	adjusted_rand_score	0.210817
4	adjusted_mutual_info_score	0.407913

In [66]:
```
cm = contingency_matrix(labels, preds)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, size=(15,15))
```



Q 11 and 12:

Using UMAP to find the best n_components to reduce the data.

Compared the performance with metric = "eucledian" and metric = "cosine"

In [52]:
```
import umap
```

Find best n_components with metric = 'eucledian':

In [68]:
```
#with metric = 'eucledian'
scores = []
for n in n_components:
    umapfit = umap.UMAP(n_components = n, metric = 'euclidean')
    inputs_umap = umapfit.fit_transform(inputs_tfidf)
    preds = km.fit_predict(inputs_umap)
    get_cluster_metrics(labels, preds)
    scores.append(get_cluster_metrics(labels, preds)['score'].tolist())
metrics = get_cluster_metrics(labels, preds)['metric'].tolist()
```

```
metrics_df = pd.DataFrame(scores, columns=metrics, index=n_components)
metrics_df
```

Out[68]:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.313804	0.328794	0.321124	0.205228	0.318863
2	0.468820	0.486809	0.477645	0.369443	0.475912
3	0.489580	0.515532	0.502221	0.392948	0.500556
5	0.499953	0.532349	0.515643	0.392992	0.514013
10	0.497766	0.537320	0.516788	0.396259	0.515141
20	0.502224	0.543511	0.522052	0.402544	0.520413
50	0.508013	0.546018	0.526330	0.409757	0.524718
100	0.514402	0.542460	0.528059	0.419452	0.526467
300	0.515771	0.535938	0.525661	0.417134	0.524086

Using metric ='eucledian', best n_components is 5 (n_components = 300 gives metrics superior only in the third decimal of the fraction)

The evaluation metrics and contingency matrix in this case would be:

In [69]:

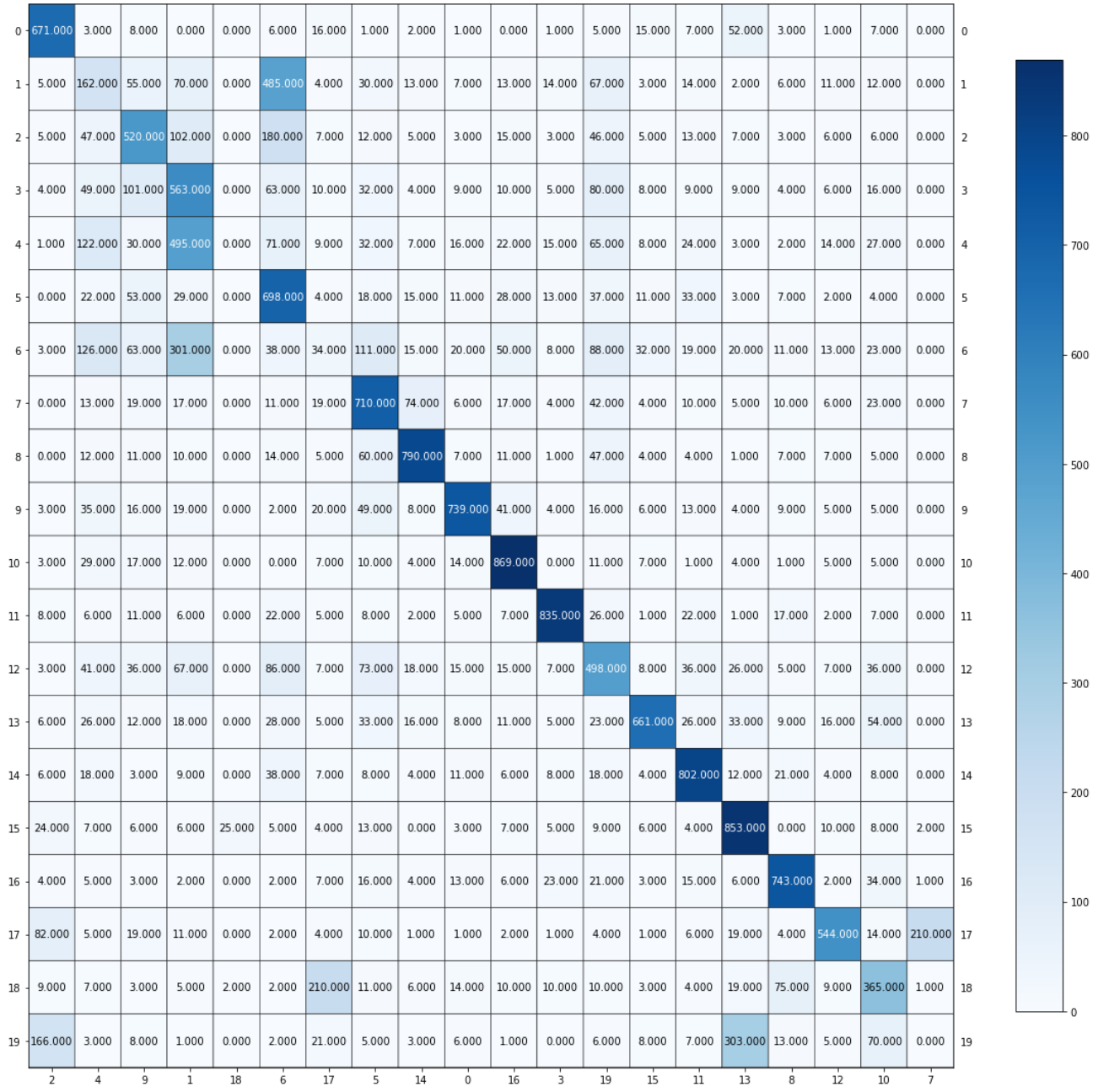
```
inputs_umap = umap.UMAP(n_components = 5, metric = 'euclidean').fit_transform(inputs_tfidf)
preds = km.fit_predict(inputs_umap)
get_cluster_metrics(labels, preds)
```

Out[69]:

	metric	score
0	homogeneity_score	0.511601
1	completeness_score	0.531974
2	v_measure_score	0.521589
3	adjusted_rand_score	0.414142
4	adjusted_mutual_info_score	0.520000

In [70]:

```
cm = contingency_matrix(labels, preds)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, size=(15,15))
```



Q12 Analysis of contingency matrix

Since certain categories are close to each other like religion, atheism, and christianity or IBM and mac hardware, the model must distinguish the other topics well enough. However, it is prone to get confused in clustering categories that are close to each other. This is observed in the above matrix as:

1. Christianity and talks about miscellaneous religions clustered together: The cluster (column) labelled 13 has most of its documents from categories (rows) 15 and 19. This is an encouraging sign because categories 15 and 19 pertain to christianity and talks about miscellaneous religions. The topics being highly similar, were clustered together by the model.

2. IBM PC hardware and Mac hardware clustered together: The cluster (column) labelled 1 has most of its documents from categories (rows) 3 and 4. This is an encouraging sign because categories 3 and 4 pertain to IBM harware and Mac hardware. The topics being highly similar, were clustered together by the model.

3. Atheism and religion clustered together: The cluster (column) labelled 2 has most of its documents from categories (rows) 1 and 19. This is an encouraging sign because categories 1 and 19 pertain to atheism and miscellanous religions. The topics being related, were clustered together by the model.

Similar behaviour although with different labelling is observed on using the 'cosine' metric, as seen below:

Find best n_components with metric = 'cosine':

In [71]:
```
#with metric = 'cosine'
scores = []
for n in n_components:
    umapfit = umap.UMAP(n_components = n, metric = 'cosine')
    inputs_umap = umapfit.fit_transform(inputs_tfidf)
    preds = km.fit_predict(inputs_umap)
    get_cluster_metrics(labels, preds)
    scores.append(get_cluster_metrics(labels, preds)['score'].tolist())
metrics = get_cluster_metrics(labels, preds)['metric'].tolist()
metrics_df = pd.DataFrame(scores, columns=metrics, index=n_components)
metrics_df
```

Out[71]:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.326141	0.337057	0.331509	0.220160	0.329310
2	0.475285	0.491008	0.483019	0.375309	0.481307
3	0.485451	0.509786	0.497321	0.393881	0.495632
5	0.501004	0.532950	0.516484	0.399789	0.514858
10	0.502372	0.530903	0.516244	0.407605	0.514623
20	0.510078	0.531100	0.520377	0.414975	0.518783
50	0.500182	0.541944	0.520227	0.394553	0.518588
100	0.498076	0.534279	0.515543	0.387900	0.513894
300	0.497215	0.537664	0.516649	0.394478	0.515015

Using metric = 'cosine', best n_components is: 3 (however, n_components = 5 also give very close metrics)

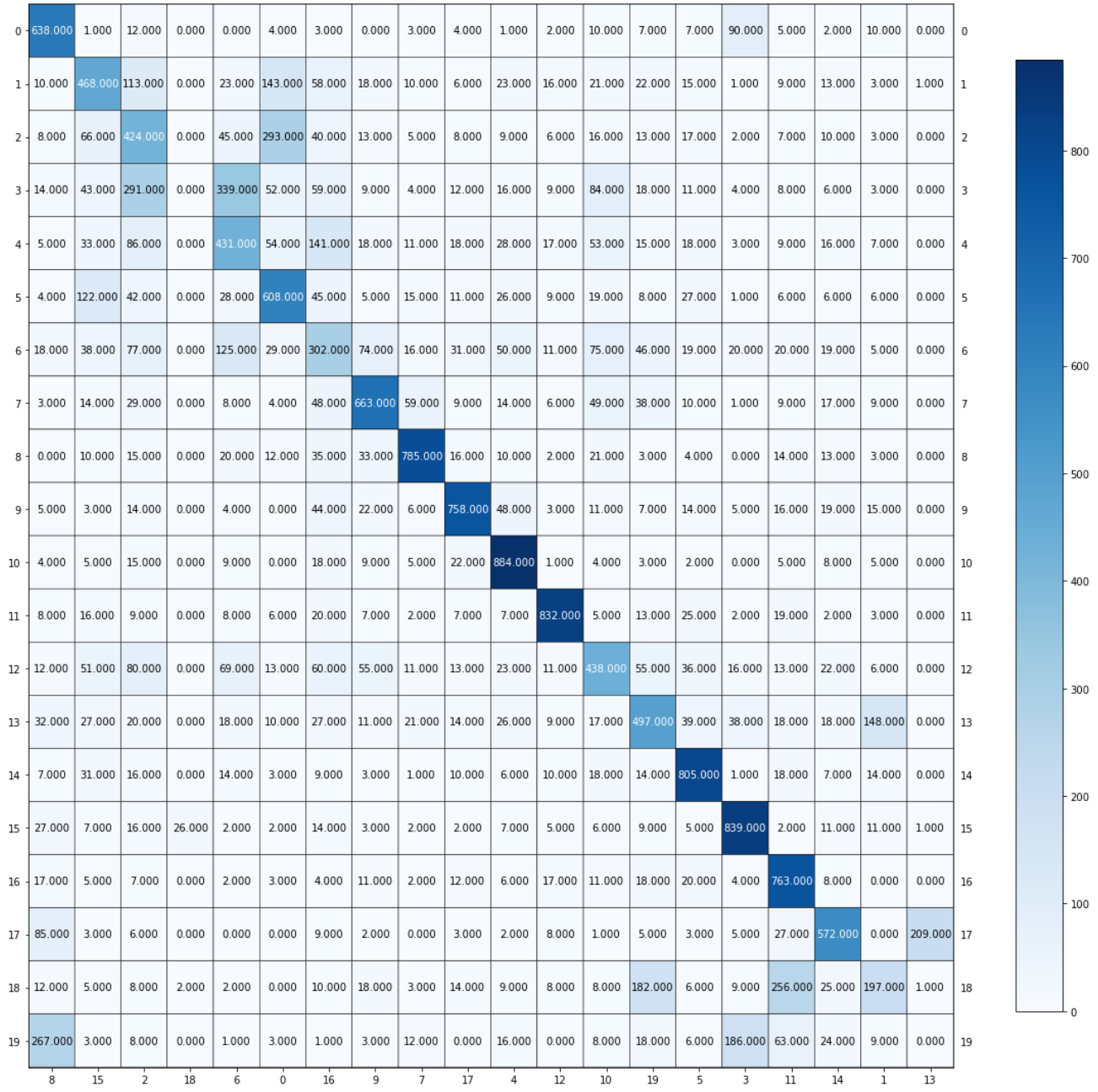
The evaluation metrics and contingency matrices (using n_components = 3) are:

In [72]:
```
inputs_umap = umap.UMAP(n_components = 3, metric = 'cosine').fit_transform(inputs_tfidf)
preds = km.fit_predict(inputs_umap)
get_cluster_metrics(labels, preds)
```

Out[72]:

	metric	score
0	homogeneity_score	0.491172
1	completeness_score	0.505385
2	v_measure_score	0.498177
3	adjusted_rand_score	0.394992
4	adjusted_mutual_info_score	0.496519

In [73]:
```
cm = contingency_matrix(labels, preds)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, size=(15,15))
```



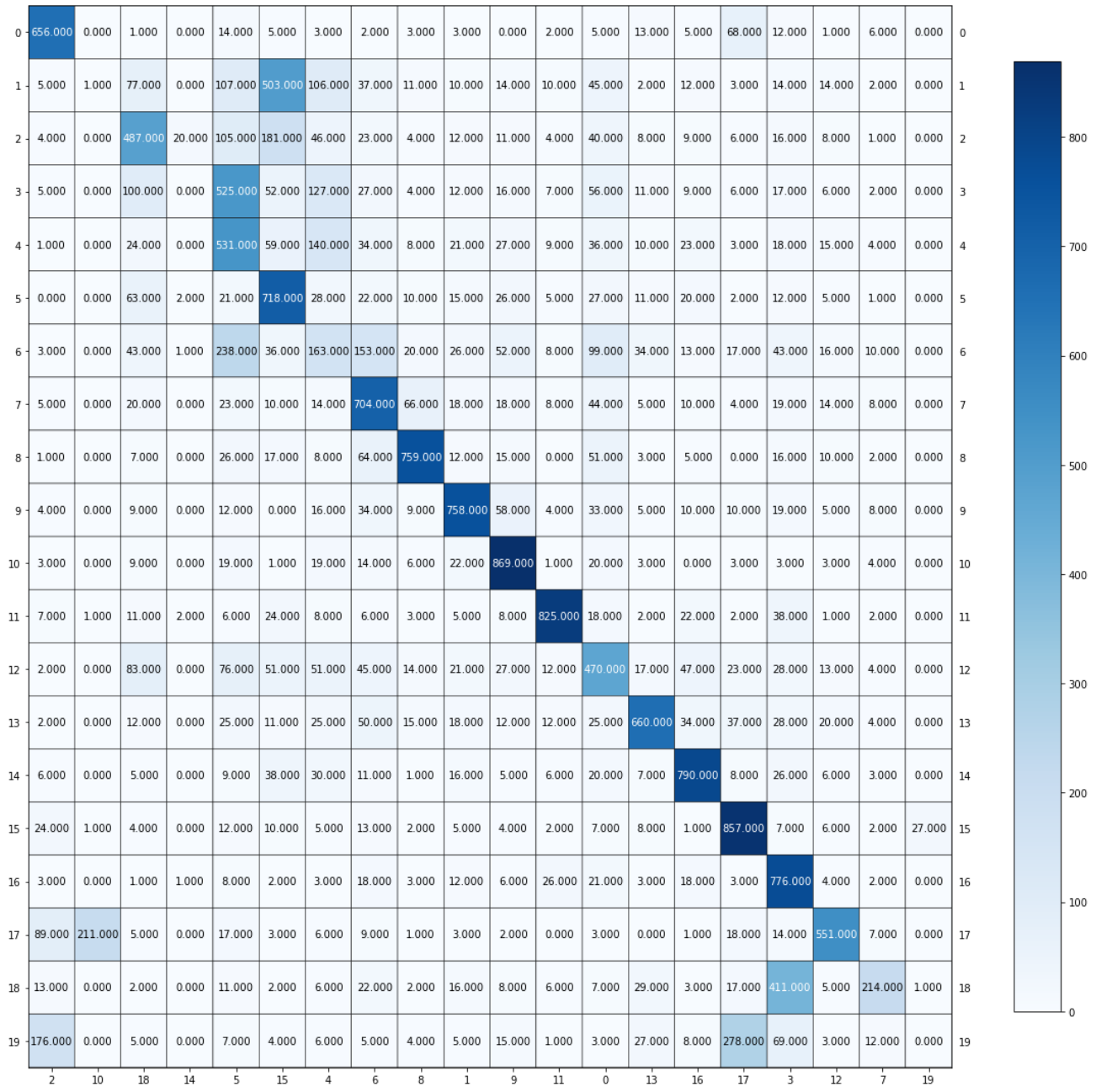
The 'cosine' metric with n_components = 5 gives:

In [27]:
```
inputs_umap = umap.UMAP(n_components = 5, metric = 'cosine').fit_transform(inputs_tfidf)
preds = km.fit_predict(inputs_umap)
get_cluster_metrics(labels, preds)
```

Out[27]:

	metric	score
0	homogeneity_score	0.493709
1	completeness_score	0.522841
2	v_measure_score	0.507858
3	adjusted_rand_score	0.397501
4	adjusted_mutual_info_score	0.506206

In [75]:
```
cm = contingency_matrix(labels, preds)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, size=(15,15))
```



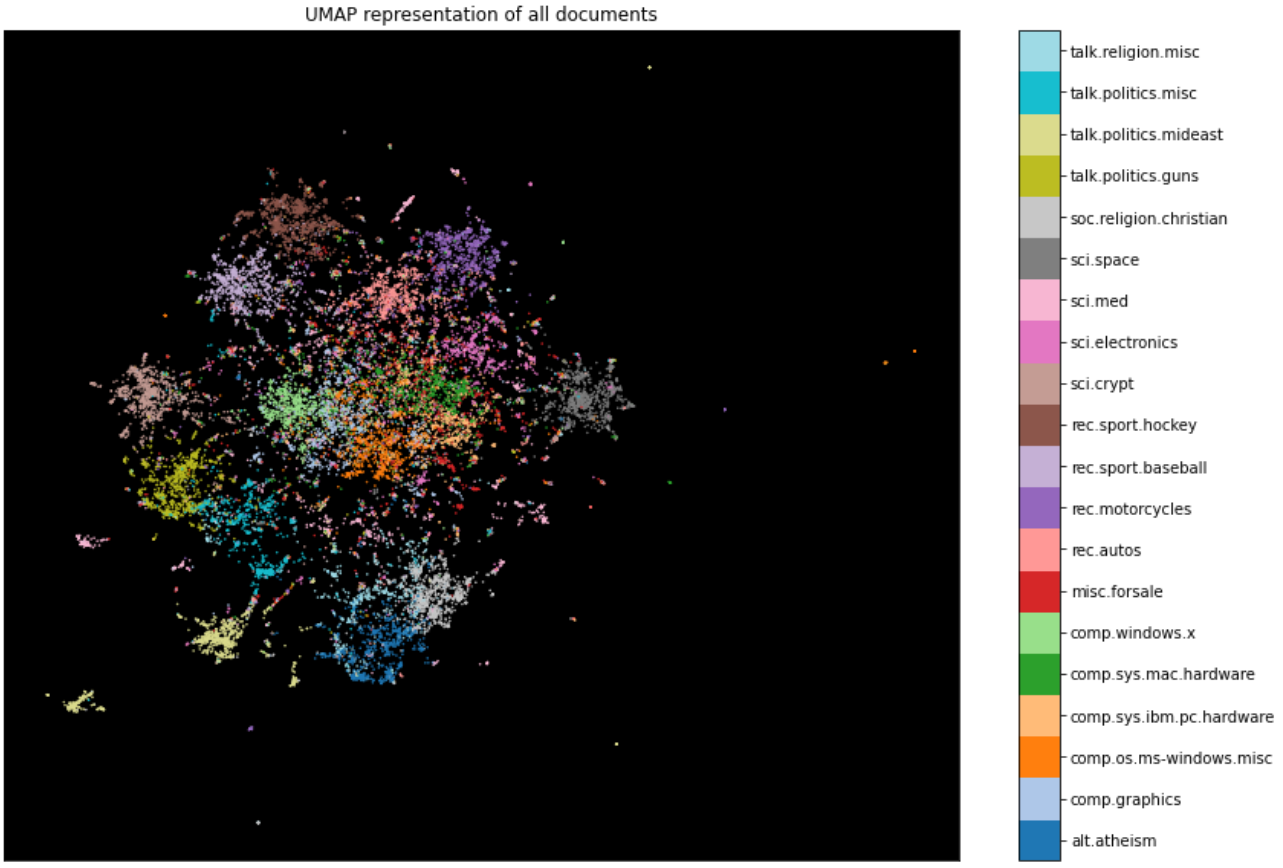
Q 13: Agglomerative clustering

A UMAP representation of the inputs is plotted, colored according to the category each document belongs to.

In [8]:
```
embedding = umap.UMAP().fit_transform(inputs_tfidf)
```

In [14]:
```
fig, ax = plt.subplots(1, figsize=(14, 10))
ax.set_facecolor('k')
plt.scatter(*embedding.T, s = 0.3, c = labels, cmap = 'tab20', alpha = 1.0)
plt.setp(ax, xticks=[], yticks=[])
cbar = plt.colorbar(boundaries = np.arange(21) - 0.5)
cbar.set_ticks(np.arange(20))
cbar.set_ticklabels(target_names)
plt.title('UMAP representation of all documents')
```

Out[14]:
```
Text(0.5, 1.0, 'UMAP representation of all documents')
```



Comparison between 'ward' and 'single' linkage criteria.

Used UMAP with metric = 'cosine' and n_components = 3, as it gave the best results for k means in the above task

In [53]:
```
from sklearn.cluster import AgglomerativeClustering
```

Agglomerative clustering is a bottom-up hierarchial clustering technique where at each step, the pair of clusters that are closest to each other according to some distance metric are merged into one cluster. This process is iterated so that each iteration lowers the total number of clusters by 1. The point at which we stop the clustering is determined by the number of clusters we decide to group the data into.

Using 'ward' linkage:

Ward's method aims to minimize the total within-cluster variance. However, it need not give the optimal lowest within-cluster variance since it is contrained by the choices made in forming the previous clusters.

At any stage, those two clusters shall be merged that lead to the smallest increase in within-cluster varaince after the merger. This can equivalently be described by saying that the two clusters merged have the lowest weighted centroid distance between all pairs of clusters. The metric is given by:

$\frac{nAnB}{nA+nB}\|m_A - m_b\|^2$ where m_j is the centroid of cluster j and n_j is the number of points in it.

The ward method works well for the 'cloud' type of clusters which are dense about a centroid and relatively sparse as we move away from it. Since our UMAP representation of data shows similar behaviour, using the 'ward' criterion serves our clustering well.

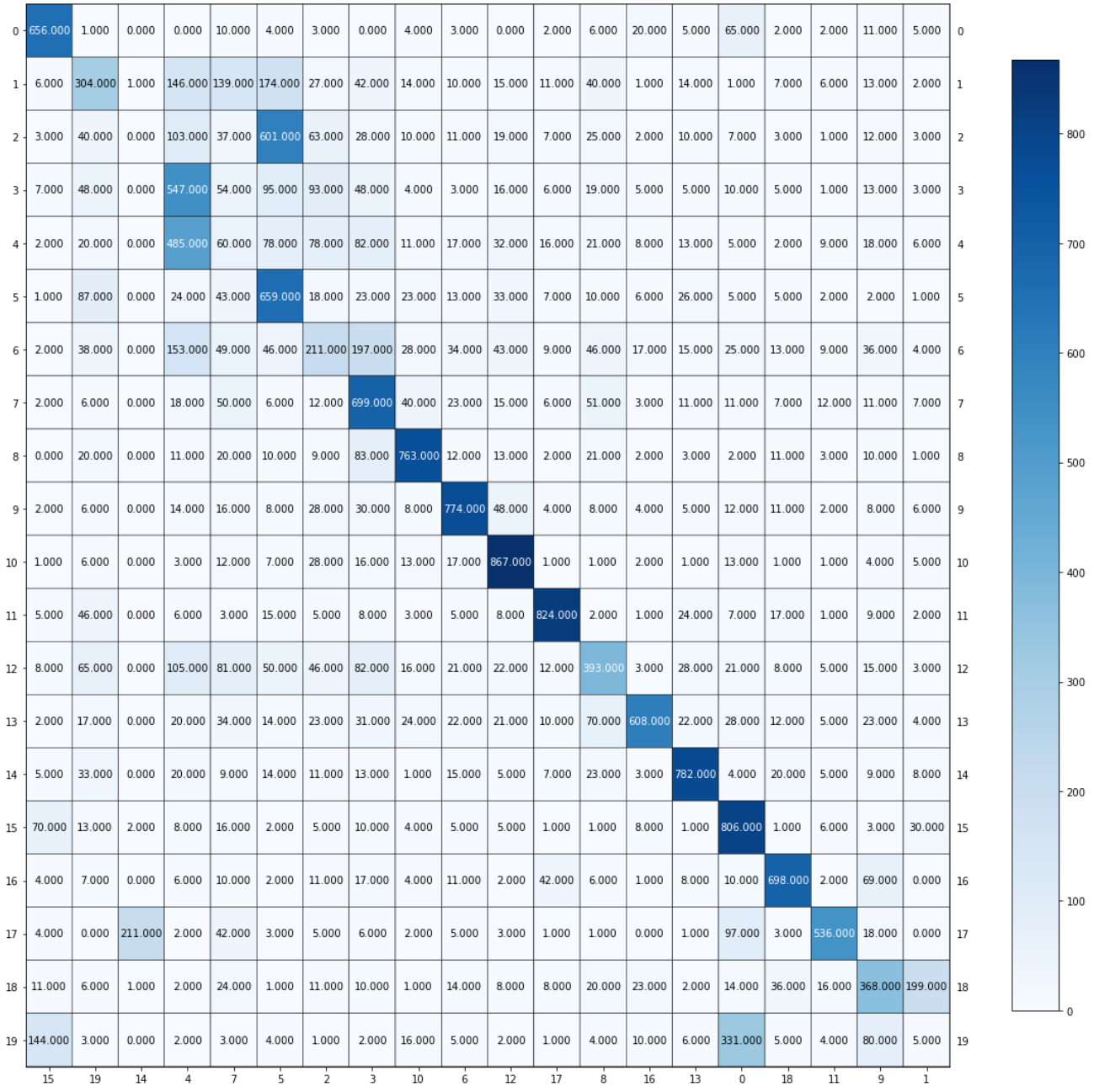
In [46]:
```
inputs_umap = umap.UMAP(n_components = 3, metric = 'cosine').fit_transform(inputs_tfidf)

agglo = AgglomerativeClustering(n_clusters = len(target_names), linkage = 'ward')
preds = agglo.fit_predict(inputs_umap)
get_cluster_metrics(labels, preds)
```

Out[46]:

	metric	score
0	homogeneity_score	0.495834
1	completeness_score	0.510624
2	v_measure_score	0.503120
3	adjusted_rand_score	0.391852
4	adjusted_mutual_info_score	0.501491

In [47]:

```
cm = contingency_matrix(labels, preds)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, size=(15,15))
```



Using 'single' linkage:

The single-linkage clustering criterion determines the distance between two clusters as the smallest distance between any pair of points, one coming from each of the clusters. Thus, the distance between two clusters A and B is given by:

$$d_{AB} = min\{\|a - b\|^2 : a \in A, b \in B\}$$

At each stage the two closest clusters in this regard are merged. Since all the distances between points in a cluster are not considered in making the merger decision, this clustering generally works only in the case of long 'chain' like clusters.

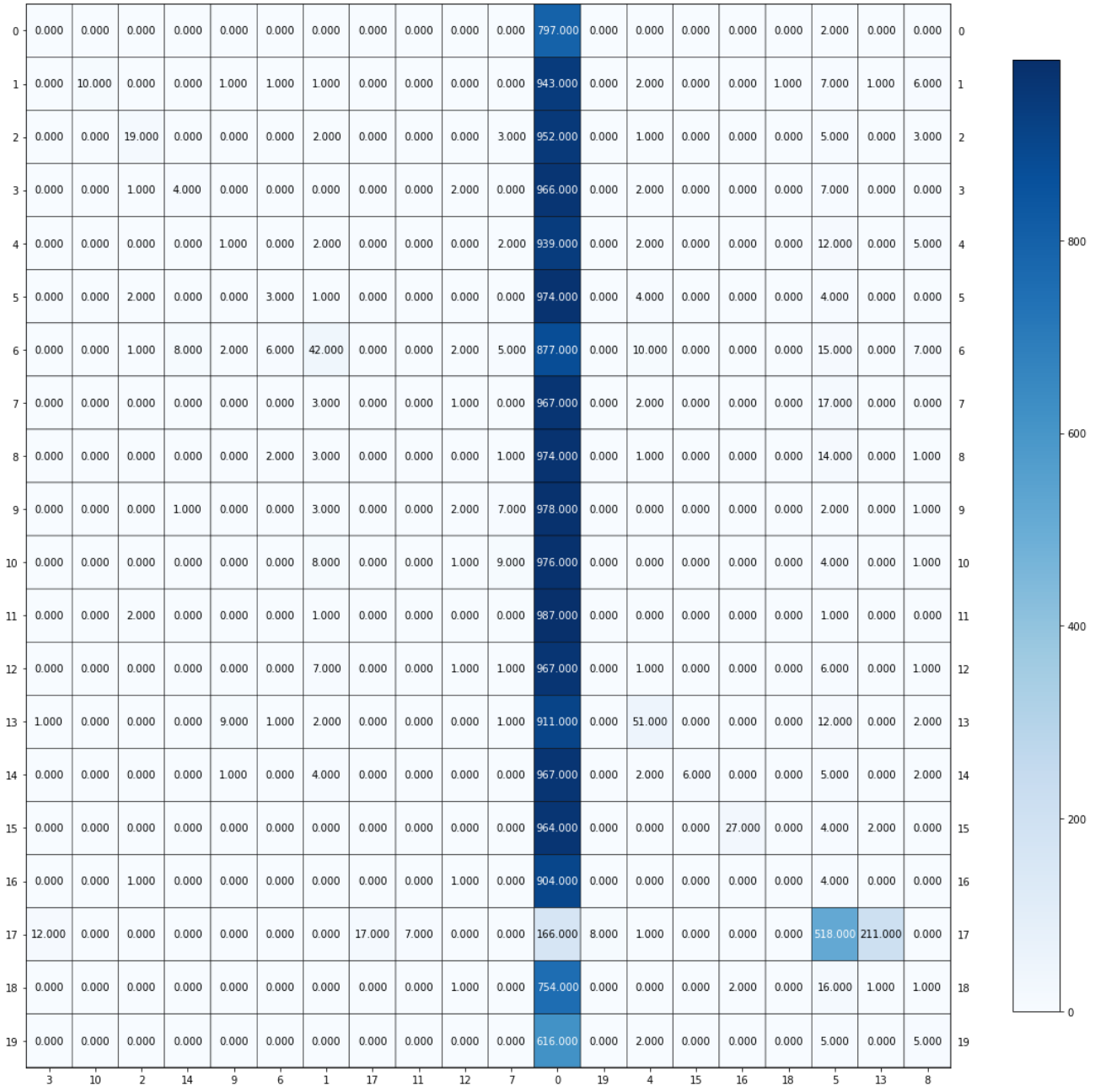
Since our document clusters do not have this representation (according to the UMAP representation above), single-linkage criterion has very poor behavior in clustering the given dataset.

In [48]:
```
agglo = AgglomerativeClustering(n_clusters = len(target_names), linkage = 'single')
preds = agglo.fit_predict(inputs_umap)
get_cluster_metrics(labels, preds)
```

Out[48]:

	metric	score
0	homogeneity_score	0.053183
1	completeness_score	0.435936
2	v_measure_score	0.094801
3	adjusted_rand_score	0.005671
4	adjusted_mutual_info_score	0.089461

In [49]:
```
cm = contingency_matrix(labels, preds)
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, size=(15,15))
```



14.

DBSCAN and HDBSCAN:

They are density based clustering approachs. They form clusters based on higly dense areas and hence could take any shape. The advantage of both methods is that in forming clusters from given data points, they also assign some points to be noise. This is a useful feature, especially in large nosiy datasets. The noisy points are those that are in regions of extremely sparse density (ie, there are no neighboring data points close to these noisy points).

The noisy points of the dataset are given the label -1

The basic principle is in identifying core samples (those that have at least a given number of neighbors within a set distance). All core samples that are close to each other than the set distance and their neighbors form a clusters. Points that do not belong to any cluster are labelled as noise (with the label -1).

REF:

Load HDBSCAN: https://pypi.org/project/hdbscan/

Link: https://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html?highlight=dbscan#dbscan

Link: https://www.datanovia.com/en/lessons/dbscan-density-based-clustering-essentials/#:~:text=Two%20important%20parameters%20are%20required,neighbors%20within%20%E2%80%9Ceps%E2%8(

Link:https://towardsdatascience.com/machine-learning-clustering-dbscan-determine-the-optimal-value-for-epsilon-eps-python-example-3100091cfbc#:~:text=DBSCAN%20works%20by%20determining%20whether,maximum%20distance%20between%20two%2

DBSCAN and HDBSCAN using min_cluster_size = 100

Experimenting on the hyperparameters:

DBSCAN

Let epsilon vary from 0.05 to 1, in steps of 0.05 and min_samples vary from 5 to 500 in steps of 10

In [88]:
```
from sklearn.cluster import DBSCAN
inputs_umap = umap.UMAP(n_components = 5, metric = 'cosine').fit_transform(inputs_tfidf)
```

In [57]:
```
eps_list = [x * 0.05 for x in range(1, 21)]
min_samples_list = list(range(5,500,10))

scores = []
for ep in eps_list:
    for min_samp in min_samples_list:
        preds = DBSCAN(eps = ep, min_samples = min_samp, n_jobs = -1).fit_predict(inputs_umap)
        row = get_cluster_metrics(labels, preds)['score'].tolist()
        row.append(ep)
        row.append(min_samp)
        scores.append(row)
        #scores.append(get_cluster_metrics(labels, preds)['score'].tolist())

titles = get_cluster_metrics(labels, preds)['metric'].tolist()
titles.append("Epsilon")
titles.append("min_samples")
#metrics = get_cluster_metrics(labels, preds)['metric'].tolist()
metrics_df = pd.DataFrame(scores, columns=titles)
metrics_df.to_excel('DBSCAN.xlsx')
```

For DBSCAN, the best model obtained had eps = 0.55 and min_samples = 115. The top performing models are:

In [37]:
```
top_DBs = pd.read_excel('DBSCAN.xlsx', header = 0, index_col = 0)
top_DBs.nlargest(5,'adjusted_mutual_info_score')
```

Out[37]:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score	Epsilon	mi
511	0.456490	0.583557	0.512261	0.195460	0.510766	0.55	
512	0.455876	0.584093	0.512081	0.182874	0.510587	0.55	
565	0.456140	0.581712	0.511329	0.204580	0.509931	0.60	
620	0.456960	0.579070	0.510819	0.214268	0.509423	0.65	
621	0.454159	0.579352	0.509173	0.207271	0.507768	0.65	

The best DBSCAN model, ran below suggests the number of clusters as: 16

In [69]:
```
preds = DBSCAN(eps = 0.55, min_samples = 115, n_jobs = -1).fit_predict(inputs_umap)
get_cluster_metrics(labels, preds)
```

Out[69]:

	metric	score
0	homogeneity_score	0.457614
1	completeness_score	0.576439
2	v_measure_score	0.510200
3	adjusted_rand_score	0.206263
4	adjusted_mutual_info_score	0.508709

In [70]:
```
print(np.unique(preds))
```

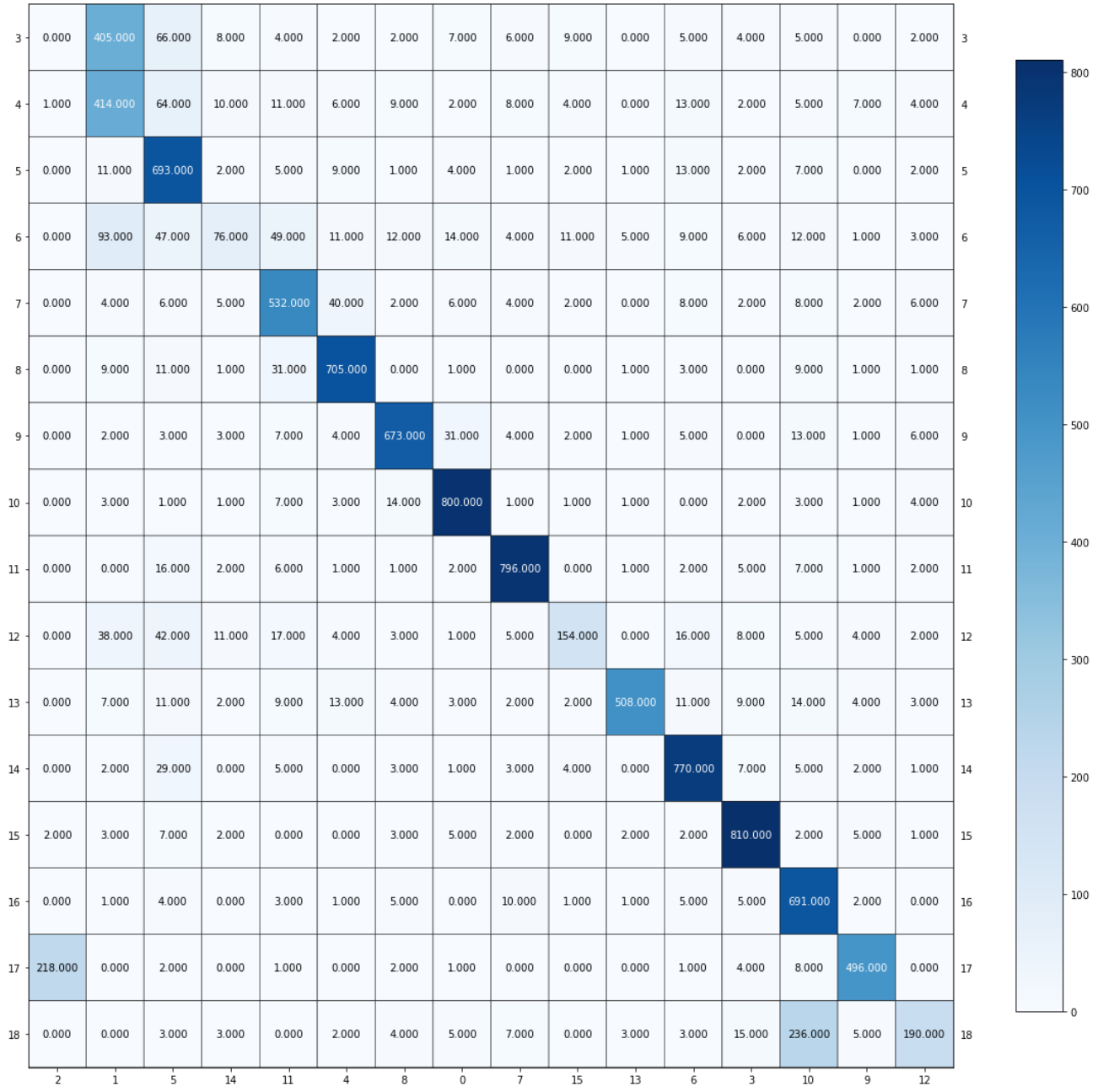
```
[-1  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
```

In [71]:
```
no_clusters = np.max(preds) + 1
print("Number of clusters as per best DBSCAN is", no_clusters)
```

```
Number of clusters as per best DBSCAN is 16
```

Plotting the permuted matrix:

In [72]:
```
cm = contingency_matrix(labels[preds!=-1], preds[preds!=-1]) #noise labels removed
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, size=(15,15))
```



Plotting the non-permuted matrix (for better analysis):

Included the noise points too.

In [73]:

```
cm = contingency_matrix(labels, preds)

cmap = sns.light_palette("blue", as_cmap=True)
x=pd.DataFrame(cm)
x=x.style.background_gradient(cmap=cmap)
display(x)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	67	0	0	0	711	3	4	4	1	1	1	4	0	2	1	0	0
1	479	8	60	0	3	4	378	8	9	4	4	6	5	1	0	2	2
2	372	2	92	0	2	2	483	7	3	3	2	2	4	1	0	8	2
3	457	7	405	0	4	2	66	5	6	2	0	5	4	2	0	8	9
4	403	2	414	1	2	6	64	13	8	9	7	5	11	4	0	10	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
5	235	4	11	0	2	9	693	13	1	1	0	7	5	2	1	2	2
6	622	14	93	0	6	11	47	9	4	12	1	12	49	3	5	76	11
7	363	6	4	0	2	40	6	8	4	2	2	8	532	6	0	5	2
8	223	1	9	0	0	705	11	3	0	0	1	9	31	1	1	1	0
9	239	31	2	0	0	4	3	5	4	673	1	13	7	6	1	3	2
10	157	800	3	0	2	3	1	0	1	14	1	3	7	4	1	1	1
11	149	2	0	0	5	1	16	2	796	1	1	7	6	2	1	2	0
12	674	1	38	0	8	4	42	16	5	3	4	5	17	2	0	11	154
13	388	3	7	0	9	13	11	11	2	4	4	14	9	3	508	2	2
14	155	1	2	0	7	0	29	770	3	3	2	5	5	1	0	0	4
15	151	5	3	2	810	0	7	2	2	3	5	2	0	1	2	2	0
16	181	0	1	0	5	1	4	5	10	5	2	691	3	0	1	0	1
17	207	1	0	218	4	0	2	1	0	2	496	8	1	0	0	0	0
18	299	5	0	0	15	2	3	3	7	4	5	236	0	190	3	3	0
19	173	0	0	0	370	2	3	9	0	2	3	57	2	5	2	0	0

Column 0 has points from all ground labels and hence indicates the noisy points.

Cluster (read column) 2 has grouped categories (read rows) 3 and 4. These categories are IBM hardware and Mac hardware and hence the model clustered these very similar labels together.

Another cluster having highly similar topics is cluster (column) 4. It has most of the points from categories (rows) 0, 15, and 19. These categories pertain to atheism, christianity, and miscellaneous religions. The closeness between the topics made it difficult for the model to make three clusters out of them.

Custer 6 has grouped categories 1, 2, and 5 which are graphics, os ms-windows, and windows.x.

In [66]:
```
conda install -c conda-forge hdbscan
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

## Package Plan ##

  environment location: C:\Users\sriva\anaconda3

  added / updated specs:
    - hdbscan


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    hdbscan-0.8.26             |   py38h347fdf6_3         506 KB  conda-forge
    ------------------------------------------------------------
                                           Total:         506 KB

The following NEW packages will be INSTALLED:

  hdbscan            conda-forge/win-64::hdbscan-0.8.26-py38h347fdf6_3


Downloading and Extracting Packages

hdbscan-0.8.26       | 506 KB    |            |   0%
```

```
hdbscan-0.8.26        | 506 KB    | 3           |   3%
hdbscan-0.8.26        | 506 KB    | ########## | 100%
hdbscan-0.8.26        | 506 KB    | ########## | 100%
Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done

Note: you may need to restart the kernel to use updated packages.
```

HDBSCAN

In [89]:
```
import hdbscan
```

In [13]:
```
clust_eps_list = [x * 0.05 for x in range(1, 21)]
min_samples_list = list(range(5,500,10))

scores = []
for ep in clust_eps_list:
    for min_samp in min_samples_list:
        preds = hdbscan.HDBSCAN(min_cluster_size=100, min_samples = min_samp, cluster_selection_epsilon =
        row = get_cluster_metrics(labels, preds)['score'].tolist()
        row.append(ep)
        row.append(min_samp)
        scores.append(row)
        #scores.append(get_cluster_metrics(labels, preds)['score'].tolist())

titles = get_cluster_metrics(labels, preds)['metric'].tolist()
titles.append("Cluster_sel_Epsilon")
titles.append("min_samples")
#metrics = get_cluster_metrics(labels, preds)['metric'].tolist()
metrics_df = pd.DataFrame(scores, columns=titles)
metrics_df.to_excel('HDBSCAN.xlsx')
```

HDBSCAN suggests the best model to be obtained with min_samples = 215, and cluster_selection_epsilon = 0.05. The top performing models are:

In [80]:
```
top_HDBs = pd.read_excel('HDBSCAN.xlsx', header = 0, index_col = 0)
top_HDBs.nlargest(5,'adjusted_mutual_info_score')
```

Out[80]:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score	Cluster_sel_E
21	0.35603	0.572526	0.43904	0.160122	0.437981	
71	0.35603	0.572526	0.43904	0.160122	0.437981	
121	0.35603	0.572526	0.43904	0.160122	0.437981	
171	0.35603	0.572526	0.43904	0.160122	0.437981	
221	0.35603	0.572526	0.43904	0.160122	0.437981	

The best HDBSCAN, ran below suggests the number of clusters as: 9 (shown below)

In [90]:
```
preds = hdbscan.HDBSCAN(min_cluster_size=100, min_samples = 215, cluster_selection_epsilon = 0.05, core_d
get_cluster_metrics(labels, preds)
```

Out[90]:

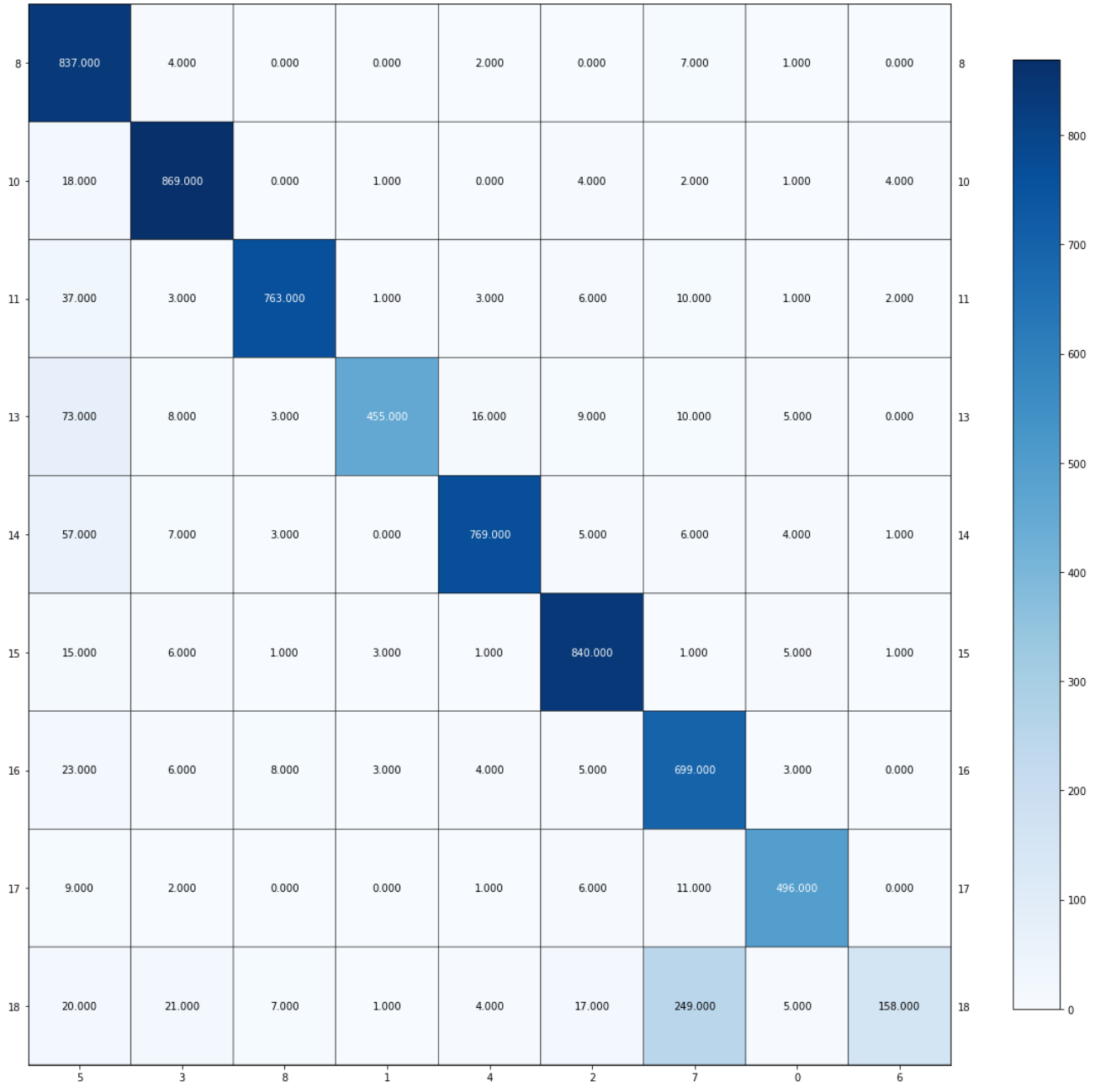
	metric	score
0	homogeneity_score	0.354008
1	completeness_score	0.566701
2	v_measure_score	0.435787
3	adjusted_rand_score	0.161766

	metric	score
4	adjusted_mutual_info_score	0.434725

In [91]:
```
print("The unique labels in case of best HDBSCAN model are:", np.unique(preds))
print("The number of clusters as per the best HDBSCAN model is:", np.max(preds)+1)
```

```
The unique labels in case of best HDBSCAN model are: [-1  0  1  2  3  4  5  6  7  8]
The number of clusters as per the best HDBSCAN model is: 9
```

In [92]:
```
cm = contingency_matrix(labels[preds!=-1], preds[preds!=-1])
rows, cols = linear_sum_assignment(cm, maximize=True)
plot_mat(cm[rows[:, np.newaxis], cols], xticklabels=cols, yticklabels=rows, size=(15,15))
```



Plotting the non-permuted matrix (for better analysis)

In [93]:
```
cm = contingency_matrix(labels, preds)

cmap = sns.light_palette("blue", as_cmap=True)
```

```
x=pd.DataFrame(cm)
x=x.style.background_gradient(cmap=cmap)
display(x)
```

	0	1	2	3	4	5	6	7	8	9
0	55	1	6	715	1	4	10	2	4	1
1	337	5	1	4	17	3	594	1	4	7
2	200	1	0	1	13	8	754	0	4	4
3	208	0	2	5	10	4	741	2	4	6
4	238	9	6	1	17	13	661	3	6	9
5	184	1	1	2	10	8	778	0	3	1
6	514	2	14	7	29	10	385	2	10	2
7	271	1	0	2	12	4	685	6	7	2
8	145	1	0	0	4	2	837	0	7	0
9	208	2	0	4	716	5	40	5	11	3
10	100	1	1	4	869	0	18	4	2	0
11	165	1	1	6	3	3	37	2	10	763
12	432	5	4	7	14	24	489	1	5	3
13	411	5	455	9	8	16	73	0	10	3
14	135	4	0	5	7	769	57	1	6	3
15	124	5	3	840	6	1	15	1	1	1
16	159	3	3	5	6	4	23	0	699	8
17	415	496	0	6	2	1	9	0	11	0
18	293	5	1	17	21	4	20	158	249	7
19	179	3	3	366	4	9	10	4	50	0

The first column indicates noise as it has points from all ground labels.

Because the model forms only 9 clusters, all ‘computer’ related categories were grouped together (cluster 6). A surprising find is that category (row) 12 which deals with ‘electronics’ is also included in the same cluster having the computer topics. A major defect in this clustering is the grouping of ‘misc.forsale’, ‘rec.autos’, and ‘rec.motorcycles’ in the same cluster 6.

Other groups are appropriately clustered.

In [ ]:

libraries, helper functions and constants

Here are all the libraries that we used for question 16 and some untility functions such as get_cluster_metrics which output the metrics.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from umap import UMAP
from sklearn.decomposition import NMF
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
from hdbscan import HDBSCAN
from scipy.optimize import linear_sum_assignment
from sklearn.metrics.cluster import (
    contingency_matrix,
    homogeneity_score,
    v_measure_score,
    completeness_score,
    adjusted_rand_score,
    adjusted_mutual_info_score
)
from plotmat import plot_mat

def get_cluster_metrics(y_true, y_pred, metrics=None):
    if not metrics:
        metrics = [
            homogeneity_score,
            completeness_score,
            v_measure_score,
            adjusted_rand_score,
            adjusted_mutual_info_score
        ]
    d = {}
    for m in metrics:
        d[m.__name__] = m(y_true, y_pred)
    df = pd.DataFrame(d, index=[0]).T
    df.reset_index(inplace=True)
    df.rename(columns={'index': 'metric', 0: 'score'}, inplace=True)
    return df

RANDOM_SEED = 42
```

load in data and transform to TF-IDF vectors

We load the data into a pandas dataframe from csv file, and uses factorize function to assign unique ID to each category. category_id_df dataframe holds category name and corresponding ID, and labels dataframe holds the label corresponding to training data.

In [2]:
```
df = pd.read_csv("./BBC_News_Train.csv")
df['category_id'], _ = df['Category'].factorize()
category_id_df = df[['Category', 'category_id']].drop_duplicates()
labels = df.category_id
```

Here we uses Term Frequency-Inverse Document Frequency (TF-IDF) metric to build representation vectors for training.

In [3]:
```
tfidf = TfidfVectorizer(min_df=5, stop_words='english')
tfidf_data = tfidf.fit_transform(df.Text)
```

dimension reduction (Truncated SVD, NMF and UMAP with K-mean)

We evaluated three different dimension reduction method (Truncated SVD, NMF and UMAP) with components number ranging from 1 to 800. We evaluate the effectiveness of the dimension reduction using K-mean and clustering algorithm. The performance resulting from each dimension reduction data are shown in tables below.

In [4]:
```
km = KMeans(
    n_clusters=category_id_df.Category.count(),
    random_state=0,
    max_iter=5000,
    n_init=50
)

n_components = [1, 2, 3, 5, 10, 20, 50, 100, 300, 500, 800, 1000]
```

Truncated SVD

In [5]:
```
scores = []
for r in n_components:
    input_svd = TruncatedSVD(
        n_components=r, random_state=RANDOM_SEED).fit_transform(tfidf_data)
    preds = km.fit_predict(input_svd)
    scores.append(get_cluster_metrics(labels, preds)['score'].tolist())
metrics = get_cluster_metrics(labels, preds)['metric'].tolist()
svd_metrics_df = pd.DataFrame(scores, columns=metrics, index=n_components)
svd_metrics_df
```

Out[5]:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.094331	0.104362	0.099094	0.049817	0.095885
2	0.303752	0.346610	0.323769	0.193236	0.321323
3	0.448852	0.502422	0.474129	0.355048	0.472247
5	0.636503	0.685090	0.659903	0.572929	0.658711
10	0.633928	0.707318	0.668616	0.543747	0.667433
20	0.616071	0.693878	0.652664	0.514318	0.651419
50	0.635522	0.718284	0.674373	0.536805	0.673204
100	0.620452	0.698453	0.657146	0.517965	0.655918
300	0.715419	0.757440	0.735830	0.662944	0.734912
500	0.782276	0.799019	0.790559	0.780867	0.789844
800	0.721031	0.756067	0.738133	0.689256	0.737227
1000	0.632077	0.710491	0.668994	0.532924	0.667809

NMF

In [6]:
```
scores = []
for r in n_components:
    inputs_nmf = NMF(n_components=r, init='random',
                     random_state=RANDOM_SEED).fit_transform(tfidf_data)
    preds = km.fit_predict(inputs_nmf)
    scores.append(get_cluster_metrics(labels, preds)['score'].tolist())
metrics = get_cluster_metrics(labels, preds)['metric'].tolist()
nmf_metrics_df = pd.DataFrame(scores, columns=metrics, index=n_components)
nmf_metrics_df
```

```
/home/boyuan/.local/lib/python3.8/site-packages/sklearn/decomposition/_nmf.py:1090: ConvergenceWarning: Max
imum number of iterations 200 reached. Increase it to improve convergence.
  warnings.warn("Maximum number of iterations %d reached. Increase it to"
/home/boyuan/.local/lib/python3.8/site-packages/sklearn/decomposition/_nmf.py:1090: ConvergenceWarning: Max
imum number of iterations 200 reached. Increase it to improve convergence.
  warnings.warn("Maximum number of iterations %d reached. Increase it to"
/home/boyuan/.local/lib/python3.8/site-packages/sklearn/decomposition/_nmf.py:1090: ConvergenceWarning: Max
imum number of iterations 200 reached. Increase it to improve convergence.
  warnings.warn("Maximum number of iterations %d reached. Increase it to"
```

Out[6]:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.094331	0.104362	0.099094	0.049817	0.095885
2	0.247705	0.308859	0.274922	0.127401	0.272187
3	0.477529	0.493823	0.485539	0.343355	0.483772
5	0.663295	0.708868	0.685325	0.612784	0.684226
10	0.472978	0.627064	0.539229	0.319264	0.537447
20	0.254254	0.456657	0.326642	0.074724	0.323695
50	0.143036	0.427838	0.214395	0.049344	0.210228
100	0.068191	0.370604	0.115188	0.011561	0.109439
300	0.084514	0.349946	0.136147	0.015954	0.131116
500	0.032743	0.290324	0.058849	0.005426	0.052556
800	0.012738	0.241210	0.024199	0.001587	0.016873
1000	0.006937	0.210721	0.013432	0.000603	0.006606

UMAP

For UMAP, we tested on both Eucledian and Cosine distance, and the result are shown below

euclidean

In [7]:

```
scores = []
for n in n_components:
    umapfit = UMAP(n_components = n, metric = 'euclidean')
    inputs_umap = umapfit.fit_transform(tfidf_data)
    preds = km.fit_predict(inputs_umap)
    get_cluster_metrics(labels, preds)
    scores.append(get_cluster_metrics(labels, preds)['score'].tolist())
metrics = get_cluster_metrics(labels, preds)['metric'].tolist()
metrics_df = pd.DataFrame(scores, columns=metrics, index=n_components)
metrics_df
```

Out[7]:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.648914	0.652242	0.650574	0.682402	0.649391
2	0.774596	0.774988	0.774792	0.807148	0.774031
3	0.772620	0.773033	0.772827	0.806810	0.772059
5	0.774802	0.775294	0.775048	0.809163	0.774288
10	0.768688	0.769228	0.768958	0.801502	0.768178
20	0.764969	0.765726	0.765348	0.797540	0.764555
50	0.761241	0.762329	0.761785	0.795600	0.760980
100	0.767745	0.768666	0.768206	0.800770	0.767422
300	0.763184	0.764151	0.763667	0.798226	0.762869

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
500	0.765523	0.766331	0.765927	0.800668	0.765136
800	0.765306	0.766059	0.765683	0.797566	0.764891
1000	0.768891	0.769484	0.769187	0.802630	0.768408

cosine

In [8]:
```
scores = []
for n in n_components:
    umapfit = UMAP(n_components = n, metric = 'cosine')
    inputs_umap = umapfit.fit_transform(tfidf_data)
    preds = km.fit_predict(inputs_umap)
    get_cluster_metrics(labels, preds)
    scores.append(get_cluster_metrics(labels, preds)['score'].tolist())
metrics = get_cluster_metrics(labels, preds)['metric'].tolist()
metrics_df = pd.DataFrame(scores, columns=metrics, index=n_components)
metrics_df
```

Out[8]:

	homogeneity_score	completeness_score	v_measure_score	adjusted_rand_score	adjusted_mutual_info_score
1	0.684861	0.683499	0.684179	0.693706	0.683114
2	0.697573	0.695054	0.696311	0.702069	0.695288
3	0.760475	0.762211	0.761342	0.794154	0.760535
5	0.763035	0.763932	0.763483	0.797935	0.762684
10	0.766929	0.767975	0.767451	0.802639	0.766666
20	0.757243	0.758450	0.757846	0.791636	0.757028
50	0.759846	0.760991	0.760418	0.794313	0.759609
100	0.763592	0.764522	0.764057	0.799254	0.763260
300	0.764780	0.765805	0.765292	0.799528	0.764499
500	0.770041	0.770846	0.770443	0.804797	0.769668
800	0.756697	0.757925	0.757311	0.790345	0.756490
1000	0.766731	0.767452	0.767091	0.804319	0.766305

As we can see, Truncated SVD achieves the best score of 0.78 to 0.80 in all scoring categories; UMAP score is very close to that of Truncated SVD, and even surpass it in adjusted Rand score; NMF perform the worst, which might be caused by the non-negative weighing constrain making it fail to capture matrix.

clustering (K-mean, Agglomerative, DBSCAN, HDBSCAN)

Here we use TruncatedSVD with 500 component for K-mean and cosine distance UMAP with 1000 component for Agglomerative, DBSCAN and HDBSCAN (these three work well with UMAP data). The results are shown in the tables below.

In [9]:
```
svd_train = TruncatedSVD(
    n_components=500, random_state=RANDOM_SEED).fit_transform(tfidf_data)

umap_train = UMAP(
    n_components = 1000, metric = 'cosine').fit_transform(tfidf_data)
```

K-mean

In [10]:
```
preds = km.fit_predict(svd_train)

get_cluster_metrics(labels, preds)
```

Out[10]:

	metric	score
0	homogeneity_score	0.782276
1	completeness_score	0.799019
2	v_measure_score	0.790559
3	adjusted_rand_score	0.780867
4	adjusted_mutual_info_score	0.789844

Agglomerative

In [11]:
```
preds = AgglomerativeClustering(n_clusters = category_id_df.Category.count(), linkage = 'ward').fit_predi

get_cluster_metrics(labels, preds)
```

Out[11]:

	metric	score
0	homogeneity_score	0.775997
1	completeness_score	0.776436
2	v_measure_score	0.776217
3	adjusted_rand_score	0.813692
4	adjusted_mutual_info_score	0.775461

DBSCAN

For DBSCAN, we tested Epsilon ranging from 1 to 21 and min sample ranging from 5 to 200

In [12]:
```
eps_list = [x * 0.05 for x in range(1, 21)]
min_samples_list = list(range(5,200,10))

scores = []
for ep in eps_list:
    for min_samp in min_samples_list:
        preds = DBSCAN(eps = ep, min_samples = min_samp, n_jobs = -1).fit_predict(umap_train)
        row = get_cluster_metrics(labels, preds)['score'].tolist()
        row.append(ep)
        row.append(min_samp)
        scores.append(row)

titles = get_cluster_metrics(labels, preds)['metric'].tolist()
titles.append("Epsilon")
titles.append("min_samples")
metrics_df = pd.DataFrame(scores, columns=titles)
metrics_df.to_excel('DBSCAN.xlsx')
```

Detailed result can be found in DBSCAN.xlsx, here I picked the one with highest score and show it below

In [13]:
```
preds = DBSCAN(eps = 0.95, min_samples = 75, n_jobs = -1).fit_predict(umap_train)

get_cluster_metrics(labels, preds)
```

Out[13]:

	metric	score

	metric	score
0	homogeneity_score	0.722026
1	completeness_score	0.647094
2	v_measure_score	0.682510
3	adjusted_rand_score	0.669950
4	adjusted_mutual_info_score	0.681241

HDBSCAN

For HDBSCAN, we set min cluster size to 100 and tested min cluster size ranging from 1 to 21 and min sample ranging from 5 to 500

In [14]:
```
clust_eps_list = [x * 0.05 for x in range(1, 21)]
min_samples_list = list(range(5,500,10))

scores = []
for ep in clust_eps_list:
    for min_samp in min_samples_list:
        preds = HDBSCAN(min_cluster_size=100, min_samples = min_samp, cluster_selection_epsilon = ep, core
        row = get_cluster_metrics(labels, preds)['score'].tolist()
        row.append(ep)
        row.append(min_samp)
        scores.append(row)

titles = get_cluster_metrics(labels, preds)['metric'].tolist()
titles.append("Cluster_sel_Epsilon")
titles.append("min_samples")
metrics_df = pd.DataFrame(scores, columns=titles)
metrics_df.to_excel('HDBSCAN.xlsx')
```

Detailed result can be found in HDBSCAN.xlsx, here I picked the one with highest score and show it below

In [15]:
```
preds = HDBSCAN(min_cluster_size=100, min_samples = 15, cluster_selection_epsilon = 0.05, core_dist_n_jobs

get_cluster_metrics(labels, preds)
```

Out[15]:

	metric	score
0	homogeneity_score	0.554131
1	completeness_score	0.773971
2	v_measure_score	0.645856
3	adjusted_rand_score	0.565880
4	adjusted_mutual_info_score	0.644796

As we can see, K-mean is the best performing algorithm. It outperforms DBSCAN/HDBSCAN and slightly outperforms Agglomerative by a very slim margin. We believe this is caused by the sparse nature of the dataset, which favors K-mean over DBSCAN/HDBSCAN. Moreover, K-mean and Agglomerative have the advantage of knowing the exact number of clusters, while DBSCAN/HDBSCAN doesn't.