Project 2: Data Representations and Clustering

Large-Scale Data Mining: Models and Algorithms

Ananya Deepak Deoghare, Hariram Veeramani, Madhav Sankar Krishnakumar

005627628, 005528336, 405692669

ECE 219 Winter 2022

Question 1

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

Dimensions: (7882, 23522)
The dataset given to us is the "20 Newsgroups dataset". For this question, we are considering only 8 of these 20 classes. We are converting this into a 2 class clustering problem and hence we are merging all the classes related to 'comp' into 1 and those related to 'rec' into another.

Feature Extraction:

- Remove headers and footers while importing the dataset.
- No stemming or lemmatization applied.
- Use CountVectorizer() to convert the text into a bag-of-words matrix. Remove English stop-words. Set min_df to 3. This removes all the rare words that appear in less than 3 documents.
- Convert the matrix of token counts to TF-IDF matrix using TfidfTransformer. The TF-IDF matrix highlights the discriminating words in each document against the more common words and this can help while clustering to differentiate the various classes.

```
# TF-IDF Code
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']
dataset = fetch_20newsgroups(subset = 'all', categories = categories,/
    shuffle = True, random_state = 0,remove=('headers','footers'))

count_vec = CountVectorizer(stop_words="english", min_df=3)
tfidf_tranformer = TfidfTransformer(use_idf=True)

word_count_vec = count_vec.fit_transform(dataset.data)
word_count_vec_tf = tfidf_tranformer.fit_transform(word_count_vec)
word_count_array = word_count_vec_tf.toarray()
word_counts = pd.DataFrame(data=word_count_array,/
        columns = count_vec.get_feature_names())

y_output = []
for label in dataset.target:
    if(label < 4):
        y_output.append(0)
    else:
        y_output.append(1)

print(word_counts.shape)
```

Question 2

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

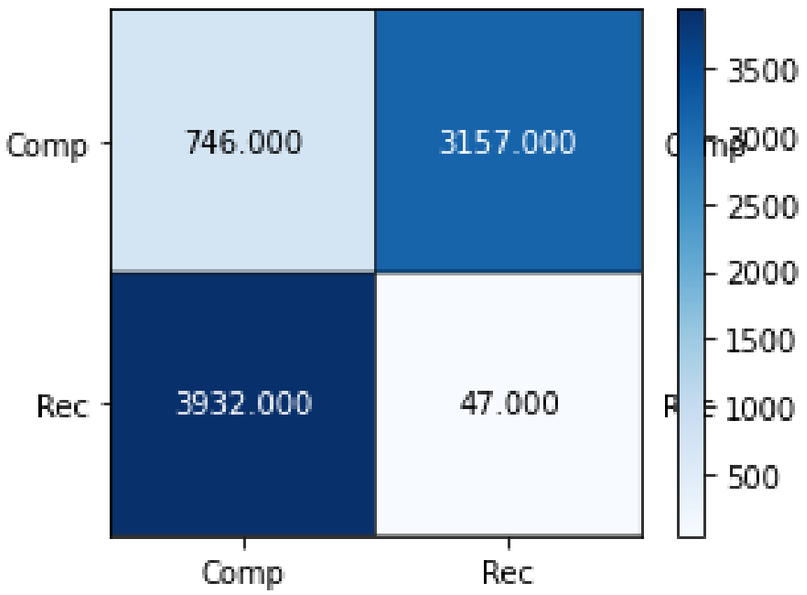

Figure 1: K-means Contingency Matrix Table

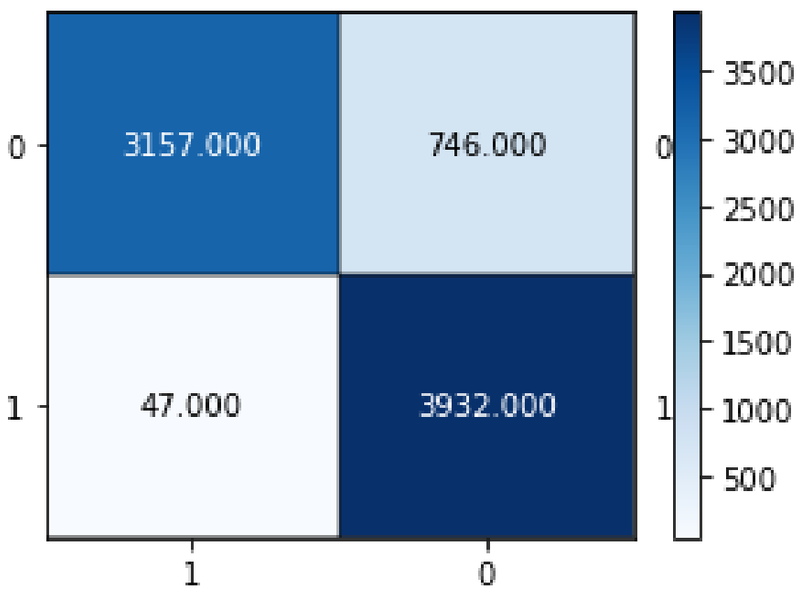

Figure 2: K-means Permuted Contingency Matrix Table

Process of grouping data that shares similarity in the feature space without using annotated classes is defined as Clustering. Each value of the contingency matrix, M is given by M_{ij} such that i is the true class and j is the predicted class/cluster. As can be observed, 3157 documents of Comp class were assigned to a single cluster which we can assume is the comp cluster. Similarly, 3932 of the rec documents are clustered together. As can be seen the comp cluster has 47 of the rec documents and rec cluster has 746 of the comp class documents.

Observing the contingency matrix, we can conclude that the clustering algorithm has worked accurately in categorizing majority of the sample points to their corresponding clusters.

Does the contingency matrix have to be square-shaped?

As far as this question is concerned, Contingency Matrix has to be square shaped. This is because we have 2 classes and k-means algorithm has 2 clusters. Contingency matrix is a broader extension of Confusion matrix's idea, where we have rows showing the number of samples in ground truth clusters while columns depict the number of samples segregated to those clusters by the clustering algorithm. Counts for Accurate conclusion is represented by the diagonal entries.

But in a general sense, the rows of the contingency matrix depict the ground truth clusters/classes and columns depict the predicted clusters. So for instance, if our data has 20 classes and our clustering algorithm generates 30 clusters, the contingency matrix will have non-zero entries only in the 20*30 rectangle. Similarly if the clustering solution created only 10 clusters, it would have non-zero entries only in 20*10 rectangle. **As far as we have seen, we have always encountered the contingency matrix to be displayed as a square with 0 in the remaining entries.**

```
kmeans = KMeans(init='k-means++', max_iter=1000,/
n_clusters=2, n_init=30, random_state=0)
kmeans.fit(word_counts)
plot_mat(contingency_matrix(y_output, kmeans.labels_),/
size = (4,3), xticklabels = ['Comp','Rec'],yticklabels = ['Comp','Rec'])
```

Question 3

Report the 5 clustering measures explained in the introduction for Kmeans clustering.

- Homogeneity score : 0.579400
- Completeness score : 0.594446
- V-measure score : 0.586826
- Adjusted Rand Index score : 0.638007
- Adjusted mutual information score : 0.586788

We are asked to report 5 standard measures of clustering metrics when labels were given for the results in the last question.

Following is the list of metrics:

- Homogeneity: A clustering result satisfies homogeneity if all of its clusters contain only data points which belong to a single class. We observe that Information Entropy tends to get maximized When the sizes of the partitions are equal, and get minimized when some group within the partition takes up all the data points.
- Completeness: Completeness property of clustering results is when all data points that are members of a given class tend to be the elements of the same cluster.
- V-measure is a symmetric metric and is useful for measuring agreement between independent cluster assignment strategies on the same dataset.
- Adjusted Rand Index is a pairwise measure of similarity or correlation between assigned clusters and ground truth labels
- Adjusted mutual information score measures the mutual information between the cluster label distribution and the ground truth label distribution.

Homogeneity metric of 58% indicates that 42% of the samples on an average are not from the assigned corresponding ground truth label. Completeness of 60 % shows that 60 % of the samples for a given class label are part of the same cluster. V-measure metric value of 59% stands at the harmonic mean of both Homogeneity and completeness, being a symmetric metric which is independent of the absolute values of the labels. Other scores such as Adjust Rand-Index (ARI) and Adjusted Mutual Information depict how the clustering distribution is closely related to the ground truth label distribution considerably.

Question 4

Report the plot of the percentage of variance that the top r principle components retain v.s. r, for r = 1 to 1000

The x axis of the Fig 3. represents the values of r, and the y axis has the percentage of variance retained.
For a given r, Truncated SVD/LSI chooses the top r principal components in the descending order of the variance retained by the component. We notice that around 50% of the variance is covered by around 1000 features i.e. r = 1000. The plot's concavity suggests that as we increase the values of r, the variance also increases up to a point but the increase gets slower with larger r. For example, the difference between the variance explained for r=50 and r=100 is more than that of r=950 and r=1000. This makes sense as the components are chosen in decreasing order of their variance explanation ratio.

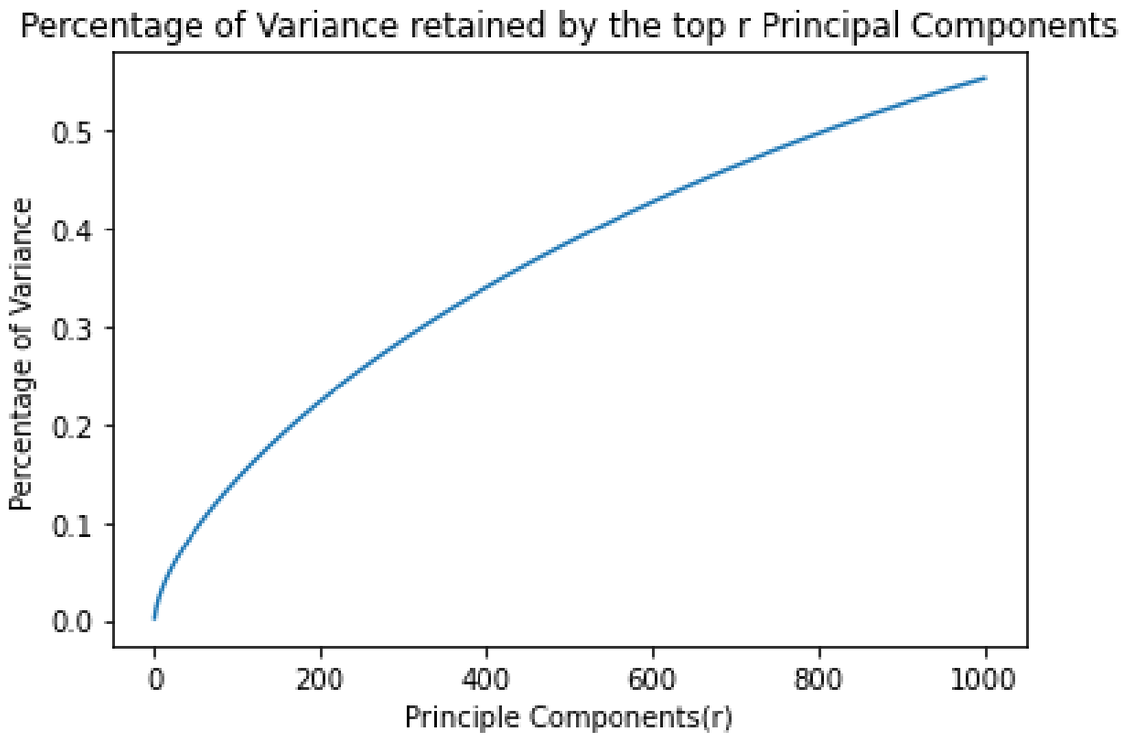

Figure 3: Percentage of Variance retained by top r Principal Components

Question 5

Let r be the dimension that we want to reduce the data to (i.e. n components). Try r = 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. Note: In the choice of r, there is a trade-off between the information preservation, and better performance of k-means in lower dimensions.

As we increase 'r' i.e., number of components, amount of semantic/ complex information and inherent patterns in data present for potential clustering increases. But this also causes the Clustering performance to drop as the critical metric for K-means clustering which is the euclidean distance converges to a constant value between all sample points(equidistant features).

Hence, it is important to pick 'r' that contains just enough information without being too large/vast, as K-means clustering performance drops once we enter into higher dimensions.

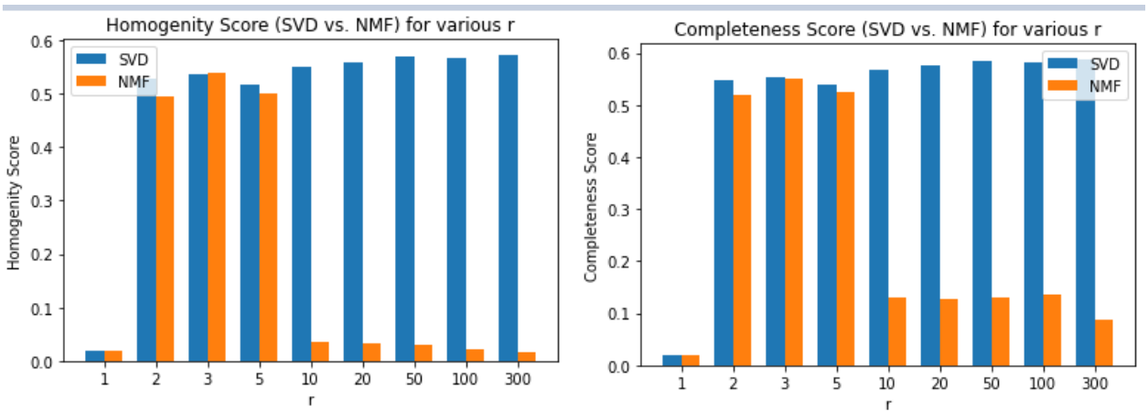

Figure 4: Homogeneity and Completeness Score vs r

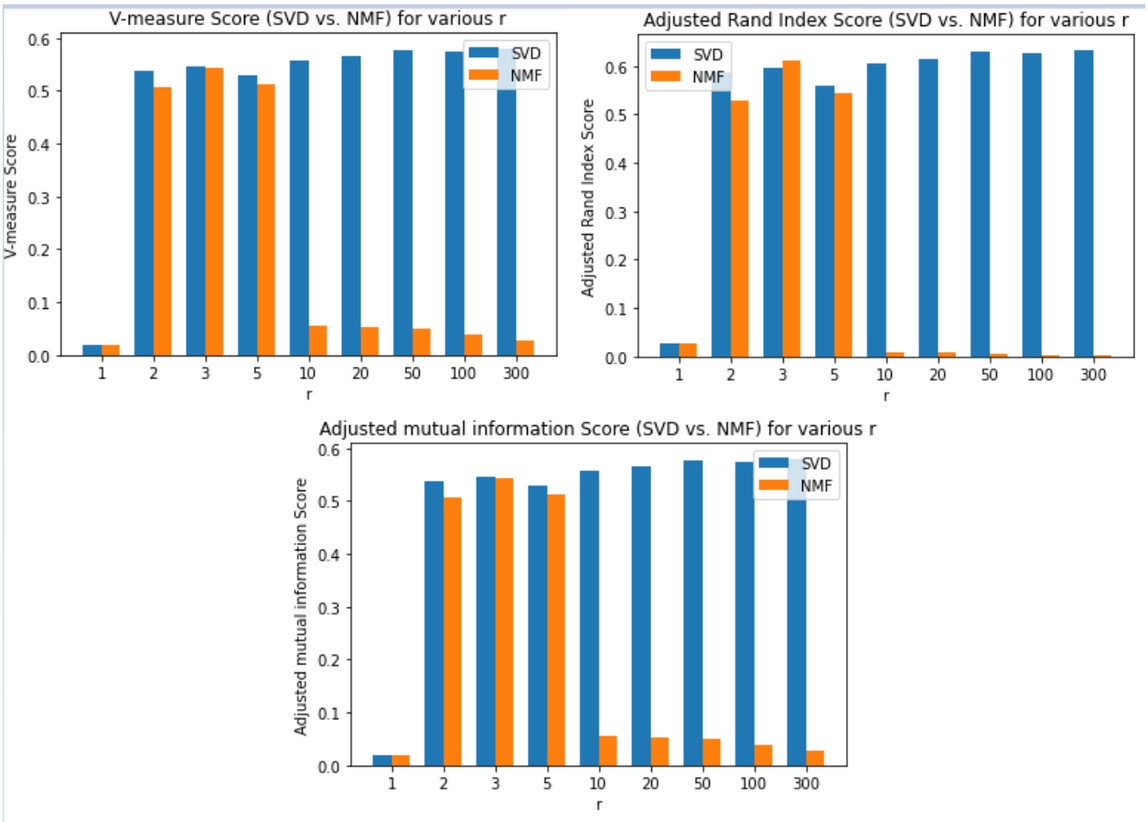

Figure 5: V-measure, ARI and AMI Scores vs r

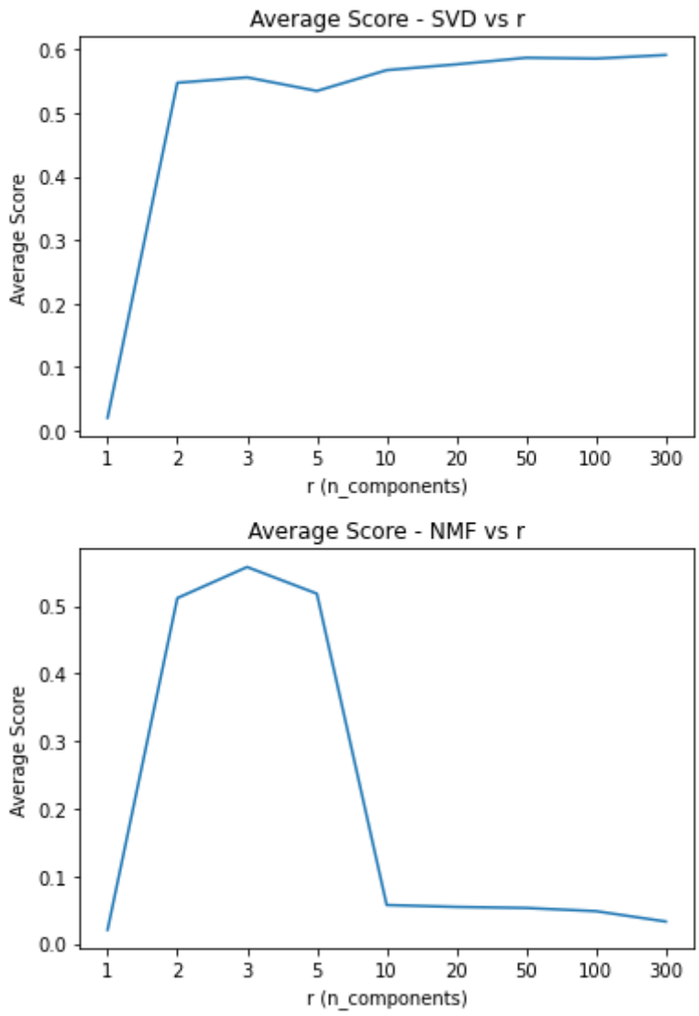

Figure 6: Average Scores vs r

```
Best in terms of Average score - SVD
Best r: 300
Best Score:0.5912913300386178

Best in terms of Average Rand Index score - SVD
Best r: 300
Best Score:0.6335553979860815

Best in terms of Average score - NMF
Best r: 3
Best Score:0.5578767772563564

Best in terms of Average Rand Index score - NMF
Best r: 3
Best Score:0.6123239707892058
```

There is a trade-off between the information preservation, and better performance of k-means in lower dimensions observed in case of NMF very clearly and r = 3 is the best choice considering the two factors. But in case of SVD we observe that the information preservation marginally outweights the poor performance of k-means in higher dimensions even at r = 300.

We can observe that for SVD r = 300 provides the best scores across all metrics and r = 3 gives best results for NMF. So for SVD even though, r = 300 provides the best score, the difference in the scores between r = 50 and for the higher dimensions are fairly negligible. Therefore it makes better sense to choose r = 50 as increasing the dimensions further doesn't provide much better results.

Question 6

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

- Clustering metrics do not increase uniformly with increasing values of r.
- Logically, one might think that increasing the r or the number of components should help the model identify even more higher order rich features leading to better clustering properties(visually even more separate clusters). As we increase 'r' i.e. number of components, amount of semantic/ complex information and inherent patterns in data which could be used for potential clustering increases.
- However, it is important to note that, as we enter into higher dimensions, this would make data become more sparse along each dimension.
- This would eventually cause the Euclidean distances on the projected higher dimension to converge to a nearly constant value between all sample points in that space.
- Hence, this effect of making all feature points equidistant without any normalization, seems to leave the Clustering algorithm helpless in figuring out centroids with sufficient inter-cluster distances amidst them.
- This effect is very clear in the NMF charts. But in case of SVD, we see that the performance continues to increase with increasing r, though only very marginally.

Now let us try to understand why only NMF shows the non-monotonic behavior:

- NMF only allows positive entries in the reduced-rank feature matrix. This is not the case for SVD and therefore SVD can represent higher dimensional feature matrix more accurately.
- SVD is more deterministic than NMF and considers the geometry in the feature space basis that is critical for clustering in higher dimensions.
- SVD sorts the features in the decreasing order of variance explanation. Therefore the more important features are higher in the hierarchy. Thus with increasing dimensions, there is a marginal increase in information with not much addition to the noise. Therefore, SVD's performance increases marginally when r is increased from 20 to 200, unlike NMF which displays a more non-monotonic behavior.

Question 7

Are these measures on average better than those computed in Question 3?

Question 3 (Without Dimensionality Reduction) measures:

```
Homogeneity score: 0.579400
Completeness score: 0.594446
V-measure score: 0.586826
Adjusted Rand Index score: 0.638007
Adjusted mutual information score: 0.586788
```

Question 5 Measures:
NMF:

```
Homogeneity score: 0.538405
Completeness score: 0.550210
V-measure score: 0.544244
Adjusted Rand Index score: 0.612324
Adjusted mutual information score: 0.544201
```

SVD (r = 50):

```
Homogeneity score: 0.568952
Completeness score: 0.583977
V-measure score: 0.576367
Adjusted Rand Index score: 0.629119
Adjusted mutual information score: 0.576327
```

SVD (r = 300):

```
Homogeneity score: 0.573359
Completeness score: 0.588206
V-measure score: 0.580688
Adjusted Rand Index score: 0.633555
Adjusted mutual information score: 0.580649
```

Measures reported for Question 5 are on an average very similar or slightly lower compared to the metrics that were found for Question 3.

- We know that while SVD has no restriction on the nature of values in the reduced-rank feature matrix, NMF allows only positive entries in the reduced rank feature matrix. This helps SVD to better represent the higher dimensional feature matrix, providing a deeper factorization with lower information loss. SVD is much more deterministic than NMF as it operates based on relevant geometric basis as opposed to NMF which neglects the feature space basis which is an important element for Higher-Dimensional clustering. Therefore, the NMF scores are much lesser than that of SVD and the case with no dimensionality reduction.

- Hence, comparing SVD vs No Dimensionality Reduction, from K means point of view, SVD with higher value of r is revealing very little additional information that could be useful for Clustering but it doesn't mask any important available information as well with excessive noise due to its structured organization of features.

- So SVD performance increases very marginally with increasing r beyond r = 50 and no dimensionality reduction can be considered as a very large r. Therefore, it makes sense that the scores without dimensionality reduction is very similar but marginally better than that of SVD with r = 50 or r = 300. This might be attributed to the less important features that were removed by SVD.

Question 8

Visualize the clustering results for:

- **SVD with your optimal choice of r for K-Means clustering;**
- **NMF with your choice of r for K-Means clustering.**

To recap, you can accomplish this by first creating the dense representations and then once again projecting these representations into a 2-D plane for visualization.

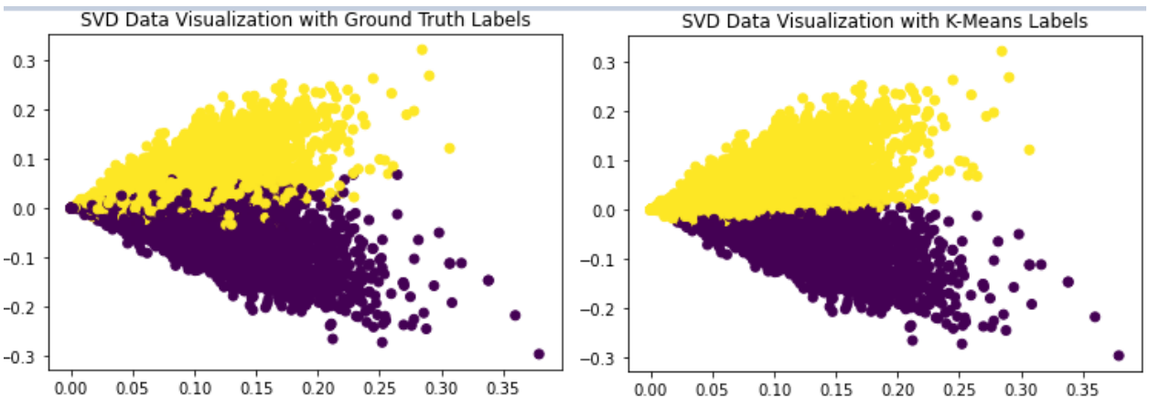

Figure 7: Plot of SVD feature matrix

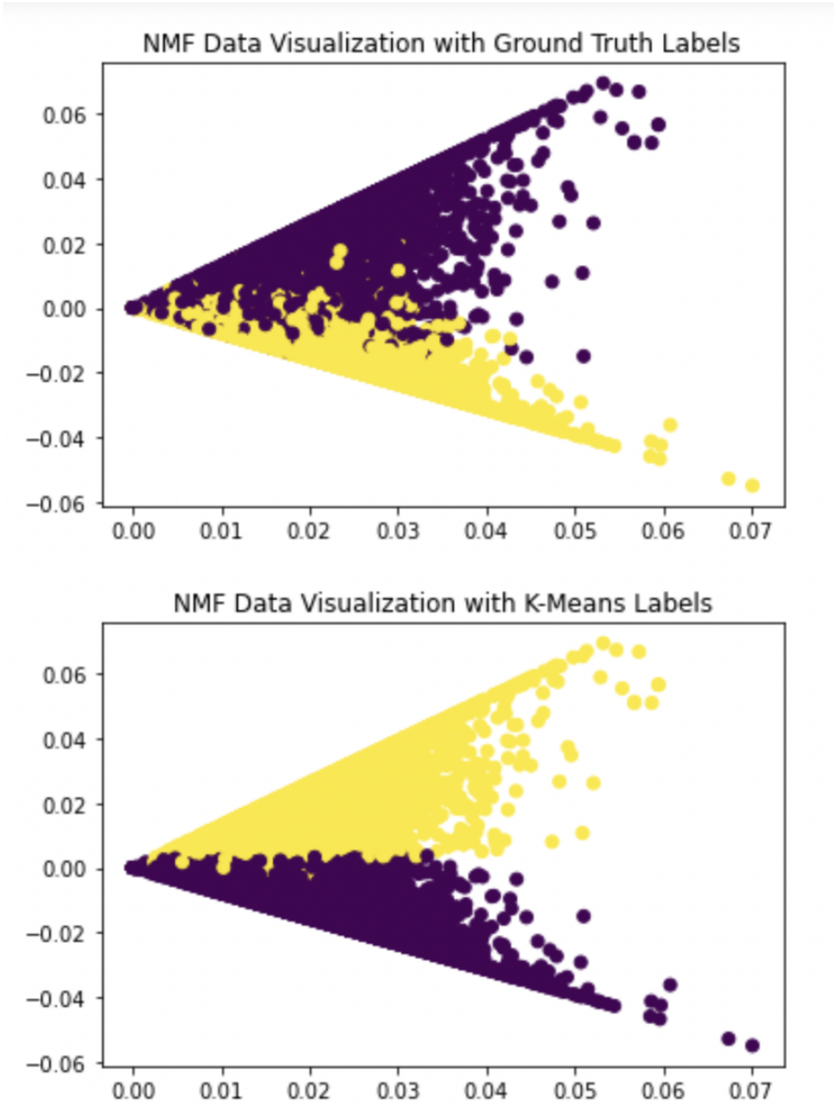

Figure 8: Plot of NMF feature matrix

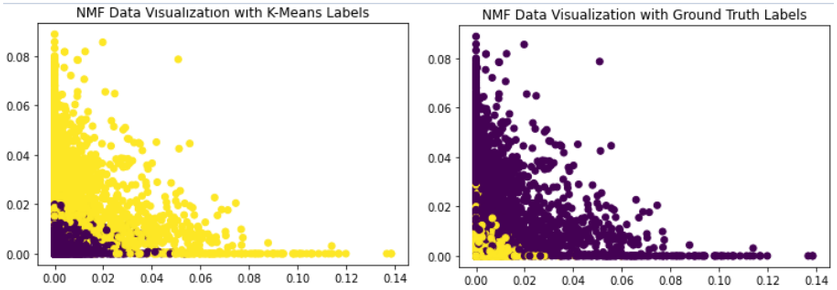

Figure 9: Plot of NMF feature matrix - only the first 2 dimensions

We have applied SVD with n_dimension = 2 to project the representations into a 2-D plane for visualization. Figure 7 and 8 are constructed this way. Figure 9 on the other hand consists of the first 2 features of NMF.

Question 9

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

- K-Means algorithm is based on the assumption that the clusters under consideration belong to Gaussian distribution describing a type of data which consists of observations on only a single feature/class/attribute (referred to as univariate). On the contrary, here, we observe the presence of two clusters with unequal variance.
- During the initialisation step, K-Means++, assumes the starting clusters to be far from each other. But, here, we can see that clusters returned by both SVD and NMF have considerably high overlap. This in turn means a very minimal euclidean distance between the centroids of the two clusters. This consequently makes the decision boundary finding process tedious and convoluted. Some metrics like Homogeneity and V-means scores also reflect our understanding of this.
- In addition to these assumptions, K-means utilizes euclidean distance which implicitly operates on the assumption that the clusters are isotropic and convex structures. Our results show that this is clearly not the case. Clusters returned by SVD and NMF resemble unobvious structures as opposed to the expected well-defined spherical 2D distribution. NMF's clusters are contradicting this assumption with their irregular shapes as we can observe that the NMF clusters are not even close to a spherical structure.

Conclusion

- All these results convey that the dataset's inherent distribution is skewed and henceforth not very ideal for K-means clustering. That being said, it is still fairly reasonable to use k-means on it. Even though it doesn't follow all the above said requirements perfectly, we do see that the two clusters can be divided by an axis to a decent accuracy. Moreover the shape is also not too convoluted.
- If logarithmic transformations are considered, the data points could be made to spread more uniformly.
- By smoothening the outliers, logarithmic transformation can improve the clustering results as these outliers can drag the centroids or form their own clusters. This is handled better by clustering algorithms later which assign outliers to a class -1.
- Given the large dimensionality of the dataset, it is not very ideal for any solution that uses distance-based similarity measure. Same is applicable to k-means. As the number of dimensions increases, a distance-based similarity measure converges to a constant value between any given examples. This is called Curse of Dimensionality. The plots show how the ratio of the standard deviation to the mean of distance between examples decreases as the number of dimensions increases. This convergence means k-means becomes less effective at distinguishing between examples. (Reference 1)
- The dataset is not purely convex and hence k-means cannot inhenerently generate an accurate cluster boundary as shown in Figure 11.

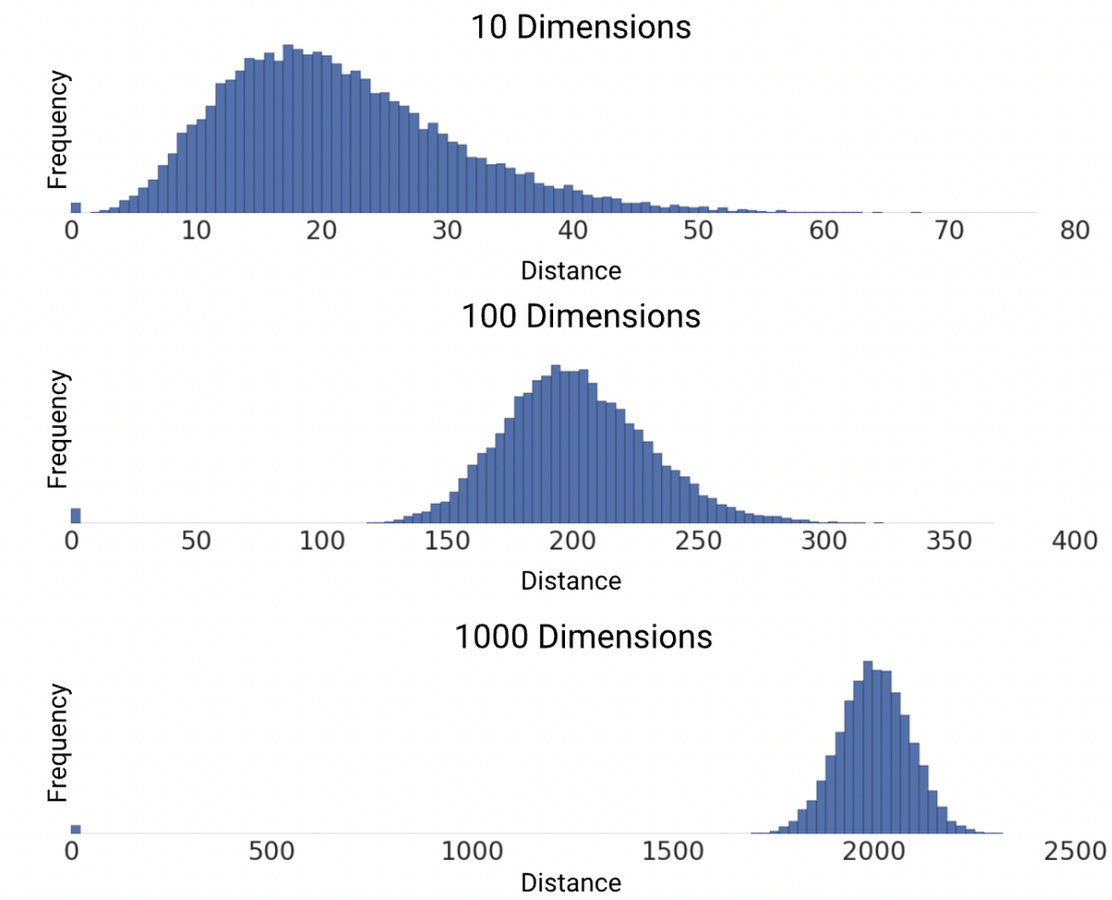

Figure 10: Each plot shows the pairwise distances between 200 random points.

Figure 11: Non-Convex boundary

Question 10

Load documents with the same configuration as in Question 1, but for ALL 20 categories. Construct the TF-IDF matrix, reduce its dimensionality properly using either NMF or SVD, and perform K-Means clustering with k=20. Visualize the contingency matrix and report the five clustering metrics.There is a mismatch between cluster labels and class labels. For example, the cluster-3 may correspond to the class #8. As a result, the high-value entries of the 20 $\times$ 20 contingency matrix can be scattered around, making it messy to inspect, even if the clustering result is not bad. One can use scipy.optimize.linearsumassignment to identify the best-matching cluster-class pairs, and permute the columns of the contingency matrix accordingly.

We follow the same steps as before but with all 20 categories now.

- Remove headers and footers while importing the dataset.
- No stemming or lemmatization applied.
- Use CountVectorizer() to convert the text into a bag-of-words matrix. Remove English stop-words. Set min_df to 3. This removes all the rare words that appear in less than 3 documents.
- Convert the matrix of token counts to TF-IDF matrix using TfidfTransformer. The TF-IDF matrix highlights the discriminating words in each document against the more common words and this can help while clustering to differentiate the various classes.
- Reduce dimensionality with SVD/NMF. Try for various values of r in the range {1, 2, 3, 5, 10, 20, 50, 100, 300}
- Use k-means to cluster into 20 classes.

SVD:
Average score is the mean across all the 5 scores mentioned above. Results for the various values of r:

```
r = 1
Average Score: 0.020699099772094783
Adjusted Rand Index Score: 0.0052377752057146945

r = 2
Average Score: 0.1866356606182336
Adjusted Rand Index Score: 0.06522862222326206

r = 3
Average Score: 0.2210624664991327
Adjusted Rand Index Score: 0.08430423856596442

r = 5
```

```
Average Score: 0.2933199644577863
Adjusted Rand Index Score: 0.1264834813291519

r = 10
Average Score: 0.2934249864519825
Adjusted Rand Index Score: 0.12150471868687898

r = 20
Average Score: 0.3088622981853003
Adjusted Rand Index Score: 0.12061901723196018

r = 50
Average Score: 0.3008653247451001
Adjusted Rand Index Score: 0.09873548453280839

r = 100
Average Score: 0.3010667727493633
Adjusted Rand Index Score: 0.10233658634623306

r = 300
Average Score: 0.278357458025868
Adjusted Rand Index Score: 0.0926830632169624

Best r in terms of average score: 20
Best SVD Score: 0.3088622981853003

Best r in terms of Adjusted Rand Score: 5
Best SVD Score: 0.1264834813291519
```

Best in terms of average score: r = 20

```
Homogeneity score: 0.336158
Completeness score: 0.378021
V-measure score: 0.355862
Adjusted Rand Index score: 0.120619
Adjusted mutual information score: 0.353652
```

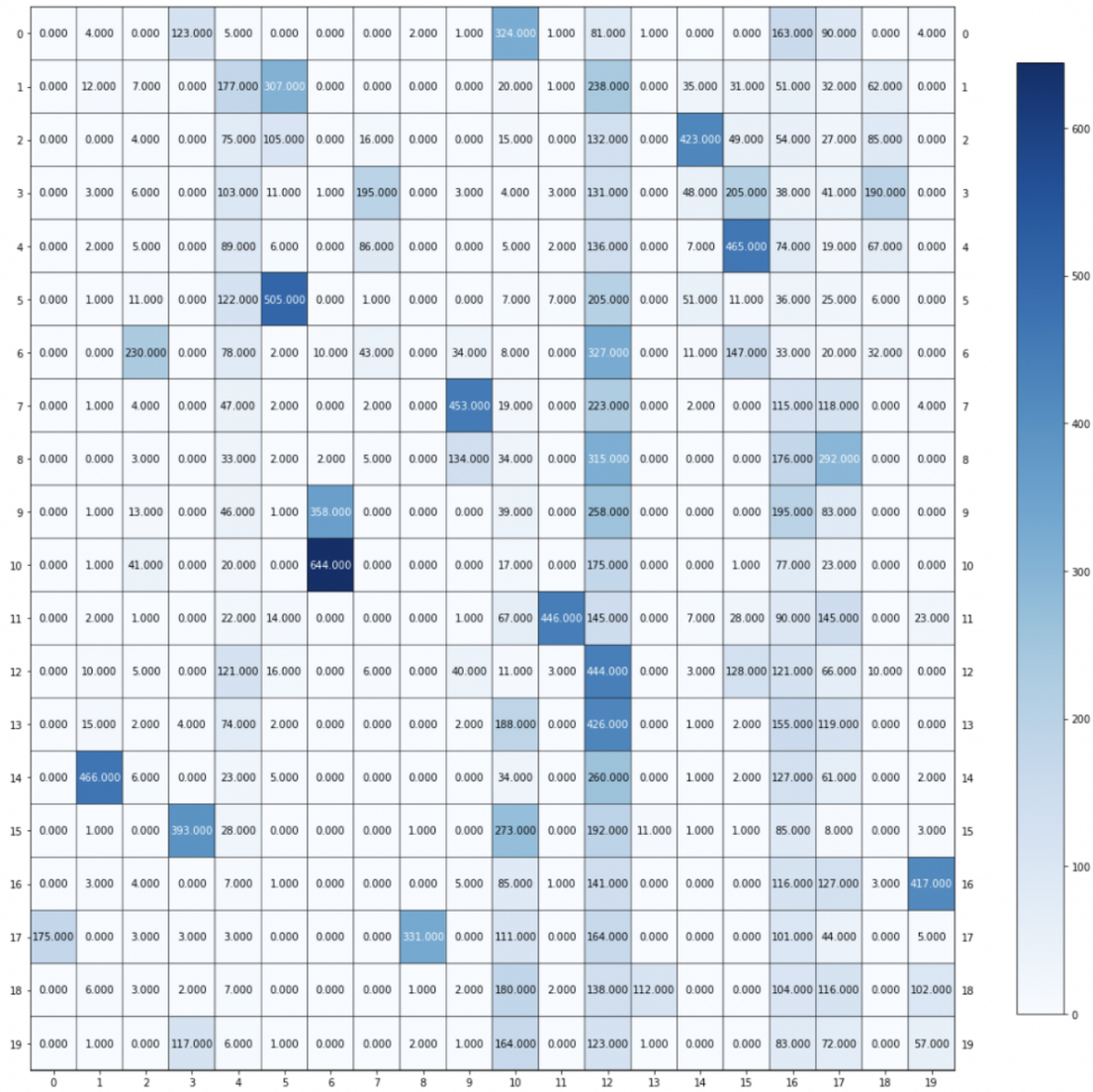

Figure 12: K-means with SVD (r = 20) Contingency Matrix Table

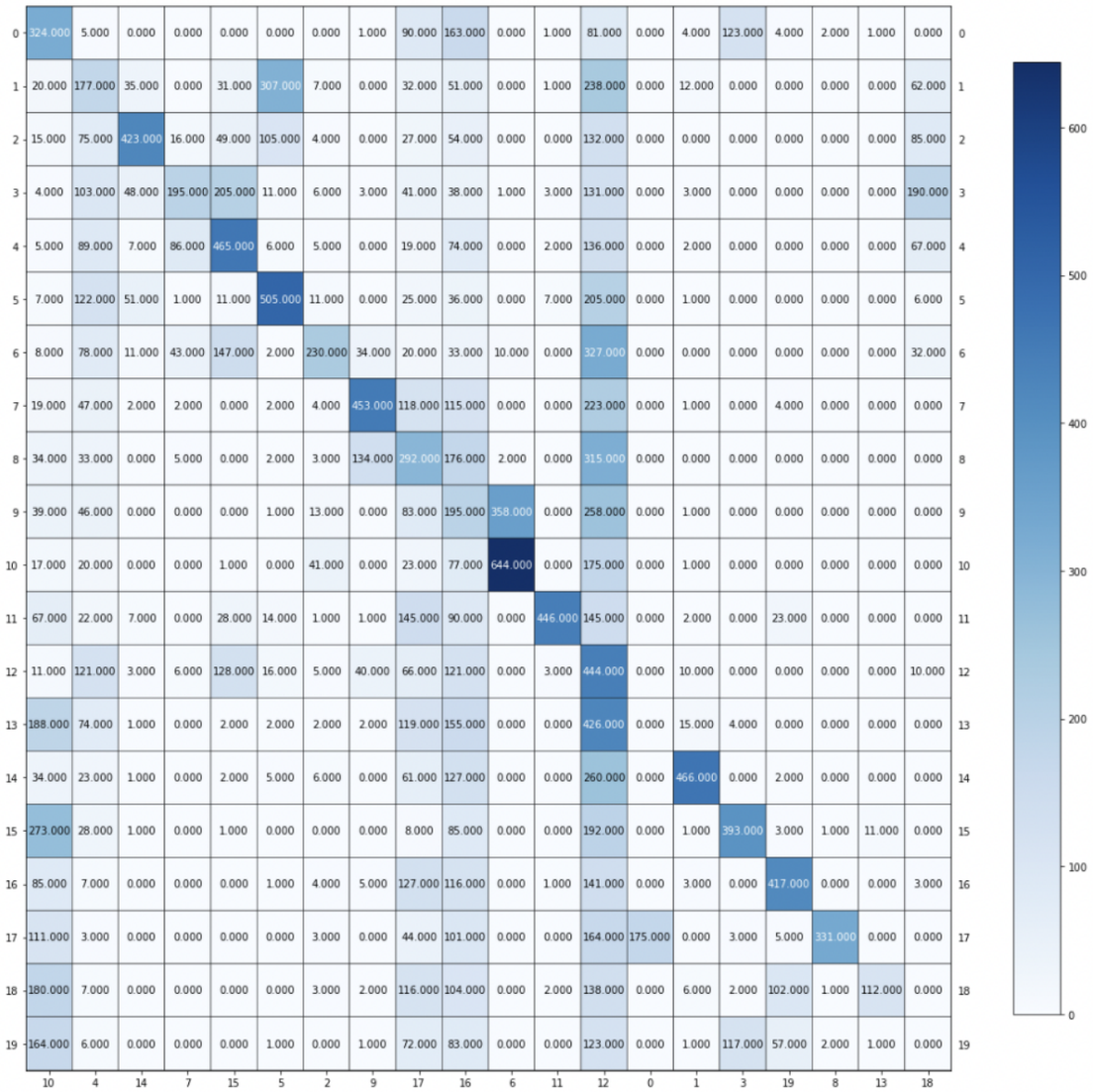

Figure 13: K-means with SVD (r = 20) Permuted Contingency Matrix Table

Best r in terms of Adjusted Rand Score: r = 5

`Homogeneity score:` 0.322084
`Completeness score:` 0.349661
`V-measure score:` 0.335306
`Adjusted Rand Index score:` 0.126483
`Adjusted mutual information score:` 0.333065

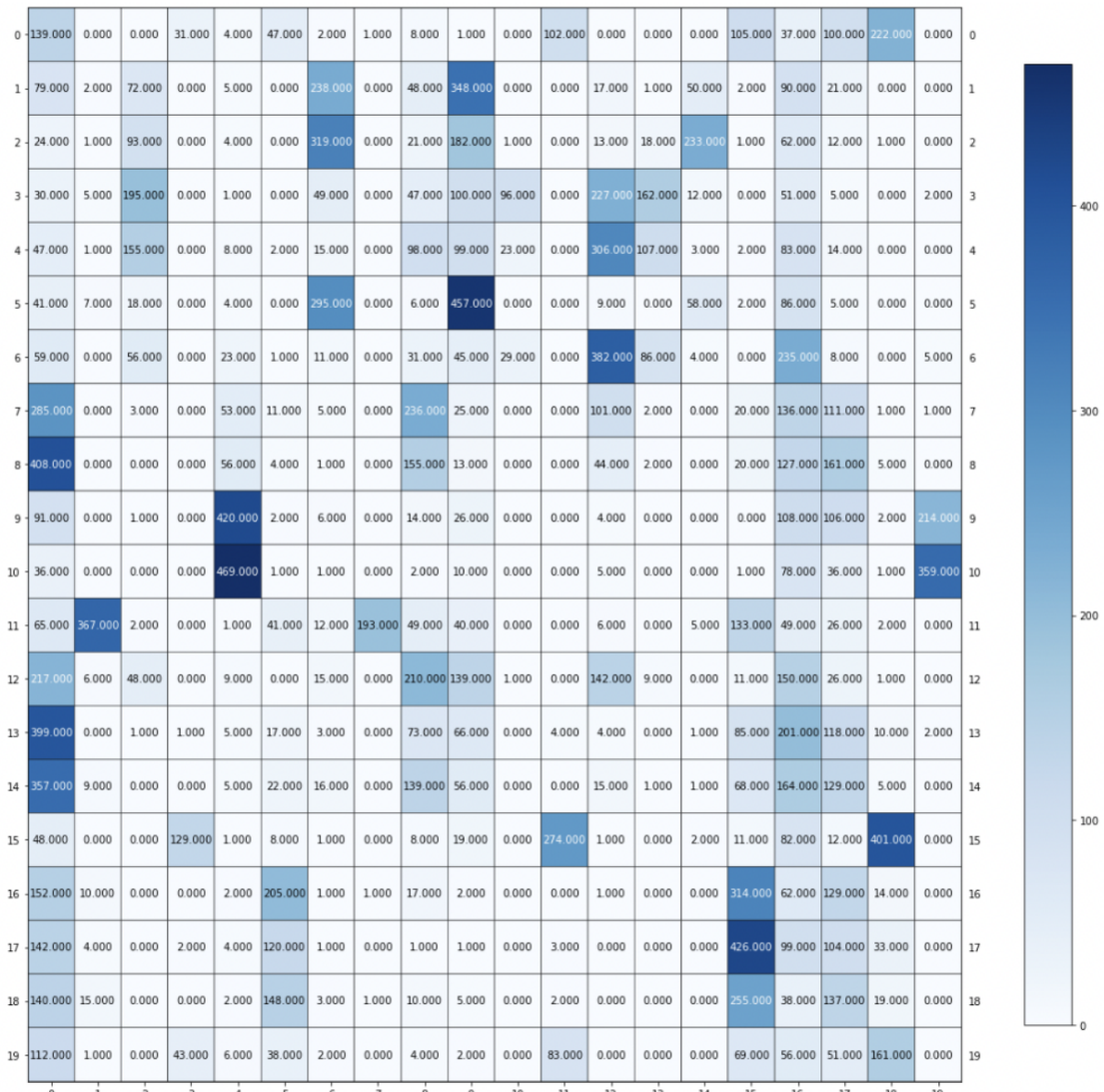

Figure 14: K-means with SVD (r = 5) Contingency Matrix Table

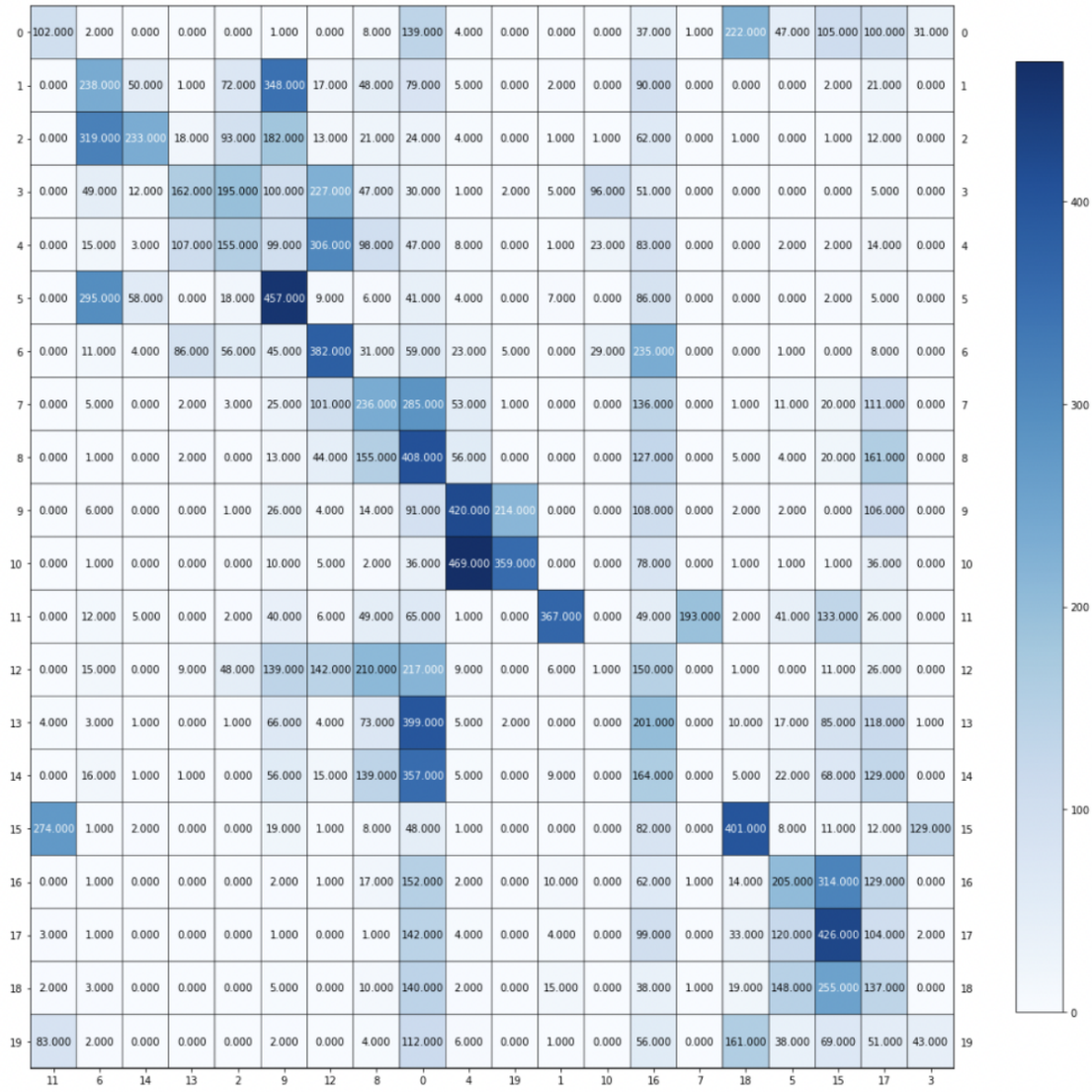

Figure 15: K-means with SVD (r = 5) Permuted Contingency Matrix Table

NMF:

```
Best r in terms of Adjusted Rand Score: 10
Best NMF Adjust Rand Score: 0.12288393405698932

Best r in terms of Average score: 10
Best NMF Average Score: 0.3080249921821888

Homogeneity score: 0.332317
Completeness score: 0.378952
V-measure score: 0.354105
Adjusted Rand Index score: 0.122884
Adjusted mutual information score: 0.351868
```

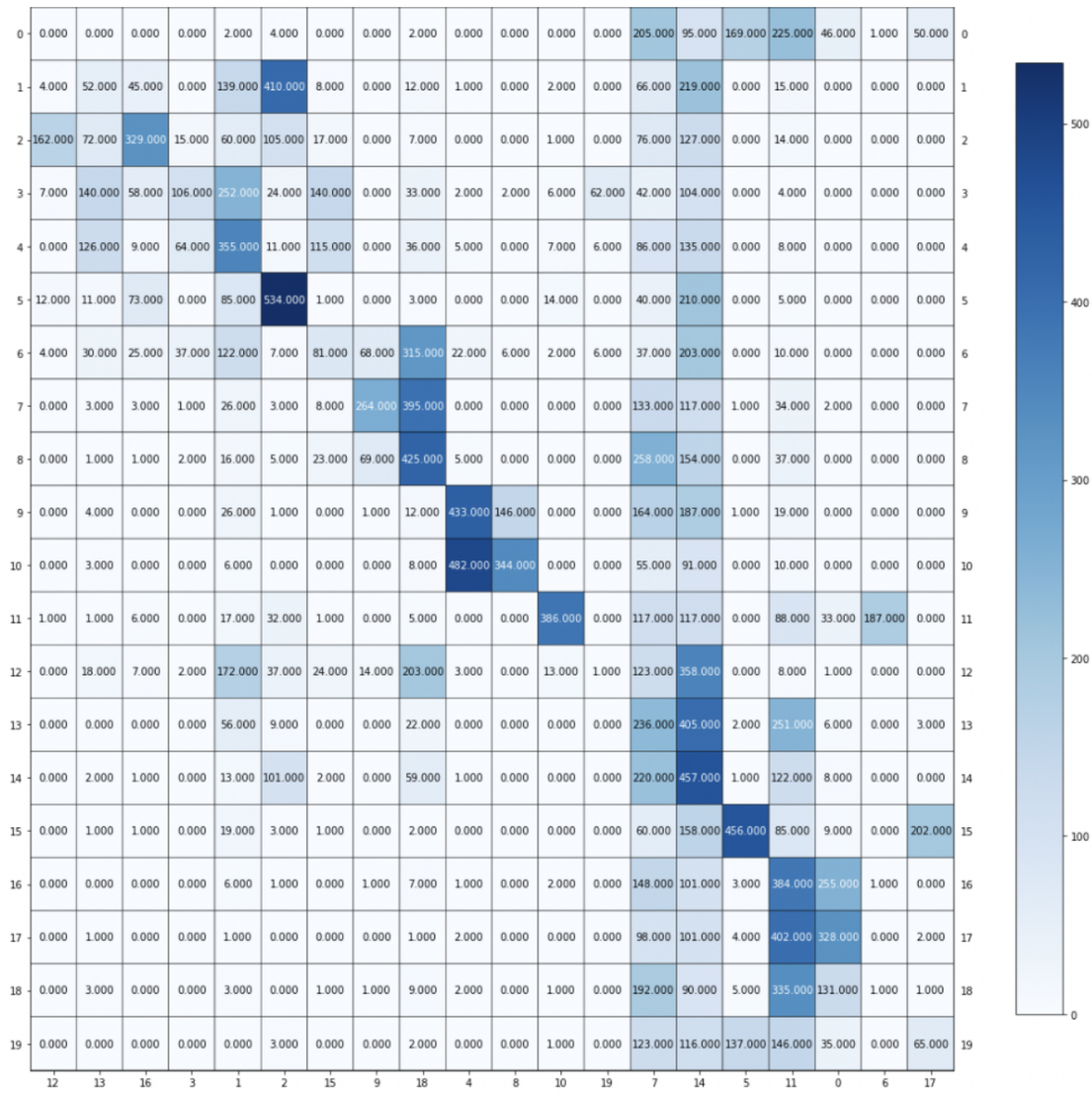

Figure 16: K-means with NMF (r = 10) Permuted Contingency Matrix Table

Question 11

Use UMAP to reduce the dimensionality of the 20 category TF-IDF matrix, and apply K-Means clustering with n_components=20. Find a good n_components choice for UMAP, and compare the performance of two metrics by setting metric="euclidean" and metric="cosine" respectively.
Report the permuted contingency matrix and the five clustering evaluation metrics for "euclidean" and "cosine".

```
n_components = 1
Euclidean - Average Score: 0.011659970714660623
Euclidean - Adjusted Rand Index Score: 0.0011295727155154144
Cosine - Average Score: 0.36682839488031094
Cosine - Adjusted Rand Index Score: 0.24835561340182444

n_components = 2
Euclidean - Average Score: 0.009233754868241337
Euclidean - Adjusted Rand Index Score: 0.0017955497426913746
Cosine - Average Score: 0.5349270395670415
Cosine - Adjusted Rand Index Score: 0.42473084210867085

n_components = 3
Euclidean - Average Score: 0.009700451521090385
Euclidean - Adjusted Rand Index Score: 0.002519239806148265
Cosine - Average Score: 0.5462503764507097
Cosine - Adjusted Rand Index Score: 0.4557831159281759

n_components = 5
Euclidean - Average Score: 0.011166397831138417
Euclidean - Adjusted Rand Index Score: 0.002427968998191207
Cosine - Average Score: 0.558580767044962
Cosine - Adjusted Rand Index Score: 0.4569154074672965

n_components = 10
Euclidean - Average Score: 0.012489899366956242
Euclidean - Adjusted Rand Index Score: 0.0032872120779449095
Cosine - Average Score: 0.5535767526541606
Cosine - Adjusted Rand Index Score: 0.45350646353504553

n_components = 20
Euclidean - Average Score: 0.012594589510638046
Euclidean - Adjusted Rand Index Score: 0.003538731411959756
Cosine - Average Score: 0.5587034208966832
Cosine - Adjusted Rand Index Score: 0.4486136067589542

n_components = 50
Euclidean - Average Score: 0.011790361302429343
```

```
Euclidean - Adjusted Rand Index Score: 0.003016874683491359
Cosine - Average Score: 0.5541780633647445
Cosine - Adjusted Rand Index Score: 0.45429248845307263

n_components = 100
Euclidean - Average Score: 0.012024396879918364
Euclidean - Adjusted Rand Index Score: 0.003495646985333093
Cosine - Average Score: 0.5574151181279057
Cosine - Adjusted Rand Index Score: 0.4502352884642707

n_components = 300
Euclidean - Average Score: 0.01071185369760264
Euclidean - Adjusted Rand Index Score: 0.0030171762691013895
Cosine - Average Score: 0.5531304838705926
Cosine - Adjusted Rand Index Score: 0.4426321319136454

Best n_components in terms of Adjusted Rand Score Euclidean: 20
Score: 0.003538731411959756

Best n_components in terms of Average Score for Euclidean: 20
Score: 0.012594589510638046

Best n_components in terms of Adjusted Rand Score Cosine: 5
Score: 0.4569154074672965

Best n_components in terms of Average Score for Cosine: 20
Score: 0.5587034208966832
```

Euclidean (n_components = 20):

```
Homogeneity score for Euclidean UMAP: 0.013954
Completeness score for Euclidean UMAP: 0.014534
V-measure score for Euclidean UMAP: 0.014238
Adjusted Rand Index score for Euclidean UMAP: 0.003035
Adjusted mutual information score for Euclidean UMAP: 0.010984
```

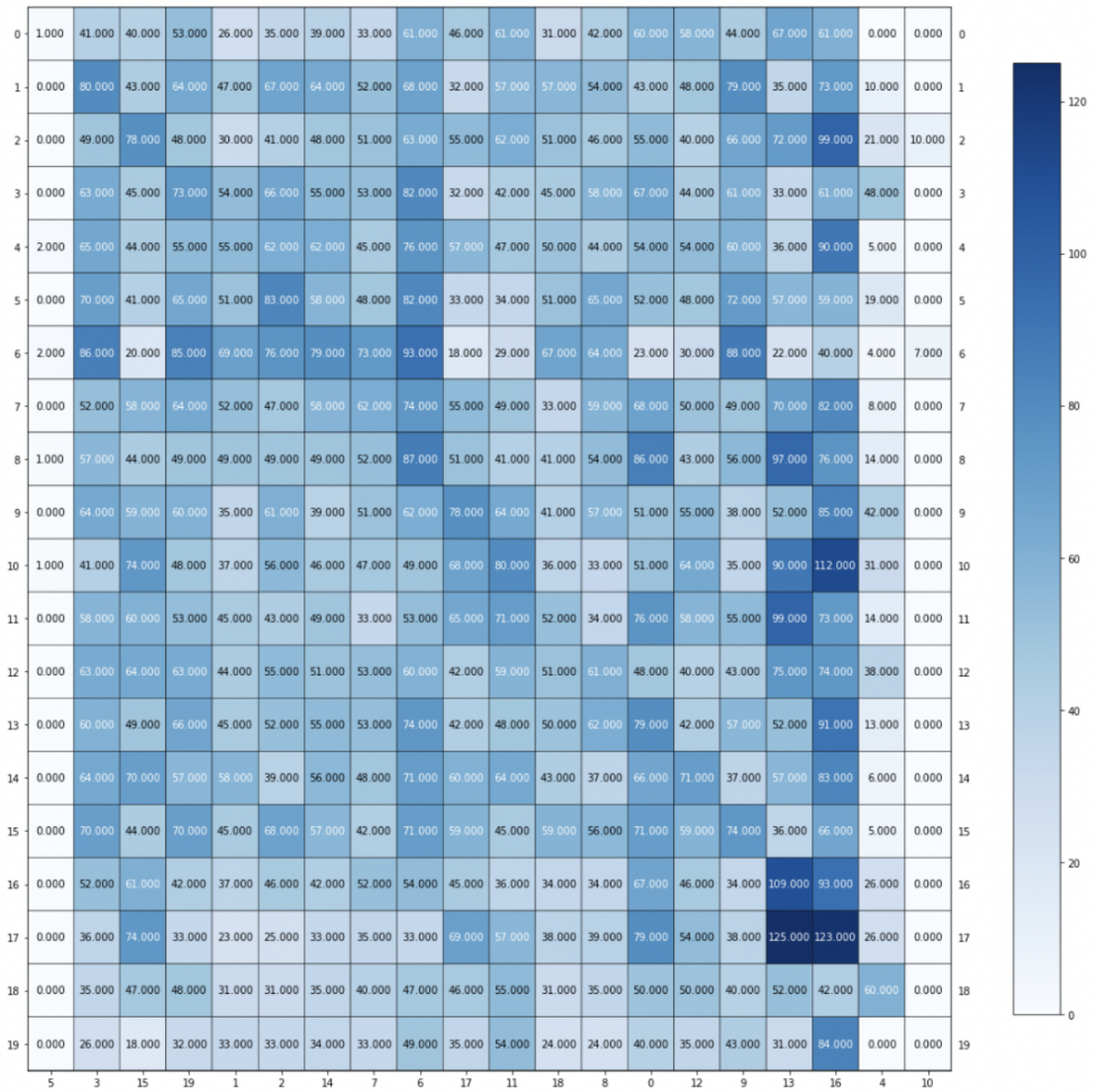

Figure 17: K-Means - UMAP Euclidean (n_components = 20) Contingency Matrix Table

Cosine:

Best in terms of Adjusted Rand Score: n_components = 5

```
Homogeneity score for Cosine UMAP: 0.556166
Completeness score for Cosine UMAP: 0.584718
V-measure score for Cosine UMAP: 0.570085
Adjusted Rand Index score for Cosine UMAP: 0.438110
Adjusted mutual information score for Cosine UMAP: 0.568648
```

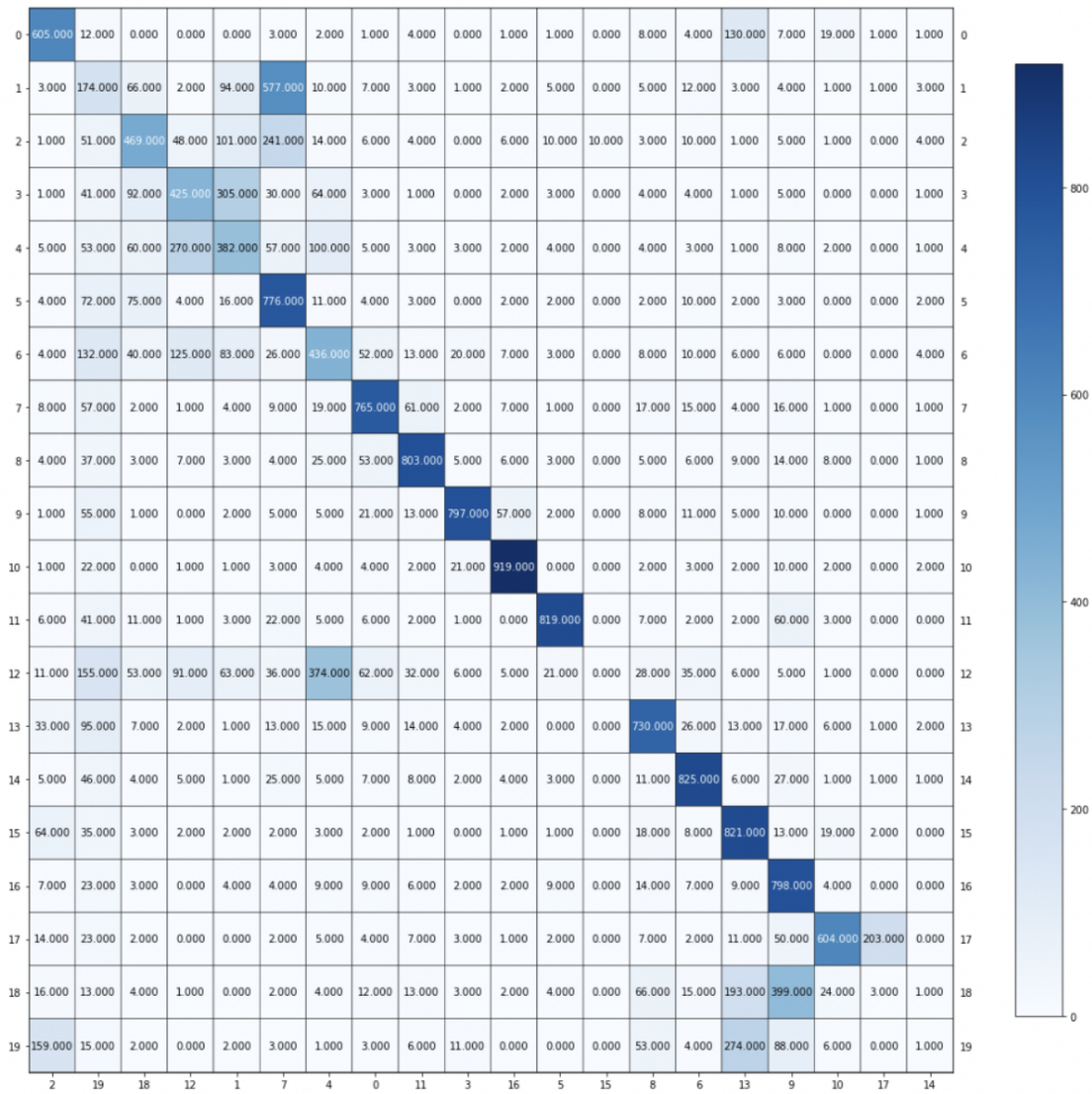

Figure 18: K-Means - UMAP Cosine (n_components = 5) Contingency Matrix Table

Best in terms of Average Score: n_components = 20

```
Homogeneity score for Cosine UMAP: 0.576348
Completeness score for Cosine UMAP: 0.596052
V-measure score for Cosine UMAP: 0.586034
Adjusted Rand Index score for Cosine UMAP: 0.461406
Adjusted mutual information score for Cosine UMAP: 0.584664
```

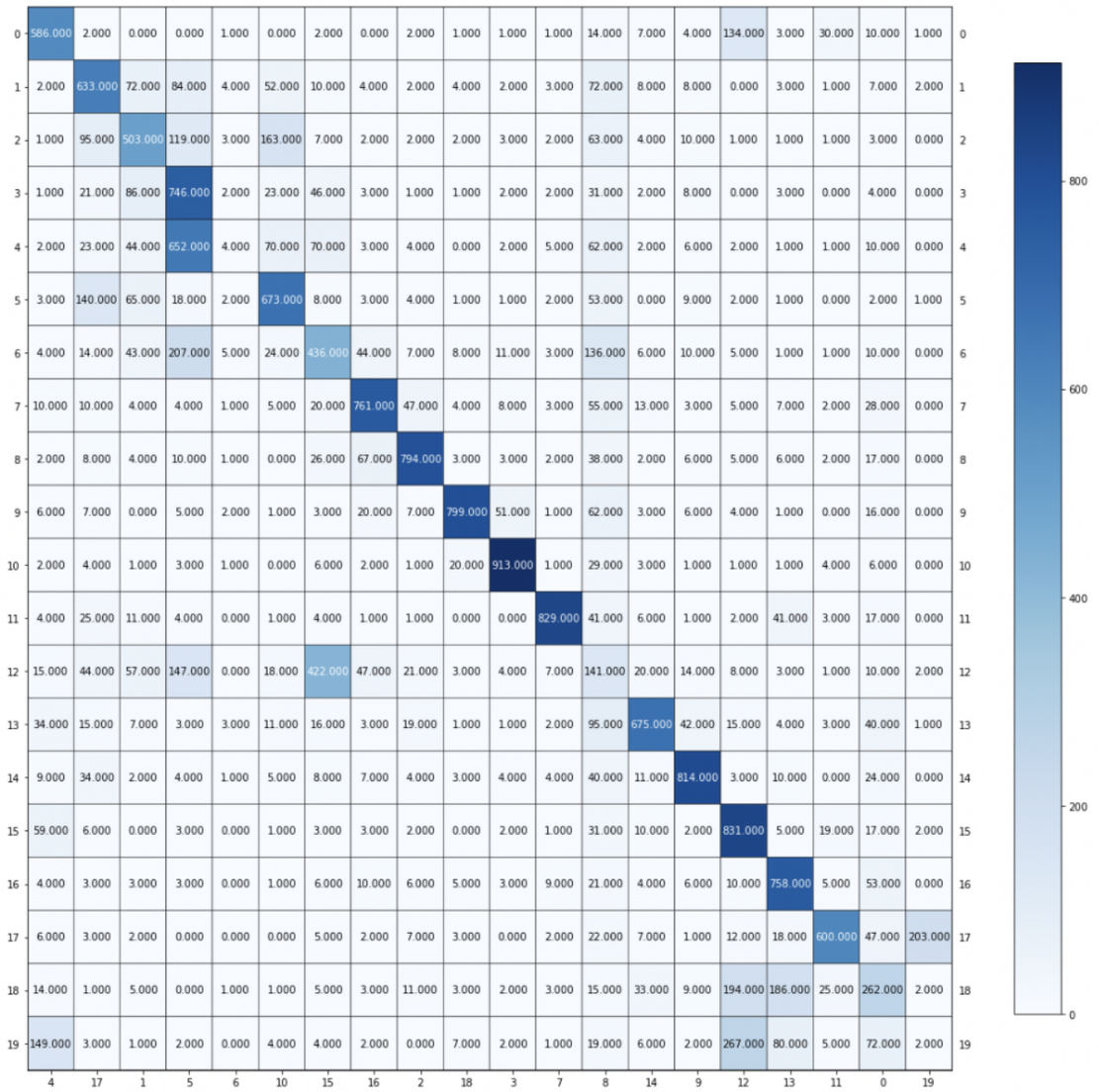

Figure 19: K-Means - UMAP Cosine (n_components = 20) Contingency Matrix Table

We can observe that UMAP with cosine distance metric outperforms Euclidean distance. This is expected because this makes use of the relative angles between the various sample points to cluster them instead of the magnitude of the distances. This helps overcome the issue with respect to the higher frequency of words deciding the cluster to which it belongs. Moreover, in larger dimensions, Euclidean distances converge to a constant value between all the sample points due to the sparsity in each dimensions. This is evident from the figures provided. We have represented the UMAP with cosine and euclidean distances for 2 dimensions. This gives an idea about how the cosine distance helps to differentiate the various clusters even at 2 dimensions.

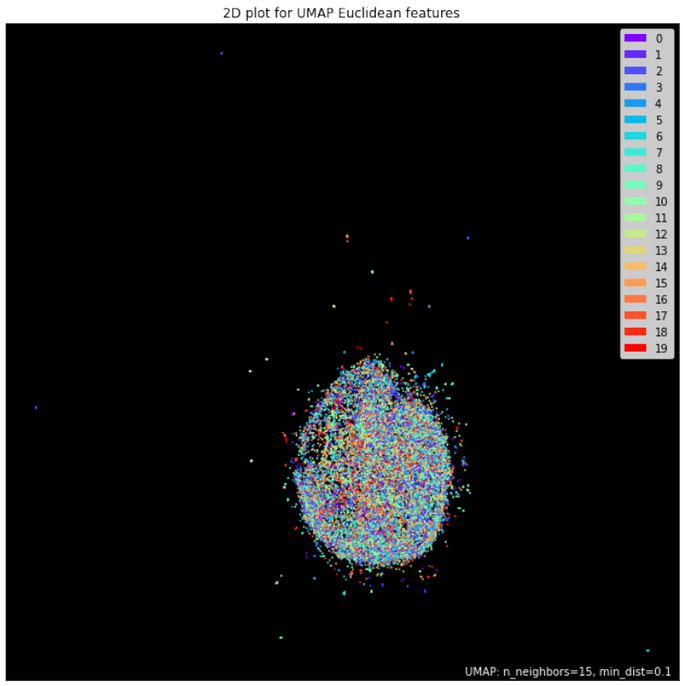

Figure 20: UMAP Euclidean

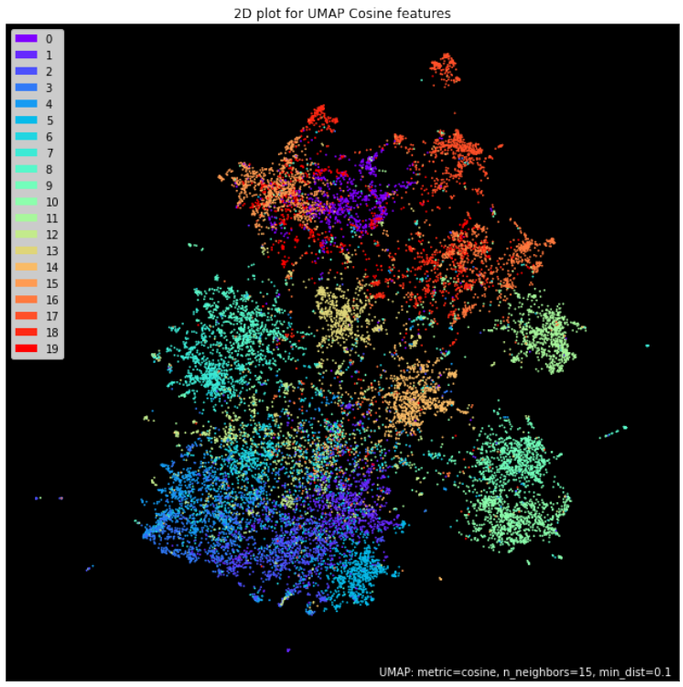

Figure 21: UMAP Cosine

Question 12

Analyze the contingency matrix

- First, we could make two key inferences. On noticing Euclidean UMAP's contingency table, we observe that the matrix not only lacks a fine structure but also it is non-diagonal, void of any patterns and stochastic.
- Therefore we only need to explain the cosine contingency matrix. As we can see, this does fairly well across most classes. This is easy to observe from the diagonal centrix matrix. But if we look closely, we can observe that some of the classes/clusters do not follow this pattern. This might be attributed to common words across classes whose semantic information might have been lost due to the removal of distinguishing metrics such as headers and footers.
- Also, as we had mentioned previously, Application of K-means algorithm on this dataset also has some striking shortcomings/limitations
  - K means being very sensitive to outliers and noisy samples, weakens its clustering ability while it witnesses such samples.
  - Operates on the assumption of clusters coupled with well defined centroids
  - Euclidean operation restricts to isotropic, globular, convex clusters.
  - It lacks the ability to handle datasets with irregular shapes and mutually disparate sizes.

Question 13

So far, we have attempted K-Means clustering with 4 different representation learning techniques (sparse representation, PCA, NMF, UMAP). Compare and contrast the results from the previous sections, and discuss which approach is best for the K-Means clustering task on the 20-class text data.

Sparse Representation:

```
Homogeneity score for Sparse Representation: 0.342959
Completeness score for Sparse Representation: 0.392091
V-measure score for Sparse Representation: 0.365883
Adjusted Rand Index score for Sparse Representation: 0.121723
Adjusted mutual information score for Sparse Representation: 0.363694
```

PCA/SVD:

```
Homogeneity score: 0.322084
Completeness score: 0.349661
V-measure score: 0.335306
Adjusted Rand Index score: 0.126483
Adjusted mutual information score: 0.333065
```

NMF:

```
Homogeneity score: 0.332317
Completeness score: 0.378952
V-measure score: 0.354105
Adjusted Rand Index score: 0.122884
Adjusted mutual information score: 0.351868
```

UMAP:

```
Homogeneity score for Cosine UMAP: 0.576348
Completeness score for Cosine UMAP: 0.596052
V-measure score for Cosine UMAP: 0.586034
Adjusted Rand Index score for Cosine UMAP: 0.461406
Adjusted mutual information score for Cosine UMAP: 0.584664
```

Observations:

- We observe that UMAP provides the best results (46% in Adjusted Rand Index Score). This is due to the fact that SVD and NMF are least squares techniques and enormously liable to outliers and noise as compared to UMAP. Moreover, both SVD and NMF are linear projections of the large dimensional features which maximize the variance of the dataset, however are not able to capture nonlinear dependencies in the data. UMAP, on the other hand, tries to ensure that the semantic and non-linear dependencies, in

addition to global structure of the data is preserved such that similar embeddings are clustered collectively and different ones as far apart as possible. This aids the clustering innately. In fact, as UMAP tries to preserve the information entropy regardless of the dimensions, the overall performance of UMAP is consistent throughout a huge range of r.

- UMAP is a local-density based dimensionality reduction method. It gives a more flexible interpretation and works fine even if the data is without any prior distribution. PCA and NMF treat all clusters as a whole and therefore can lose local structure. This is something UMAP tries to overcome. UMAP scale better to large sample size and higher dimensionality.

- Now let us compare SVD and NMF. Both provide similar results though NMF provides slightly lesser ARI (12.2% compared to SVD's 12.6%). This could be because NMF only allows positive entries in the reduced-rank feature matrix. This is not the case for SVD and therefore SVD can represent higher dimensional feature matrix more accurately. SVD is more deterministic than NMF and considers the geometry in the feature space basis that is critical for clustering in higher dimensions. SVD sorts the features in the decreasing order of variance explanation. Therefore the more important features are higher in the hierarchy.

- Sparse representation provides the least scores(12.1%). This is expected as the dataset is very large in 20 classes as compared to the 2 classes we saw earlier. Therefore, in this case the large sparse data causes a lot of noise which makes it difficult for the k-means algorithm to cluster or classify the data points accurately.

UMAP with cosine distance metric is the best representation learning technique for K-Means clustering on the 20-class text data.

Question 14

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.

```
r = 1
Ward - Average Score: 0.37280719554437003
Ward - Adjusted Rand Index Score: 0.25499290629074656
Single - Average Score: 0.09894774627791939
Single - Adjusted Rand Index Score: 0.0011583407478178026

r = 2
Ward - Average Score: 0.5213396697993213
Ward - Adjusted Rand Index Score: 0.4054495051431426
Single - Average Score: 0.0918021807735636
Single - Adjusted Rand Index Score: 0.0004794021782465533

r = 3
Ward - Average Score: 0.5252274397588991
Ward - Adjusted Rand Index Score: 0.40808944756741156
Single - Average Score: 0.09215160953150422
Single - Adjusted Rand Index Score: 0.0006487202324916832

r = 5
Ward - Average Score: 0.5373638423827882
Ward - Adjusted Rand Index Score: 0.41922979197682786
Single - Average Score: 0.09474693013632571
Single - Adjusted Rand Index Score: 0.0004640604374673964

r = 10
Ward - Average Score: 0.5421441761875938
Ward - Adjusted Rand Index Score: 0.4135373003982577
Single - Average Score: 0.09163247979160813
Single - Adjusted Rand Index Score: 0.0004771725684970719

r = 20
Ward - Average Score: 0.541151006627852
Ward - Adjusted Rand Index Score: 0.4201402789033057
Single - Average Score: 0.0896138985366776
Single - Adjusted Rand Index Score: 0.00047550869216948116

r = 50
Ward - Average Score: 0.5421622015907681
Ward - Adjusted Rand Index Score: 0.4205767589824054
Single - Average Score: 0.094573096768697
```

```
Single - Adjusted Rand Index Score: 0.0004649996198351806

r = 100
Ward - Average Score: 0.5352827628493888
Ward - Adjusted Rand Index Score: 0.41049416048323184
Single - Average Score: 0.0926476502478393
Single - Adjusted Rand Index Score: 0.0005177735243756604

r = 300
Ward - Average Score: 0.5410097595175583
Ward - Adjusted Rand Index Score: 0.42362421567082825
Single - Average Score: 0.09349409570434383
Single - Adjusted Rand Index Score: 0.0005029717749594421

Ward:
Best r with respect to Average Score: 50
Best r with respect to Adjusted Rand Score: 300

Single:
Best r with respect to Average Score: 1
Best r with respect to Adjusted Rand Score: 1
```

WARD:

Best with respect to average score: r = 50

```
Homogeneity score for Agglomerative Clustering - Ward: 0.557938
Completeness score for Agglomerative Clustering - Ward: 0.591014
V-measure score for Agglomerative Clustering - Ward: 0.574000
Adjusted Rand Index score for Agglomerative Clustering - Ward: 0.424335
Adjusted mutual information score for Agglomerative Clustering - Ward: 0.572572
```

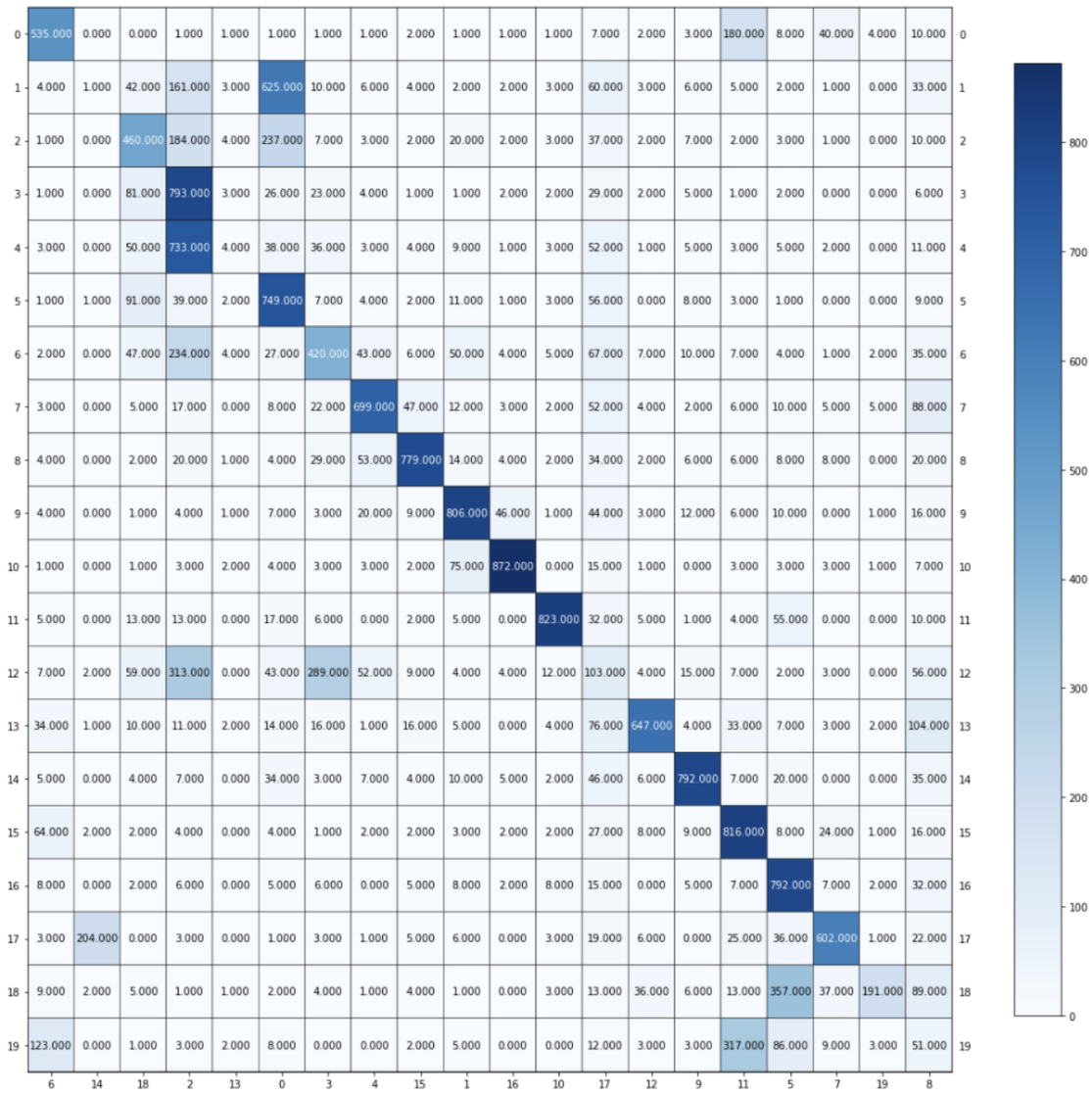

Figure 22: Agglomerative Clustering - WARD (r = 50) Contingency Matrix

Best with respect to ARI score: r = 300

```
Homogeneity score for Agglomerative Clustering - Ward: 0.561838
Completeness score for Agglomerative Clustering - Ward: 0.595401
V-measure score for Agglomerative Clustering - Ward: 0.578133
Adjusted Rand Index score for Agglomerative Clustering - Ward: 0.425093
Adjusted mutual information score for Agglomerative Clustering - Ward: 0.576726
```

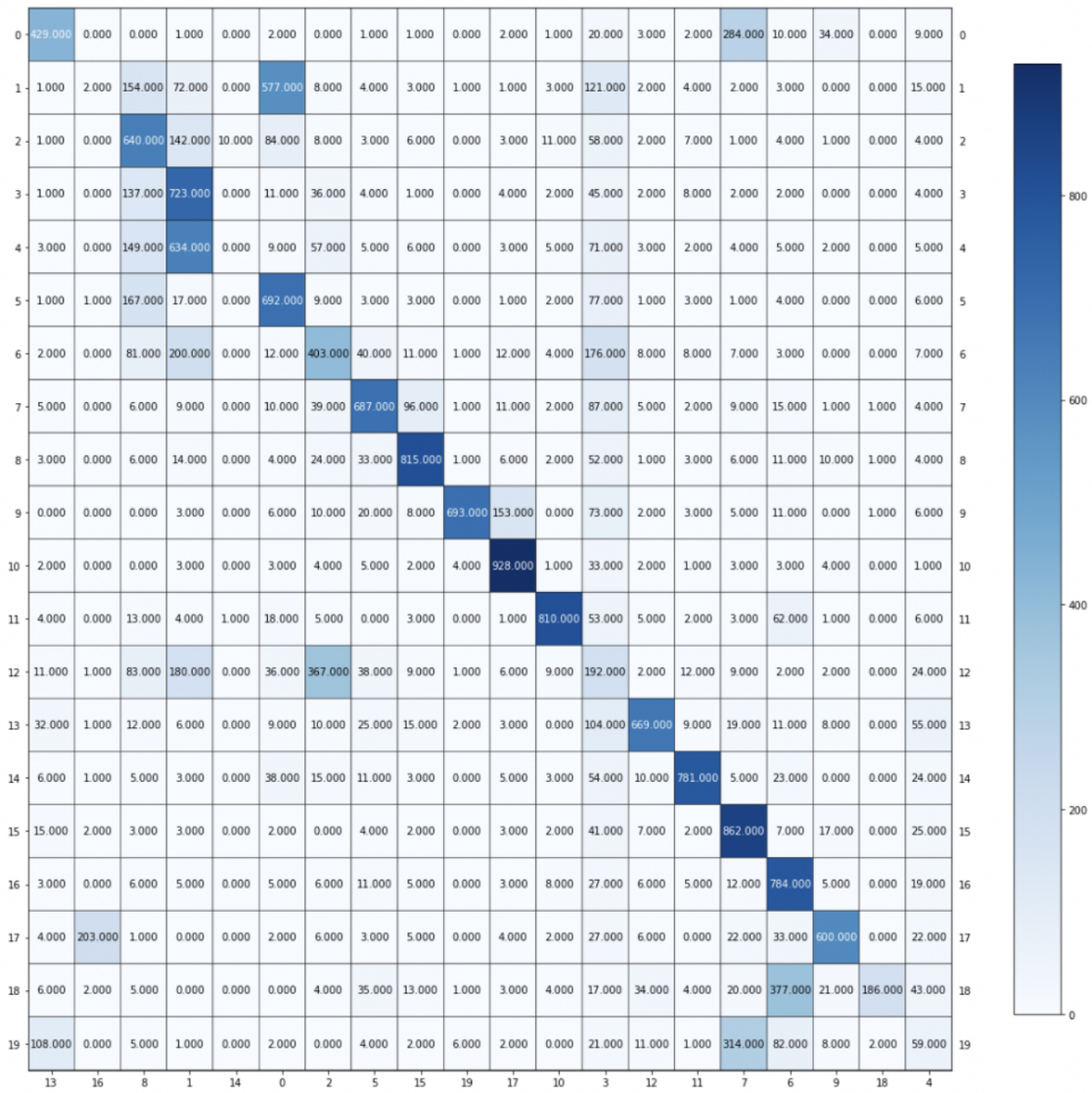

Figure 23: Agglomerative Clustering - WARD (r = 300) Contingency Matrix

SINGLE (r = 1):

```
Homogeneity score for Agglomerative Clustering - Single: 0.026486
Completeness score for Agglomerative Clustering - Single: 0.357980
V-measure score for Agglomerative Clustering - Single: 0.049323
Adjusted Rand Index score for Agglomerative Clustering - Single: 0.000900
Adjusted mutual information score for Agglomerative Clustering - Single: 0.044019
```

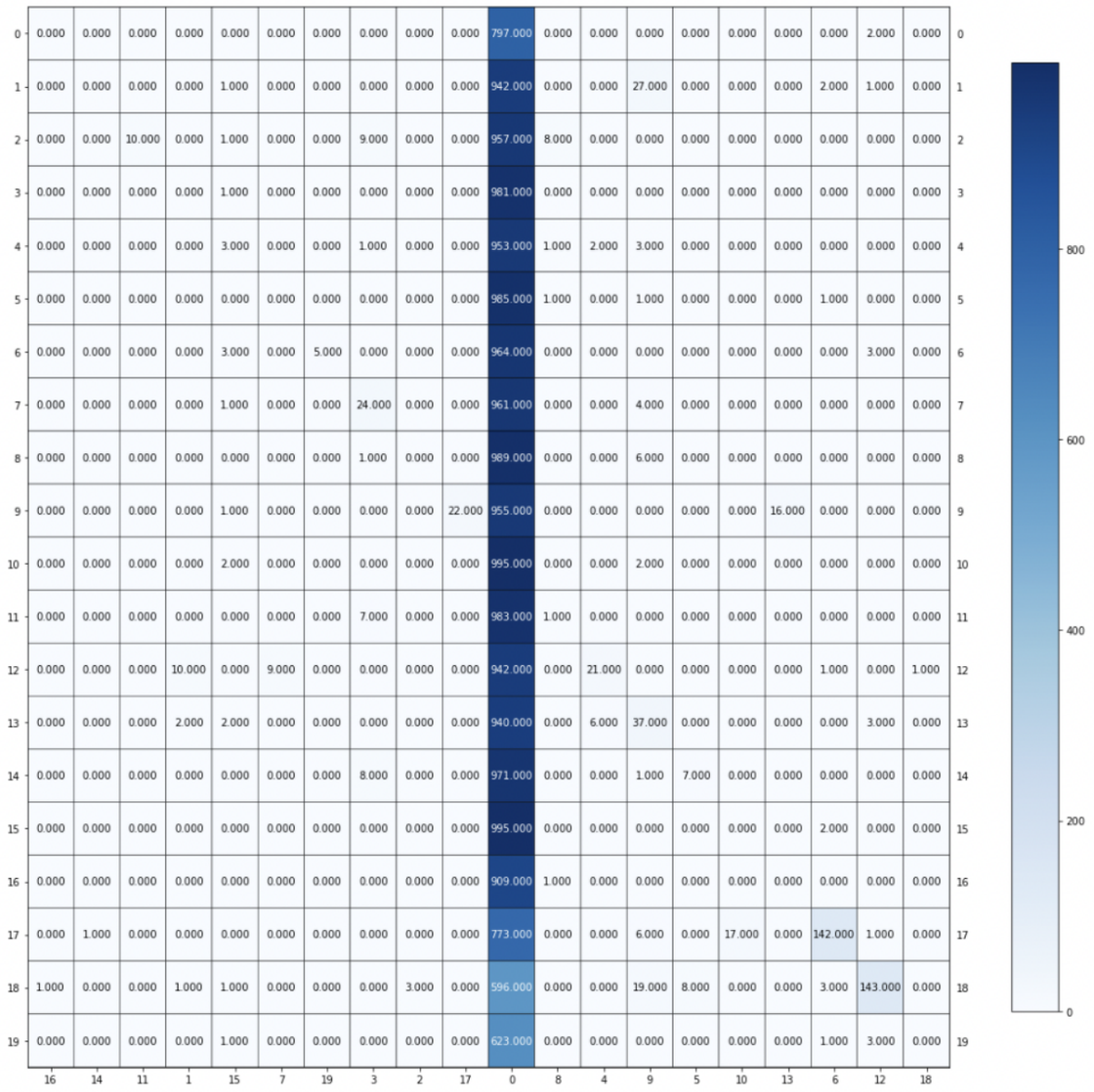

Figure 24: Agglomerative Clustering - Single (r = 1) Contingency Matrix

Comparing Ward and Single, we find that Ward is consistently outperforming in terms of both the metrics- Average score and Adjusted Rand Index Score. Agglomerative clustering groups clusters in a bottom-up fashion. Initially all documents are considered to be separate clusters and they are merged based on different metrics.

- Single Linkage Clustering tries to combine two clusters that contain the closest pair of elements not yet belonging to the same cluster as each other.
- In single-linkage clustering, the distance between two clusters is determined by a single pair of elements: those two elements (one in each cluster) that are closest to each other. The shortest of these pairwise distances that remain at any step causes the two clusters whose elements are involved to be merged. The method is also known as nearest neighbour clustering.
- A drawback of this method is that it tends to produce long chain like merging of clusters. This makes the process of defining the boundaries of the sub classes corresponding to the different clusters very tedious.
- As similarity is calculated for the closest pair, one drawback is that groups with close pairs can merge sooner than is optimal, even if those groups have overall dissimilarity.

This is clear from the Contingency Matrix. We see that a large number of data from various classes are clustered together into cluster 0. Therefore it forms one major cluster and the documents very far away from it are categorized as different clusters.

- Ward's clustering method is also termed as Minimum variance Method
- This is because Ward proposed his clustering algorithm which seeks to do a constrained optimization on the cluster space on the basis of the metric 'Error sum of squares'
- More specifically, during the implementation, at each step, this tends to find the pair of clusters that leads to minimum increase in total within-cluster variance post merging.
- In the initial step, all clusters are singletons (clusters containing a single point)
- Minimum variance criterion minimizes the total within-cluster variance. This increase in minimum variance criterion increases the weighted squared distance between cluster centers

The image below shows the difference between Ward and Single for different types of data. The third and fourth rows of the dataset depict the shortcoming of single linkage. As it takes the minimum distance points between clusters as the metric, it tends to merge clusters in a chain fashion and most of the dataset ends up being in one cluster. This is precisely what has occured with out data as well, as is observed from the Contingency matrix.

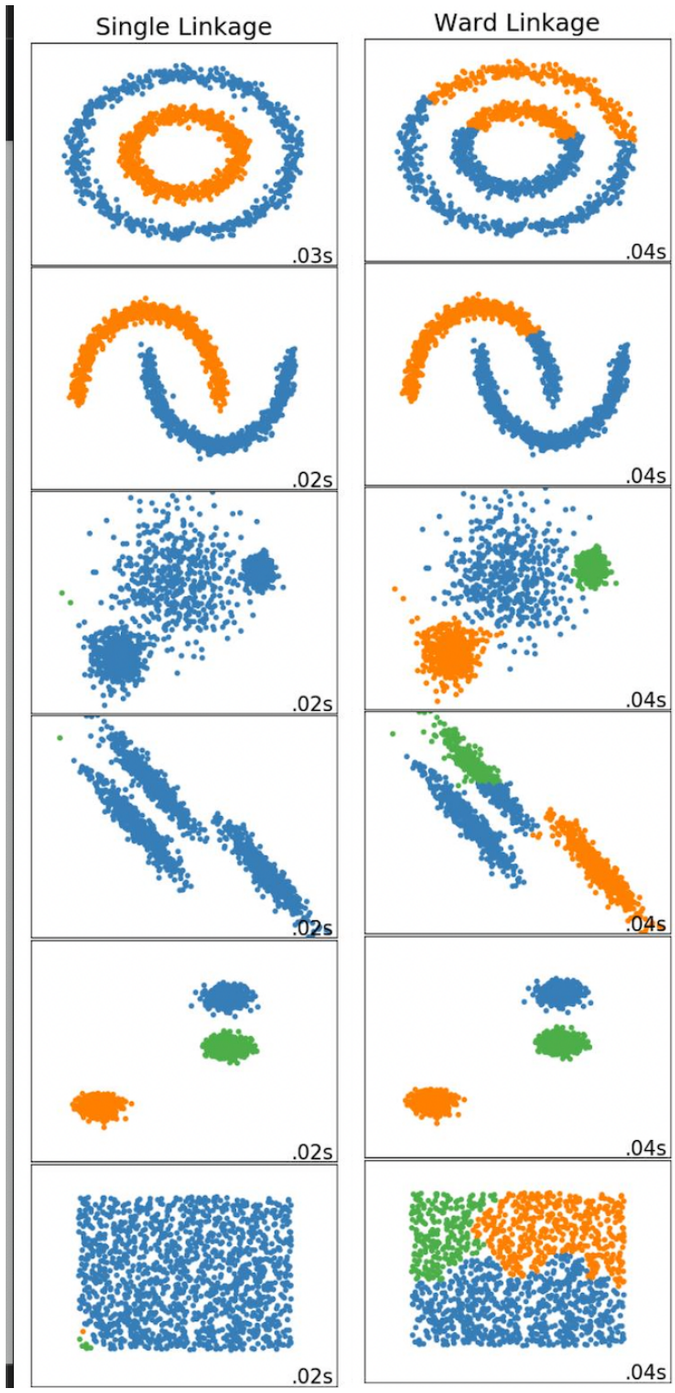

Figure 25: Single vs Ward linkage

Question 15

Apply DBSCAN and HDBSCAN on UMAP-transformed 20-category data. Use minclustersize=100 . Experiment on the hyperparameters and report your findings in terms of the five clustering evaluation metrics.
Here, we are asked to use DBSCAN (Density-Based Spatial Clustering of Application with Noise) and HBSCAN (Hierarchical DBSCAN) clustering algorithms to operate on UMAP transformed data and experiment with hyperparameters.
Used UMAP with n_components as 20.
DBSCAN:

```
eps = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 5.0, 10.0, 25.0, 50.0]
min_samples = [5,10,30,60,75,100,200,500,1000]

Best in terms of Average Score:
Best eps for DBSCAN: 0.7
Best Min Samples for DBSCAN: 200
Best Score for DBSCAN: 0.48828953640917516

Best in terms of Adjusted Rand Index score:
Best eps for DBSCAN: 0.3
Best Min Samples for DBSCAN: 10
Best Score:0.31800441281345015
```

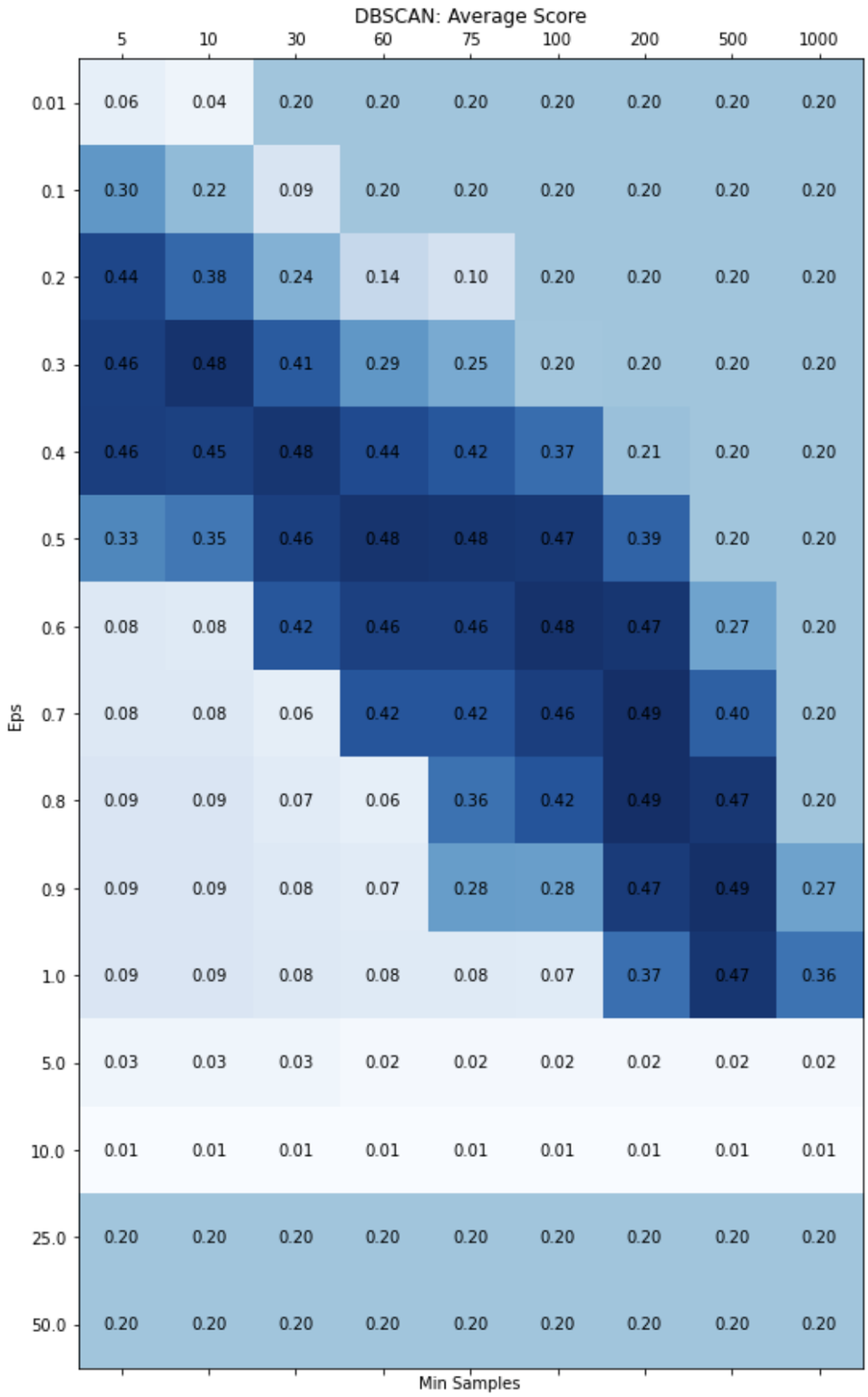

Figure 26: Average Score over various values of epsilon and min samples

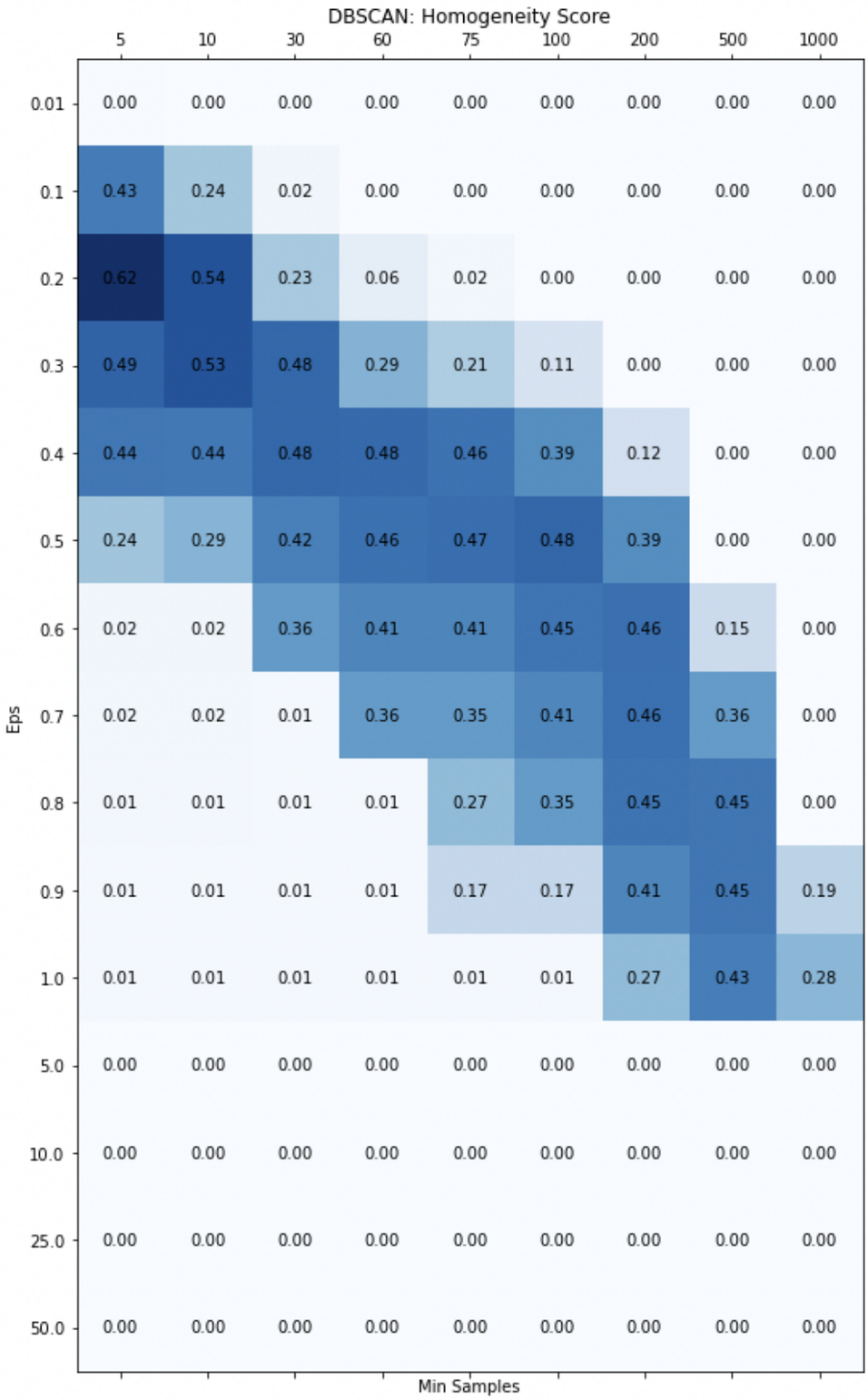

Figure 27: Homogeneity Score over various values of epsilon and min samples

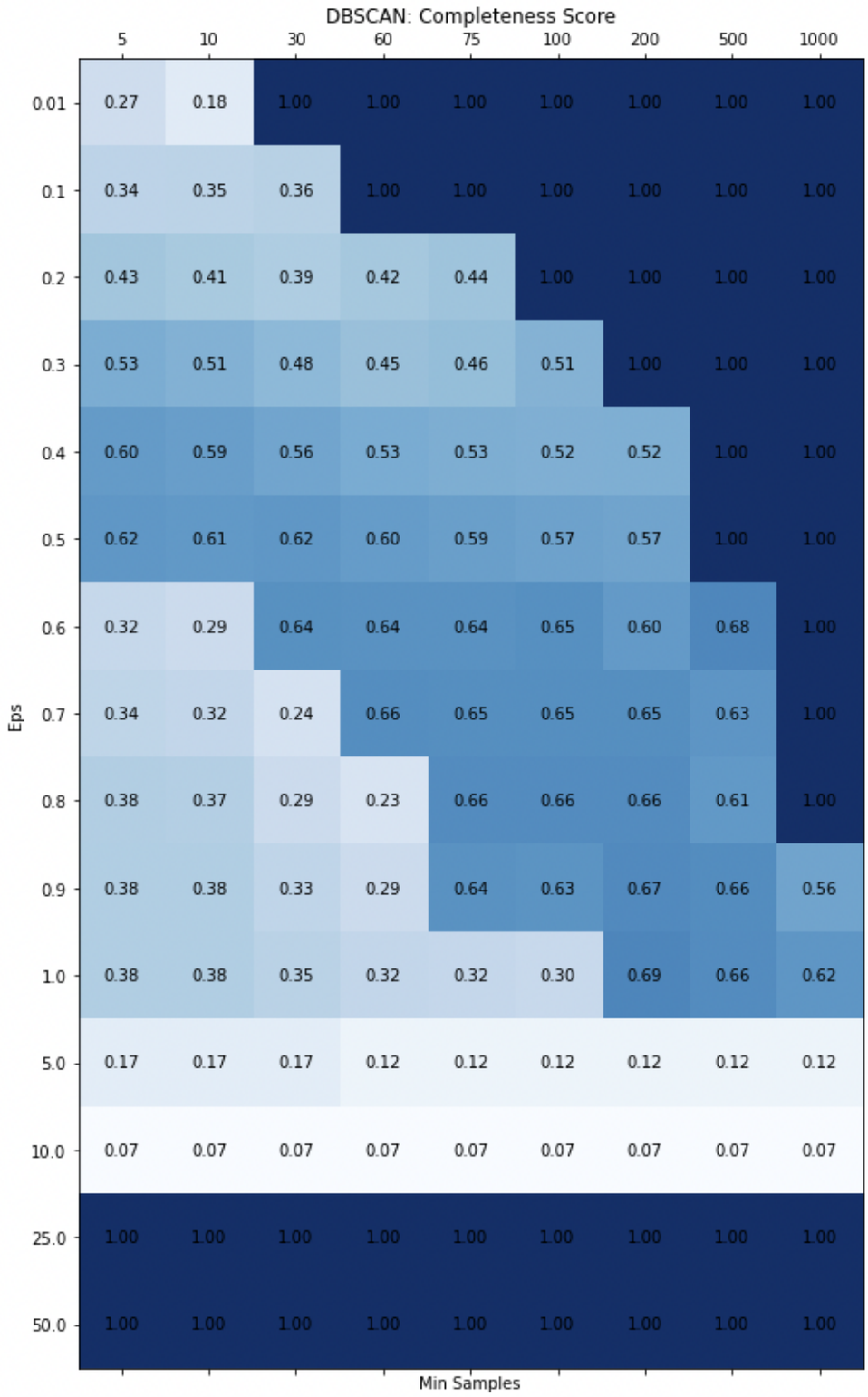

Figure 28: Completeness Score over various values of epsilon and min samples

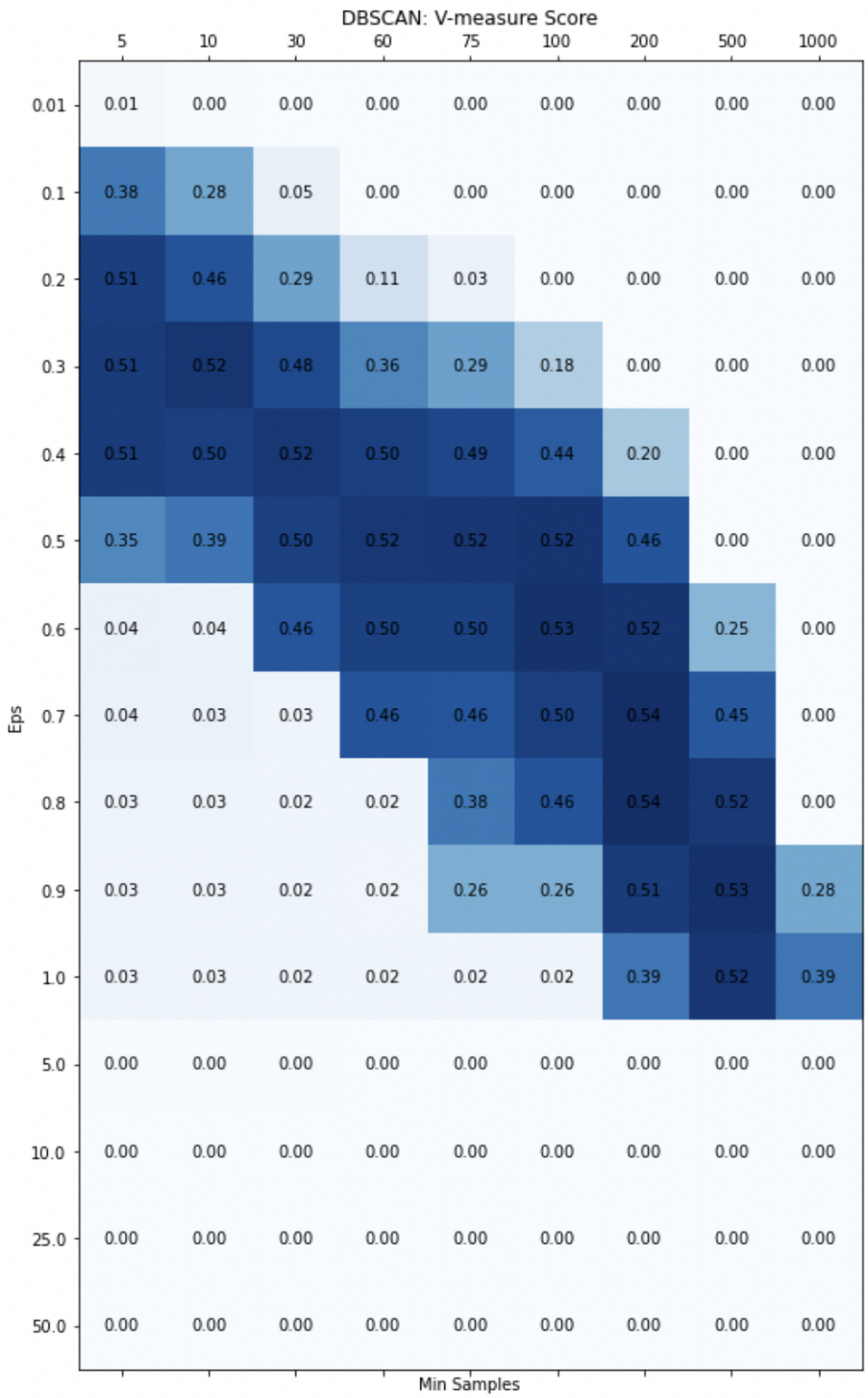

Figure 29: V-Measure Score over various values of epsilon and min samples

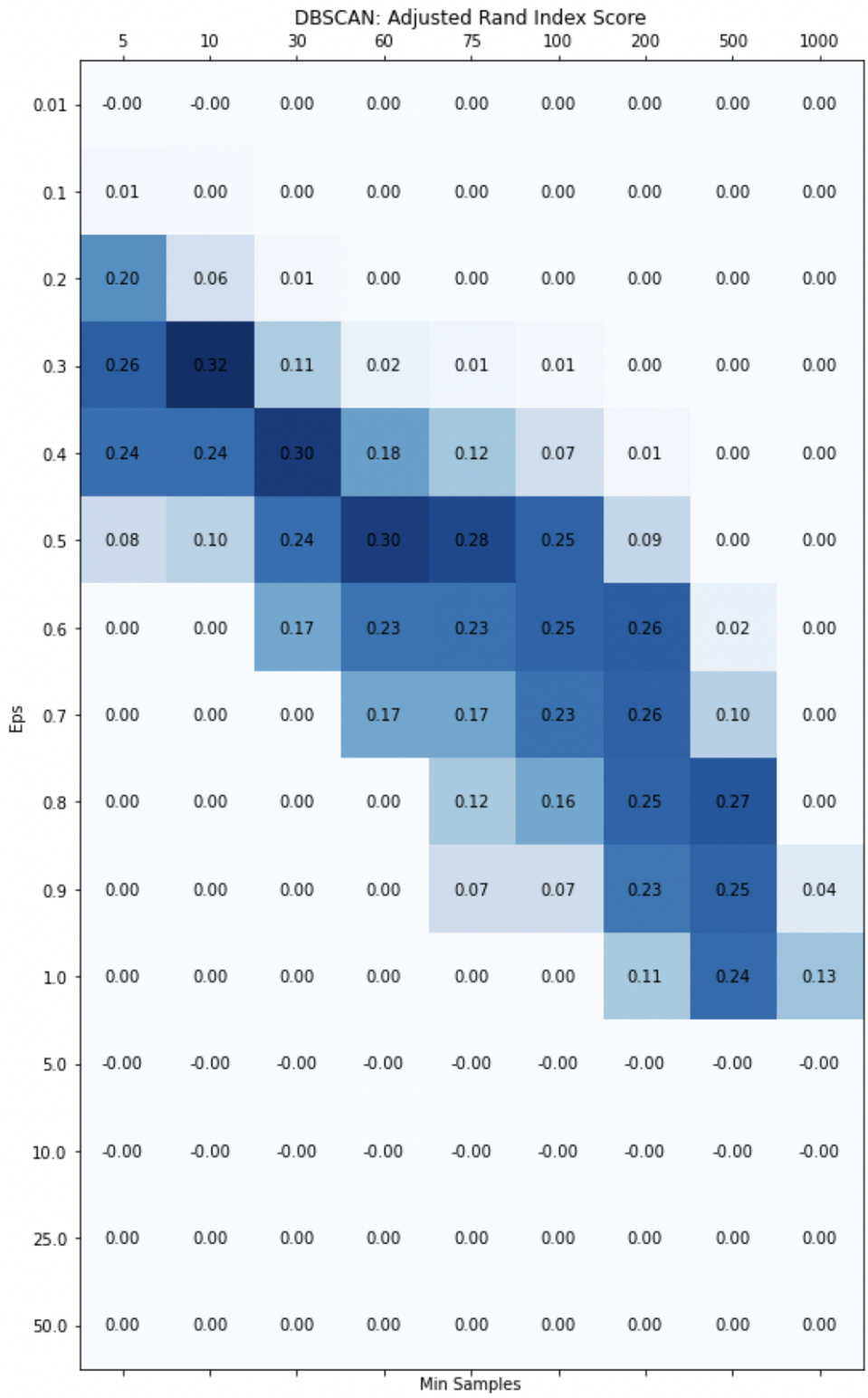

Figure 30: Adjusted Rand Index Score over various values of epsilon and min samples

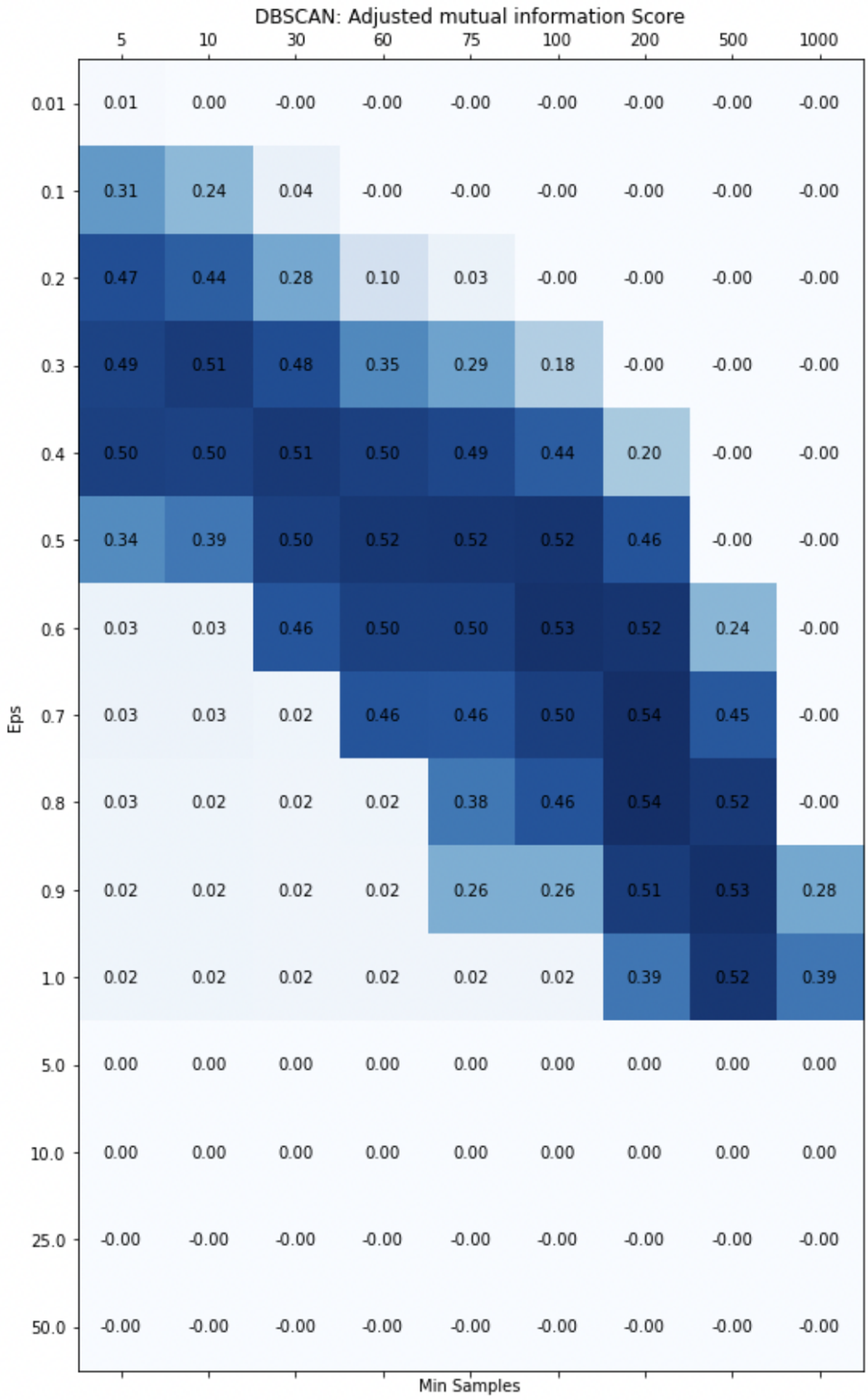

Figure 31: Adjusted Mutual Information Score over various values of epsilon and min samples

eps = 0.7, min_samples = 200:

```
Homogeneity score for DBSCAN: 0.458913
Completeness score for DBSCAN: 0.650827
V-measure score for DBSCAN: 0.538276
Adjusted Rand Index score for DBSCAN: 0.256169
Adjusted mutual information score for DBSCAN: 0.537263
```

eps = 0.3, min_samples = 10:

```
Homogeneity score for DBSCAN: 0.533378
Completeness score for DBSCAN: 0.511711
V-measure score for DBSCAN: 0.522320
Adjusted Rand Index score for DBSCAN: 0.318004
Adjusted mutual information score for DBSCAN: 0.511704
```

We were asked to restrict the epsilon range to 0.5 and 5.0. The best hyperparameters given this range were:

```
Best in terms of Average Score:
Best Score for DBSCAN: 0.4773516179716598
Best eps for DBSCAN: 0.5
Best Min Samples for DBSCAN: 60

Best in terms of Adjusted Rand Index score:
Best Score:0.2985271584692788
Best eps for DBSCAN: 0.5
Best Min Samples for DBSCAN: 60
```

eps = 0.5, min_samples = 60:

```
Homogeneity score for DBSCAN: 0.456488
Completeness score for DBSCAN: 0.597696
V-measure score for DBSCAN: 0.517635
Adjusted Rand Index score for DBSCAN: 0.298527
Adjusted mutual information score for DBSCAN: 0.516412
```

HDBSCAN:

```
eps = [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 5.0, 10.0, 25.0, 50.0]
min_samples = [5,10,30,60,75,100,200,500,1000]

Best in terms of Average Score:
Best eps for HDBSCAN: 0.3
Best Min Samples for HDBSCAN: 5
Best Score for HDBSCAN: 0.469018642279162

Best in terms of Adjusted Rand Index score:
Best eps for HDBSCAN: 0.3
Best Min Samples for HDBSCAN: 5
Best Score:0.29270045478535195
```

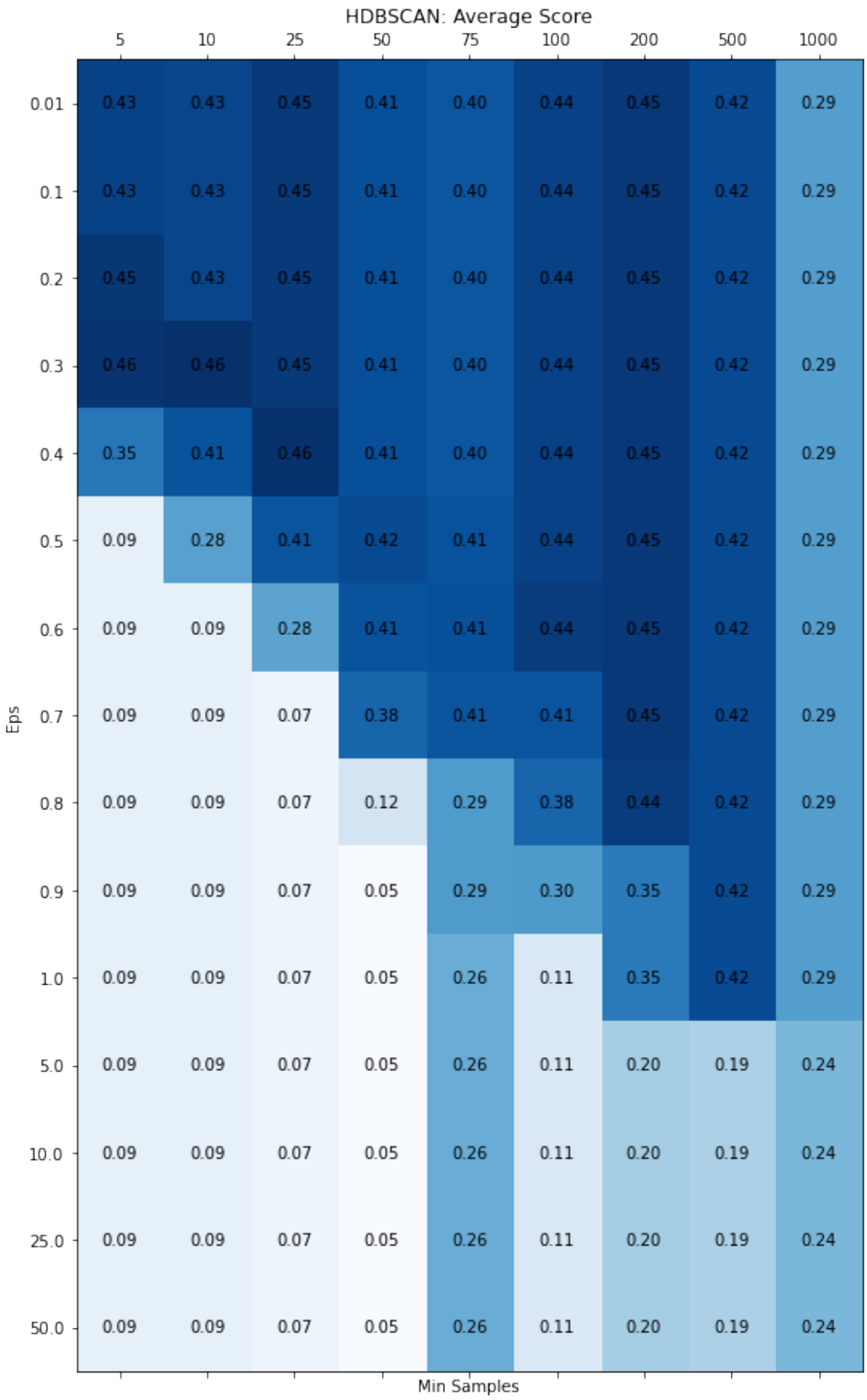

Figure 32: Average Score over various values of epsilon and min samples

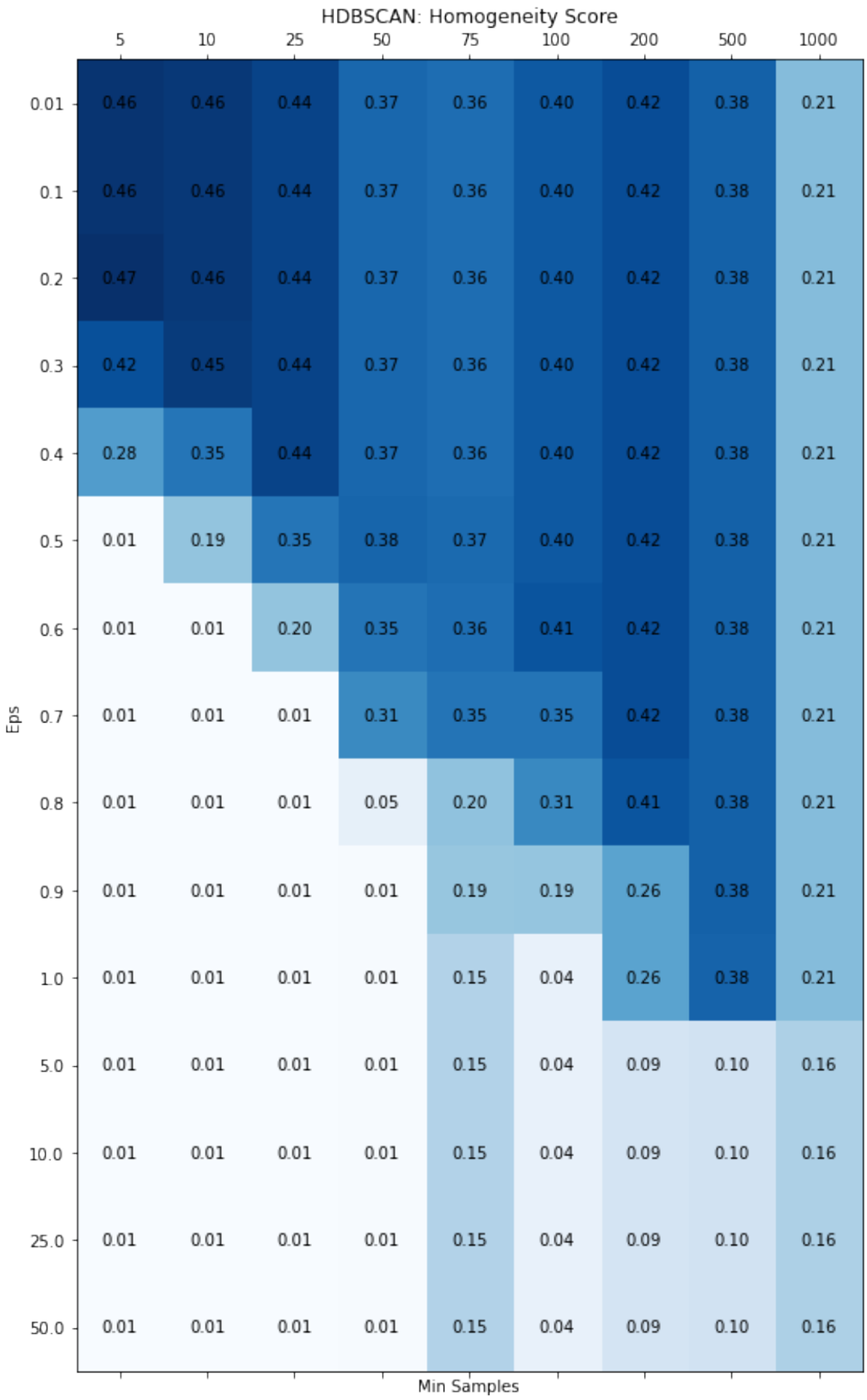

Figure 33: Homogeneity Score over various values of epsilon and min samples

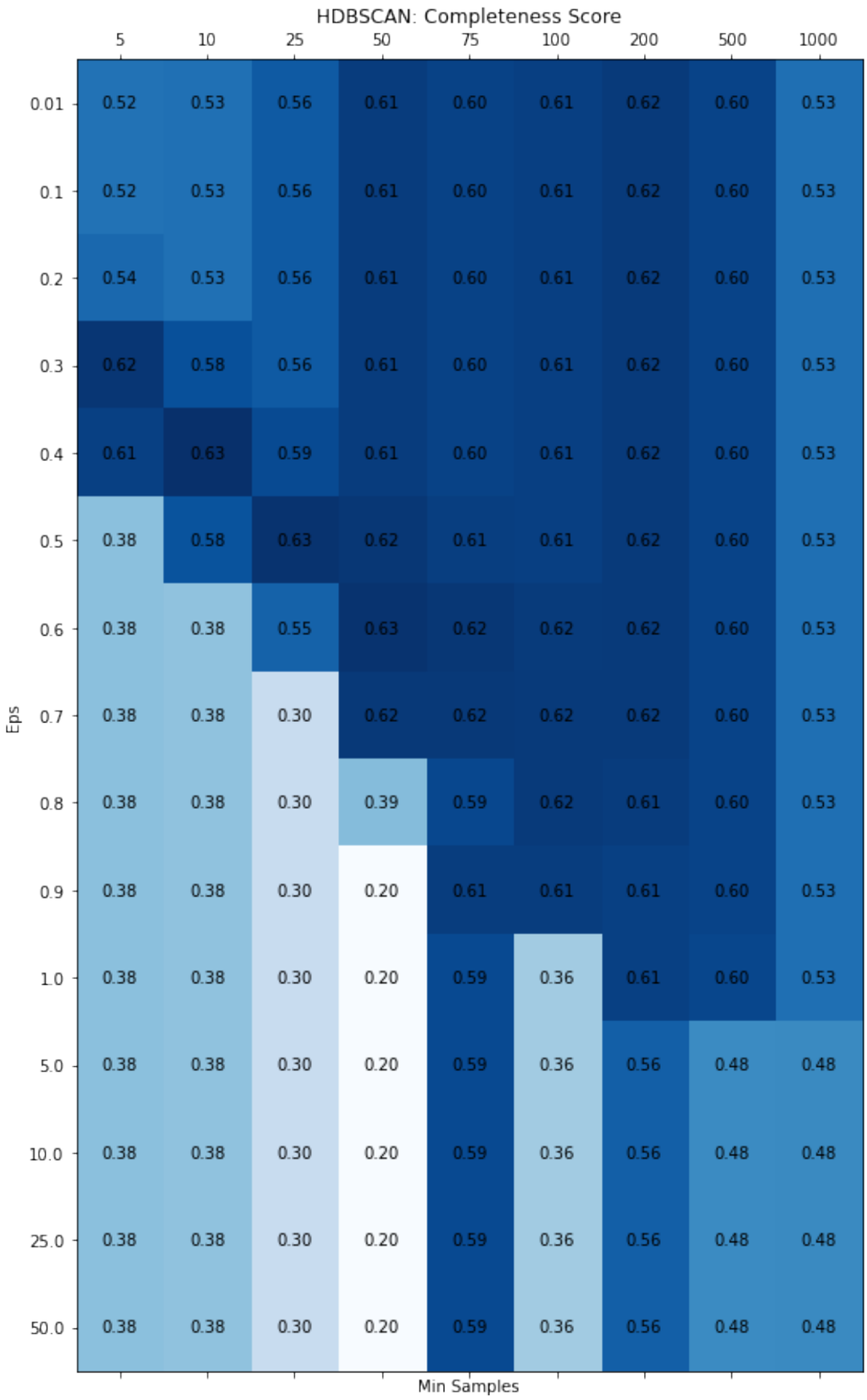

Figure 34: Completeness Score over various values of epsilon and min samples

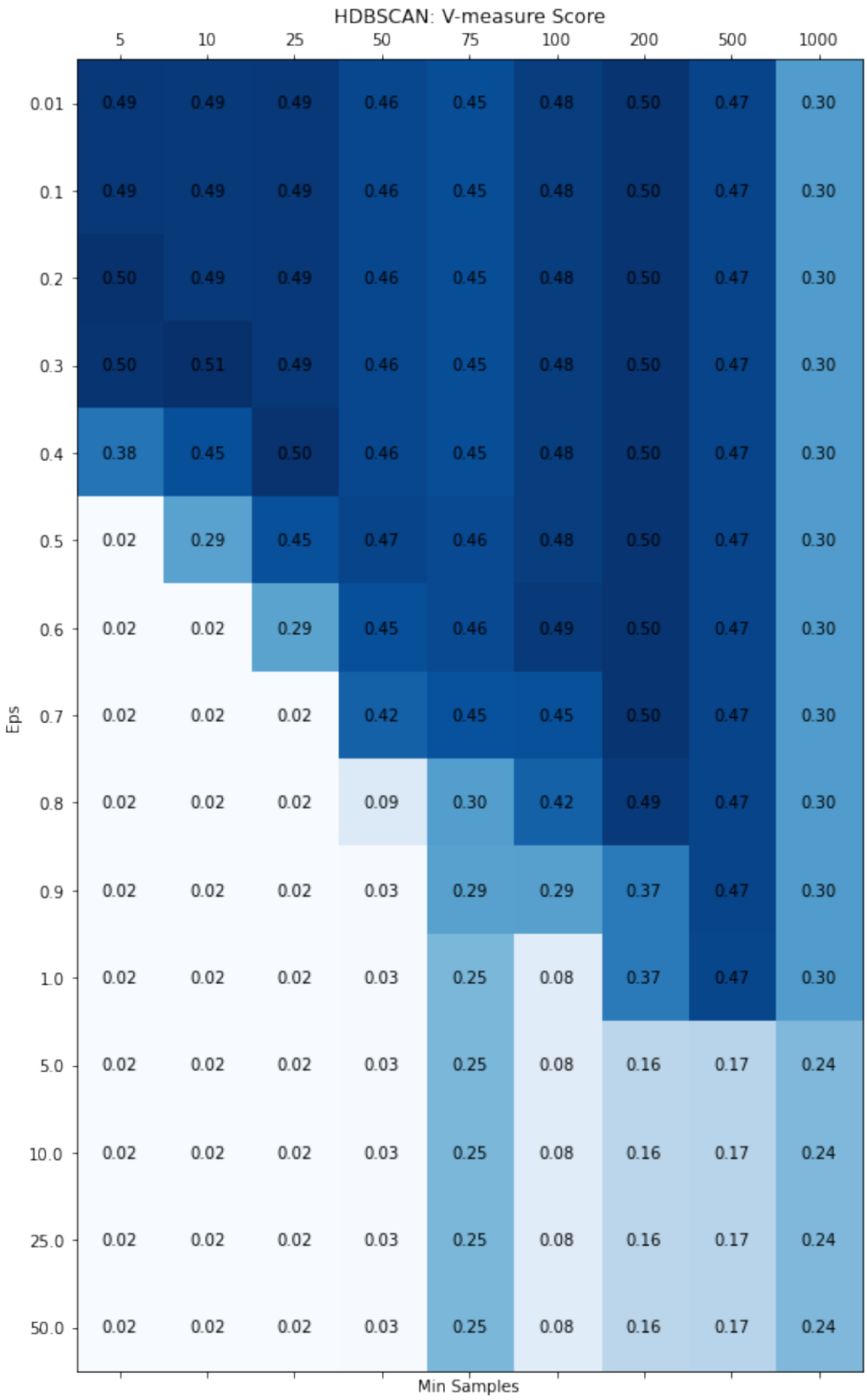

Figure 35: V-Measure Score over various values of epsilon and min samples

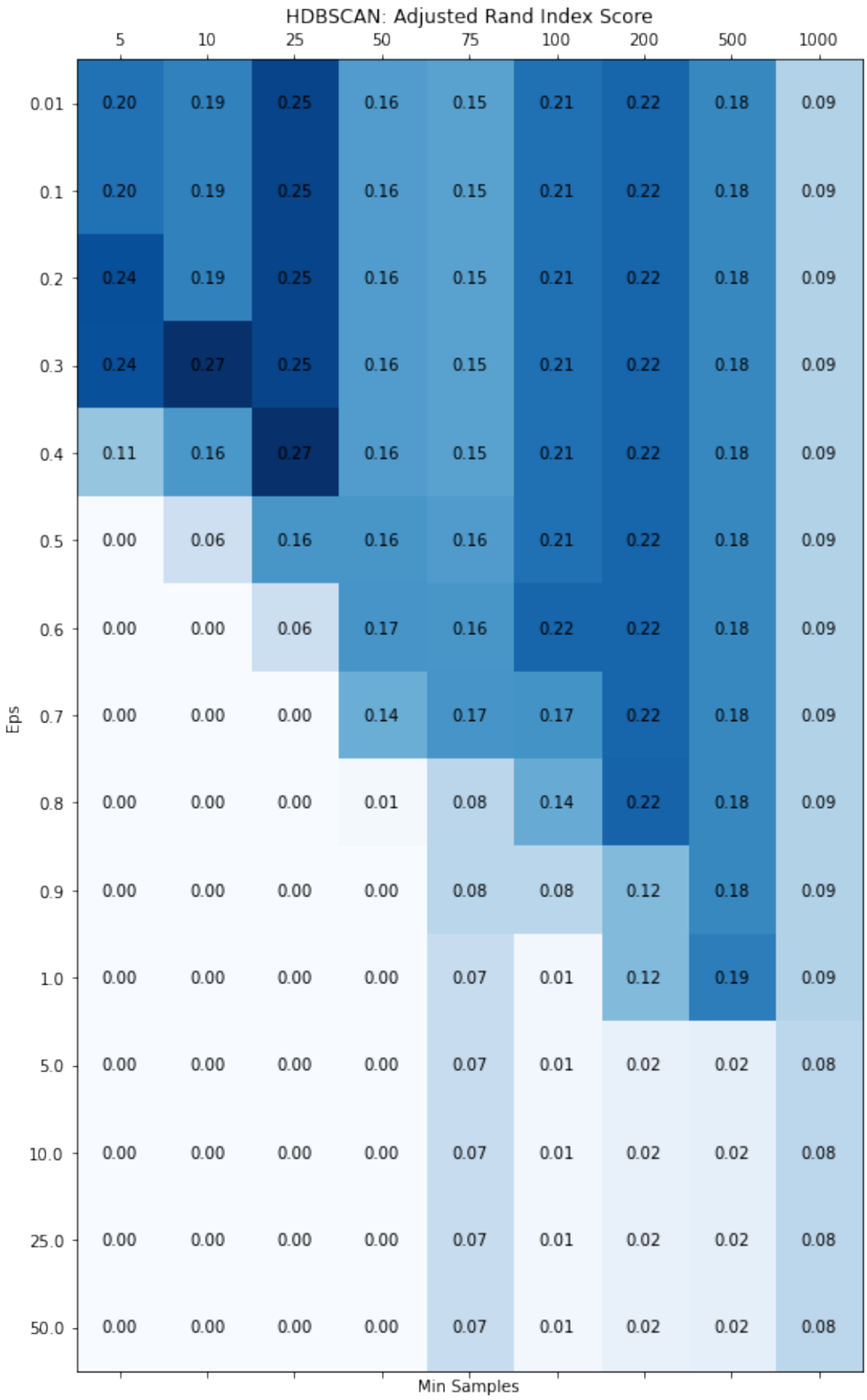

Figure 36: Adjusted Rand Index Score over various values of epsilon and min samples

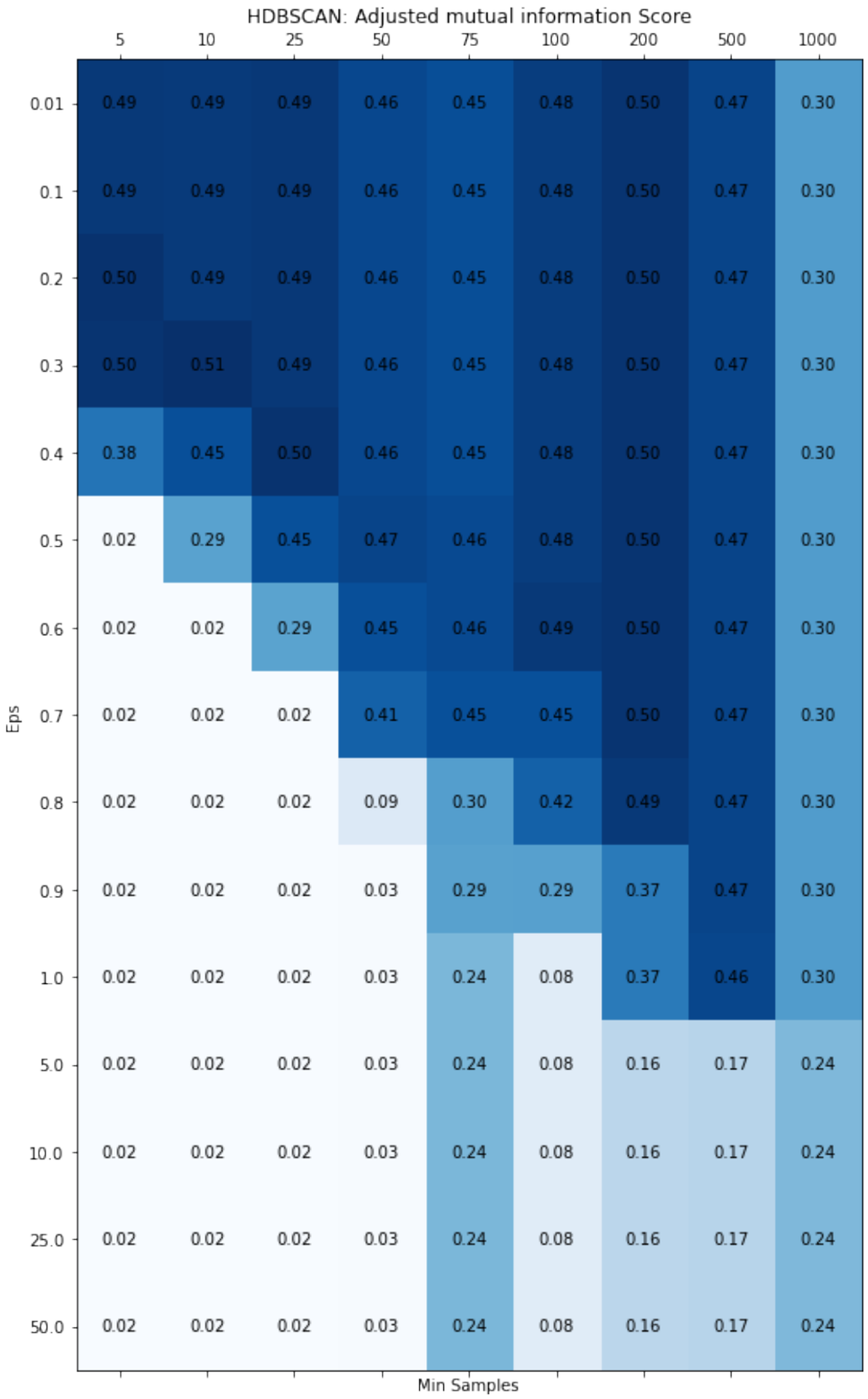

Figure 37: Adjusted Mutual Information Score over various values of epsilon and min samples

eps = 0.3, min_samples = 5:

```
Homogeneity score for HDBSCAN: 0.444601
Completeness score for HDBSCAN: 0.592694
V-measure score for HDBSCAN: 0.508076
Adjusted Rand Index score for HDBSCAN: 0.292700
Adjusted mutual information score for HDBSCAN: 0.507022
```

We were asked to restrict the epsilon range to 0.5 and 5.0. The best hyperparameters given this range were:

```
Best in terms of Average score:
Best Score for HDBSCAN: 0.4657999560785793
Best eps for HDBSCAN: 0.5
Best Min Samples for HDBSCAN: 60

Best in terms of Adjusted Rand Index score:
Best Score:0.24054549884581522
Best eps for DBSCAN: 0.5
Best Min Samples for DBSCAN: 60
```

eps = 0.5, min_samples = 60:

```
Homogeneity score for HDBSCAN: 0.426188
Completeness score for HDBSCAN: 0.639910
V-measure score for HDBSCAN: 0.511626
Adjusted Rand Index score for HDBSCAN: 0.240545
Adjusted mutual information score for HDBSCAN: 0.510731
```

We do not mention the number of clusters for DBSCAN or HDBSCAN and hence the number of clusters formed can be larger or smaller than expected unlike k-means or Agglomerative Clustering. As Adjusted Ran Index Score consider pairwise elements in the clusters to estimate the cluster equivalent of accuracy, these models perform poorly there. But they perform well in the other metrics. As the number of clusters are lesser than actual classes, Completeness score is pretty high. It just measure if the elements of the same class are clustered together.

The two most important hyperparameters for both algorithms are epsilon and minimum samples per cluster. If epsilon is too small, most of the samples will be treated as outliers, while a large value erroneously merges different clusters together. With increasing minimum samples per cluster we can observe that the cluster count decreases. This can observed with the epsilon = 0.3 and min_samples = 10 case. We observed over 200 clusters being formed for this case in DBSCAN. On the other hand epsilon = 0.7 and min_samples = 200 gave a mere 13 clusters.

From the 5 clustering metrics, we see that both HDBSCAN and DBSCAN performed very similarly without much differences. This is expected because the only difference between HDBSCAN and DBSCAN is the ability to deal with varying density clusters. Based on the results, it is fair to assume that our data did not have varying cluster densities and hence the final results are fairly similar.

When restricted to epsilon 0.5 and 5,
In case of both DBSCAN and HDBSCAN, the estimated best values of epsilon and min samples turns out to be eps = 0.5, min_samples = 60.

Question 16

Plot the contingency matrix for the best clustering model from Question 15. How many clusters are given by the model? What does "-1" mean for the clustering labels? Interpret the contingency matrix considering the answer to these questions.

DBSCAN needs a minimum cluster size and a distance threshold epsilon as user-defined input parameters, HDBSCAN is basically a DBSCAN implementation combined with hierarchical clustering attributes.

The main disadvantage of DBSCAN is that is much more prone to noise, which may lead to false clustering. On the other hand, HDBSCAN focus on high density clustering, which reduces this noise clustering problem and allows a hierarchical clustering based on a decision tree approach

Basically, DBSCAN is known to work well with non convex and non-isotropic ensuring stable performance across run with datasets having large variance. This is because, HDBSCAN operates based on the heuristic that all closely related samples in the feature space are closely packed with distinct areas of separation amidst them. DBSCAN uses euclidean metric to combine points that that are closer in space to each other. DBSCAN considers points lying in sparsely populated regions as obvious outliers or undefined noise. **Hence, they segregate all those undefined points and do not include them as part of any other clusters, this is characeterized by the -1 labelled cluster in the contingency table which essentially indicates the noise factor).** In one sense, DBSCAN is intertwined with agglomerative clustering as it uses single linkage criteria to create dendrograms once it completes transforming the feature space according to data density. But still, agglomerative clustering is only a form of partitioning and does not take into account, the absolute data density metric.

HDBSCAN addresses the limiting factors of DBSCAN and extends its capabilities along those lines. It essentially translates DBSCAN into a hierarchical clustering algorithm, with the goal of still permitting varying density clusters. The underlying operational framework is very similar, except that the formed dendrograms are segregated at different variations of heights to selectively choose stable clusters. This eventually results in a smaller tree with lesser number of clusters that lose out certain points.

The chosen values for hyperparameters training include:

```
eps= [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 5.0, 10.0, 25.0, 50.0]
min_samples = [5,10,30,60,75,100,200,500,1000]
```

DBSCAN:
In case of DBSCAN, the estimated best values of epsilon and min samples turns out to be eps = 0.5, min_samples = 60:

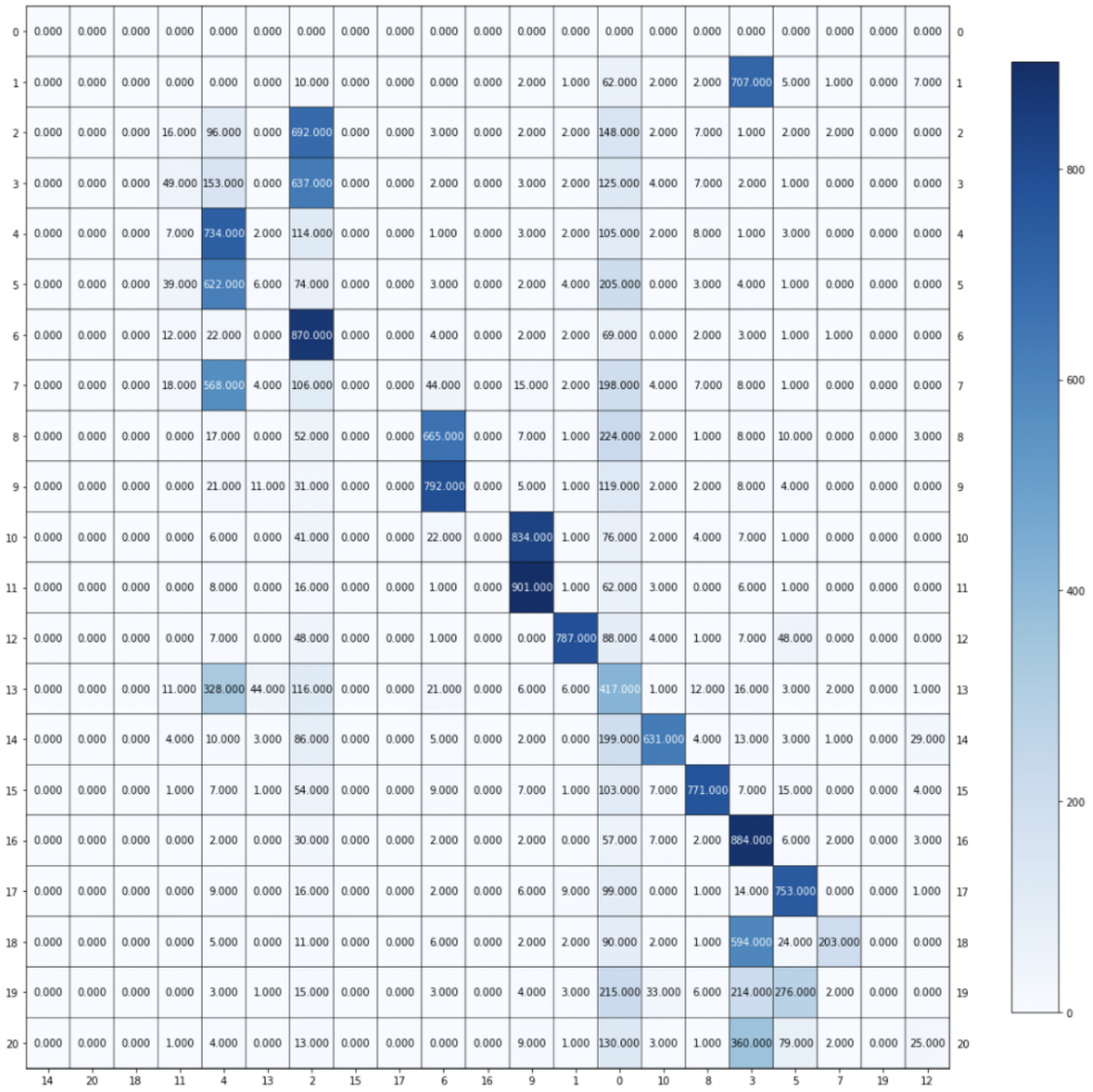

Figure 38: DBSCAN (eps = 0.5, min_samples = 60) Contingency Matrix

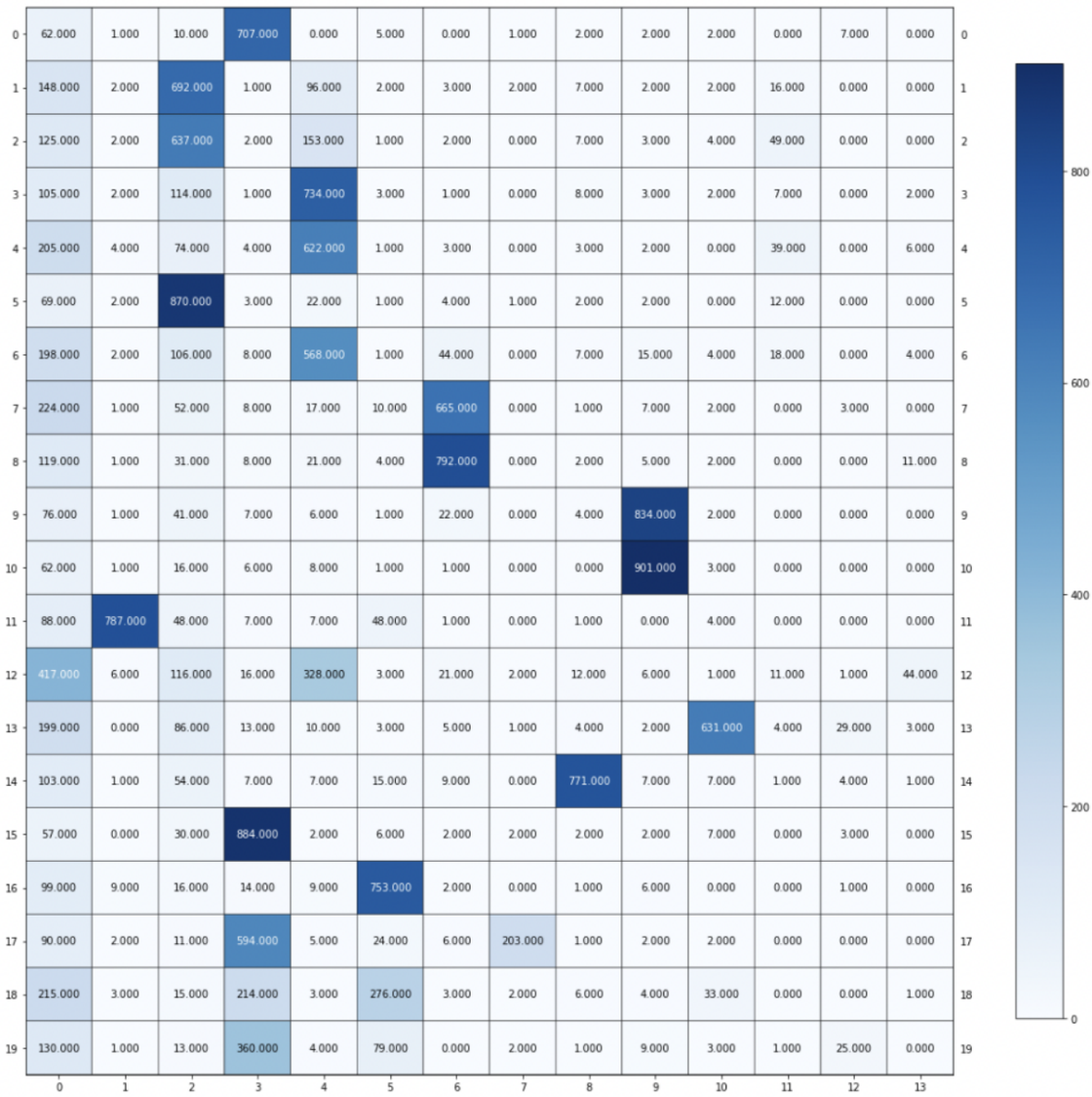

Figure 39: DBSCAN (eps = 0.5, min_samples = 60) Non-permuted Contingency matrix with all zero columns removed

```
Estimated number of clusters: 13
Estimated number of noise points: 2791
```

HDBSCAN:

In case of HDBSCAN, the estimated best values of epsilon and min samples turns out to be: eps = 0.5, min_samples = 60:

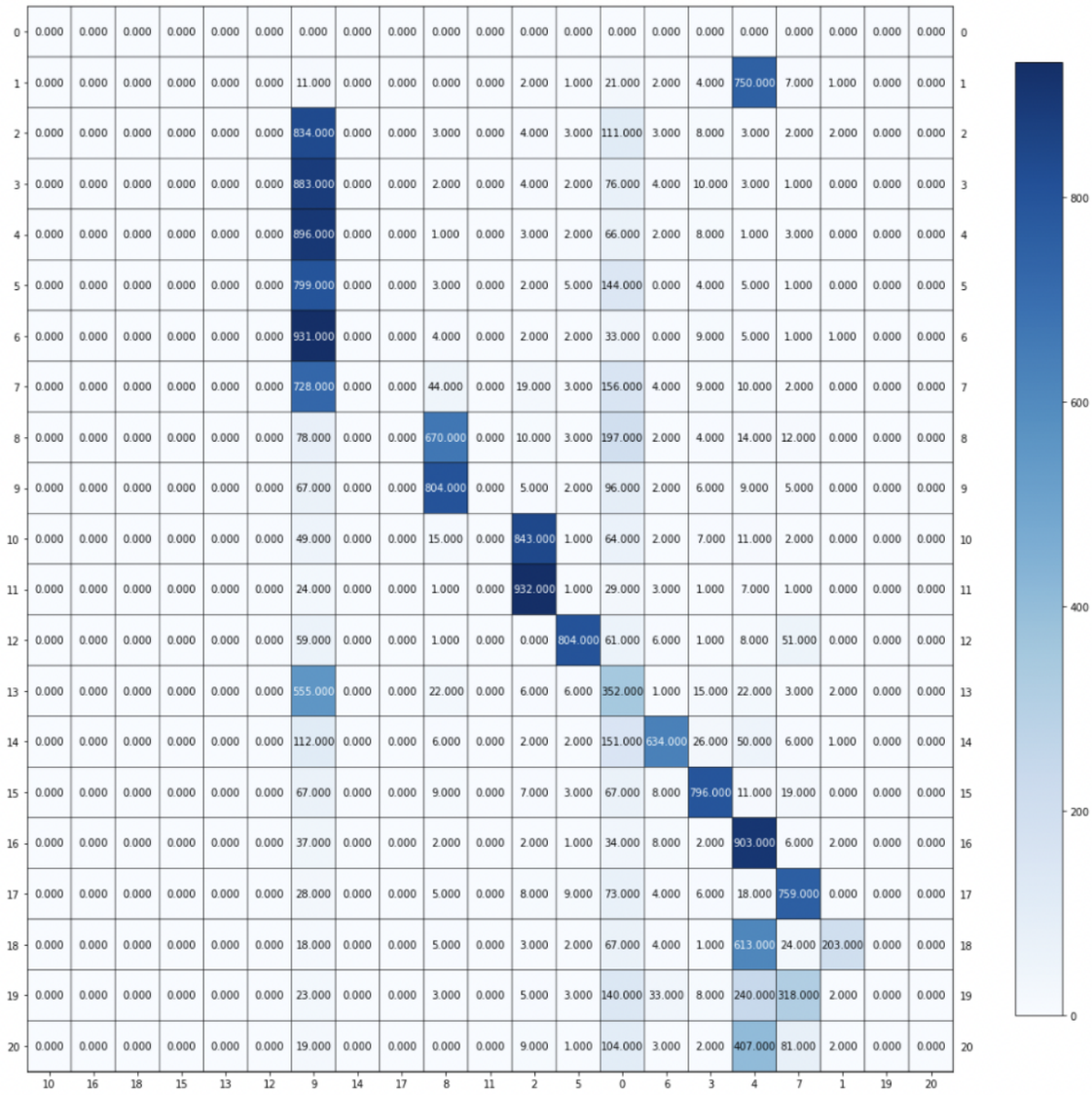

Figure 40: HDBSCAN (eps = 0.5, min_samples = 60) Contingency Matrix

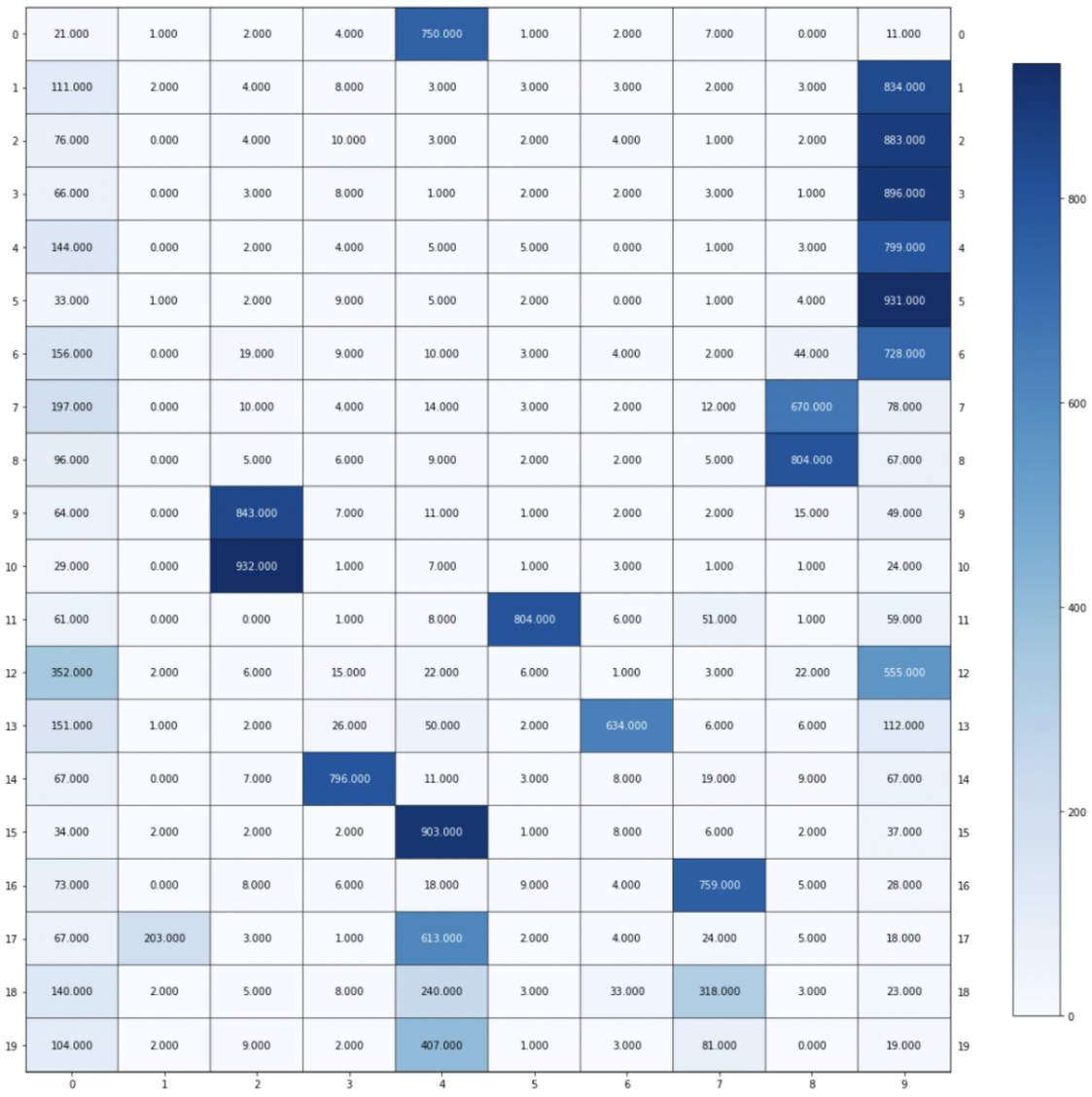

Figure 41: HDBSCAN (eps = 0.5, min_samples = 60) Non-permuted Contingency matrix with all zero columns removed

```
Estimated number of clusters: 9
Estimated number of noise points: 2042
```

DBSCAN has created 13 clusters without including the '-1' class. -1 is the label for all the outliers or noise data. HDBSCAN created 9 clusters excluding the outliers. These can be observed from the contingency matrix as well. Most of the columns are all empty because the number of clusters are lesser than the number of true labels/classes in the dataset(20). Therefore, we can see a much smaller prominent diagonal as well. Moreover one of the columns is the cluster -1, which is basically a collection of all outliers. We have removed the 'all 0' columns to obtain the rectangular representation of the contingency matrix. This highlights how the output clusters are lesser. This is one major reason why the metrics are low for DBSCAN and HDBSCAN. We do not mention the number of clusters and hence the number of clusters formed can be larger or smaller than expected unlike k-means or Agglomerative Clustering. As Adjusted Rand Index Score consider pairwise elements in the clusters to estimate the cluster equivalent of accuracy, these models perform poorly there.

The problem here is that noise is not a cluster and almost all metrics assume the data is strictly partitioned into disjoint clusters. Most implementations will then evaluate noise like

a cluster, and the result will appear much worse than it is. There is a density based cluster evaluation metric called DBCV which might be a better metric.

Comparing the metrics, we see that both DBSCAN and HDBSCAN similarly without any drastic differences in the clustering performance metrics. We do not see HDBSCAN perform superior when compared with DBSCAN. This is because of the fact that HDBSCAN's robust performance is based on its ability to deal with irregular and fluctuating cluster densities. If we do not have clusters exhibiting such variations, we could not reap the additional performance offered by HDBSCAN. Hence, it turns out that, in this case, HDBSCAN performs similar to DBSCAN algorithm.

Question 17

Based on your experiments, which dimensionality reduction technique and clustering methods worked best together for 20-class text data and why? Follow the table below.

Hint: DBSCAN and HDBSCAN do not accept the number of clusters as an input parameter. So pay close attention to how the different clustering metrics are being computed for these methods.

We are comparing based on Adjusted Rand Index Score.// Top 5 Adjusted Rand Index scores:

1. 0.467275

```
UMAP(metric='cosine',n_components=200)
KMeans(max_iter=1000, n_clusters=20, n_init=30, random_state=0)
```

2. 0.466947

```
UMAP(metric='cosine',n_components=20)
KMeans(max_iter=1000, n_clusters=20, n_init=30, random_state=0)
```

3. 0.442801

```
UMAP(metric='cosine',n_components=5)
KMeans(max_iter=1000, n_clusters=20, n_init=30, random_state=0)
```

4. 0.426737

```
UMAP(metric='cosine',n_components=20)
AgglomerativeClustering(n_clusters=20)
```

5. 0.419747

```
UMAP(metric='cosine',n_components=5)
AgglomerativeClustering(n_clusters=20)
```

Observations:

- Sparse representation provides the least scores (12%). This is expected as the dataset is very large in 20 classes as compared to the 2 classes we saw earlier. Therefore, in this case the large sparse data causes a lot of noise which makes it difficult for any algorithm to cluster or classify the data points accurately.
- We observe that SVD and NMF (both 15%) perform poorly across all the various clustering techniques. UMAP outperforms all other feature reduction techniques considerably.
- We observe that UMAP provides the best results (46.7% in Adjusted Rand Index Score). This is due to the fact that SVD and NMF are least squares techniques and enormously liable to outliers and noise as compared to UMAP. Moreover, both SVD and NMF are linear projections of the large dimensional features which maximize the variance of the dataset, however are not able to capture nonlinear dependencies

in the data. UMAP, on the other hand, tries to ensure that the semantic and non-linear dependencies, in addition to global structure of the data is preserved such that similar embeddings are clustered collectively and different ones as far apart as possible. This aids the clustering innately. In fact, as UMAP tries to preserve the information entropy regardless of the dimensions, the overall performance of UMAP is consistent throughout a huge range of r.

- UMAP is a local-density based dimensionality reduction method. It gives a more flexible interpretation and works fine even if the data is without any prior distribution. PCA and NMF treat all clusters as a whole and therefore can lose local structure. This is something UMAP tries to overcome. UMAP scale better to large sample size and higher dimensionality.

- Now let us understand the various clustering algorithms' performance. Both Hierarchical clustering methods and Density based methods in general do not scale well with the large dataset and large dimensions. Considering that we have a high dimensional data over 20 classes, they perform poorly.

- Moreover, DBSCAN doesn't work well as the clusters have varying density (as observed in the figure below) and that the data is high dimensional.

- Agglomerative clustering being a hierarchical clustering method suffers from the issue of large dimension. But considering that the various clusters are fairly equal sized, this does work fairly well and provides good results.

- HDBSCAN suffers from the same shortcomings faced by DBSCAN and this affects its result considerably.

- The figure below shows the UMAP dimensional reduced data further reduced to 2 dimensional space using UMAP - Cosine distance metric. From the figure we can observe that the densities of the various clusters are different. They form distinct spherical structures that are well differentiated. The cluster sizes are fairly equal and the various clusters have a convex shape. Therefore it makes sense that k-means work perfectly on them. k-means also scale very well to large datasets.

- Finally, it makes sense that kmeans with number of clusters = 20 outperforms all other scenarios as our dataset has 20 classes.

- DBSCAN and HDBSCAN can have more or less than 20 clusters formed as the classes are not mentioned as parameters. As Adjusted Rand Index Score consider pairwise elements in the clusters to estimate the cluster equivalent of accuracy, these models perform poorly there. But they perform well in the other metrics. If the number of clusters are lesser than actual classes, Completeness score can be pretty high. It just measure if the elements of the same class are clustered together. If clusters are more, Homogeneity can increase. Therefore other metrics need to be considered to truely understand how good these measures. This is also evident from the poorer performance of K-means when number of clusters are set to 10 or 50.

- Moreover there is an outlier class -1 mentioned as well. These can affect the metrics calculated as the classes count don't match the cluster count. This could also be one reason why their values are significantly lower than those of k-means and agglomerative clustering (Number of clusters are passed for them). The problem with most metrics

is that noise is not a cluster and it assumes the data is strictly partitioned into disjoint clusters. Most implementations will then evaluate noise like a cluster, and the result will appear much worse than it is. There is a density based cluster evaluation metric called DBCV which might be a better metric.

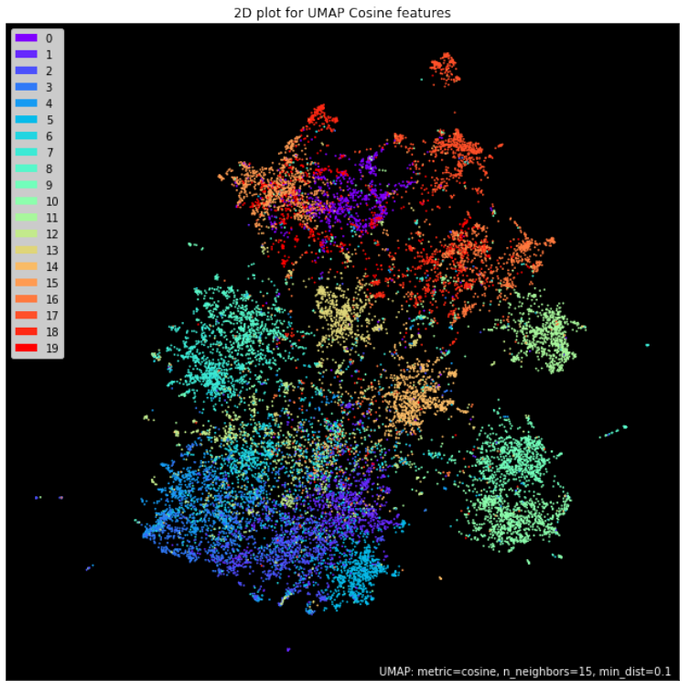

Figure 42: UMAP Reduced to 2 dimensions

Note: HDBSCAN with no feature extraction was not running successfully and hence was skipped.

Question 18

Bonus. If you can find creative ways to further enhance the clustering performance, report your method and the results you obtain.

The method we tried and the Adjusted Rand Index Score are as follows:

Method 1: Clean Data and Lemmatize
We tried to lemmatize the data and clean the data by removing numbers and punctuations. Though the results were competetive, it did not beat the 46.7% from above.
Results: 0.44558165545848627

Method 2: GloVe Embeddings
Instead of using Tf-Idf matrix which is just the word counts, we wanted to make use of the word semantics to enhance their embeddings. But the GloVe Embeddings when mapped onto 2 dimensions was further evidence that this method would not succeed.

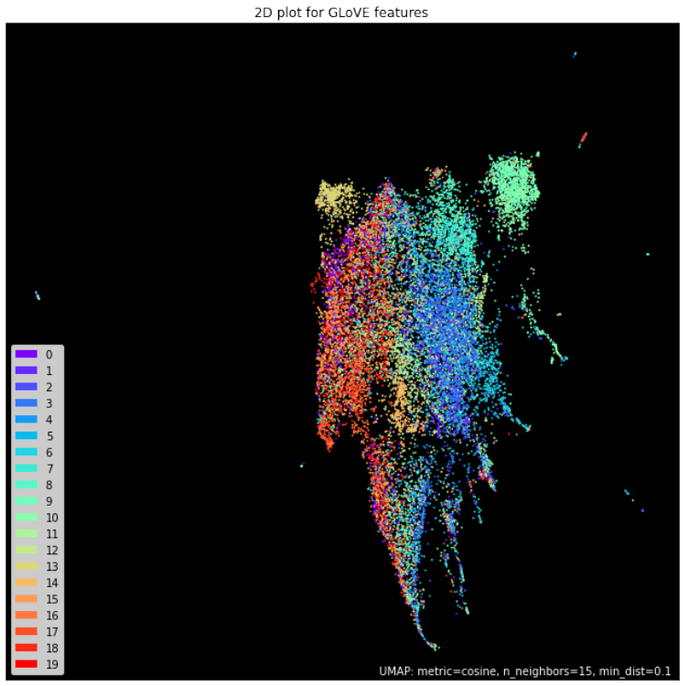

Figure 43: 2D plot of GloVe Features

Method 3: BERT Embeddings
GloVe embeddings doesn't make use of the context. So we tried to use a context-dependent embedding - the BERT Embeddings. We tried running this solution but due to the large dataset, huge dimensions and lack of processing power, the solution failed to run to completion.

Method 4: Gaussian Mixture Models
We tried to make use of GMM to cluster the documents.
Results: 0.4152672331773858

Method 5: Self Organizing Map

We tried to make use of SMM, which is a clustering and feature reduction solution.

Results: 0.2377693005090737

Method 6: Cluster Ensembles

We ran multiple different models. Different runs of kmeans, agglomerative clustering and GMM. This method then combines the outputs of various methods to give an ensembled result. We observe that the results are fairly good for this method.

Results: 0.4630521505516385

BEST METHOD:

Method 7: Normalization

We tried normalizing the tf-idf matrix before the dimensionality reduction. Then normalize the UMAP dimensionality reduced data before K-Means clustering. Normalizing data is especially useful before applying k-means. This is evident from the clear spike in accuracy. Applying normalization only before UMAP increased accuracy only to 47%. But applying before k-means as well enabled us to get our adjusted rand score to cross 50%. The results were varying from 47% to 51% in different runs of the code. But it has always performed better than the best case scenario from Q17.

Results: 0.5114313619252064

```
Homogeneity score for Normalization: 0.578952
Completeness score for Normalization: 0.584585
V-measure score for Normalization: 0.581755
Adjusted Rand Index score for Normalization: 0.511431
Adjusted mutual information score for Normalization: 0.580399
```

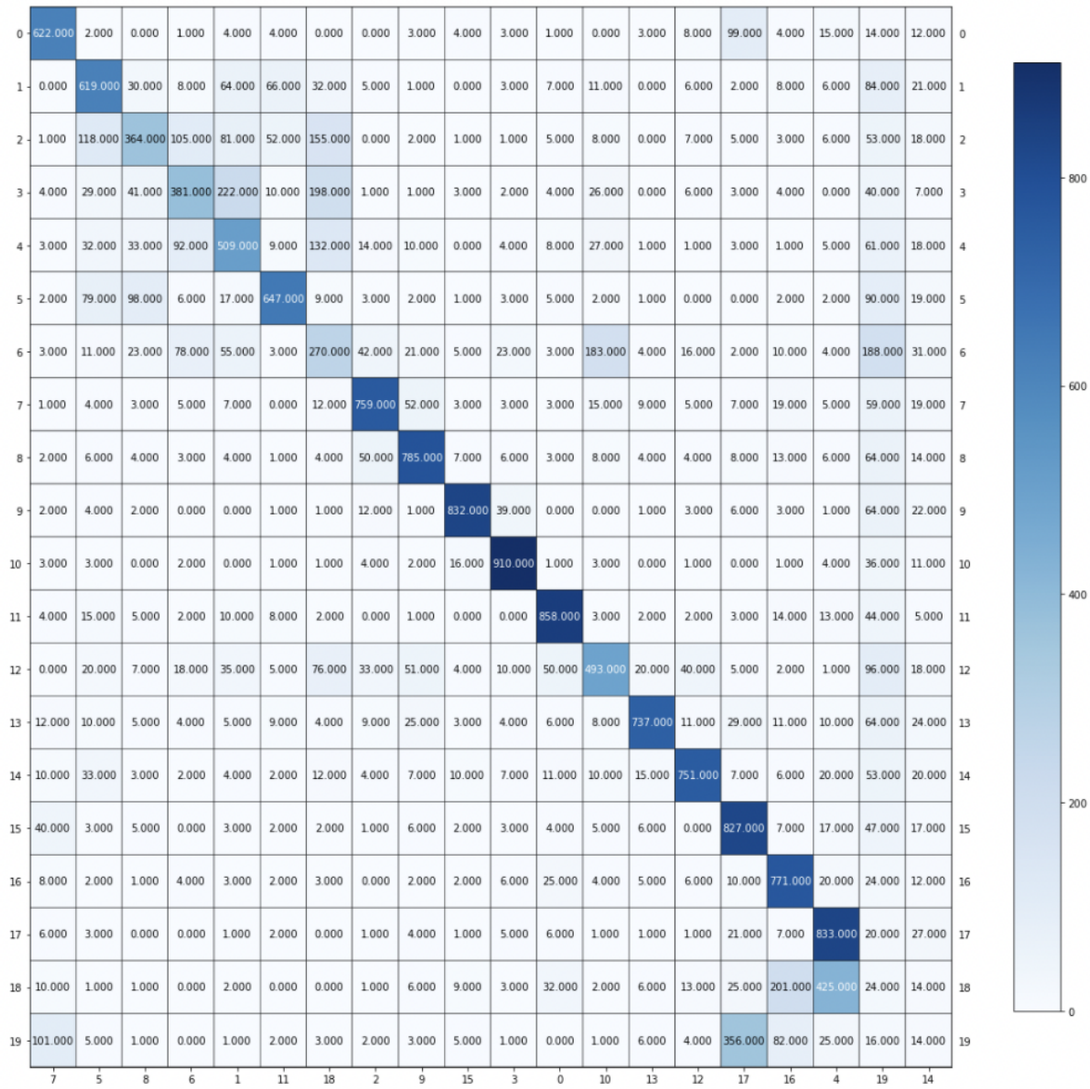

Figure 44: Normalization - Contingency Matrix

Question 19

In a brief paragraph discuss: If the VGG network is trained on a dataset with perhaps totally different classes as targets, why would one expect the features derived from such a network to have discriminative power for a custom dataset?
Discriminative power is a measure of the ability of a test to distinguish between two or more groups being assessed. We could use the VGG network trained on a dataset with different classes as target to train another dataset using the concept of Transfer learning.
Transfer learning (TL) is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. In transfer learning, we first train a base network on a base dataset and task, and then we repurpose the learned features, or transfer them, to a second target network to be trained on a target dataset and task. This process will tend to work if the discriminative features are general, meaning suitable to both base and target tasks, instead of specific to the base task. The idea is that if the tasks are similar enough, then the features at later layers of the network ought to be good features for this new task.
The ImageNet dataset consists of 1000 classes. The "daisy" class is present in both Imagenet and tf_flowers dataset. There are a few other flowers and plants present in the ImageNet dataset. The network can learn about the edges, corners of these and can help classify the flowers in our dataset. Imagenet is a good representation for classifying the tf_flowers dataset.

Question 20

In a brief paragraph explain how the helper code base is performing feature extraction.

The VGG16 model is loaded with the pretrained weights for the imagenet dataset. VGG16 model is a series of convolutional layers followed by one or a few dense (or fully connected) layers. From the input layer to the last max pooling layer (labeled by 7 x 7 x 512) is regarded as feature extraction part of the model, while the rest of the network is regarded as classification part of the model. After defining the model, we need to load the input image with the size expected by the model, in this case, 224×224. Next, the image PIL object needs to be converted to a NumPy array of pixel data and expanded from a 3D array to a 4D array with the dimensions of [samples, rows, cols, channels], where we only have one sample. The pixel values then need to be scaled appropriately for the VGG model.

```
List of VGG Features: Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace=True)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace=True)
  (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (18): ReLU(inplace=True)
  (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace=True)
  (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): ReLU(inplace=True)
  (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (25): ReLU(inplace=True)
  (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (27): ReLU(inplace=True)
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): ReLU(inplace=True)
  (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

The input to conv1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv layers (not all the conv layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

Usually, Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 5-way tf_flowers classification and thus contains 5 channels (one for each class). The final layer is the soft-max layer.

Question 21

How many pixels are there in the original images? How many features does the VGG network extract per image; i.e what is the dimension of each feature vector for an image sample?

The images in the Flowers dataset are not all the same size. The sizes for the first 5 images are printed below.

```
Image 1 shape: (333, 500, 3) label: 2
Image 2 shape: (212, 320, 3) label: 3
Image 3 shape: (240, 320, 3) label: 3
Image 4 shape: (240, 320, 3) label: 4
Image 5 shape: (317, 500, 3) label: 3
```

Figure 45: The input image size for the first 5 images

The total number of images in the dataset is 3670. The input image gets resized to 224 × 224 × 3, i.e., 150,5286 pixels, where W = 224, H = 224, and 3 stands for RGB channel.
The number of features extracted per image until the last layer = [512, 7, 7] = 25,088. (We ignored the batch size for the network).
The above features are then passed through a Fully connected layer to reduce the number of features. The number of features extracted in the last layer per image = 4096.

Question 22

Are the extracted features dense or sparse? (Compare with sparse TF-IDF features in text.)

The extracted features are dense as that have relatively high density and they have very small spaced intervals. We saw that as when we used the np.count_nonzero on the 4096 features, we get almost no zero values, which proves that the extracted features are actually dense, not sparse. On the other hand, the TF-IDF features in the text were sparse. There were a lot of non-zero entries in the TF-IDF matrix. This does make sense as the vocabulary across all documents would be much larger than the words present in a particular document. Setting min_df does help make it a bit more dense but when compared to the extracted features here, it is still very sparse.

Question 23

In order to inspect the high-dimensional features, t-SNE is a popular off-the-shelf choice for visualizing Vision features. Map the features you have extracted onto 2 dimensions with t-SNE. Then plot the mapped feature vectors along x and y axes. Color-code the data points with ground-truth labels. Describe your observation.

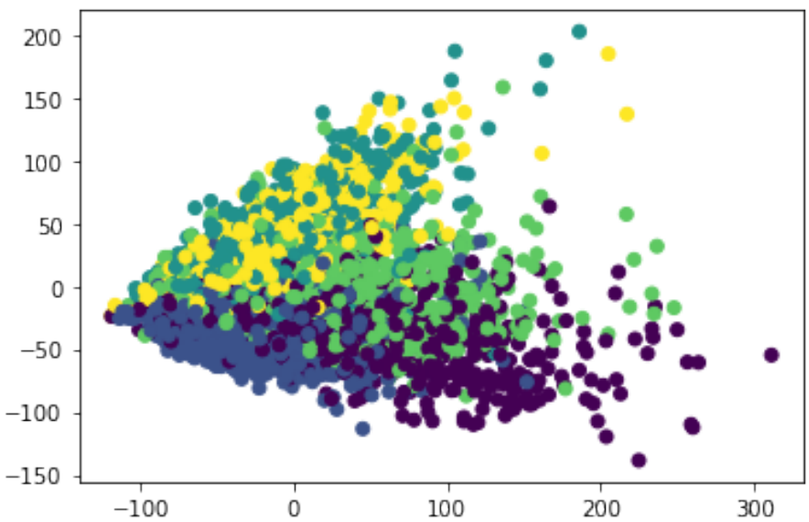

Figure 46: Scatter plot of mapped feature vectors for PCA

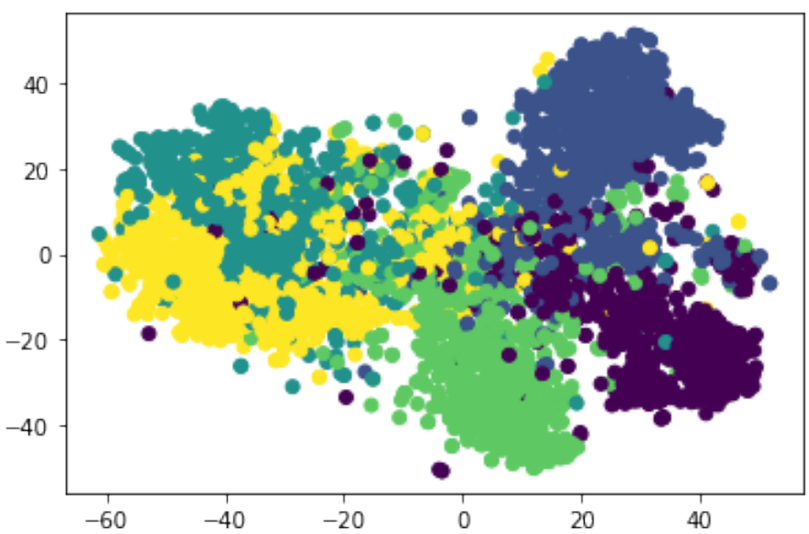

Figure 47: Scatter plot of mapped feature vectors for t-SNE

PCA is not able to cluster the data very well as the images have a lot of common features. PCA is a linear feature extraction method. But the data we have is obviously not linear and hence it makes sense that PCA is not the right dimensionality reduction method.
t-SNE stands for T- distributed Stochastic Neighbor Embedding. t-SNE is used for non-linear dimensionality reduction. It is an unsupervised, non-parametric method. Therefore, the data given to us is clustered better using t-SNE. We can notice the various clusters in t-SNE scatter plot. t-SNE finds the structure in the data where other dimensionality reduction techniques can't.

One of the most major differences between PCA and t-SNE is it preserves only local similarities whereas PCA preserves large pairwise distance maximize variance. It takes a set of points in high dimensional data and converts it into low dimensional data.

Question 24

Report the best result (in terms of rand score) within the table below. For HDBSCAN introduce your own reasonable grid over min cluster size and min samples.

We tried the below variations:

Module	Alternatives	Hyperparameters
Dimensionality Reduction	None	N/A
	SVD	r = 50
	UMAP	n_components = 50
	Autoencoder	num_features = 50
Clustering	K-Means	k = 5
	Agglomerative Clustering	n_clusters = 5
	HDBSCAN	min_cluster_size=2, min_samples=15
		min_cluster_size=3, min_samples=15
		min_cluster_size=5, min_samples=15
		min_cluster_size=10, min_samples=15
		min_cluster_size=2, min_samples=45
		min_cluster_size=3, min_samples=45
		min_cluster_size=5, min_samples=45
		min_cluster_size=10, min_samples=45

We notice that the best result was obtained for Dimensionality Reduction - **UMAP**, Clustering - **K-Means**, result - **0.464398**

Observations are very similar to the ones we had in the textual data. Similar to then, we observe UMAP and K-Means as the best combination:

- Sparse representation provides the least scores. This is expected as the dataset is very large. Therefore, in this case the large data causes a lot of noise which makes it difficult for any algorithm to cluster or classify the data points accurately.
- We observe that SVD and NMF perform poorly across all the various clustering techniques. UMAP outperforms all other feature reduction techniques considerably.
- We observe that UMAP provides the best results. This is due to the fact that SVD and NMF are least squares techniques and enormously liable to outliers and noise as compared to UMAP. Moreover, both SVD and NMF are linear projections of the large dimensional features which maximize the variance of the dataset, however are not able to capture nonlinear dependencies in the data. UMAP, on the other hand, tries to ensure that the semantic and non-linear dependencies, in addition to global structure of the data is preserved such that similar embeddings are clustered collectively and different ones as far apart as possible. This aids the clustering innately. In fact, as UMAP tries to preserve the information entropy regardless of the dimensions, the overall performance of UMAP is consistent throughout a huge range of r.

- Now let us understand the various clustering algorithms' performance. Both Hierarchical clustering methods and Density based methods in general do not scale well with the large dataset and large dimensions. Considering that we have a high dimensional data over multiple classes, they perform poorly.

- Moreover, DBSCAN doesn't work well as the clusters have varying density (as observed in the figure below) and that the data is high dimensional.

- Agglomerative clustering being a hierarchical clustering method suffers from the issue of large dimension. But considering that the various clusters are fairly equal sized, this does work fairly well and provides good results.

- HDBSCAN suffers from the same shortcomings faced by DBSCAN and this affects its result considerably.

- The densities of the various clusters are different and the cluster sizes are fairly equal. Therefore it makes sense that k-means work perfectly on them. k-means also scale very well to large datasets.

Question 25

Report the test accuracy of the MLP classifier on the original VGG features. Report the same when using the reduced-dimension features (you have freedom in choosing the dimensionality reduction algorithm and its parameters). Does the performance of the model suffer with the reduced-dimension representations? Is it significant? Correlate your classification results with the clustering results obtained for the same features in Question 24.

We notice that we have obtained an average accuracy of **91%** for the MLP classifier on the original VGG features.

We used various dimension-reduction techniques. We noticed a fall in the accuracy values.

Feature Reduction Techniques	**Best Accuracy Values**
With No feature reduction	93.46049
PCA with 2 Components	55.17711
PCA with 50 Components	91.55313
t-SNE	80.79019
Truncated SVD with 50 Components	91.68937
UMAP with 50 Components	82.97002

As we can see from the above table the performance of the model does suffer with the reduced-dimension representations. The features which we may think are redundant are not that redundant and get used by the Neural network. Therefore, once we perform feature reduction we are losing valuable information that was required for the classsification. Unlike textual data, the features here are pretty dense and hence losing these features can cause quite a significant drop in accuracy. as can be observed, the drop in accuracies is very significant for most of the feature reduction techniques.

The classification method does comparatively much better than the clustering method for the tf_flowers dataset. This makes sense considering that a supervised learning given the outputs will work better than an unsupervised problem. Moreover, the accuracy equivalent measure for clustering is adjusted rand index. Adjusted rand index is the similarity measure between the clusters and hence is evaluated pairwise between the various clusters. This is a very different metric from classification accuracy and hence cannot be compared directly.