ECE 219 Project 2: Data Representations and Clustering

Sarah Wilen
UCLA
ECE Department
swilen1@g.ucla.edu
UID: 305070897

Marco Venegas
UCLA
ECE Department
venmarco310@g.ucla.edu
UID: 805461921

Henry Peters
UCLA
CS Department
hpeters@g.ucla.edu
UID: 205704526

1. Question 1

The TD-IDF matrix shape: $(4732, 17131)$

2. Question 2

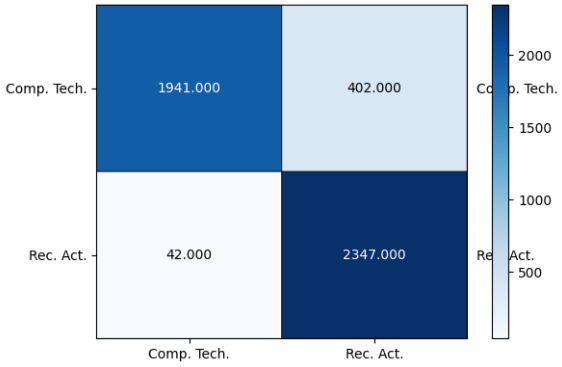

Figure 1. Contigency Matrix for means (k=2)

The contingency matrix does not have to be square. Let's say there are 2 underlying classes and we want to cluster them into 3 clusters. Therefore, contingency matrix A would have dimensions 2x3. In our example, we have a square contingency matrix because we have 2 underlying classes and chose to have 2 clusters.

3. Question 3

Homogeneity	0.589
Completeness	0.601
V-measure	0.595
Adjusted Rand Index	0.660
Adjusted Mutual Information Score	0.595

Table 1. Scores for Kmeans(k=2)

4. Question 4

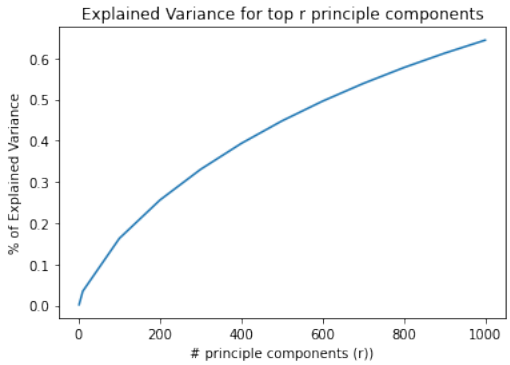

Figure 2. Explained Variance vs Principle Components

5. Question 5

Looking at the graphs below, a good choice for SVD would be $r = 20$ and for NMF a good choice would be $r = 20$

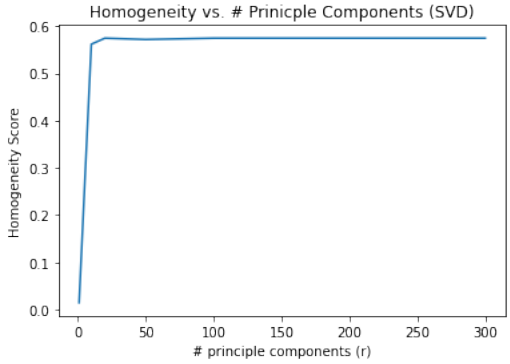

Figure 3. Homogeneity (SVD)

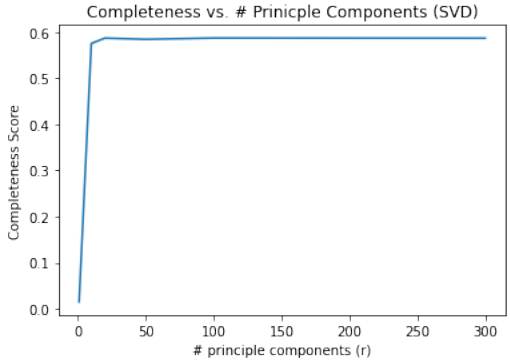

Figure 4. Completeness (SVD)

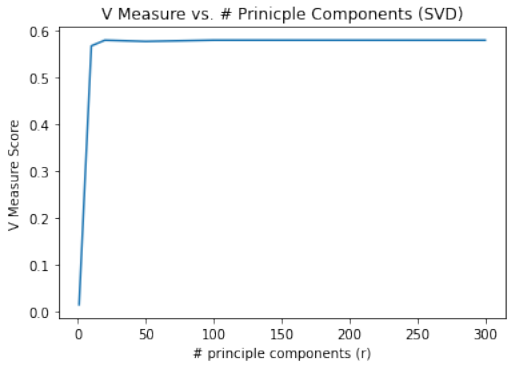

Figure 5. V-measure (SVD)

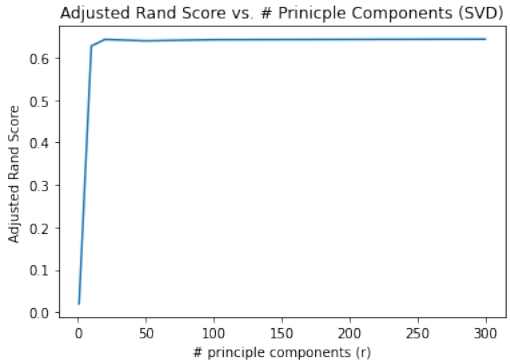

Figure 6. Adjusted Rand Index (SVD)

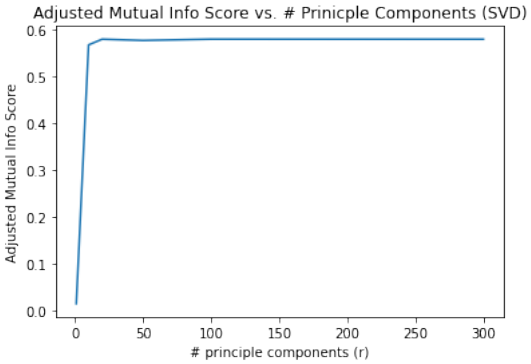

Figure 7. Adjusted Mutual Info Score (SVD)

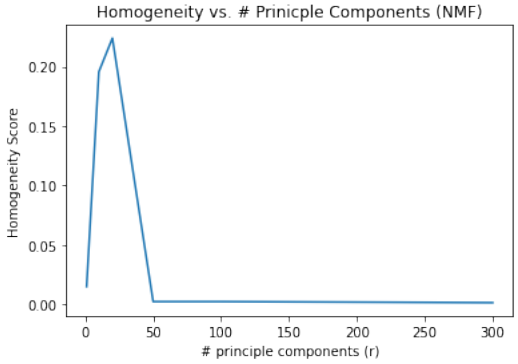

Figure 8. Homogeneity (NMF)

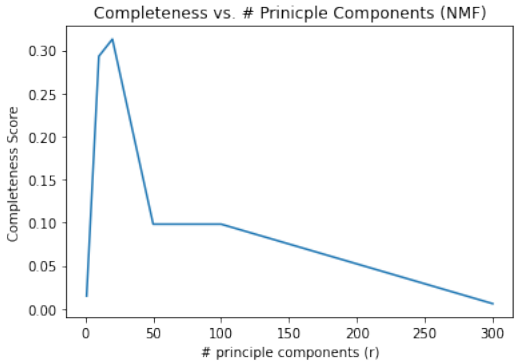

Figure 9. Completeness (NMF)

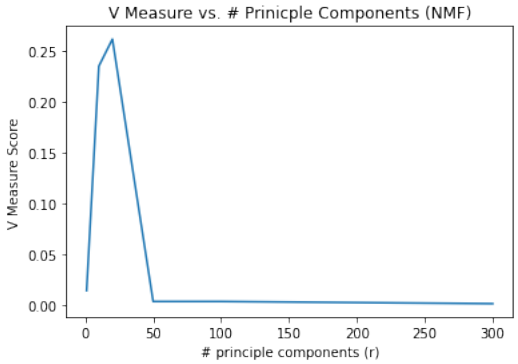

Figure 10. V-measure (NMF)

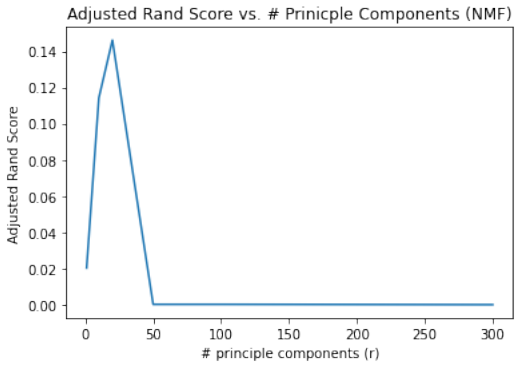

Figure 11. Adjusted Rand Index (NMF)

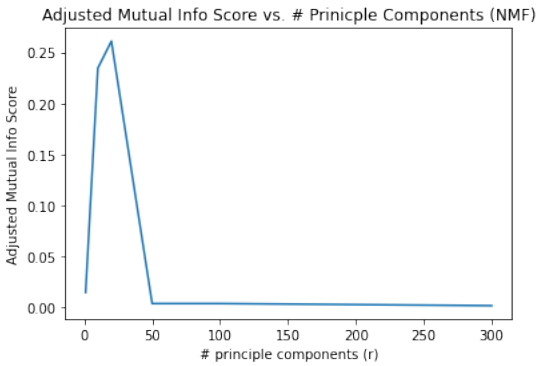

Figure 12. Adjusted Mutual Info Score (NMF)

6. Question 6

As the number of principal components increases for SVD, the score increases and then converges to a steady score. As we increase the number of principal components, we think that the score should increase too as the number of features is larger and we have a more descriptive feature space. However, we also know that as we increase the number of principal components, there is more sparsity in the data, which leads to the curse of dimensionality, so the score does not increase proportionally as much as the increase in features from more principal components may suggest an improvement.

As the number of components increases for NMF, there is a sharp increase and then a sharp decrease in score. This makes sense due to the curse of dimensionality. The higher the dimension, the larger the distance all points are away from each other, hence, the score is a lot lower. Therefore, we found the number of components that not only retains a good amount of feature information but also keeps the dimension low enough so that we don't encounter the curse of dimensionality.

7. Question 7

On average, no we do not see a better performance, which goes to reinforces our notions in question 6 about non-monotonic behavior.

8. Question 8

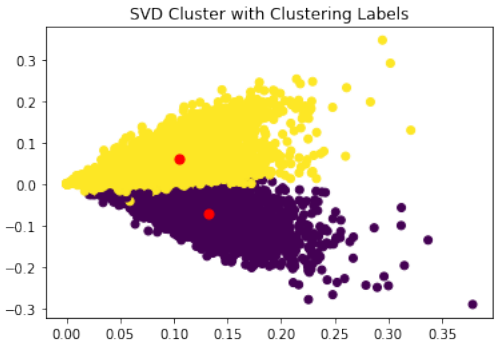

Figure 13. Data Visualization with SVD

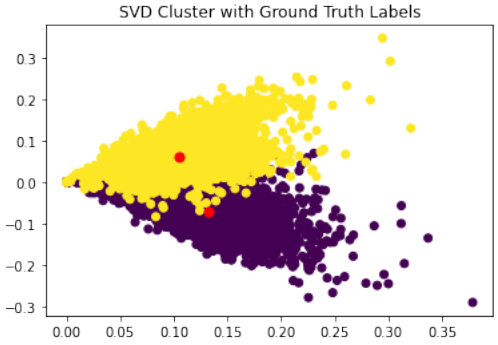

Figure 14. Data Visualization with SVD Ground Truth

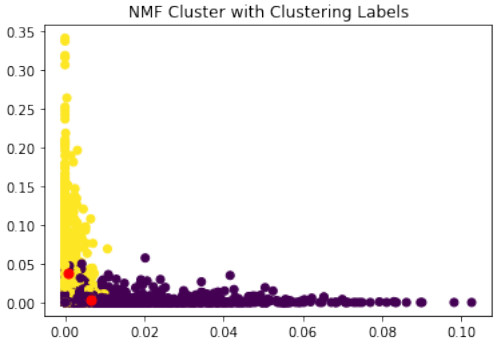

Figure 15. Data Visualization with NMF

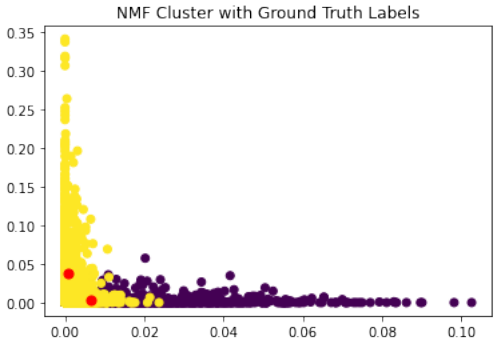

Figure 16. Data Visualization with NMF Ground Truth

9. Question 9

One thing worth noting is that none of the clusterings look spherical, as expected. Both SVD and NMF clustering take on really odd shapes, which can seem like an uneven distribution. We also see that both clustering classes have really close boundaries when labeling the data, which can explain our poor measures shown in our earlier report regarding the curse of dimensionality, especially with NMF clustering. We can see the variance for both methods is not too good, which can be seen as evidence of the noise that is introduced in our models.

10. Question 10

We started by sweeping the number of components between 1-300 for SVD and NMF on the new 20 class data in order to determine the best number of principal components to use. We noticed as we changed the number of components closer to the optimal number, the contingency matrix started out scattered with values all over the table but became more defined on the diagonal from the top left to the bottom right as we approached the optimal value, which is what we are looking for. An example of the contingency matrices for the different number of components for SVD is shown. We did the same analysis for NMF but did not include the example because it is similar to SVD.

We ultimately determined that n_components = 300 bore best for SVD and n_components of 20 was best for NMF. With this said, the scores for SVD's n_components = 300, 100, 50, and 20 were close together, so we ended up using 20 since we don't want to use high dimensions in case of (1) cures of dimensionality and (2) long processing time.

Optimal n_components: SVD - 20, NMF - 20

10.1. SVD

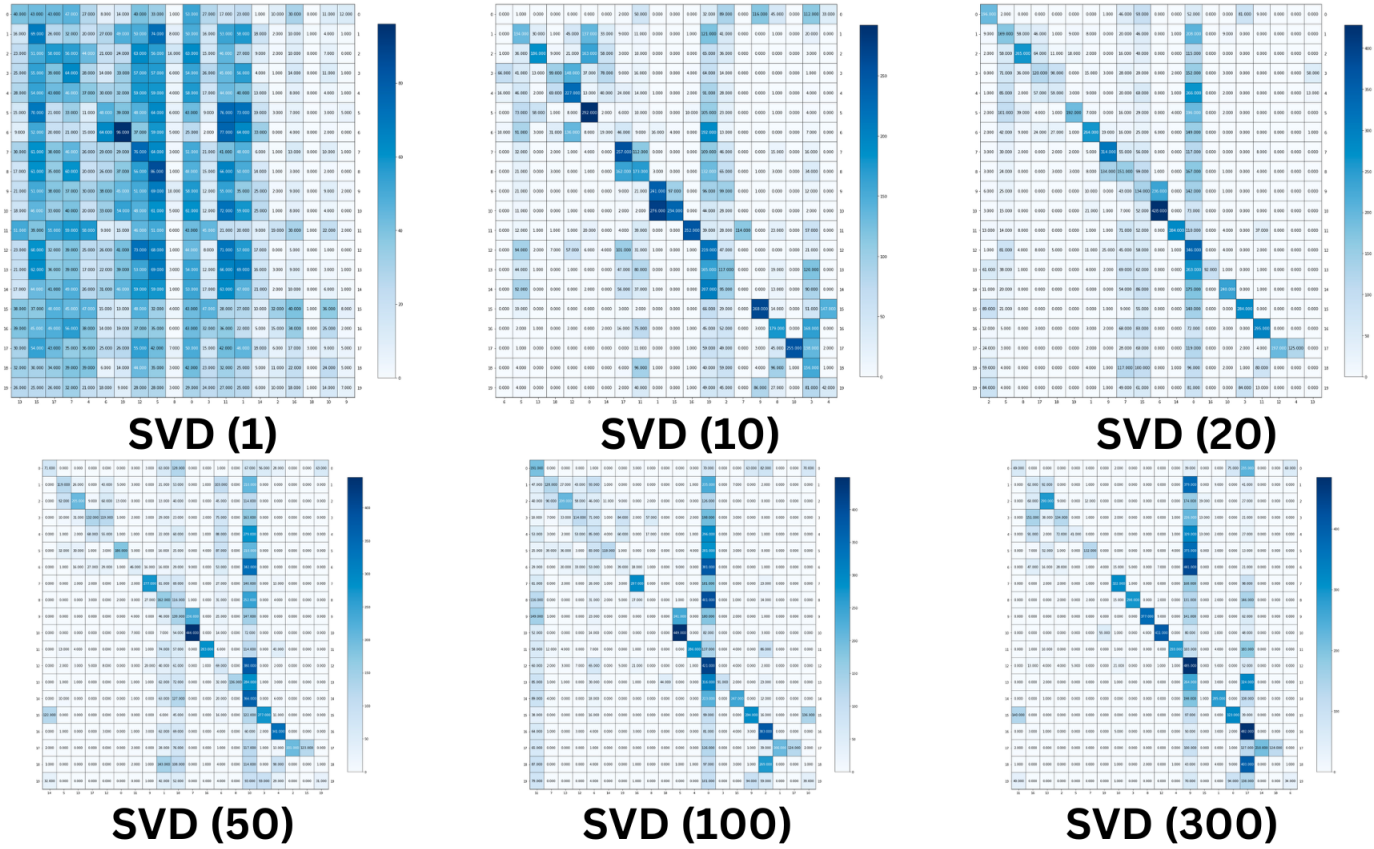

Figure 17. Example: Choosing the number of components for SVD

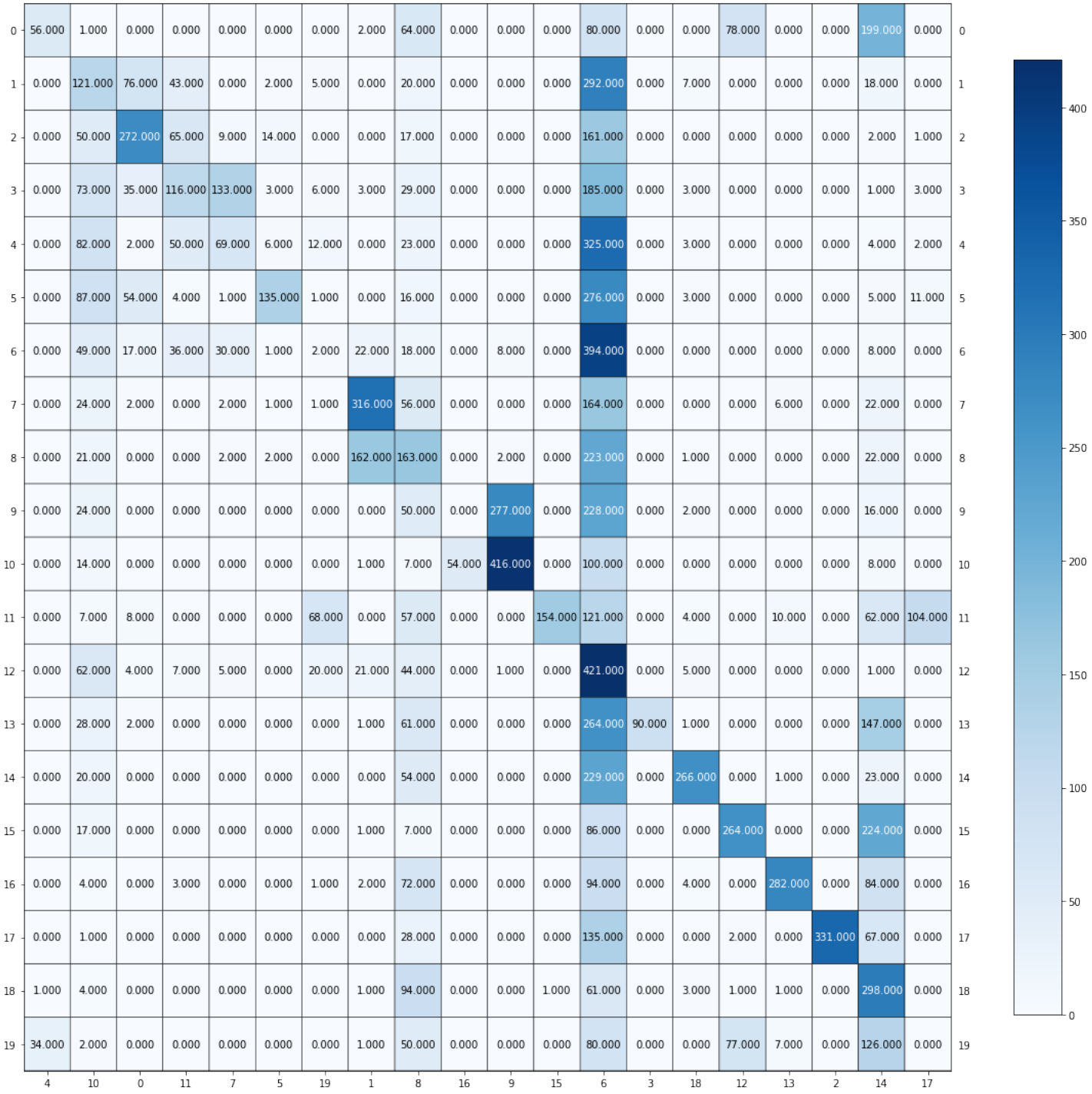

Figure 18. 20 Cat - SVD Contingency Matrix

Homogeneity	0.342
Completeness	0.431
V-measure	0.381
Adjusted Rand Index	0.100
Adjusted Mutual Information Score	0.378

Table 2. Scores for SVD K-means(k=20)

10.2. NMF

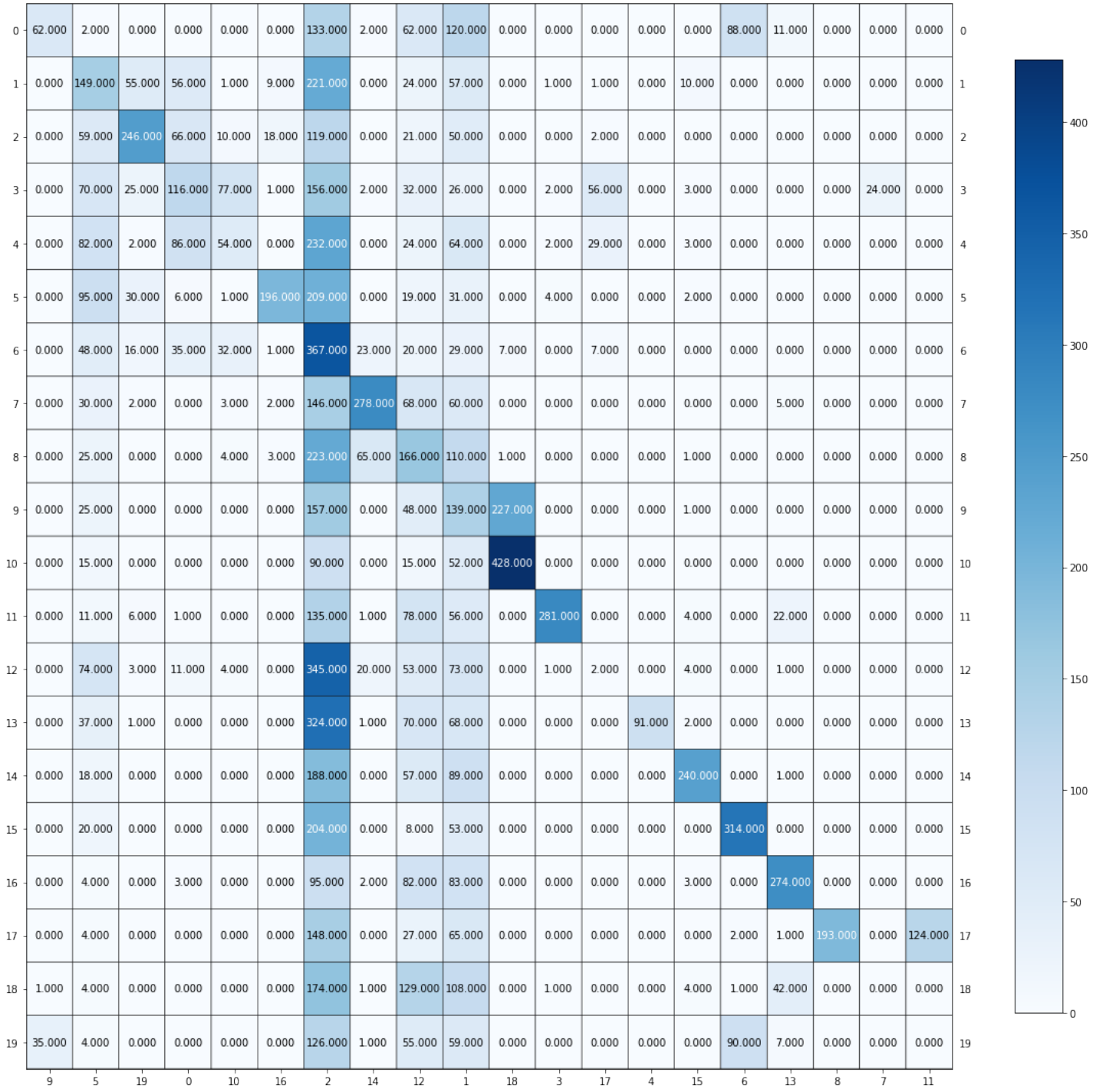

Figure 19. 20 Cat - NMF Contingency Matrix

Homogeneity	0.301
Completeness	0.379
V-measure	0.335
Adjusted Rand Index	0.079
Adjusted Mutual Information Score	0.331

Table 3. Scores for NMF K-means(k=20)

11. Question 11

11.0.1 5 Components Euclidean

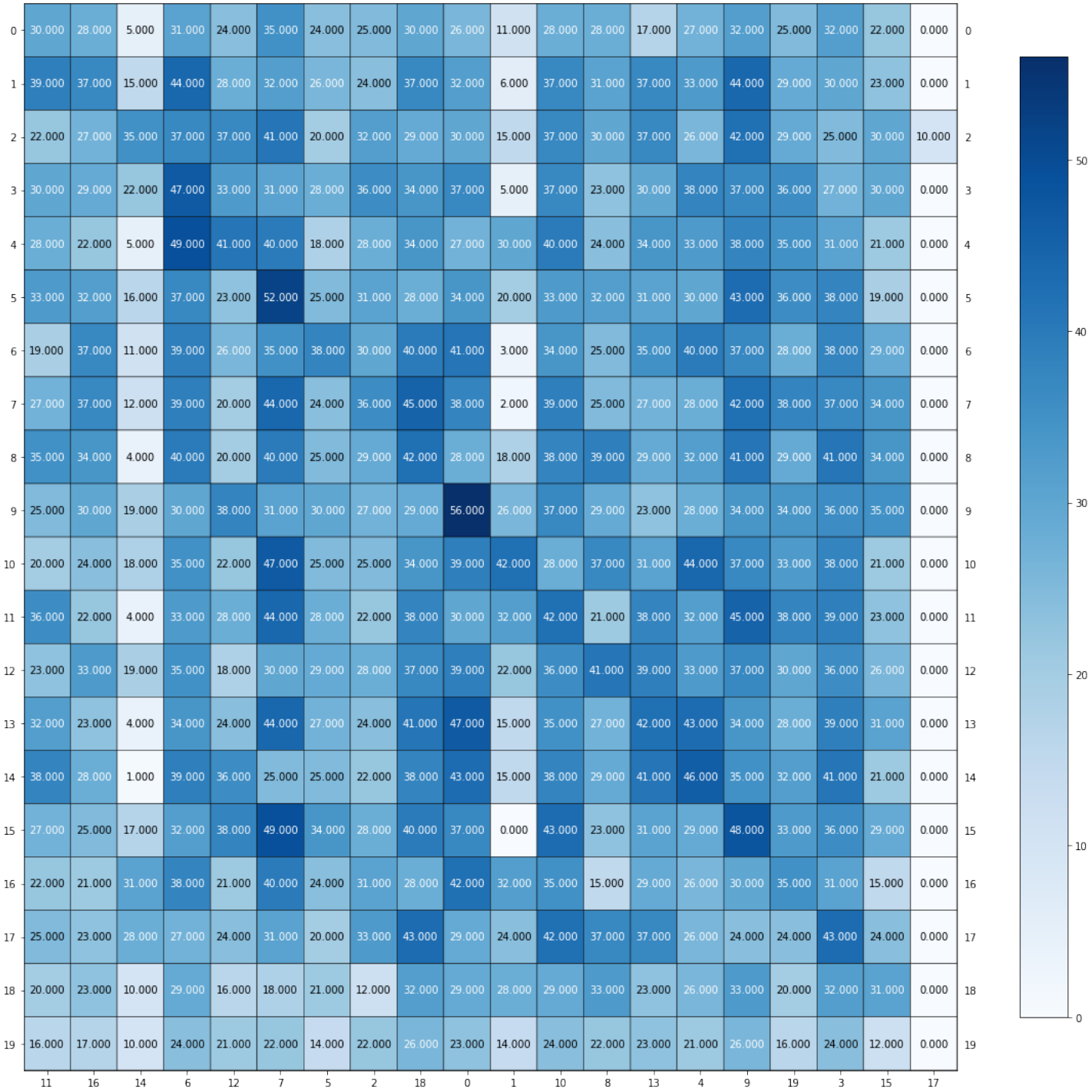

Figure 20. UMAP (5 components, euclidean) Contingency Matrix

Homogeneity	0.010
Completeness	0.010
V-measure	0.010
Adjusted Rand Index	0.001
Adjusted Mutual Information Score	0.005

Table 4. Scores for UMAP (5 components, euclidean)

11.0.2 20 Components Euclidean

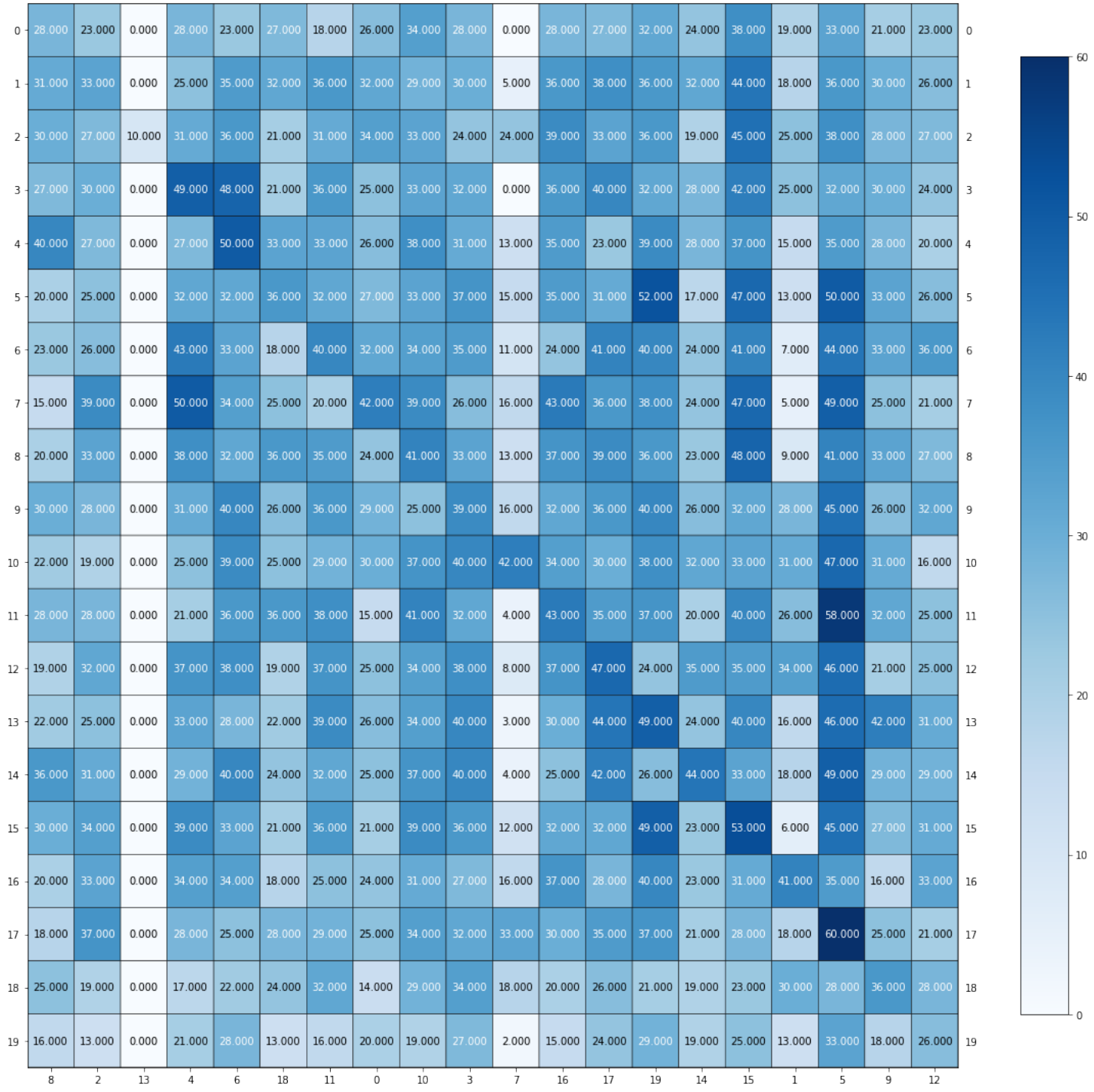

Figure 21. UMAP (20 components, euclidean) Contingency Matrix

Homogeneity	0.031
Completeness	0.023
V-measure	0.027
Adjusted Rand Index	0.005
Adjusted Mutual Information Score	0.021

Table 5. Scores for UMAP (20 components, euclidean)

11.0.3 200 Components Euclidean

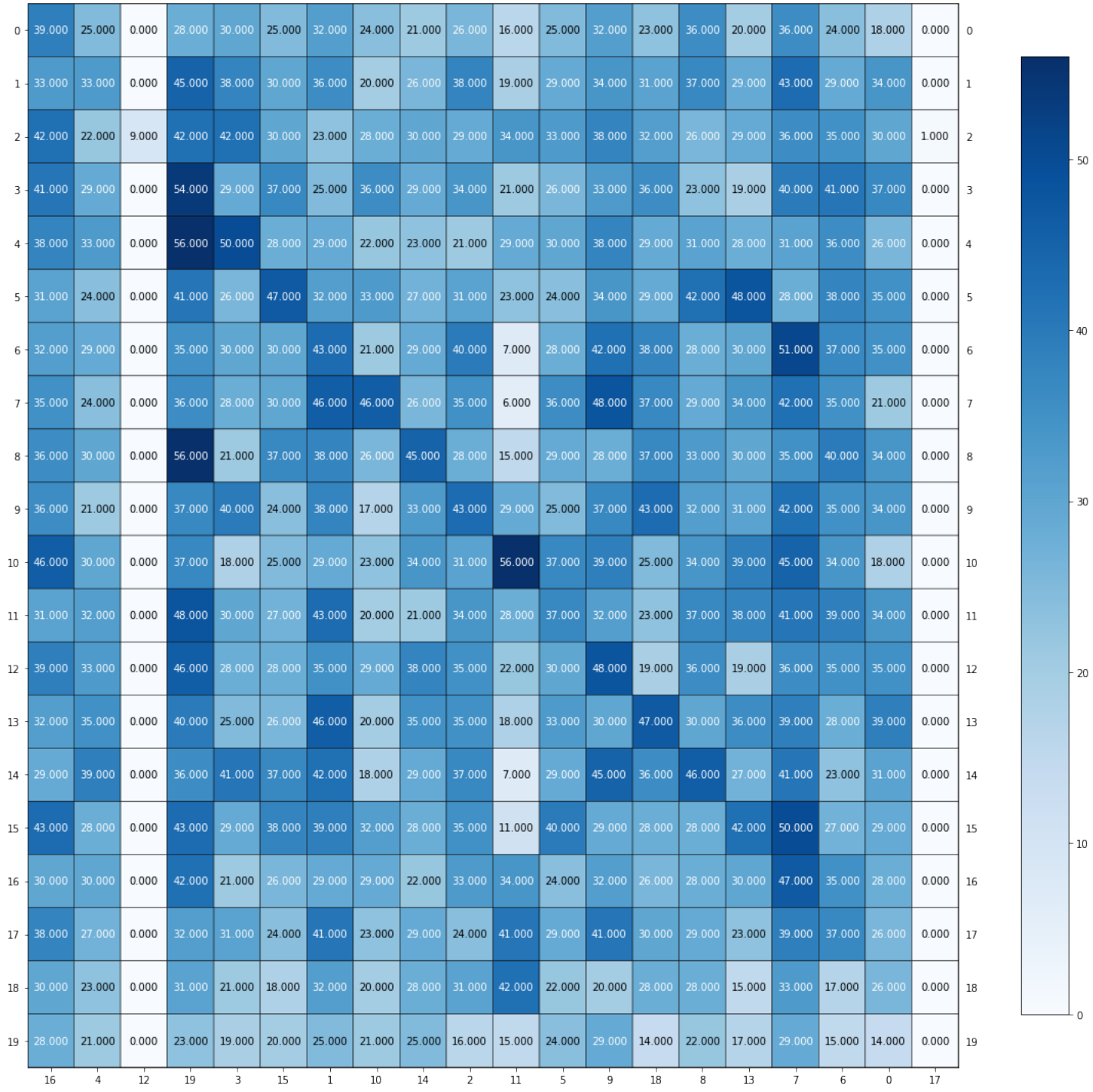

Figure 22. UMAP (200 components, euclidean) Contingency Matrix

Homogeneity	0.033
Completeness	0.024
V-measure	0.026
Adjusted Rand Index	0.004
Adjusted Mutual Information Score	0.022

Table 6. Scores for UMAP (200 components, euclidean)

11.0.4 5 Components Cosine

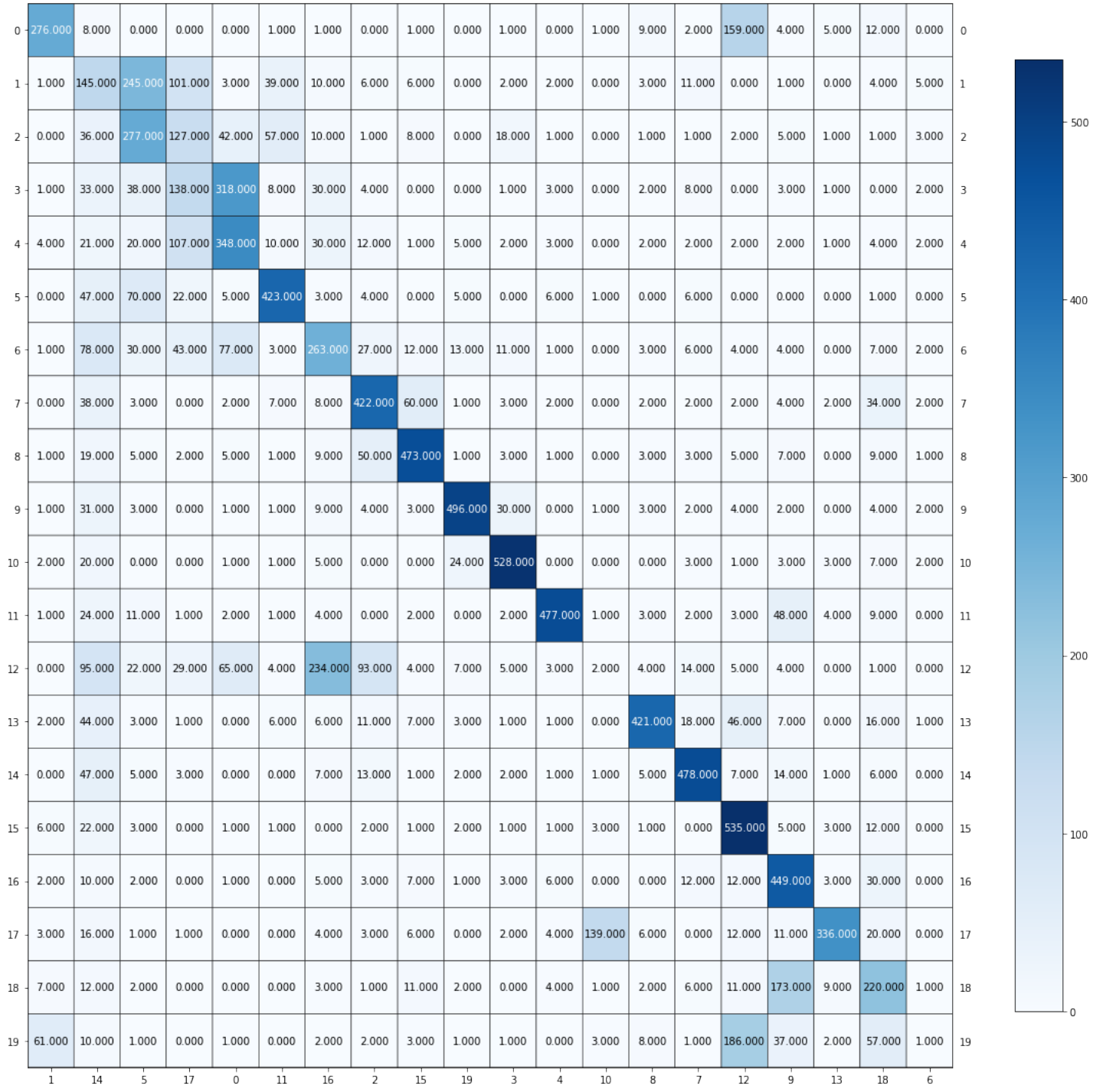

Figure 23. UMAP (5 components, cosine) Contingency Matrix

Homogeneity	0.567
Completeness	0.585
V-measure	0.576
Adjusted Rand Index	0.436
Adjusted Mutual Information Score	0.573

Table 7. Scores for UMAP (5 components, cosine)

11.0.5 20 Components Cosine

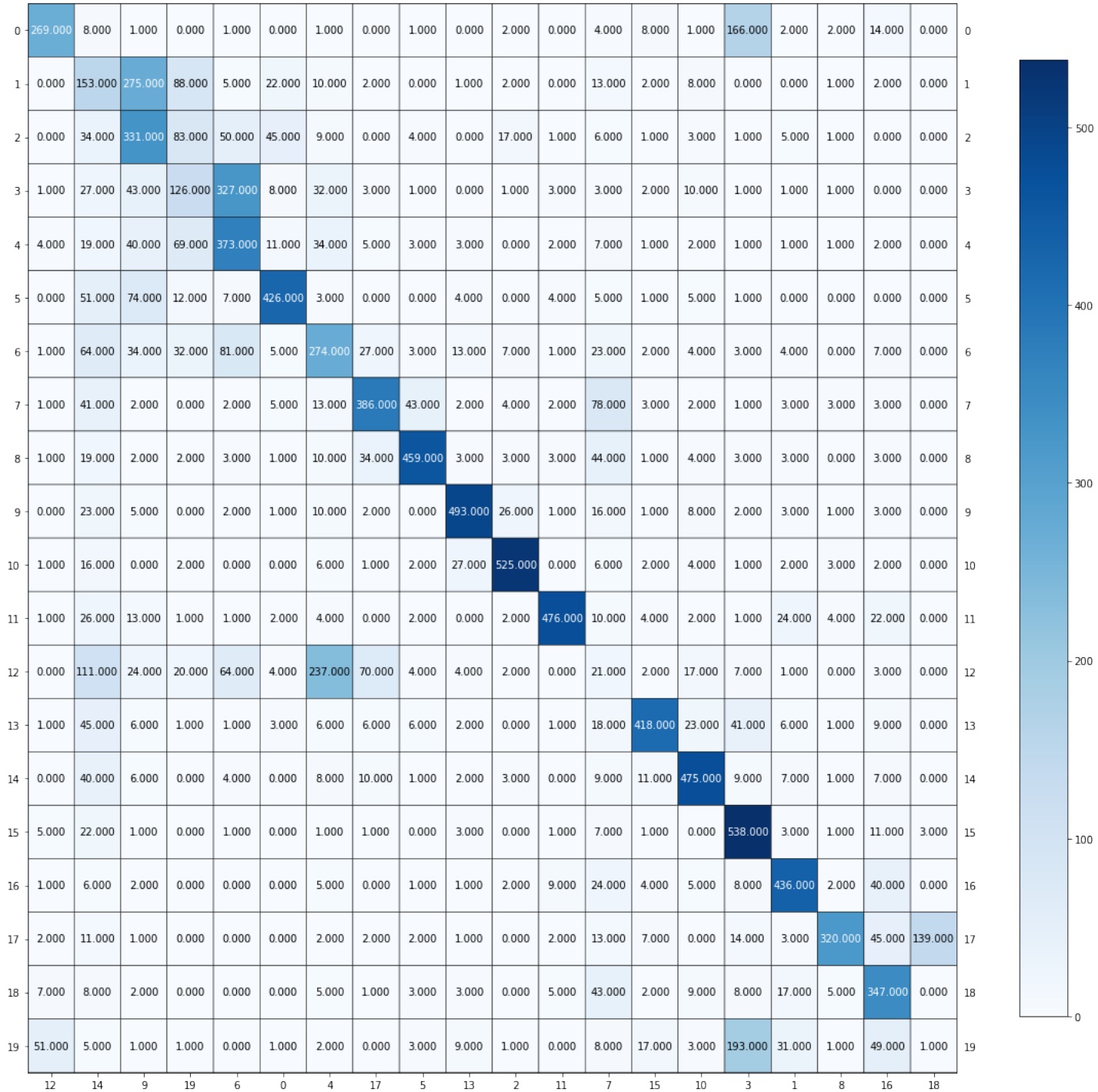

Figure 24. UMAP (20 components, cosine) Contingency Matrix

Homogeneity	0.580
Completeness	0.591
V-measure	0.585
Adjusted Rand Index	0.448
Adjusted Mutual Information Score	0.583

Table 8. Scores for UMAP (20 components, cosine)

11.0.6 200 Components Cosine

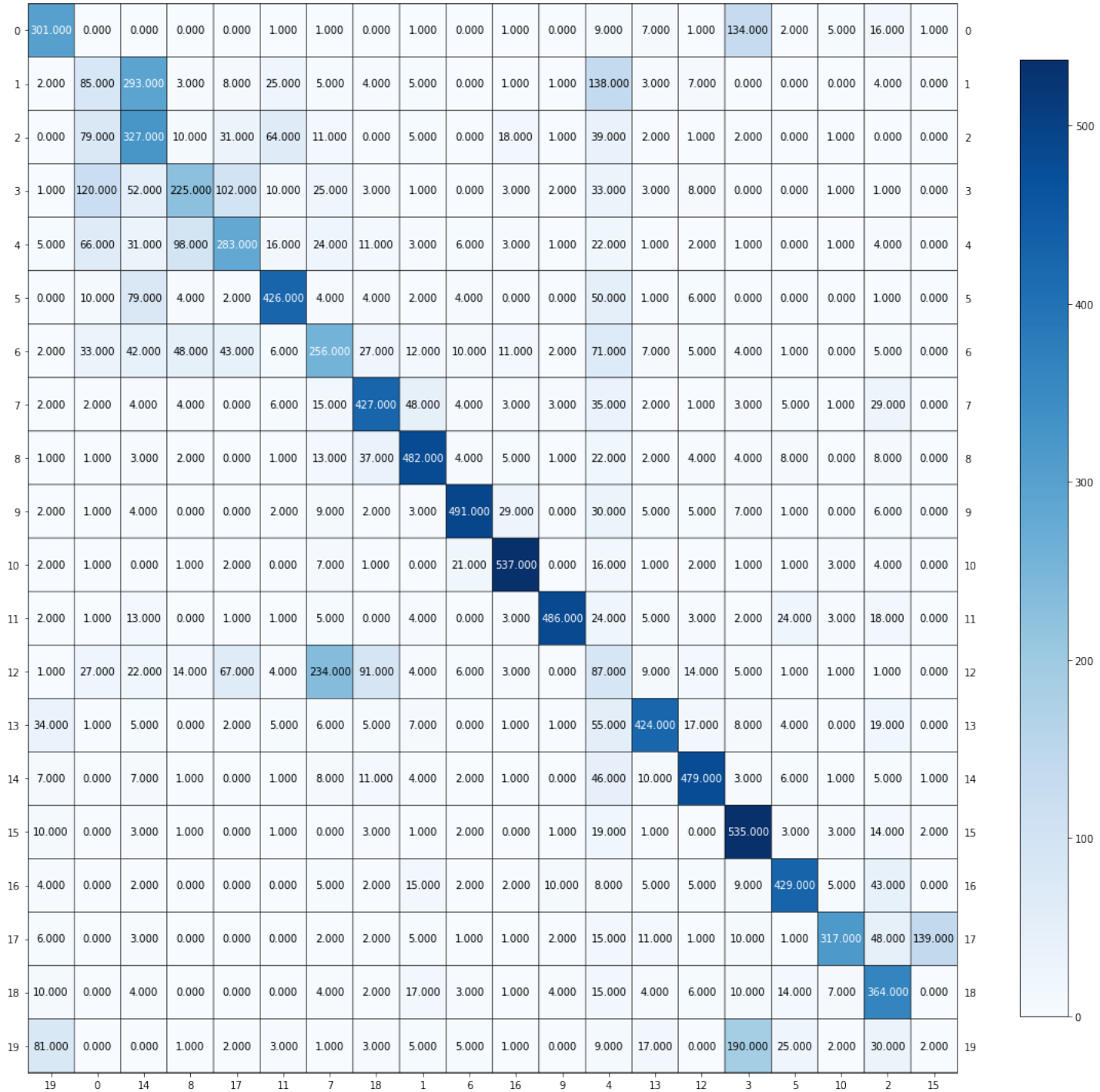

Figure 25. UMAP (200 components, cosine) Contingency Matrix

Homogeneity	0.578
Completeness	0.587
V-measure	0.582
Adjusted Rand Index	0.454
Adjusted Mutual Information Score	0.580

Table 9. Scores for UMAP (200 components, cosine)

12. Question 12

For both COS and EUC models, we got that 20 components were the best number of components. However, we can see that using the cosine method is far superior compared to the euclidean distance metric. The reason for this is that cosine is not affected by the magnitude of vectors, rather it uses the angles between sample points. So the documents can have varying lengths which lead to varying distances, but this will be factored in when using the cosine, which makes it better than euclidean. Equally important, in higher dimensions, we have mentioned that data become sparse, so in euclidean measure, the distance can converge when a document becomes too large between sample points. Furthermore, when features start to become equidistant, UMAP will not be able to find significant trends with respect to the distance of the data.

13. Question 13

Based on all 5 metrics, the best clustering method is UMAP reduced data with the cosine distance metric and using twenty principal components. SVD reduction also performs fairly well, but its scores are noticeably worse than the best method of UMAP. NMF performs slightly worse than SVD reduction, and UMAP with euclidean distance performs worst of all by a large margin.

14. Question 14

Ward linkage criteria performed better than single linkage. We may guess that this is because ward is a variance-minimizing approach, which minimizes the sum of squared distances within all clusters. Whereas, the single linkage minimizes the distance between the closest observation of pairs of clusters. In this way, Single linkage is not robust for noisy data, so did not perform the best.

14.1. WARD

Homogeneity	0.563
Completeness	0.583
V-measure	0.573
Adjusted Rand Index	0.422
Adjusted Mutual Information Score	0.570

Table 10. Scores for WARD

14.2. SINGLE

Homogeneity	0.108
Completeness	0.642
V-measure	0.185
Adjusted Rand Index	0.020
Adjusted Mutual Information Score	0.179

Table 11. Scores for SINGLE

15. Question 15

Homogeneity	0.453
Completeness	0.491
V-measure	0.471
Adjusted Rand Index	0.113
Adjusted Mutual Information Score	0.459

Table 12. Scores for HDBSCAN (min_cluster_size=20)

Homogeneity	0.382
Completeness	0.619
V-measure	0.473
Adjusted Rand Index	0.172
Adjusted Mutual Information Score	0.472

Table 13. Scores for HDBSCAN (min_cluster_size=100)

Homogeneity	0.351
Completeness	0.593
V-measure	0.441
Adjusted Rand Index	0.151
Adjusted Mutual Information Score	0.439

Table 14. Scores for HDBSCAN (min_cluster_size=200)

16. Question 16

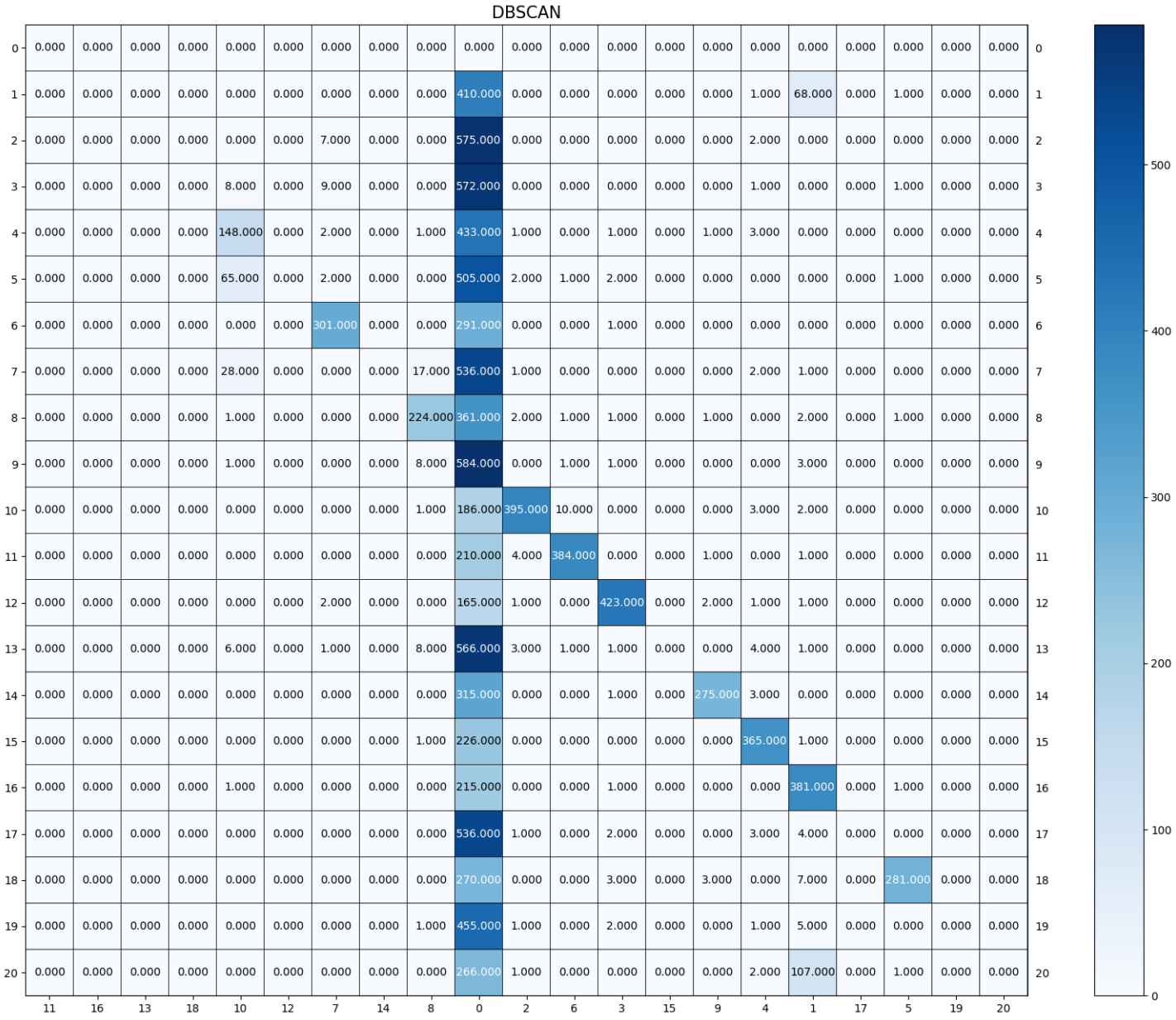

Figure 26. 20-Cat Contingency Matrix (HDBSCAN)

Print out the unique labels, we see there is [0, 10] meaning there are 11 clusters. Looking at the contingency matrix, we have 11 major clusters. We can see this in the deep blue within the contingency matrix, which can represent our clusters. Noisy samples are given the label -1, which means it is an outlier or noise that was clustered by the algorithm. We can see there are a lot of labelings that did not have many clusters associated with them. This is due to when running the code with the sample size. This is a hyperparameter than can affect our clustering. What does not help is the noise when going to higher categories.

17. Question 17

After running all possibilities of dimensionality reduction techniques and clustering methods provided in the table, we noticed that kmeans with 50 clusters using the top 20 features of NMF reduction was actually the best model. This is reinforced by the evidence from previous questions on this project. However, we knew that UMAP would actually perform the best when given the right conditions. Hence, take a look at our next question where we found a creative way to further enhance our performance.

	Dim Red	Dim Red Param	Clustering	Clustering Param	Homogeneity	Completeness	V-measure	Adjusted Rand-Index	Adjusted Mutual Information
89	UMAP	200	Kmeans	20	0.580000	0.604759	0.592121	0.457428	5.898660e-01
77	UMAP	5	Kmeans	20	0.565415	0.585196	0.575135	0.444008	5.727952e-01
91	UMAP	200	Agglomerative	20	0.567220	0.581735	0.574386	0.438778	5.720547e-01
83	UMAP	20	Kmeans	20	0.562184	0.585492	0.573601	0.421849	5.712455e-01
79	UMAP	5	Agglomerative	20	0.549878	0.573298	0.561343	0.412517	5.589183e-01

Figure 27. Top few dimensionality reduction and clustering parameters ranked

18. Question 18

Following the evidence in Question 17 and the previous UMAP question on this project, we knew that a UMAP with cosine distance metric would actually perform very well. Since we ran Question 17 with the euclidean method, UMAP did not get a chance to prove its power. Therefore, we used cosine as a distance metric in this extra credit, and to our expectation, it outperformed all of our other models. The best model combination ended up being UMAP dimensionality reduction with 200 dimensions with kmeans clustering of 20 groups.

19. Question 19

We take advantage of the transfer learning concept. We use a previously loaded neural network (VGG) and modify it to fit our scenario changing the dataset's input feature size and expected output size. VGG is trained on a task that has a larger scope than our classification problem. The idea is we know more initially and we apply it to learn something new. We take the sensitivity of VGG and apply it to different tasks.

VGG will take in an input dataset and extract important features and it will break it down to lower-level features, such as edges. The more varying different data sets fed into the neural network the lower level features it will extract, and it begins to train its weights within the neural network. When a VGG network is done learning, it can keep its trained weights. So now this network has information on a smaller data set, and it can be used as a backbone for another that will have some discriminative powers to extract lower-level features from the custom dataset.

20. Question 20

The helper code first imports the flower images from tensor flow. A FeatureExtractor object is initialized.

FeatureExtractor().eval() is to set dropout and batch normalization layers to evaluation mode before running inference. These layers prevent overfitting and stabilize training.

Then, the dataset is resized, cropped to its center, converted to a tensor, and normalized. A data loader is created from the dataset. The data loader wraps an iterable around the Dataset to enable easy access to the samples to allow us to pass samples in mini-batches (set size to 64), shuffle (benefit for training), and multiprocess.

Afterward, the features are extracted. Two arrays are created: one feature 2d and one target array. For each minibatch in the data loader, the following feature extraction occurs. For the target array, the original targets are just concatenated. For the feature array, the feature extractor is run.

From VGG-16, the feature and pooling layers are taken. The feature layer extracted from VGG-16 is 16 convolutional layers. These layers and trained weights can detect generic features from our pictures, much more sensitive than our flower classification requires (transfer learning principle). Essentially the convolution layer is a kernel filter that is a layer on top of the image. This filter is then used to extract features, where it outputs a feature map which is a reduced and summarized version of the input data.

The pooling layer is the process of merging for the purpose of reducing the size of the data. This process removes some of the noise in the data and extracts significant ones. This reduces overfitting and speeds up computation. Pooling methods

depend on the user, which is meant to reduce the spatial dimensions of the feature map while trying to retain as much information as possible. The convolution layer occurs 16 times, and the pooling layer occurs every 2 - 3 a convolution layer occurs.

Then, the image is flattened, which means it is converted into a 1D vector. This vector is fed into a fully connected network, meaning the output flattened feature is connected to every neuron, where the connection between features and neurons has weights (the weights can be trained). Lastly, we extract the first part of the full-connected network to get out classification.

21. Question 21

There are 224x224 pixels in the original image. There are 4096 features extracted. The dimension for each feature vector for an image sample is (1x4096)

22. Question 22

About 71 percent of values in the feature map are close to zero, so the features are sparse.

23. Question 23

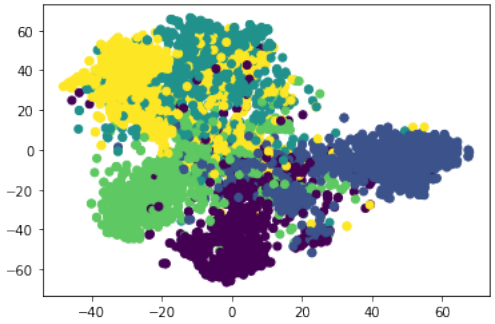

Figure 28. Plot of features projected onto two dimensions using t-SNE

There are four distinct clusters in the t-SNE projection and the fifth class's projections are intermingled with the other four classes. This makes sense since there are five ground truth classes in the source dataset. This likely means this class is more difficult to classify than the other four, and we would expect this to manifest itself in quantitative measures of classifier performance as well as the qualitative view we get from the t-SNE plot.

24. Question 24

Our best result: UMAP(cosine, n_components=50) and agglomerative clustering (n_cluster=5, linkage='ward') with adjusted rand index of 0.4998

Our conservative HDBSCAN grid was over min_cluster_size=[1, 25, 50, 100, 200] and min_samples=[5, 15, 30, 60, 100, 200, 500]. We found that the best results were min_cluster_size of 50 and min_samples of 60. Although, our resulting best parameters used agglomerative clustering.

25. Question 25

We trained the classifier on a random train test split with 80% of the data points as the training set and the remaining 20% as the validation/test set. Our classifier was able to achieve a final accuracy of 89.78% on the testing set. The classifier was trained for 100 epochs with negative log-likelihood loss and a learning rate of 10^{-3}. The performance suffers a little for UMAP and Autoencoder reduced representations (79% and 86% accuracy, respectively), but the classifier achieves similar accuracy with no dimensionality reduction or with SVD reduction. These results are a little surprising in the context of our clustering results, as this accuracy is better than the homogeneity of our clustering methods would have us believe.